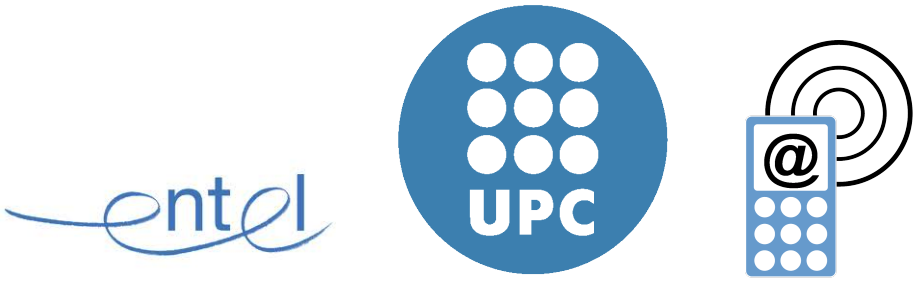
Contributions to Presence-Based Systems for Deploying Ubiquitous Communication Services



Victoria Beltran Martinez

Wireless Network Group, Department of Telematics

Technical University of Catalonia

Advisor: PhD Josep Paradells Aspas

Co-advisor: PhD Henning Schulzrinne

A thesis submitted for the degree of

PhilosophiæDoctor (PhD)

2011 December

Abstract

Next-Generation Networks (NGNs) are expected to build a single network infrastructure for ubiquitous connectivity and service access. This platform will network any person and device from anywhere and at any time through intelligent interfaces and with enriched media. The ultimate goal of NGNs is to bring value to human life through new experiences and convenient services as well as to provide a playground for everybody to create, share, compose, and deliver services. Personalization is a required feature in any next-generation service. Users should be capable to customize their services' behavior and appearance based on their needs over time. Presence information is considered as a key enabler of next-generation services' personalization. Presence information greatly contributed to the worldwide success of applications such as instant messengers. Presence information includes all the information about users that applications need to take intelligent decisions for establishing and managing user communications. SIP for Instant Messaging and Presence Leveraging Extensions (SIMPLE) is the framework that will be used to handle presence information and instant messages in NGNs. Next-generation services are expected to rely on context dissemination to a large extent, which entails scalability and performance issues in network operators. The fact that it is desirable that these services be used worldwide in order to maximize operator revenue makes scalability crucial in NGNs. However, SIMPLE is a verbose subscription-based protocol that entails much signaling traffic for maintaing subscriptions and disseminating presence updates. Thus, presence-based services may harmfully impact centralized serves in operator networks and user devices with scarce resources. On the other hand, presence-based services are not still part of everyday life. It is necessary to find out intuitive, easy-to-use services that rely on presence information for adapting to the needs of general users,

thereby incorporating technology into their daily lives. This would bring value to human life through new experiences and convenient applications. Although nowadays there is a multitude of convenient services spread over Internet, they do not cooperate with each other; they can not be composed automatically, and hence the potential of web services is not exploited. The main goal of this thesis is to contribute to the deployment and success of next-generation services. To this end, this thesis addresses the scalability of the presence service, and the design and composition of presence-enabled, value-added services. In addition, this thesis provides a comprehensible analysis of SIMPLE, and an unified view of what the presence service is and its usefulness in ubiquitous computing.

Acknowledgements

Contents

List of Figures

List of Tables

Glossary

3GPP 3rd Generation Partnership Project

ACL Access Control List

API Application Programming Interface

AS Application Server

ASE Application Service Element

BPEL Business Process Execution Language

CC/PP Composite Capability/Preference Profile

CIR Communications Initiation Request

CLCD Connected Limited Device Configuration

CLI Command Line Interface

CLP Command Line Protocol

CN Common Notify

CoA Care-Of-Address

CoAP Constrained Application Protocol

CPM Client Presence Middleware

CPP Common Profile for Presence

CPU Central Processing Unit

CS Common Subscribe

CSCF Call State Control Function

CSP Client-Server Protocol

CSS Cascading Style Sheets

DHT Distributed Hash Table

DIAL Device Independent Authoring Language

DNS Domain Name System

DOM Document Object Model

EDGE Enhanced Data rates for GSM of Evolution

EIM Enterprise Instant Messaging

EMAIL electronic mail

ETSI European Telecommunications Standards Institute

EXI Efficient XML Interchange

FCFS First Come First Served

FCS Federated Common Subscribe

FIPA Foundation for Intelligent Physical Agents

FMC Fixed Mobile Convergence

GCalendar Google Calendar

GContacts Google Contacts

GEOPRIV GEOgraphic Location/PRIVacy

GLatitude Google Latitude

GloServ Global Service Discovery Architecture

GMail Google e-Mail

GMaps Google Maps

GUI Graphical User Interface

GUP Generic User Profile

GVoice Google Voice

GZip GNU Zip

HA Home Agent

HSDPA High Speed Downlink Packet Access

HSS Home Subscriber Server

HTTP HyperText Transfer Protocol

HTTPS HTTP Secure

GLOSSARY

IETF Internet Engineering Task Force

IM Instant Messaging

IMP Instant Messaging and Presence

IMPP Instant Messaging and Presence Protocol

IMPS Instant Messaging and Presence Service

IMS IP Multimedia Subsystem

IRT Internet Real Time lab

ITU International Telecommunication Union

IP Internet Protocol

J2ME Java 2 Micro Edition

J2SE Java 2 Standard Edition

JCR Journal Citation Report

JSR Java Specification Request

LBS Location-Based System

LoST Location-to-Service Translation Protocol

MGCF Media Gateway Control Function

MGW Media Gateway

MIDP Mobile Information Device Profile

MIME Multipurpose Internet Mail Extensions

MLP Mobile Location Protocol

MPEG Moving Picture Experts Group

NGN Next-Generation Network

NO Notify Optimization

OMA Open Mobile Alliance

OS Operating System

OTA Over-The-Air

OWL Ontology Web Language

OWL-DL OWL Description Logic

P2P Peer to Peer

PIDF Presence Information Data Format

PC Personal Computer

PGP Pretty-Good-Privacy

PIDF-LO PIDF Location Object

PP Personal Proxy

PS Presence Server

PSTN Public Switched Telephone Network

PTM Push To Multimedia

PTT Push To Talk

PUA Presence User Agent

RAM Random-Access Memory

QoE Quality of Experience

QoS Quality of Service

RDF Resource Description Framework

REST Representational State Transfer

RFC Request For Comments

RFID RFID Radio Frequency IDentification

RLMI Resource List Meta Information

RLS Resource List Server

S/MIME Secure Multipurpose Internet Mail Extensions

SAP Service Access Point

SASL Simple Authentication and Security Layer

SBI Sojourn-Based Interval

SER SIP Express Router

SIMPLE SIP for Instant Messaging and Presence Leveraging Extensions

SIP Session Initiation Protocol

SMIL Synchronized Multimedia Integration Language

SMS Short Message Service

SMTP Simple Mail Transfer Protocol

SOAP Simple Object Access Protocol

SPM Server Presence Middleware

SSP Server-Server Protocol

TISPAN Telecoms and Internet converged Services and Protocols for Advanced Network

TLS Transport Layer Security

UA User Agent

UAC User Agent Client

UAProf User Agent Profile

UAS User Agent Server

UDDI Universal Description, Discovery and Integration

URI Uniform Resource Identifier

URL Uniform Resource Locator

VoD Video on Demand

VoIP Voice over IP

VS View Sharing

W3C World Wide Web Consortium

WAP Wireless Application Protocol

WBXML WAP Binary XML

WG Working Group

WiMAX Worldwide Interoperability for Microwave Access

WSDL Web Service Description Language

WSP Wireless Session Protocol

WURLF Wireless Universal Resource FiLe.

WWW World Wide Web

XCAP XML Configuration Access Protocol

XML eXtensible Markup Language

XMPP eXtensible Messaging and Presence Service

1

Introduction

Next-Generation Networks (NGNs) are expected to build a single IP (Internet Protocol) network infrastructure for ubiquitous connectivity and service access. This platform will network any person and device from anywhere and at any time through intelligent interfaces and with enriched media. NGNs will enable the desired global convergence of wired and wireless networks, which is known as Fixed Mobile Convergence (FMC). The foundation for such a new generation of communication networks is the IP Multimedia Subsystem (IMS) [1], introduced in the Universal Mobile Telecommunication System (UMTS) by Release 5/6. IMS evolves mobile operators towards an all IP technology for the support and integration of advanced multimedia services. The ultimate goal of NGNs is to bring value to human life through new experiences and convenient services as well as to provide a playground for everybody to create, share, compose and deliver services. Thus, a key factor in the success of network convergence is to provide users with value-added services that encourage them to communicate in an always-on, more dynamic way. These services will improve users' Quality of Experience (QoE) and bring greater revenues to service and network providers. Personalization is a required feature in any next-generation service. Users should be capable to customize their services' behavior and appearance based on their needs, which may change over time. Presence information is considered as a key enabler of next-generation services' personalization. Presence information greatly contributed to the worldwide success of applications such as instant messengers. Presence information includes all the information about users that applications need to take intelligent decisions for establishing and managing user communications. The presence service forms part of the IMS specification, and plays an

increasingly important role in existing and emerging multimedia services. The Session Initiation Protocol (SIP) [2] is the session signaling protocol chosen by the IMS. Thus, the SIP for Instant Messaging and Presence Leveraging Extensions (SIMPLE) [3] has become the de facto protocol for managing presence.

Three players take part in the implantation and success of NGNs: technology, operators and end users. Technology provides the means of implementating and deploying ubiquitious services, which operators provide to end users. The industry has entered in a new era in which the level of technology development exceeds the level of customer desire [4]. There is an overwhelming emergence of mobile and wired technologies; UMTS, Worldwide Interoperability for Microwave Access (WiMAX), Wireless MeshNetworks, The Semantic Web, and The Internet of Things are just some examples. This boom in technology makes the operators' investment in infrastucture more complex and costly. Such investment is restraining operators from moving towards NGNs. On one hand, operators need to adopt a customer-need-driven model for deploying the services that attract the largest number of people, thereby maximizing revenue the most. On the other hand, operators need to be sure that such revenue will compensate for the investment done and the impact of the provided services on the network. The capacity impact of next-generation services on the operator network is far from trivial due to multiple reasons. These services will be ubiquitous, device- and access-network-independent. They will in large measure rely on context information, enriched data, and social relationships. These features require complex functionality and introduce traffic load, which operators should be capable to bear for world-wide used services. Furthermore, the presence service is responsible of timely disseminating all the context needed by applications to handle user communications in NGNs. SIMPLE is a subscription-based framework, which periodically notifies subscribers of the presence information of interest. Presence subscriptions must be refreshed periodically to prevent their lifetime from expiring, which would result in their elimination. Whenever some presence information changes, all the entities subscribed to this information are notified. Moreover, SIMPLE encodes presence information by the eXtensible Markup Language (XML), which provides interoperability but results in verbose, large documents. These features cause presence-based applications to generate a great amount of traffic. Such overload may be specially critic in location-based systems (LBSs) that need to disseminate location updates very frequently and large-scale applications that distribute presence

information among million of users worldwide. IMS relies on a set of centralized SIP servers that handle all the messages sent and received by end users. The number of nodes in the signaling path and the stateful nature of these servers involve scalability issues in the IMS. Thus, the IMS presence service is particularly challenging because of its constant flows of signaling traffic, which may impact the IMS performance severely and introduce end-to-end delays.

On the other hand, context-aware computing is not yet a seamless part of everyday life. Intuitive, easy-to-use, and ubiquitous services that incorporate technology into the daily lives of general users in an always-on mode remain to be find out. Nowadays millions of people use instant messages, Short Message Service (SMS), electronic mail (email), Twitter and Facebook everyday. Technology is somewhat part of these users' life. However, these Internet services are not automated and programmable by end-users, decreasing their utility. Moreover, although these services handle very similar information (e.g., calendar, buddies status, presence, messages and user history), they do not communicate with each other. Such a lack of service cooperation and automation forces users to check services one after another and manually copy data or configure services based on other services. Unfortunately, there is currently no easy way to create new services that integrate multiple third-party services such as location, presence, calendar, address book, Instant Messaging (IM), SMS, calls, email, Facebook, and Twitter. Networked sensors and actuators for lights, temperature, humidity, smoke, and motion are also becoming popular both in residential and commercial environments. Sensor information may therefore play an important role in user-created service composition. As web services proliferate, a framework is needed where multiple services can be automatically discovered and composed for a particular user within a certain context. The Semantic Web is aimed to provide automatic web service discovery, composition and execution through ontological descriptions of web services. Such semantic-based automatic composition of web services will drastically increase the potential of web services. However, to date, there is not any platform for context-aware, automatic or user-created, service composition. A platform of this kind should be proactive and provide tools for creating compositionsintuitive enough for non-technical users.

The main goal of this thesis is to contribute to the deployment and success of next-generation ubiquitous services. To this end, this thesis addresses the above-mentioned

key issues in NGNs: the scalability of the presence service, and the design and composition of presence-enabled, value-added services. Moreover, we provide a comprehensible analysis of SIMPLE, all its technical specifications and related work. This is intended to enlighten readers about the SIMPLE framework, which is spread over numerous Internet Engineering Task Force (IETF) specifications and may therefore be hard to understand.

2

Background

The goal of this section is to provide a complete picture and state of the art of presence systems so as to help the reader to comprehend the contributions discussed in this thesis. This section discusses what presence information is, what it is useful for, and the underlying technologies involved in presence applications. Moreover, this section addresses the current and future use of presence services as well as the challenges this kind of service needs to face in ubiquitous, proactive applications.

2.1 Instant Messaging

Instant Messaging (IM) is the killer application that along with e-mail has most contributed to the success of the Internet. IM allows sending instantaneous text messages to the users specified in a contact list (also so-called "buddy list" or "friend list"). Grouping known users in a contact list differentiates IM from online chat. The success of IM stems from the perceived synchronicity of communications by users since messages are exchanged in real-time. Today's IM applications go beyond simple real-time text-based communication. IM applications usually include additional features such as video-conference, file sharing, subgroups of contacts, backgrounds, avatars, emoticons, user profiles, and offline auto-reply messages, among others. Lastly, some IM clients have integrated social networks such as Facebook, Linkedln, and MySpace. Figure 2.1 shows some typical IM interfaces, which basically list a set of known users and some convenient information to consider when initiating a communication. This "convenient information" is the birth of presence information and a key feature that differentiates

Figure 2.1: IM graphical user interfaces

IM from e-mail. Presence was conceived as an indicator of the willingness of users to communicate with others through basic states such as online, idle, busy, and so on. This novel feature was crucial for the great acceptance of IM. Without presence, a sender using short messaging might interrupt a receiver because the sender did not know the receiver's status. However, the sender can choose a better time for IM by learning the receiver's presence. A survey of 443 IM users [5] (in particular, college students in three universities in the United States) demonstrates the important role of social presence in IM. This study also finds out five motivations for IM: social utility, interpersonal utility, convenience, entertainment/relaxation and information. The authors of [6] finds that IM provides a more social experience than email communications. Other study [7] determines that students mainly instant message for social entertainment and attention.

The first IM applications, such as *write*, *talk*, *who*, or *finger*, had a great acceptance among the most assiduous users of personal computers. These basic applications did not provide, however, presence information. The first application that incorporated some basic presence information was Internet Relay Chat (IRC) in 1988. This basic information included the user's IP address, "away" status, descriptive textual message, and last input time. In 1996, ICQ ("I seek you") showed up with a simple and easy-to-use Graphical User Interface (GUI) that first provided a list of users with status icons. ICQ had a great success, and its appearance and mode of operation have been

adopted by many posterior IM applications. After that, the telecommunications magnates Yahoo, Microsoft, and America Online formed part of the IM industry with so popular applications such as *Yahoo! Messenger (YMSG)*, *Microsoft Messenger (MSN)*, and *AOL Instant Messenger (AIM)*. From then, other IM clients have emerged with their own proprietary protocols such as *Excite*, *Skype*, *Ubique*, *Brosix*, and *Zephir*. This variety of propriety Instant Messaging and Presence (IMP) protocols and clients made users run multiple client applications when they wish to connect to multiple IM networks. To ease this lack of interoperability, numerous third-party clients that connect to multiple IM services have showed up such as *Adium*, *Digsby*, *Meebo*, *Pigdin*, *QuteCom*, and *iChat* among others. These all-in-one instant messengers require users to add a user account for each IM service to which they wish to connect. Basically, these applications control the user's connections to each service, and merge each service's contacts, updates and profiles in a single GUI. The success of all-in-one instant messengers and IM users' clamor for interoperability have led the biggest IM services to move on towards interoperability. In 2006, YMSG and MSN launched clients that are able to interoperate with each other without the need to create an account on the other service. In 2007, Google and AOL announced their interoperability through the Google e-Mail (GMail) Integrated chat. However, full interoperability between Google Talk and AIM clients have not yet been launched despite of several promising announcements from the companies. Actually, these two business agreements evidence a growing battle for the largest IM customer database. The two camps, with Google and AOL on one side and Yahoo and Microsoft on the other, support competing IMP protocols. The former supports eXtensible Messaging and Presence Protocol (XMPP) and the later SIP for Instant Messaging and Presence Leveraging Extensions (SIMPLE), both part of standards-making processes in the IETF. XMPP has received a great acceptance, and is used by popular IM applications such as *Google Talk*, *Facebook*, and *Gizmo5*. Most multi-protocol clients interoperate with XMPP as for example *Adium*, *eBuddy*, *Digsby*, *Emphaty*, *Palringo* and *IChat*. SIMPLE took form later, after the acceptance of XMPP, which is the main reason why less IM clients support this protocol. Some IM clients that do support SIMPLE are *YMSG*, *QuteCom*, *Pidgin*, *SIP Communicator*, and *MSN*. IM has already reached the mobile world. IM providers such as Yahoo, Skype, Microsoft and AOL already provide support for mobile platforms such as mobile phones, laptops, and smart phones. In addition, numerous multi-protocol clients have

been launched to run on portable devices such as *MXit*, *Yamingo*, *Fring*, *Palringo*, *IM+*, and *Beejive*. Mobile IM is also offered by browser-based applications such as *Meebo* and *eBuddy Web Messenger*, which do not require downloading any software to the handset.

Although IM was initially conceived as a social tool, specially popular among young people, it has been also adopted in business environments. The authors of [8] show the results of an extensive survey of IM users in the United States in 2004. This survey revealed that by then 53 million American adults used IM. Around 21% of them were used to IM at work, which increased productivity and interoffice cooperation. As well, other survey [9], which was carried out in the United States in 2006, concludes that IM in the workplace promotes more frequent communications and reduces interruptions. Other studies [10][11] state that IM is desirable at workplaces due to its immediacy, presence features, and rich communications. The works [12] and [13] analyze IM applications' traffic in corporation environments and show that IM is widely used during working hours. In response to the demand for business-grade IM and the communication security that corporations need, new Enterprise IM (EIM) solutions have emerged such as *IBM Lotus Sametime*, *Microsoft Office Communication Server*, and *Cisco Unified Presence*. EIM platforms are aimed to medium and large companies by offering more Quality of Service (QoS) and interoperability with multiple IM services.

A recent study [14] conducted in 2010 reveals that there are 2.4 billion IM accounts worldwide, and this number will grow to 3.4 billion by 2014. The authors of [15] analyze behavioral data from the Microsoft Messenger system on a planetary scale. The dataset contains 30 billion conversations among 240 million people. A survey of Internet users in New York [16], which was carried out in 2009, shows that the amount of time spent with communication tools such as IM and email has declined by 8% since 2003. One of the reasons of this decline is the extreme success of social networks in the last years. This does not mean, however, that IM is past history. On the contrary, users still instant message but, instead of using traditional IM clients, social networks allow them to instant message a wider audience, from different platforms and with more context awareness.

2.2 Context Information

The word *context* is defined as the situation in which something happens and that helps you to understand it. Context plays a key role in smart environments that proactively adapt to users' needs and preferences by examining their surroundings and circumstances. Context-aware services need to obtain, interpret, describe and disseminate context information. This involves monitoring the resources and entities that participate in the service. It is therefore necessary to reach a consensus about what context information is and how it can be obtained and manipulated. In the frame of telecommunications, multiple classifications of context have emerged according to entity, role and low- and high-level information, for example. The authors of [17] provide the possibly most accepted definition of context information. This definition says that context information is any information that is susceptible of characterizing the state of an entity. Entity can be any element that is relevant in the interaction between an application and a user. The state of an entity includes innumerable aspects such as physical and computational characteristics of components that interact with the system, history of user activity, preferences, and other kinds of personal information. This definition of context information fits those that have been proposed by the main standard organizations: World Wide Web Consortium (W3C), Open Mobile Alliance (OMA), 3rd Generation Partnership Project (3GPP), and Foundation for Intelligent Physical Agents (FIPA). Based on the kind of entity to characterize, there exist different classifications of context information. For example, user context could include a diversity of entities such as the user himself/herself, other users, and his or her environment, location, preferences, circumstances, etc. Context-aware services use this information for adapting their behavior to the users' needs. This adaption could be targeted at providing diverse functionalities: user-personalized contents, device independence, multimodal interactions and so on. Due to the broad nature of context, real systems need to limit the number of entities and their characteristics according to their needs. These systems should carefully study the most appropriate methods for abstracting, representing, merging, disseminating, and reusing context information. These tasks are specially challenging since context information can come from diverse sources as for example sensor networks, localization services, operator networks, and third-party information repositories. Intelligent systems should be capable to inference

new context information from that obtained from context sources. For instance, the user's activity at a given moment may be deduced from his or her location and calendar information.

Context-aware services typically have a partial view of a problem and focus on concrete applications and environments. Context-awareness can be found in the Ubiquitous Web [18], agent-based systems, presence services, Peer-to-Peer (P2P) applications for file sharing, sensor and actuator networks, user-personalized services of mobile operators, multimedia contents, etc. Such countless context-aware applications have led to many frameworks for context handling that overlap each other. To provide interoperability, a number of proposals have come out as Table 2.1 shows. The great number of proposals along with the heterogeneity of devices and protocols make interoperability specially challenging.

Context type	**Proposed Standards**
User	GUP [19], CC/PP [20], UAProf [21], SIMPLE [22][23]
Service	GUP [19], SIMPLE [22][24], OWL-S [25], BPEL [26], WSDL [27], UDDI [28], FIPA Ontology Service [29]
Content	MPEG-21[30], CSS3 (Media Queries) [31], SMIL [32], DIAL [33]
Device	FIPA Device Ontology [34], CC/PP [20], UAProf [21], WURLF [35], Device Description Repository [36], SIMPLE [24]
Location	MLP [37], GEOPRIV [38]

Table 2.1: Proposals for standardizing context information

2.3 Presence Information

Presence information is a well-known concept on the Internet and is widely used by applications such as IM and Push-to-Talk (PTT) as explained in Section 2.1. In these applications, a user can discover the willingness of other users in his or her buddy list to communicate with him or her, through the presence states of online, offline, busy or absent, among others. This basic understanding of presence is evolving towards a much more generic and flexible concept that includes all context that allows a user or application to adapt and control communications in a more efficient, personalized manner. Presence includes a wide range of information about a user, such as his or her localization (at different levels: region, place or GPS coordinates), the activities that

the user is doing at a specific time, ambient conditions, communications preferences, and devices on which the user is available.

Although the concept of presence is understood as a type of context information, the distinction between context and presence is blurred. We know that all presence information is context but not all context information is presence. Thus, we should answer the question *What makes some context information be defined as presence?*. We answer this question by two parameters: the object and the goal of the information. Context characterizes all relevant entities in any kind of interaction between a user and an application (i.e., the object). This information helps applications to build all kind of intelligence that is aware of the user's environment (i.e., the goal). On the other hand, presence is used to characterize entities that can affect the management of user communications (i.e., the object). This information allows applications to take intelligent decisions about the start, continuation and end of user communications (i.e., the goal).

The traditional view of presence revolves around communicating with a human being. However, the concept of presence is evolving towards a more powerful and pervasive concept that is not restricted to human users. The research community reflects this evolution. For instance, the authors of [39] and [40] highlight the need to have presence systems capable of describing entities other than human users such as objects and places. The former includes an attribute in presence documents that indicates the type of presentity. The latter describes an IM application that allows interacting with physical things such as printers and rooms in a buddy list. The patent [41] models rooms, guests, and accommodation services through presence information. Other patent [42] models web services through presence documents. Thus, we propose a revised definition of presence as follows:

> "Presence is any information that characterizes the state and nature of an entity that is capable to communicate with other entities. This information is relevant to take decisions about the entity's communications or can have some kind of influence on the way others communicate with this entity. Here, the term "communication" should be understood in a wide scope. It not only means virtual communications based on text, audio, and video, but also real-world communications and any interaction relevant to the entity.

> Entity could be a human being, a software process, or a physical device, for example. An entity can be indivisible (e.g. a printer or human) or composed by subentities (e.g., a smart room, a corporation and a communication network). A composed entity's presence information is formed by its subentities' presence information."

2.3.1 Advantages and Uses Cases

Presence information offers a world of attractive possibilities for self-expression by letting our friends and contacts know how we are and by seeing how they are in an instant. We are thereby able to choose the most suitable time to contact our buddies since we know when they are most available and the condition in which we will find them. Thus, we can avoid failed call attempts that sometimes end in voicemail, which in turn allows us to save money. As described in Section 2.1, users of IM applications rely on presence information to handle their communications that, apart from instant messages, include voice calls and videoconferences. There are many authors that have addressed the use of presence information for **call handling**. For example, the authors of [43] describe a system for call forwarding based on the caller's and callee's presence information. The authors of [44] describe a router that uses presence information for handling communications and in general building application chains. UbiPhone [45] is an ubiquitous phone service that automatically handles phone calls based on the caller's and callee's presence status, location, calendar and social relationships, for example. A similar but simpler application is Live AddressBook [46], which helps users make telephone calls by providing their contacts' dynamic presence information. LESS [47] is an XML-based scripting language for call handling that can use some basic presence information (e.g., time and online status) to trigger actions.

Presence information offers other definite advantages beyond communication handling. Presence and mobility are intrinsically connected. Mobile devices are personal and pervasive; the user always keeps them close in a manner always-on, and stores personal information in them, such as a diary or favorite media. Presence-based mobile applications bring out exciting possibilities. An experimental study of the use of mobile social presence [48] from the Motorola Social Media Research Lab probes the usefulness of presence in users' daily life. Participants in this study used three presence applications that let users know their contacts' motion status, played music, and posted

photos. This study found out that participants used contextual presence to coordinate their daily activities. Thus, mobile presence systems can help people manage their everyday coordination tasks in a less disruptive, more natural way. The authors call it **perceptual microcoordination**. The authors also report other two advantages of presence information: **constant awareness** and **shared experience**. Constant awareness refers to the ability of users to connect to others' rhythms and activities throughout the day, without the need to communicate. This provides a feeling of connectedness and safety. Shared experiences let participants feel a connection to others whom they could not physically be with at a given time. As the authors state, all these advantages can persuade people to initiate communications, thereby providing better experiences to users and increasing service providers' revenues.

As presence is innately associated to the mobile world, location information is a key part of presence information. Buddies' location information has become a hot feature in some IM applications. AOL Instant Messenger already provides a plugin that enables users to check out the location of their buddies. Chatsquare [49] allows discovering and communicating with nearby people. BuddyFinder [50] tracks the location of the users' buddies through SMSs. The authors of [51] propose an algorithm for detecting when any contact gets into a user-defined vicinity. ContextContacts is an application on the ContextPhone platform [52] that aggregates the contacts' presence information from multiple sources. This information includes user location and close-by friends. The authors of [53] describe a mobile workforce management platform that integrates the employees' presence information with their vehicles' status and location information.

A new emerging field of research and posibilites comes out from the integration of presence information, specially location, and social networking. Foursquare [54] is a phone application that allows Facebook's, Twitter's and other social networks' users to share their location regarding points of interest and presence status. The authors of [55] describe an application for gathering presence information from sensors embedded in mobile phones and exporting this information to Facebook. NFCSocial [56] is a mobile application that uses Near Field Communication (NFC) technology to determine the users' location and to ease the update of presence information. Whenever a user's location is detected via NFC, her presence information is generated and updated in her social networks. The user's presence information includes a picture that represents the user's location and other picture for her mood. R-U-In? [57] is a social networking

system that relies on users' activities and other presence information to help users participating in activities of mutual interest.

The above-mentioned use cases are just a few examples; the most popular ones. Enumerating all the use cases would be impossible because the possibilities of presence information are almost infinite and multidisciplinary. The following works shows different areas in which presence information can also be useful. BusinessFinder [58] is an application for matching customer requests to nomadic vendors. This relies on customers' and vendors' presence information to find out the nearby vendors that are available and best fit the requester's needs. The patent [41] proposes a presence-enabled property management system in which rooms, guests and services have presence information associated. The patent [42] describes a system that model web services by presence information. Thus, a users can group interesting services in a contact list and access any of them from this list. The recommendation system for cellular networks described by [59] considers users' presence information to decide whether or not to send out recommendation messages. The authors of [60] propose the presence service as a mechanism for discovering web services. The work [61] describes a prototype social television system that incorporates user presence and messaging.

2.3.2 Requirements on Presence Systems

A presence-based application should only permit a user to communicate with another user by the services specified in the presence document of the latter. In addition, this communication should have the characteristics indicated in that presence document, for example, regarding target devices and content types allowed. Thus, users can restrict the way other users communicate with them by publishing information to this effect in their presence documents. For example, a user may determine which communications to accept depending on the relevance of the requesters, redirect calls to secretaries or delegates when she is busy, accept different content types depending on her circumstances (working, traveling, etc.), and so on.

Presence information has evolved into a much more complex and diverse data than the first binary states online and offline. Such basic states came from a single IM desktop application. Now, presence information may come from multiple sources such as cellular phones, sensors, Personal Computers (PCs), calendar applications, location-based systems (LBS), etc. Presence systems need to merge all these sources' information

to create an unified view of the user's presence information. This merging process could detect information conflicts between two or more sources. Thus, an important requirement that any presence system should support is **composition policy**. The goal of a composition policy is to eliminate obsolete, redundant or contradictory information, and possibly generate new presence information that is inferred as a result of composing. The diversity of presence information also leads to other requirement: **extensibility**. A presence system should be capable to extend the users' presence information with new information in a non-intrusive way for users. Due to the personal character of presence information, other two important requirements are **preference policy** and **privacy policy**. Exigencies on a presence application can greatly vary from one user to another. The user's exigencies can even vary over time. For example, one user may only be interested in his or her workmates' presence during working hours, while other user may only wish to receive presence updates about his partner at any time. Preference policies are therefore necessary to personalize presence application based on the users' needs. Privacy policy is an indispensable feature of presence applications for avoiding disclosing private information to users that are not allowed to see such information. Figure 2.2 shows a schematic representation of how presence information should be processed by a presence system that satisfies the aforementioned requirements. The left-hand box represents the retrieval and composition of the user's presence information. The user is connected to several devices that let the presence system know about the user's presence information. The presence system merges the information from all the sources according to a composition policy. This process produces the raw presence document. The right-hand box shows the tasks that are accomplished when other user is informed about the user's presence information. The raw presence document is filtered according to the user's privacy rules about the recipient. This generates the public presence document for the recipient. Before this document is delivered to the recipient, it may be filtered again according to the recipient's preferences.

The academia is actively putting efforts to develop presence-based systems that satisfies the above-mentioned requirements. There are many authors that have addressed these issues, and we mention the most relevant ones: The authors of [62] and [63] describe presence aggregation and composition within the SIMPLE framework (see Section 2.7). They also present an XML format for users to express preferences on presence composition. The survey [64] is a complete description of presence management

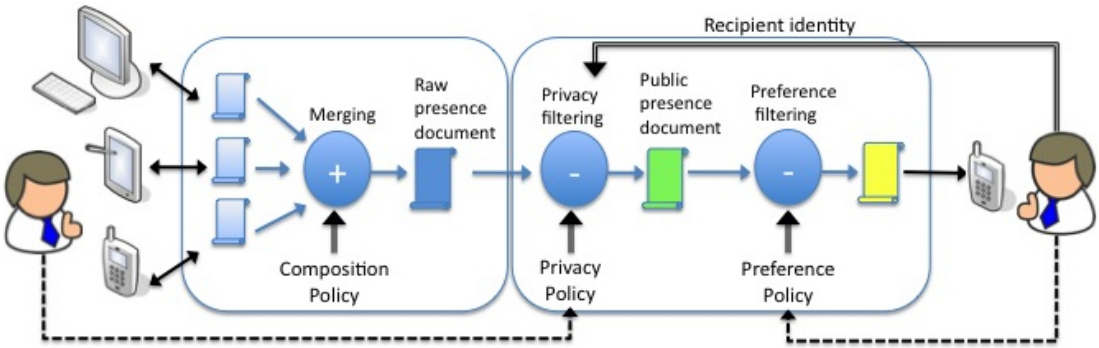


Figure 2.2: Processing of presence information

and its benefits, requirements and challenges. In [65], a hierarchical architecture for retrieving presence information from distributed sources is described. The authors of [55] and [66] aggregate information from sensors for forming the user's presence information. Moreover, the work [66] also allows users to determine how presence information is aggregated for each recipient. The authors of [67] infer the willingness level of the callee to accept calls based on the time of the day, call duration and the location. The work [68] aggregates information from multiple sources for inferring a more complete and informative description of the activity that the user is doing at a given time. The application described by [69] infers presence information from the data gathered by built-in microphones on laptops computers, access points and user calendars. These sources allows inferring the user's state (working and/or busy), activity and location. The authors [53] describe a system that uses several sources for inferring and composing the presence information of a company's employees. The presence sources are the IM applications in which the employees are logged and their cell phones' and vehicles' location. The authors of [70] propose a method for composing the presence information of the user's buddies based on activities or events in common. This work also allows users to specify privacy policies.

Due to the diverse nature of presence information, extensible formats are needed to encode this information. The standard presence frameworks (see Section 2.6) use XML-based formats, which define a basic but extensible set of common presence. Some research works have already proposed extensions of XML-based presence formats, as instance, for functions that describe the exactness of information decreasing over time [66], information about vehicles [53], information about web services [60], non-human

entities [39] and abstract things [71]. Other authors defend the use of ontologies for modeling presence information such as those of [72], [73], [74], [68], [57] and [64].

2.4 Fixed Mobile Convergence

Fixed Mobile Convergence (FMC) means the convergence of the existing wireless and wired networks independently of the end device's characteristics and network access. This network convergence is described straightforwardly by the survey paper [75]. A key factor in the success of network convergence is providing services with new and value-added services that encourage them to communicate in an always-on, more dynamic way. These services will improve users' Quality of Experience (QoE) and bring greater revenues to service and network providers. Personalization is a required feature in any FMC service. Users should be capable to customize their services' behavior and appearance based on their needs, which may change over time. Presence information is considered as a key enabler of FMC services' personalization.

FMC is based on an all-IP approach that allows any IP-enabled device to access any service regardless of the access technology. It is expected that this convergence will rely on the introduction of IMS into Next-Generation Networks (NGN) for allowing users and next-generation services from radio and fixed broadband access networks to communicate and interoperate. Section 2.4.1 briefly describes the IMS architecture, which enables SIP-based multimedia communication services. NGNs define a single network infrastructure for networking any person and device from anywhere and at any time through intelligent interfaces and with enriched media. To achieve the promised global convergence, device-independent mobile applications that are capable to adapt their contents based on the devices' hardware characteristics are necessary. Section 2.4.2 outlines this issue. The ultimate goal of NGNs is to bring value to human life through new experiences and convenient services as well as to provide a playground for everybody to create, share, compose and deliver services. International Telecommunication Union (ITU) and European Telecommunications Standards Institute (ETSI) have been actively working on the NGN standardization from 2003. Two fundamental recommendations on NGN have been produced and approved: Y.2001 (General overview of NGN) [76] and Y.2011 (General principles and general reference model for next-generation networks) [77]. The authors of [78] discourse on the NGNs' architecture,

key factors and challenges. As the authors state, to adopt this platform, network operators should face the following issues: the need to provide services over broadband accesses, the need to merge diverse network services, such as data, voice, telephony, instant message and presence among others, and the desire of users to be able to access services from anywhere.

Three players take part in the implantation and success of NGNs: technology, operators and end users. Technology provides the means of implementing and deploying ubiquitous services, which operators provide to end users. The industry has entered in a new era in which the level of technology development exceeds the level of customer desire [4]. There is an overwhelming emergence of mobile and wired technologies; UMTS, WiMAX, Wireless MeshNetworks, The Semantic Web and The Internet of Things are just some examples. This boom in technology makes the operators' investment in infrastructure more complex and costly. Such investment is restraining operators from moving towards NGN. Operators need to adopt a customer-need-driven model for deploying the services that attract the largest number of people, and hence maximize revenue the most. Moreover, operators need to be sure that such revenue will compensate for the investment done and the impact of the provided services on the network. The capacity impact of next-generation services on the operator network is far from trivial due to multiple reasons. These services will be ubiquitous, device- and access-network-independent. They will in large measure rely on context information, enriched data, and social relationships. These features require complex functionality and introduce traffic load, which operators should be capable to bear in world-wide used services. Sections 2.9.4, 2.9.5 and 2.9.6 discuss scalability issues about the deployment of presence systems. Finding out and implementing scalable and advanced services that will attract the largest population of users is currently an active research topic. We mention only some examples: the authors of [79] points out the need to integrate HyperText Transfer Protocol (HTTP) and RTSP (Real Time Streaming Protocol) proxies into IMS networks to reduce costs in web and streaming video applications. They propose a solution that integrates both protocols into a single and scalable element. The authors of [80] and [81] propose a Push-to-Multimedia (PTM) application with fancy communication features and an extensible framework that can be used to deploy IMS services. They use an IMS simulator called Open IMS Playground [82]. The paper [83] presents an IMS-based platform for managing community services such as multi-player

video games, chat, video or streaming. In [84], a platform is described for managing and coordinating multiple IMS services.

2.4.1 IP Multimedia Subsystem

UMTS Release 5/6 moves towards an all-IP network core through the IP Multimedia Subsystem (IMS). This provides 3G network operators with three advantages: QoS, charging and integration of different services. The IMS has been standardized by the 3GPP and 3GPP2 through a number of deliverables, specially the TS 23.228 [1]. The IMS uses Internet protocols, which have been traditionally standardized by the IETF. OMA also plays an important role in the IMS standardization by developing IMS services and, in turn, issuing requirements. The IMS was born from the necessity to attract customers to cellular IP-based services, that is, the mobile Internet. Although the cradle of IMS was 3G operators, it is access-network independent and is being considered as the common core to provide convergent IP-based services from any access technology. IMS forms a substantial part of NGNs, which provides fixed broadband access to IMS services. Presence is an indispensable service of IMS from its birth. The presence service provides the necessary information for customizing services according to the user's needs and preferences. IMS uses SIP for establishing and managing multimedia sessions and its extension, SIMPLE, for presence and instant messaging. The 3GPP defined the presence service over the IMS in 3GPP TS 24.141 [85] but is not actively progressing it. The definition of this service has moved to OMA, which describes the IMS presence service mainly by the OMA Presence SIMPLE specification [86]. Figure 2.3 shows the main components in IMS, which are: CSCF (Call State Control Function), MGCF (Media Gateway Control Function) and MGW (Media Gateway). The MGCF and the MGW are Public Switched Telephone Network (PSTN) gateways; they enable the communication between the IMS and the CS (circuit-switched) network. They perform the protocol translations that are necessary when an IMS user communicates with a CS user, and vice versa. The Home Subscriber Server (HSS) is a database that contains user-related information. The CSCF processes SIP signaling. There are three types of CSCF: Proxy, Interrogating and Serving CSCF. The Proxy CSCF (P-CSCF) is the first point of contact between the IMS terminal and the IMS network. This is an outbound/inbound SIP proxy, and hence all the communications initiated by or destined for the IMS terminal traverse the P-CSCF. The Serving CSCF (S-CSCF)

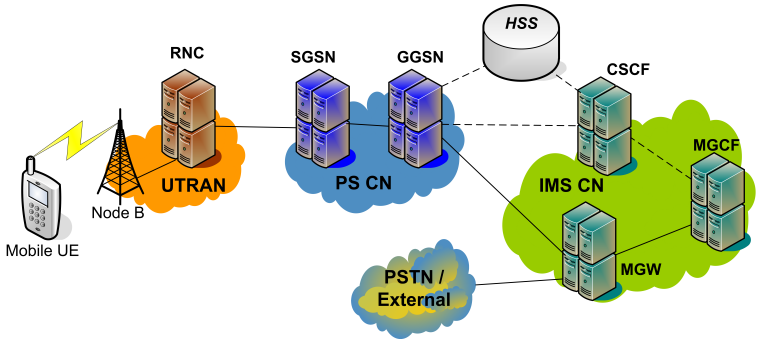


Figure 2.3: Main components of the IMS

is the brain in the IMS signaling plane. This is a SIP server and SIP registrar that performs session control as well. All the SIP signaling sent by IM terminals traverses the allocated S-CSCF. The main functions of the S-CSCF are to provide SIP routing services and to enforce the policy of the network operator. The Interrogating CSCF (I-CSCF) receives SIP requests and routes them to the appropriate destination, that is, an S-CSCF or Application Server (AS). An AS is a sever that provides a particular application service in the IMS. The address of a domain's I-CSCF is registered in the DNS (Domain Name System) records of the domain in order to permit external SIP servers to route SIP messages towards this domain. Before an IMS terminal starts any IMS-related operation, it needs to discover the IP address of the P-CSCF that will be acting as an outbound/inbound proxy server. After this, the IMS terminal needs to register within the IMS before initiating or receiving any other SIP signaling. This is accomplished by regular SIP registration. When the registration is done successfully, the IMS terminal can establish any session with other users or ASs. All the messages that the IMS terminal receives and sends go through the allocated P-CSCF and S-CSCF. Thus, with services that generate large amounts of messages, these servers could become bottlenecks. An obvious example is the presence service, which has to timely disseminate publication, subscription and notification requests among different domains.

2.4.2 Device-Independent Communications

A necessary feature of ubiquitous communication services is device independence. Currently, end users are who adapt to service requirements by choosing the devices that

best fit these requirements. Services, therefore, need to move towards service-to-service adaptations, which minimize user interaction as much as possible. Ubiquitous services need to know the software and hardware characteristics of end devices, and accordingly adapt their behavior and contents. Mobile users access the Internet and the World Wide Web (WWW) via different wireless and cellular networks, each using different radio interfaces and protocols. Mobile users also use a wide spectrum of mobile devices, ranging from limited mobile phones to more advanced devices such as PDAs, smart phones, and laptops. These devices have significant differences in memory, computation power, networking and battery lifetime. Consequently, a contextualization or profiling of user devices is needed to provide pervasive and device-independent web access. Web services and user applications should automatically adjust their operation and presentation to changes in the user environment, network state, and end devices' capabilities.

Device characterization has already been considered by various standardization organizations, especially in the world of mobile communications. The most widespread proposals are the CC/PP specification [20] presented by the W3C and the UAProf specification [21] presented by the OMA. Although these frameworks provide good tools for attaining a device-independent web environment, there are limitations in their specifications and current implementations. The CC/PP specification exceeds in flexibility since developers can define their own vocabularies. The lack of a consensus on vocabularies hinders content adaptation and presentation. Regarding UAProf, the vast majority of manufacturers only offer partial implementations. The main drawback to all these frameworks is the fact that they are focused on mobile phones and do not provide suitable solutions for other types of devices.

Content adaptation is of vital importance for services that interact with heterogenous devices with different hardware and software capabilities. Such heterogeneity affects the process of creating, delivering and presenting Internet contents. The WWW provides users with an infinite variety of contents that includes almost every conceivable kind of media in different formats and languages. Most of these contents are designed to work out on personal computers, and hence it is necessary to adapt these contents to other types of end devices. Moreover, a lot of networking and enterprise services (i.e. instant messaging and presence services, VoD (Video on Demand), VoIP (Voice

over IP) , etc.) use web applications as front-ends for users to configure account settings and even access their systems. This fact increases the interaction modalities and communication possibilities that affect the delivery of web content.

2.5 Publish/Subscribe Communication Paradigm

Publish/subscribe-based systems interconnect information producers with information consumers by means of events. Producers encapsulate information into events and consumers subscribe to those events in which they are interested. When a new event is generated and published in a publish/subscribe-based system, this system is responsible of checking out what subscriptions match the event, and delivering the event to the consumers associated with these subscriptions. There are two models for disseminating information: pull and push. In a pull model, event consumers initiate the transfer of events by requesting them from the system while, in a push model, event producers are who initiate the transfer. Both models can adopt a periodic or asynchronous communication. With a periodic communication, producers and consumers exchange information periodically. On the contrary, with an asynchronous communication, the timing of event transfers is not predetermined. A subscription is thought of as a filter that specify the interest of a consumer in particular kinds of event. Publish/subscribe systems can therefore be classified by their filtering model. The most popular systems use expressive content-based mechanisms. With this kind of mechanism, consumers can specify the notifications that they wish to receive based on the content of these notifications. Content-based filters with boolean expressions about the content of notifications are basically logical operators in the form of <*attribute*, *operator*, *value*>. The publish/subscribe communication paradigm differs from traditional point-to-point communication models on multiple aspects: anonymity (i.e., event consumers do not necessarily know the identity of event produces and vice versa), asynchronicity (i.e., event producers can send events without any event request), multicasting (i.e., a single event can be sent to multiple consumers through one message), and dynamism (i.e., the network infrastructure supports dynamic scenarios in which consumers and producers connect to and disconnect from the network frequently). The publish/subscribe paradigm allows decoupling, on one hand, event consumers from producers and, on the

other hand, the communication protocol from the underlying technology. Such a capacity of abstraction makes this paradigm applicable to diverse scenarios with different goals.

The publish/subscribe paradigm has been widely studied for constituting the base on which to build event-based distributed systems. Linda [87] proposed structuring distributed programs by using several Central Processing Units (CPUs) with a common memory indexed by attributes, which is known as tuple space. Later, systems such as ISIS [88] and Information Bus [89] provided publish/subscribe-based proposals in which event producers publish information and consumers subscribe to subsets of information. Some publish/subscribe-based systems rely on network servers, which are known as brokers, such as Sienna [90], Gryphon [91] and Kyra [92], while others have focused on Distributed Hash Tables (DHTs), such as Scribe [93], Bayeux [94] and Chord [95]. Publish/subscribe systems have relied on fixed networks traditionally. Thus, most systems are not concerned about the consumption of resources in mobile networks with scarce resources and changing nodes. The authors of [96] and [97] discuss the limitations of centralized solutions for event-based systems on mobile networks. Both works propose distributed brokers that implement content-based subscriptions.

Sensor networks are based on events in nature since sensors periodically obtain new measurements from their environment. Thus, the integration of the publish/subscribe paradigm into such restricted networks is an exciting although challenging research topic. The authors of [98] propose a technique in which the nodes organize their own P2P relations based on the similarity of their subscriptions. This proposal therefore provides a fully-distributed approach. In [99], a middleware with some centralized elements is proposed, in which sensors publish the kinds of information that they can generate. Moreover, this middleware uses some information aggregation methods for reducing the traffic at the centralized servers. Similar works that propose the use of centralized elements in sensor networks are those described in [100], SeNMi [101] and MQTT-S [102]. Other publish/subscribe middlewares for sensor networks propose flooding for propagating subscriptions (e.g., DV/DRP [103]) and publications (e.g., REBECA [104]). Constrained Application Protocol (CoAP) [105] is being defined by the CoRE IETF Working Group (WG) [106] for providing a subset of Representational State Transfer (REST) functionality on sensor networks. CoAP provides a subscribe/publish model, which presents some limitations (e.g., on content-based filtering, aggregation of events,

maintenance of subscription state) due to the fact that this protocol is based on the HTTP communication model. Some works such as [107], [109], and [108] adopt the SIP publish/subscribe model [110] (see Section 2.7) and use gateways for translating SIP to other publish/subscribe models optimized for sensor networks. In [107], a publish/subscribe model is used to interconnect remote ZigBee networks. In [108] and [109], systems for interconnecting different sensor networks with other systems based on generic events are presented.

The presence service matches the publish/subscribe paradigm perfectly: the entities that have presence information associated generate this information asynchronously (i.e., presence changes) and other entities subscribe to the entities in which they are interested. The vast majority of presence protocols are therefore based on this paradigm. Section 2.6 describes the most popular presence protocols. For example, SIMPLE defined centralized servers (i.e., brokers) for handling presence events through the distributed SIP publish/subscribe model. This protocol also includes content-based filters for classifying event subscriptions. Some works have addressed the integration of presence services into sensor networks. This requires optimizing the existing presence protocols or implementing new ones since the existing presence protocols have been designed for Internet. TinySIP [109] optimizes SIP/SIMPLE for accessing to sensor information in a resource-efficient manner. The authors of [111], [39] and [112] offer proposals for interconnecting the presence service with sensor networks.

2.6 Instant Messaging and Presence Protocols

Due to the great diversification of IM systems, as described in Section 2.1, a number of IMP protocols have emerged. Among them, the most relevant ones are XMPP, SIMPLE and Instant Messaging and Presence Service (IMPS). IMPS was thought of as a competitive IMP framework for the mobile world but lost much ground to SIMPLE. XMPP and SIMPLE are the two competing protocols for becoming the de facto standard IMP protocol worldwide. Both protocols are supported by the IETF and by different telecom magnates. This situation has caused an interoperability barrier between IMP providers that is restraining the convergence of presence-based communications. Sections 2.6.1, 2.6.2 and 2.6.3 briefly describe the main features of XMPP,

SIMPLE and IMPS, respectively. Lastly, Section 2.6.4 analyzes some important factors that may be determining to the adoption of these protocols.

2.6.1 XMPP

Jabber was released as an open-source protocol in 1999 and later, in 2004, was adopted by the IETF under the name of eXtensible Messaging and Presence Protocol (XMPP). XMPP core protocol was published as Request For Comments (RFCs) 3920 and 3921. Nevertheless, these documents were revised recently, resulting in the most up-to-date XMPP specification described in the RFCs 6120 [113], 6121 [114] and 6122 [115]. In addition to these specifications, to date, other 9 RFCs has been published and 2 Internet-Drafts are under consideration by the IETF XMPP WG [116]. Besides the IETF, the XMPP standards Foundation [117] has defined many XMPP extensions. This foundation is a community of developers that provides both open-source and commercial XMPP software.

2.6.1.1 Architecture and Operation

XMPP core is mainly defined by the RFCs 6120 [113] and 6121 [114]. The former defines its core and the latter extends it to support instant messaging and presence. XMPP supports IMP features such as group handling and off-line messages for users. However, the RFC 6121 [114] limits presence to availability (i.e., basic states such as online, busy and so on) and does not consider other kind of information. Although XMPP is independent of the transport layer, its specification describes a binding to TCP. An XMPP server is in charge of managing connections through XML streams and routing XML stanzas. An XML stream works as a container for exchanging XML elements between two entities. An XML stanza is a semantic entity that is sent through an XML stream. XMPP defines the following stanzas: *message* (for sending information), *presence* (for expressing availability) and *iq* (for queries). XML streams can convey errors (e.g., bad format or conflict), which are irrecoverable and involve closing the XML stream and the underlying TCP connection. An XMPP client needs to initialize an XML stream before sending information to any entity. This initialization requires negotiating with an XMPP server through the Simple Authentication and Security Layer (SASL) protocol. Once an XML stream is established, the XMPP client can send an undefined number of stanzas to any XMPP entity through this stream.

2.6.1.2 Instant Messaging and Presence

Message and *presence* stanzas provide IMP functionality. The former has a *Type* attribute that identifies the type of the message. The most relevant message types are: "normal", which is sent to an XMPP user's inbox, and "chat", which is an instantaneous message. Other message types are "groupchat" for multi-recipient IMs and "headline" for describing the content of web services, broadcast, etc. A *presence* stanza is a basic mechanism for subscribing and notifying presence information. XMPP only defines five availability states as presence: "chat" (willing to talk), "away" (not available), "xa" (extended not available) and "dnd" (do not disturb). These states are merely informative and do not affect the communication protocol.

2.6.2 SIMPLE

SIP is an IETF standard for initiating, modifying and terminating multimedia communications between two or more participants. This is a text-based protocol with a transaction model that similar to that in HTTP. SIP for Instant Messaging and Presence Leveraging Extensions (SIMPLE) was born in 2004 and, to date, is composed by a total of 30 RFCs that can be found on its IETF WG [3]. Although the high number of RFCs may suggest that SIMPLE is a highly complex protocol, most of these specifications are optional and deal with advanced IMP features. We estimate that a basic SIMPLE-compliant system should satisfy around 7 of these RFCs. Section 2.7 explain the whole SIMPLE framework in more detail.

2.6.2.1 Architecture and Operation

SIMPLE inherits the SIP's architecture and operation as described in Section 2.7.1. We summarize the SIP operation as follows. An end system that implements SIP is formed by a User Agent Client (UAC), which generates SIP requests, and by a User Agent Server (UAS), which responds to SIP requests. There exist three kinds of SIP servers: Registrar, Proxy and Redirect. The first permits a user to bind a SIP Uniform Resource Identifier (URI) to a contact address. Proxy and Redirect servers route or redirect SIP requests, respectively. SIP defines defines the following types of request message: INVITE (invites a user to initiate a session), ACK (confirms a response), CANCEL (cancels an uncompleted request), OPTIONS (discovers a SIP

user's capabilites), BYE (finishes an established session) and REGISTER (registers contact information). The event notification framework specified in the RFC 3265 [110] extends the SIP core with two new SIP requests: SUBSCRIBE and NOTIFY. An entity that is interested in a resource can subscribe to the resource's state information through a SUBSCRIBE message. Thus, this entity is called subscriber. When the entity that handles the resource's state information receives a SUBSCRIBE message, it sends the requester a NOTIFY message that contains the state information. Thus, this entity is called notifier. From then on, the notifier will send a NOTIFY message to the subscriber whenever the resource's state information changes. The mechanism for publishing state information is defined in the RFC 3903 [118]. This extension defines a PUBLISH request for resources to let other entities know about their state information. SUBSCRIBE, NOTIFY and PUBLISH messages contain the type of state information in an *Event* header. Extensions that define new values for the *Event* header are called event packages.

2.6.2.2 Instant Messaging and Presence

Instant messaging and presence is easily integrated into the SIP architecture by means of extensions. RFC 3428 [119] extends SIP with the MESSAGE request type, which contains an instant message within a Multipurpose Internet Mail Extensions (MIME) body. SIMPLE defines a new event type called *presence* in the RFC 3856 [120], which also introduces additional concepts. A Presence Agent (PA) receives the user's presence information. When this information comes from multiple sources, it performs merging functions to build a complete and consistent picture of the user's presence information. Other important function of PAs is handling of subscriptions. PAs receive SUBSCRIBE messages, maintain the subscriptions' state and, when a subscription's state information changes, send NOTIFY messages to the proper subscribers. A Presence Server (PS) is a physical entity that can act as a PA or Proxy Server for SUBSCRIBE requests. When a PS receives a SUBSCRIBE request to a user that is under its control, it acts as a PA. Otherwise, it acts as a proxy by redirecting the request to other PS. In addition, SIMPLE extends its event framework for supporting resource lists in the RFC 4662 [121]. A resource list is a set of zero or more resources whose state information is seen as a single state and is therefore subscribed by means of a single request. This supports the concept of contact list in IMP systems. Regarding presence information

format, SIMPLE defines the Presence Information Data Format (PIDF) in the RFC 3863 [22]. PIDF is a basic common profile for presence that is based on XML, and thereby protocol-independent and extensible. There already exist several PIDF extensions for personal information [23], information about services and devices [24], time intervals [122] and contact information [123].

2.6.3 IMPS

IMPS [124] is a set of universal specifications for IMP mobile services. IMPS was conceived into the Wireless Village iniciative, which was formed by Ericsson, Nokia and Motorola in 2001. After Wireless Village was merged with OMA, IMPS was published as OMA IMPS 1.0 in 2002. IMPS is constituted by 16 documents that describe its architecture, requirements, use cases, protocols, data formats and presence information.

2.6.3.1 Architecture and Operation

IMPS has a client-server architecture in which IMPS servers communicate with each other by the Server-Server Protocol (SSP) or other non-specified protocol in case of a mobile network. As well, an IMP client can communicate with other IMPS clients by the Client-Server Protocol (CSP) in a direct connection or through IMPS servers that act as proxies. IMPS is composed of two different layers: application and transport, which are independent from each other. There are multiple bindings between these higher-lever application layers and the lower-level transport layers. CSP and SSP are application protocols. CSP connects IMPS clients to servers and is capable to use different transport means based on the clients' capabilities. Transport bindings are split into two channels: data and Communications Initiation Request (CIR). The former is used to exchange CSP primitives and the latter serves to activate the former. The need for a CIR channel depends on the particular application and the transport protocol used. Protocol bindings defined for the data channel are Wireless Session Protocol (WSP), HTTP, HTTP Secure (HTTPS) and SMS; they all except SMS require a CIR channel. Regarding the network protocol, CSP can rely on SMS, 2.5/3G wireless IP and Mobile IP and SSP usually is on wired IP networks. IMPS also defines multiple syntax of application-level messages such as XML and WAP Binary XML (WBXML).

IMPS server are formed of a set of Application Service Elements (ASEs) that are accessible by Service Access Points (SAPs). There are four ASEs: the Presence Service Element, Instant Messaging Service Element, Group Service Element and Content Service Element. The Content Service Element permits content sharing between IMPS users, which basically means an exchange of the content's Uniform Resource Locator (URL). How contents are updated and downloaded is beyond the scope of IMPS. The remaining ASEs are related to IMP and therefore defined in Section 2.6.3.2. A SAP is the IMPS server's interface to the outside. This provides IMPS clients, other IMPS servers and external entities with a communication point to an IMS server. A SAP offers the following main functions: authentication and authorization, service discovery and negotiation, user profile management and service retransmission. Service discovery allows an application to identify the services that may be of interest. Service negotiation consists in finding out the service's capabilities. Service retransmission routes the service requests and responses through IMPS servers. IMPS defines a transaction-based communication model in which requests and responses are grouped into transactions. General transactions constitute the minimum level of interoperability. Specific transactions are defined by each ASE. Transactions are exchanged in the frame of a session. IMPS sessions are independent from the transport layer and are established by connecting to a SAP. An IMPS session is always associated with context information such as client capabilities, presence subscriptions and negotiated services. There are two kinds of IMPS client: the embedded client and the Command Line Interface (CLI) client. The former can be embedded in different mobile or fixed devices, which communicate through the CSP protocol. A CLI client is a lighter version of an embedded client and uses the Command Line Protocol (CLP) for communicating with IMPS servers.

2.6.3.2 Instant Messaging and Presence

The Instant Messaging Service Element provides operations for sending and receiving instant messages. This supports group messages and two delivery methods: push and notification/pull. In the push model, the IM service delivers the message to the recipient and in the notification/pull model, the IM service notifies the recipient, whom afterwards pulls it. This second method is appropriate for multimedia content, which is much heavier than a textual content. The Presence Service Element retrieves, handles and updates location and presence information. Presence information is structured in

presence attributes, which are composed of name, qualifier, and value. Some presence attributes are time zone, geographical information, communication capabilities and availability. IMPS classifies presence attributes into client-related (information about physical devices and software) and user-related. The Group Service Element offers the operations that are necessary to handle contact groups. A group can be public (i.e., created by a service provider) or private (i.e., created by an IMPS user).

2.6.4 Which Makes a Difference?

Determining which IMP framework is the best to win the war for standardization would be a hard task. There are many factors to take into account and the choice may depend less on technical merits than Industry interests. Instead, we compare XMPP, SIMPLE and IMPS with regard to five factors, namely interoperability, complexity, security, acceptance, and wireless communications, that are important to measure the suitability of each protocol:

Interoperability: The IETF Instant Messaging and Presence Protocol (IMPP) WG concluded a set of standard specifications for providing interoperability between IMP systems (see Section 2.7.2). SIMPLE and XMPP satisfies these specifications, which makes it easy to implement gateways between them and other IMPP-compliant systems. A mailing list called SIXPAC (SIP Interworking with XMPP in Presence Aware Clients) [125] was created in the IETF for making it easier for developers to create applications that work with both SIMPLE and XMPP. SIMPLE has the advantage of being a SIP extension since SIP is the standard protocol for VoIP. This fact promotes interoperability because there is no need for a standalone protocol to work in parallel. SIMPLE is actually used by the IMS, which is described in Section 2.4.1. IMPS complies IMPP except for its presence information format. Regarding IMPS interoperability, its SSP protocol was designed as the interface to other IMP systems through proprietary gateways.

Complexity: Regarding to the specification size, SIMPLE is the biggest protocol. However, a great part of the SIMPLE specification is dedicated to advanced features and its core is around 7 RFCs. SIMPLE has been designed to be extended

from its birth and has evolved in a modular way. IMPS is composed of 16 specifications, which are are quite complex and interconnected. XMPP seems to be the least complex specification since its total number of RFCs is 9. Nevertheless, its specification is not only composed of IETF RFCs; the XMPP Standards Foundation is defining numerous extensions (to date, 11 final extensions and much more active drafts).

Security: SIMPLE and XMPP provides similar guarantees of security. Data integrity and confidentiality is achieved by Transport Layer Security (TLS). Regarding client-server authentication, XMPP uses SASL and SIMPLE any HTTP authentication scheme. Regarding end-to-end authentication and confidentiality, XMPP and SIMPLE use Pretty-Good-Privacy (PGP) and Secure MIME (S/MIME), respectively. IMPS support user authentication through the 2-way and 4-way control mechanisms. The former consists in sending the user's identifier and password in plain text while the latter is a digest authentication based on a challenge sent by the server. IMPS does not support a consistent data integrity and confidentiality. HTTP-S can be used between client and servers but end-to-end confidentiality is not guaranteed. However, as any MIME type is allowed for content of instant messages, end-to-end authentication can be accomplished using S/MIME.

Industry acceptance: Currently there is a race on for IMP standardization between SIMPLE and XMPP. The latter is more accepted by the Internet community because of its open-source character as well as the fact it was launched before SIMPLE. Most of the current IMP clients are based on XMPP and the Internet magnate Google has selected XMPP for its Google Talk IM. Other companies that have been attracted by XMPP are Hewlett-Packard, Intel Capital and France Telecom. In 2010, Facebook announced the integration of an XMPP interface into the Facebook Chat for interoperability with other XMPP-based instant messengers. On the other side, SIMPLE has been selected for IMS (see Section 2.4.1), which means network operators are going to use this protocol. Rich Communication Suite gives guidelines for deploying presence-based rich communication services on IMS, hence through SIMPLE. Heavyweights Microsoft, Yahoo and IBM lined up behind SIP and SIMPLE. Moreover, the java community has provided support for SIP and SIMPLE through the Java Specification Requests

(JSRs) 116 and 289 (SIP Servlet Application Programming Interfaces (APIs)), 125 and 32 (JAIN SIP APIs), 164 (SIMPLE presence API), 165 (SIMPLE Instant Messaging API) and 180 (SIP API for Java 2 Micro Edition (J2ME)). IMPS is widely deployed but not so marketed. Many mobile terminals from manufacturers such as Ericsson, Nokia, Motorola and Siemens support IMPS with built-in IM clients. However, there are only a few standalone IMPS-compliant clients such as Mobjab, Agile Mobile Messenger and Yamigo. To date, there does not exist any open-source IMPS client or server.

Mobile communications: XMPP is based on XML streams and therefore is much heavier than SIMPLE. This is the main drawback of XMPP because XML streams consume much bandwidth and processing resources. SIMPLE has more concern on bandwidth consumption through several optimizations that have come out from its WG. In addition, the JAVA community supports SIP for mobile devices through the JSR 180 (SIP API for J2ME). Although IMPS was conceived for wireless devices from the beginning, the fact that it has evolved to satisfy the requirements of multiple telecom companies has made it a much heavier protocol than expected.

2.7 SIMPLE Framework

SIP for Instant Messaging and Presence Leveraging Extensions (SIMPLE) is an IETF standard for instant messaging and presence. SIMPLE is an extension of SIP and therefore relies on the SIP architecture, which is described in Section 2.7.1. SIMPLE was conceived to fully comply the IMPP requirements from its birth, and hence it is based on the IMPP presence model. Section 2.7.2 describes this model and therefore the main semantics of SIMPLE. As outlined in Section 2.6.2.2, instant messages are included into SIP through the MESSAGE method, which is defined by the RFC 3428 [119]. SIMPLE builds upon the SIP publish/subscribe communication model, which introduces the SUBSCRIBE, NOTIFY and PUBLISH methods as explained in Section 2.7.1. SIMPLE extends this model with a new event type called *presence* and introduces new concepts by the RFC 3856 [120]: a *Presence Agent (PA)* receives the user's presence information. When this information comes from multiple sources, it performs merging functions to build a complete and consistent picture of the user's

presence. Other important function of PAs is handling of subscriptions. PAs receive SUBSCRIBE messages, maintain the subscriptions' state and send NOTIFY messages to the proper subscribers when the subscriptions' state information changes. A *Presence Server (PS)* is a physical entity that can act as a PA or a Proxy Server (see Section 2.7.1) for SUBSCRIBE requests. When a PS receives a SUBSCRIBE request to a user that is under its control, it acts as a PA. Otherwise, it acts as a proxy by redirecting the request to other PS. In addition, SIMPLE extends its event framework with resource lists by the RFC 4662 [121]. A *resource list* is a set of zero or more resources whose state information is seen as a single state and are, therefore, subscribed by a single request. *Resource List Meta Information (RLMI)* describes the subscription state of the resources in a resource list. A resource's subscription state is encoded by PIDF as described in Section 2.7.3. Appendix B shows an example of RLMI document. A *Resource List Server (RLS)* receives SUBSCRIBE messages to a resource list and inform the resource list's watchers of changes in the list's resources through NOTIFY messages. Figure 2.4 outlines the operation of SIMPLE, which can be summarized as follows. Presentities, or more specifically their Presence User Agents (PUAs) (see Section 2.7.2), publish presence changes to their PSs by sending PUBLISH requests. When a presence change occurs, the PS notifies the presentity's watchers through NOTIFY messages. On the other hand, watchers, or more specifically their Watcher User Agents (WUAs) (see Section 2.7.2), subscribe to their RLSs. Although the use of an RLS is common, watchers could subscribe to their presentities directly. When a subscription between an RLS and a watcher is established, the RLS subscribes to each resource in the list on behalf of the watcher. Any SUBSCRIBE message to a presentity is routed to the presentity's domain (that is specified in its SIP URI). Then, the presentity's domain redirects the request to the presentity's PS. If the subscription is successful, the PS will notify the RLS whenever a presence change occurs. In turn, the RLS will notify the watcher of an RLMI document that contains the presence change.

The IETF is continuously working on SIMPLE and producing new extensions and related documents. For this reason, it may be hard to figure out how SIMPLE works and how the SIMPLE specifications interconnect. An Internet-Draft [126] addresses this problem by enumerating and classifying all of the SIMPLE specifications. Regarding presence, SIMPLE specifications are classified into six groups: core protocol, presence documents, privacy and policy, provisioning, federation, and optimizations. Table 2.2

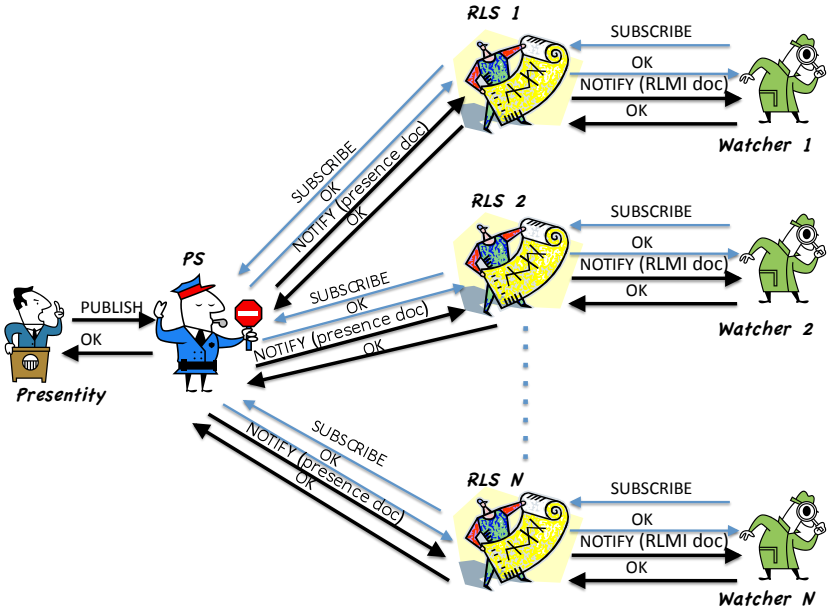


Figure 2.4: SIMPLE presence subscription and publication

shows these specifications classified by groups. SIMPLE specifications about the core protocol and presence documents form the basis for the SIMPLE presence framework, and hence are specially relevant in this thesis. Above, we introduced the core protocol specifications and Section 2.7.3 gives details about the specifications related to presence documents. Optimization specifications describe SIMPLE extensions for reducing the load of presence information, and are described by Section 2.7.4. Presence federation means the interconnection of different systems for exchanging presence and instant messages. Provisioning refers to how data is managed by users and provisioned into the presence system. Privacy and policy provides users with the capability to accept and reject presence subscriptions, and decide what information is notified to authorized watchers. Regarding instant messaging, the Internet-Draft [126] defines three groups: page mode, session mode, and IM features. In page mode, instant messages are sent by sending a SIP request. In session mode, instant messages are sent within a multimedia session (set up by an INVITE request). The choice between page and session mode is mainly a matter of efficiency: page mode is more efficient for short conversations and session mode for longer conversations. Furthermore, session mode allows all of the SIP features, such as forking and third party call control, in instant messaging. Additionally, the SIMPLE WG is considering two Internet-Drafts for IM: one defines

an alternative to relay servers [127] and the other specifies a method for mapping MSRP messages to sessions when application layer gateways change SDP contents [128]. Table 2.2 summarizes all of the SIMPLE RFCs to date for instant messaging in addition to presence.

2.7.1 Session Initiation Protocol

Session Initiation Protocol (SIP) is an application protocol for establishing, modifying and terminating multimedia sessions. RFC 3261 [2] defines its architecture and operation. SIP supports name translation and service redirection seamlessly, which permits user mobility. Each user is identified with a SIP URI in the form of "username@domain", which is a public identifier independent of network location. A user's SIP URI is associated with his or her contact address, which depends on his or her current location. Moreover, SIP secure (SIPS) URIs can be used to ensure the use of TLS for securing SIP messages. SIP relies on an infrastructure of network servers that permits users to discover other users and register their localization, among other functions. Nevertheless, SIP users are also able to communicate with each other directly, without any intermediate server. An end system, which is called User Agent (UA), represents a software entity that implements SIP. UAs are formed by a User Agent Client (UAC), which generates SIP requests, and a User Agent Server (UAS), which responds to SIP requests. There exist three kinds of SIP servers: Registrar, Proxy and Redirect. SIP Users register their contact addresses in Registrar servers in order to be reachable by other users. A Proxy server is responsible for routing SIP requests to the recipients on behalf of the requesters. Lastly, a Redirect server handles a SIP request, as a Proxy does, but informs the requester of the recipient's contact address instead of routing the request. Once the requester receives the contact address, it is able to contact the recipient directly. All the SIP servers rely on a database that is indispensable in the SIP architecture: the localization service. This database maintains all the user contact addresses. Only Registrar servers can update the localization service but Redirect and Proxy servers can query it.

SIP responses contain a state code and a text that describes the response such as "100 Trying", "200 OK" and "400 Bad Request". SIP defines the following types of request message: INVITE (invites a user to initiate a session), ACK (confirms a response), CANCEL (cancels an uncompleted request), OPTIONS (discovers a SIP

Identifier	Type	Short description
RFC 3265	Core Protocol	SIP event notification framework
RFC 3856	Core Protocol	Presence subscriptions to presentities, PS
RFC 4662	Core Protocol	Presence subscriptions to resource lists, RLS
RFC 5367	Core Protocol	Inclusion of resource lists into SUBSCRIBEs
RFC 3903	Core Protocol	Presence publication
RFC 3863	Presence Document	Presence Information Data Format (PIDF)
RFC 4479	Presence Document	Semantics of presence documents
RFC 4480	Presence Document	PIDF extension based on RFC 4479
RFC 4481	Presence Document	Addition of time conditions to PIDF
RFC 4482	Presence Document	Addition of contact information to PIDF
RFC 5196	Presence Document	Device and service elements in PIDF
RFC 4745	Privacy&Policy	Framework for expressing privacy preferences
RFC 5025	Privacy&Policy	Document format for describing presence privacy policies
RFC 3857	Privacy&Policy	Subscriptions to incoming watchers
RFC 3858	Privacy&Policy	Document format for describing incoming watchers
RFC 4825	Provisioning	Configuration Access Protocol (XCAP)
RFC 5875	Provisioning	Mechanism for learning about changes in XCAP documents
RFC 5874	Provisioning	Document format for describing changes in XCAP documents
RFC 4826	Provisioning	Document format for resource lists, including sublists
RFC 4827	Provisioning	XCAP usage to store “offline” documents
RFC 5344	Federation	IMP use cases for federating between providers
RFC 4660	Optimization	Mechanism for filtering presence notifications
RFC 4661	Optimization	Document format for expressing notification filters
RFC 5262	Optimization	Document format for partial-state presence documents
RFC 5263	Optimization	Description of partial-state presence notifications
RFC 5264	Optimization	Description of partial-state presence publications
RFC 5261	Optimization	Changes in XML documents
RFC 5112	Optimization	Dictionary for usage with Signaling Compression
RFC 3428	Page mode IM	MESSAGE method for instant messages
RFC 5365	Page mode IM	Multiple-recipient instant messages
RFC 4975	Session mode IM	Message Session Relay Protocol (MSRP)
RFC 3862	Session mode IM	Message format for providing meta-data information in MSRP
RFC 4976	Session mode IM	Extensions to MSRP for relay servers
RFC 3994	IM features	Status of message composition (such as “is-typing”)
RFC 5438	IM features	Delivery notifications of IM receipt

Table 2.2: SIMPLE RFCs

user's capabilites), BYE (finishes an established session) and REGISTER (registers contact information). SIP defines a transaction-based communication model between UACs and UASs. A SIP transaction contains the UAC's request, all the messages that are exchanged before the UAS sends a final response, and the final response. If the final response is positive, a SIP dialog is established between the UAC and the UAS. Both parts of the communication have to maintain the dialog's state until it is terminated. SIP core is extended with an event notification framework by the RFC 3265 [110]. This framework introduces two new SIP requests: SUBSCRIBE and NOTIFY. An entity that is interested in a resource can subscribe to the resource's state information through a SUBSCRIBE message. Thus, this entity is called subscriber. When the entity that handles the resource's state information receives a SUBSCRIBE request, it sends a NOTIFY message that contains the state information to the requester. Thus, this entity is called notifier. From then, the notifier will send a NOTIFY message to the subscriber whenever the resource's state information changes. The mechanism for publishing state information is defined by the RFC 3903 [118]. This extension defines a PUBLISH request that resources can use to let other entities know about their state information. SUBSCRIBE, NOTIFY and PUBLISH messages specify the type of state information in an Event header. Extensions that define new values of Event header are called event packages.

2.7.2 Instant Messaging and Presence Protocol

Due to the great success of instant messaging and the diversity of proprietary protocols, the IETF published a set of requirements that IMP protocols should satisfy for the sake of interoperability in 2000. These requirements were named as Instant Messaging and Presence Protocol (IMPP) and published by the RFCs 2778 [129] and 2779 [130]. IMPP was defined within the IETF IMPP WG [131], which is already concluded. IMPP is an abstract model for designing IMP systems that defines the involved entities and the services that these systems should provide. This protocol is the first standardized definition of an IMP system, which provides terminology and concepts that can be applied to any specific IMP protocol. IMP systems have two kinds of clients: presentities and watchers. A *presentity* is an entity that has associated presence information and somehow makes an IMP system aware of this information. A *watcher* is an entity that receives presence information from an IMP system. There are three

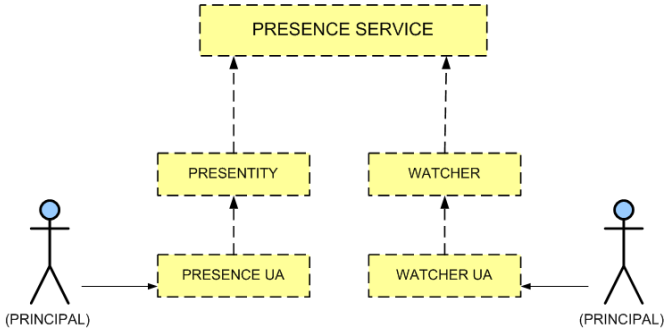


Figure 2.5: IMPP presence model

kinds of watcher: fetchers, subscribers, and pollers. *Subscribers* are the watchers that receive asynchronous notifications of presence information when changes of interest occur. *Fetchers* are the watchers that retrieve presence information from the IMP system when necessary. *Pollers* are the watchers that retrieve presence information regularly to check out whether any interesting change occurred. Presence information consists in an arbitrary number of tuples. Each tuple contains a state sign (such as offline and online) and an optional communication address. How the IMP entities interact is outlined as follows. A *Principal* is the human being or program that connects to an IMP service. A *Presence User Agent (PUA)* is the entity that allows a principal to manipulate zero or more presentities. A principal could be associated with zero, one or more presentities, each represents a different model of the principal's presence information. Likewise, a *Watcher User Agent (WUA)* is the entity that allows a principal to manipulate zero or more watchers. Figure 2.5 shows this model.

RFC 2779 states three basic principles for the IMPP standard: security and privacy, scalability, and wireless operation. The first principle states that the standard IMP protocol must provide a means for exchanging short messages and publishing presence in a secure and privacy-protected way. The second principle states that the standard IMP protocol must work even for huge amounts of users distributed on the Internet, while allowing comfortable conversational exchange of short messages. Lastly, the third principle states that the standard IMP protocol must be usable via mobile IP wireless access devices. Furthermore, IMPP includes other standards for interoperable IMP protocols such as the Common Profile for Presence (CPP) [132] and PIDF [22]. The former defines the high-level semantics and formats of information that are common to

IMP systems. The latter is a common presence data format for CPP-compliant IMP protocols, which is described in Section 2.7.3.

2.7.3 Presence Documents

SIMPLE adopts the Presence Information Data Form (PIDF) as its basic common format for presence, which was defined in the RFC 3863 [22] by IMPP (see Section 2.7.2). This is based on XML and, therefore, protocol-independent and extensible. A PIDF document is formed by a set of tuples, each representing a different segment of the presentity's presence. Presence segmentation may be caused by multiple user devices or applications as well as by different instant times at which the presence was generated. A tuple can contain four XML elements: status, contact, timestamp and note. A contact element indicates the tuple's contact address. A status element specifies the tuple's status and contains a single "basic" element. This element expresses the communication availability of the contact address indicated by the contact element. PIDF only defines two values: "open" when the tuple is available and "closed" in other case. The timestamp element indicates when the tuple was generated. A note element is used to show a sentence on the GUI.

A Data Model for Presence, which is defined in RFC 4479 [133], relies on IMPP and PIDF to go into the semantics of presence information in depth. This clarifies the role of presence documents into communication systems, what a presence document means and what it is for. This specification defines three main concepts in any presence system: Service, Device and Person. These three concepts are mapped to the XML elements tuple, device, and person. A *service* is a means of communicating with a person. A *person* represents a human user that has associated presence information. A *device* models the physical environment where one or more services are running. Person and presentity has different meanings. Person is equivalent to Principal in IMPP [129], that is, an end user. We refer the reader to Section 2.7.2 for further information about IMPP. Presentity is a complete image of a person, which combines information about the person and its associated services and devices. Figure 2.6 shows the hierarchical relationships between person, service and device elements in a particular presence document. Person, device and service elements can have static and dynamic presence attributes, which are called characteristics and state, respectively. Characteristics are information that does not change under normal circumstances while state is

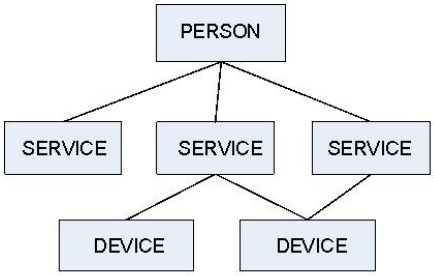


Figure 2.6: Structure of an example IMPP presence document

information that varies over time. Table 2.3 shows some examples of characteristics and state.

	Characteristics	**State**
Person	Age, weight	location, activity, humor
Service	communication type, work purpose	duplex, media preferences
Device	1GB RAM, 6400x200 display	off/on, battery level, network location

Table 2.3: Examples of characteristics and state for person, service, and device entities

PIDF is the basic frame from which new kinds of presence information can be defined. There already exist several PIDF extensions, which are described below. Appendix B shows an example of PIDF document, which includes extensions defined in the RFCs 4480 [23] and 4119 [38].

Personal information (RFC 4480 [23]): This mainly extends the person element with presence attributes such as the type of place in where the person is, its acoustic and luminous conditions, the person's humour, activities and role, etc. This also defines some information for services such as service class and privacy, status icon and identifiers of the involved devices. Device elements are enriched with class and user-input attributes.

Services and devices (RFC 5196 [24]): This introduces the servcaps and devcaps elements for describing UA and device capabilities, respectively. The servcaps element extends the tuple element for including information such as types of accepted medias, service class, supported SIP methods and events, etc. The

devcaps element extends the device element for indicating the device's mobility and textual description.

Time intervals (RFC 4481 [122]): This defines the timed-status element for specifying a service's status that depends on time; thus, this specification extends the tuple element. For example, a presentity could indicate that its availability is offline from 3 to 6 p.m.

Contact information (RFC 4482 [123]): This adds contact information to PIDF such as display name, homepage, business card, icon, and music that represent the person. This information extends the person element or, less commonly, the tuple element.

Location information (RFCs 4119 [38] and 5139 [134]): This defines a location object format that contains geographical information [38] as well as civic information [134] as an extension of the tuple element. This extension is called PIDF Location Object (PIDF-LO). The RFC 5491 [135] gives recommendations on the usage and interpretation of PIDF-LO. This extension has emerged from the Geographic Location/Privacy (GEOPRIV) WG [136], which is refining PIDF-LO and other aspects of location handling such as authorization, integrity, and privacy.

2.7.4 Optimizations

The verbose nature of presence documents and the fact that presence publications and notifications need to be timely disseminated introduce presence overload into SIMPLE systems. This excessive traffic may have harmful effects on network servers, wireless network accesses and devices with limited resources. Sections 2.9.4, 2.9.5 and 2.9.6 tackle presence information overload more deeply. To address this problem, the IETF has proposed the following traffic optimization techniques:

Partial publication and notification: The RFC 5264 [137] defines a mechanism for only publishing the presence changes that have occurred from the last publication. Likewise, the RFC 5263 [138] describes how presence notifications can only contain the presence changes that have occurred from the last notification. These two mechanisms use the document format for partial-state presence information described in the RFC 5262 [139].

Notification Filtering: The RFC 4660 [140] defines event notification filtering for subscribers to express what information presence notifications should include and when notifications should be delivered. Notification filters are encoded by the XML format described in the RFC 4661 [141].

Signalling compression: Sigcomp [142] defines a flexible framework for compressing SIP messages in end-to-end communications. The RFC 5112 [143] provides a Sigcomp dictionary for presence information in order to improve the Sigcomp efficiency on presence messages.

XML Patch operations with XPath: The RFC 5261 [144] defines an XML structure for representing changes in XML documents. It avoids sending the whole XML document when it changes, and is used by several SIMPLE optimizations (e.g., partial presence)

Notification rate control: As the SIP event framework [110] mandates, each event package specification defines an absolute maximum on the rate at which notifications are allowed to be generated by a single notifier. For example, the watcher-info event package [145] recommends that the server generate notifications at a rate no faster than once every five seconds, while the message-summary event package [146] suggests a maximum of one notification per second.

Conditional notification: The RFC 5839 [147] proposes a mechanism that suppresses the sending of unnecessary notifies when subscriptions are refreshed or terminated. This basically consists in including a *Supress-If-Match* header in the SUBSCRIBE message that should not result in a notification if there are not pending changes. This header must contain the entity-tag that the notifier previously sent to the subscriber in the last NOTIFY message's *SIP-ETag* header.

2.8 Platforms for Presence-Aware Services and Automatic Service Composition

As described in Section 2.4, network convergence is intended to be a playground for deploying new, more convenient ubiquitous services that will improve users' QoE. The

users' presence information is fundamental to the success of these services. This information allows applications to adapt user communications according to the users' circumstances and preferences. A key factor in the success of network convergence is providing users with value-added services that encourage them to communicate in an always-on, more dynamic way. Personalization is a required feature in any FMC service. Users should be capable to customize their services' behavior and appearance based on their needs, which may change over time. Simplified, intuitive and easy-to-use interfaces are needed to attract general users. Section 2.3.1 introduces some research works on applications that offer innovate and intelligent functionality based on presence and context information. The industry and academia are alarmed by the scalability issues that convergent presence-aware services are expected to generate, as introduced in Section 2.4 and further explained in Section 2.9. Thus, supporting software platforms that are scalable, interoperable and optimized for mobile environments are necessary to deploy large-scale ubiquitous applications. Many authors have addressed context management but only a few of them have presented solutions for disseminating presence information. For instance, the work [148] is a generic platform for provisioning and handling context information in mobile environments. Although this work does not consider presence information about the user and his or her buddies, it presents some desirable features such as lighter user devices, and context management, reasoning, privacy and dissemination controlled by the user. The works [52] and [149] present software platforms in which the user devices take the responsibility for context management, which is not suitable for limited mobile devices. The authors of [150] propose a RESTful web service for providing lighter-weight presence services. However, HTTP is not suitable for subscriptions and presents a number of issues difficult to overcome. Subscriptions are simulated with persistent connections and chunked encoding. No resource list subscriptions are defined and the mapping between SIP addresses and URLs is not evident. The authors of [151] present a platform for providing service integration in the personal domain. This mainly consists of a personal proxy server that handles all the user's communication services and his or her end terminals' presence information. The authors claim that this solution is scalable since it is implemented in personal domains. This personal proxy server recollects the user's presence information that comes from his or her devices. This acts as the user's presence agent by publishing the

user's aggregated presence information to the PS. This solution therefore relies on the operator's centralized PS.

Today's users own advanced smart phones that support both cellular communication protocols (e.g., UMTS, High Speed Downlink Packet Access (HSDPA), and Enhanced Data rates for GSM of Evolution (EDGE)) and wireless data services (e.g., WiFi and Bluetooth). Communication is not limited to telephony anymore, as millions of users use IM, SMS, email, Twitter, and Facebook everyday. There is a growing trend among users to trust web services to keep their personal information, calendars, pics, presence and so on. Although these services handle very similar information, they do not interoperate with each other. Such a lack of service cooperation and automation forces users to check services one after another and manually copy data or configure services based on other services. The WWW is advancing towards greater personalization. Services on the Web, such as social networking, e-commerce, or search sites, store user information in order to profile the user and target specific products or ads of interest. Since web service functionality is increasingly relying on user information, a user's context is becoming more crucial towards creating a personalized set of services within the Web. There is a world of communication and information technologies, and the user context is available from multiple sources in the WWW. Users however do not have the tools to exploit such a world full of possibilities. A framework is, therefore, needed where multiple services can be composed and executed proactively for a particular user within a certain context. This framework should let the end user create service compositions and execute these compositions based on his circumstances proactively. Although many authors have been interested in this exciting topic in the last decade, complete solutions do not yet exist. Most authors describe or propose theoretical work. The few that present real implementations are partial solutions or domain-specific. CPL [152], LESS [47], SPL [153], VisuCom [154] and DiaSpec [155] are attempts to allow end users to create services, but they are all limited to controlling call routing. Also, CPL and LESS use XML, and hence even simple services require long specifications. Moreover, XML-based languages are difficult to read and write for non-technical end-users. Although the CPL specification does not consider presence information, the authors of [156] and [157] add some basic presence attributes to CPL. DiaSpec is very low level. Writing a specification in DiaSpec and then developing a service using the generated framework is definitely not suitable for non-technical end

users. The authors of DiaSpec extended their initial work to support services beyond telephony [158], which include sensors and actuators. However, DisSpec is still only suitable for advanced developers. SPL is a scripting language that is suitable for end-users but only for telephony events. VisuCom has the same functionality as SPL, but allows users to create services visually via GUI components. CybreMinder [159] is a context-aware tool that allows users to setup email, SMS, print out and on-screen reminders based not only on time but also location and presence status of other users. This tool uses local sensors to detect a user's location, and only displays reminders to the end user (i.e., this does not take any actions). Also, CybreMinder is not as powerful as scripting-based systems due to its form-based nature. Task.fm [160] is a similar SMS and email remainder system that uses natural language to describe time instants when email or SMS reminders will be sent. However, Task.fm only supports time-based rules and does not include information from sensors. This tool does not take actions other than reminding users via SMS, email or phone call.

Some authors have addressed the need to provide proactive, user-centric services. Ubiphone [45] is a human-centered ubiquitous phone system for handling phone calls proactively. This system takes intelligent and proactive decisions based on the user context. Users, however, do not seem to have control on the actions to take when particular events happen. This platform relies on a centralized server that retrieves user context and handles it through a tree of Ontology Web Language (OWL) ontologies. Other research works on call handling are [44], [43], and [161]. The authors of [44] present an application router that determines what composition chain to follow based on presence information. The bindings between the received SIP requests and the invoked applications are statically configured in XML documents. The authors of [43] provide an inbound call routing service within an IMS AS (see Section 2.4.1), which allows incoming calls to be routed to the right person dynamically. This service makes suggestions to the callee based on its, and caller's, presence information. The work described in [161] composes widgets on web pages based on Event-Condition-Action (ECA) rules that are encoded by XML. This composition is only triggered by call requests.

In the WWW, full proactivity will come with the automatic discovery, composition and invocation of web services, as described in Section 2.8.1. SWORD [162] was one of the first prototypes for web service composition. However, this tools offers a quite

limited composition that is not automatic and its scripting language is targeted at developers. Ezweb [163] is a graphical tool whereby users can connect web services manually. However, this tool does not provide automatic web service discovery or a language for composing services. Moreover, service composition is not context-aware and proactive. Yahoo Pipes [164] is other graphical tool for web service composition. However, it presents the same limitations as Ezweb and its GUI is not really easy-to-use and intuitive, which makes it difficult for non-technical users. A prototype described in a research paper [165] offers event-based web service composition. This means that service composition is triggered by events such as changes in the user's context rather than end users. However, this work does not provide any language or tool for specifying the web service compositions and events that trigger them. The authors seem to implement low-level compositions that may be personalized according to user preferences. Thus, this work does not offer end users control of service composition. This prototype seems not to be available in the Internet. The authors of [166] describe a platform for users to create context-aware event-based compositions. Users define their compositions through statecharts, which are translated into lower-level control tuples. These tuples determine pre/post-conditions, service invocation and exception handling. The platform only considers location and temporal conditions as context, and provides a GUI rather than a scripting language. The work [167] tackle event-based service composition, although it does not provide any platform. Only two illustrative examples are given, which execute an image render service that modifies an image when some time event occurs or the user types his or her mood.

To sum up, to the best of our knowledge, there is no implemented platform for allowing end users to compose services of different kind based on events. The current solutions are not proactive because the end-user is who triggers the composite services or only provides template-based compositions (i.e., the user is not who defines the compositions). There is neither a platform for event-based web service discovery. The composition tools that take user context into account only consider a limited set of context. The studied tools's GUIs are quite limited and not flexible for non-technical users. The scripting languages provided by some tools are neither suitable for non-technical users and only support a limited set of context information. Moreover, none of the studied tools proactively discover web services based on the user preferences.

2.8.1 Automatic Web Service Discovery, Composition, and Invocation

Web services have emerged as a standard mechanism for accessing information and software components in an automatic and interoperable way. To call a web service, a program has to send the service one or more messages, normally encoded by XML, and then the program receives XML replies containing the returned values. This exchange of messages has been standardized mainly through two technologies: Simple Object Access Protocol (SOAP) [168] and Web Services Description Language (WSDL) [27]. WSDL defines the syntax of the input and output messages of the web service, as well as other details needed for the invocation of the service. Although, WSDL is independent from the underlying protocol and encoding, its specification only defines the bindings to SOAP, HTTP GET/POST and MIME. SOAP is a protocol that allows web services to exchange XML-based objects over HTTP. SOAP provides a request/response handshake protocol and message formats for letting web services communicate with each other. These technologies standardize web service communication and, hence, provide interoperability. However, they do not deal with the semantics of web services and, therefore, it is developers' responsibility to compose semantically correct web services. Rich semantic specification of web services is necessary to enable flexible automation of service invocation and composition. To meet this need, the Semantic Web [169] is working on languages and architectures for automating web service discovery, invocation and composition. Resource Description Framework (RDF) [170] and OWL are standard languages for describing ontologies, and constitute the basis for the Semantic Web. An ontology is a formal representation of the knowledge within a domain by a set of concepts and the relationships between these concepts. An ontology consists of "classes" of objects, their relationships and axioms that place constrains on classes and the types of relationships permitted between them. These axioms provide semantics by allowing systems to infer additional information based on the data explicitly provided.

OWL-S [25] is an ontology for web services expressed in OWL, which includes three primary sub-ontologies: the service profile, service model, and grounding. The service profile is used to describe what the service provides for and requires of agents (i.e., functional, classification and non-functional aspects). The service model is used to describe how the service is used (i.e., inputs, outputs, preconditions and results) and the grounding is used to describe how to interact with the service (i.e., binding to WSDL).

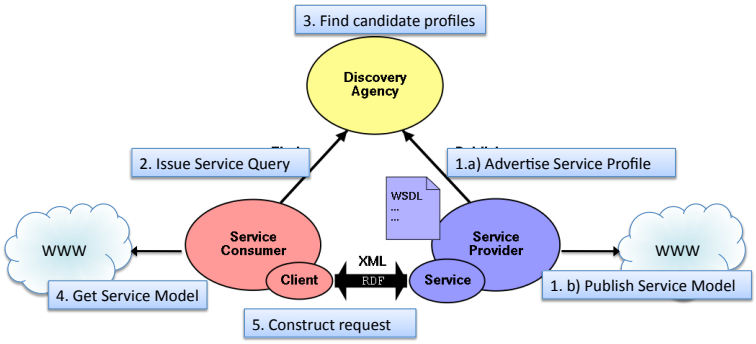


Figure 2.7: Web Service Composition

The first two ontologies are thought of as abstract characterizations of services, while the third one provides necessary concrete details to interact with services. Figure 2.7 shows a typical interaction to discover and invoke a web service automatically. A third-party, which is called "Discovery Agency" in Figure 2.7, keeps advertised service profiles and matches them with service discovery requests. To make a service discoverable and reachable, the service provider registers the service's profile into the Discovery Agency and makes the service's model available in the Web. When a service customer needs to look up for a service that satisfies certain properties or capabilities, it sends a request to the discovery agency. This request is contrasted with the advertised services' profiles and, as a result, the matching ones are sent to the requester. Then, the service consumer selects the service with which to communicate and downloads its model from the web (i.e., the service profile specifies the model's web link). Once the service consumer knows the service's model it is able to interact with the service via HTTP. Automatic web service composition and interoperation is a step further from web service discovery. This involves discovering, selecting, composing and interoperating web services automatically in order to perform some complex task given a high-level description of an objective. OWL-S supports it by specifying services' prerequisites and consequences as well as composite data flow interactions. Service composition could be performed by the service consumer itself or a third party.

2.9 Challenges in Presence Services

This section analyzes the main challenges and difficulties presence services need to overcome for its success, which basically concern interoperability, privacy, user-centric

customization, scalability, large-scale scenarios, mobile applications, and information consistency. This section also gives an overview of the research efforts related to these topics.

2.9.1 Interoperability

Two systems are interoperable when they are able to exchange information and use this information. Presence information came into being in IM and PTT applications and has been extended to social networks. The fact that these applications typically compete for the largest user population has derived a serious interoperability problem. In a fight for attracting the greatest number of users, IM and social networks are reluctant to let their users communicate with each other and share their presence information. To face this problem, the IETF has proposed two standards for IMP, namely XMPP and SIMPLE. There are detractors and supporters of both protocols. However, the standardizing organisms are inclined towards SIMPLE for several reasons. SIMPLE is an extension of SIP, which inherits most of its characteristics from HTTP and Simple Mail Transfer Protocol (SMTP). This fact is an advantage because these two protocols are the most successful Internet protocols. Since SIP is based on HTTP, SIP service developers can use all the service frameworks for HTTP, such as CGI (Common Gateway Interface) and Java servlets. Moreover, SIP is a text-based protocol, which makes it easier to extend, debug and use to build services. These reasons led the 3GPP to adopt SIP as the session control protocol for the IMS, and hence SIMPLE for managing presence and IM in the IMS. SIMPLE has become the standard protocol for providing presence-based services in NGNs. We refer the reader to Section 2.4 for further information about the IMS and NGN.

Besides the industry, the academia and research community is concerned about the interoperability barrier in presence-based systems. The authors of [73] address the need to provide user-centric, provider-agnostic presence services. They claim that service providers offer presence services that are tightly bound to the provider network, even when these services are built upon standard protocols. These provider-centric presence services are restraining creation of ubiquitous and dynamic presence applications. The authors of [171] performed experiments on interoperability testing of the presence service in an open source IMS platform. They applied some OMA test cases for SIMPLE to two IMS clients from different vendors. The experiment results show

that the tested IMS clients do not support presence functionalities such as composition rules, notification filtering and partial presence information. Moreover, these clients are not capable to handle presence information that comes from multiple user devices. Furthermore, the tested PS does not work properly when the watchers unsubscribe. These results show that ensuring that all the components involved in a presence service behave in the same way is hard even when they agree on a standard protocol. The authors of [172] study the interoperability between SIMPLE and IMPS, and propose a protocol mapping and framework for making them interoperable. In [173], translating gateways are proposed to solve the problem of interoperability between MSN, AOL, Jabber (XMPP) and Yahoo IM networks. The authors of [174] propose a middleware that allows users to roam among IMP service providers with different IMP protocols while maintaining their IMP sessions. This middleware permits to load the visited network's IMP implementation into the user's device and reconstruct her presence information by communicating with her home IMP network. The authors of [175] describe an approach for letting Internet services know about the presence of cellular users. A centralized server discovers a user's presence information by analyzing cellular events such as call establishments, location updates, network registration and SMS reception. This server interacts with Internet services via SIP/SIMPLE. In [176], an architecture model for multimedia communications based on NGN is described. The authors' goal is to ensure that a suitable level of interoperability is reached by all the components involved in the communication process. The authors performed numerous experiments of end-to-end residential communications between different IMS providers in Europe. Regarding the presence service, half of the tests failed by internal AS problems and SIP interoperability between terminals and the network. The authors conclude that this interoperability issues are due to the flexibility of SIP and the coexistence of 3GPP, Telecoms and Internet converged Services and Protocols for Advanced Network (TISPAN), and IETF standards. Moreover, user acceptance tests showed that the ergonomics of presence-based applications needs improvements, and users demand homogeneous user interfaces regardless of the terminal used.

2.9.2 Privacy

The success of presence information is to a great extent due to our innate curiosity to see the context and circumstances of others that are important to us. However,

a user's presence information is likely to contain very sensitive information such as location, meetings, likes and social relationships. A number of questions, therefore, arise such as "Where is the limit between the satisfaction of watchers and the privacy of users?" and "Can users trust service providers to maintain their presence information?". The answers are fuzzy and always depend on the needs and requirements of each user. Presence systems should provide privacy mechanisms flexible enough to accommodate the needs of users. This would empower users to hand their presence information in, and hence privacy policy is a required precondition to the success of presence applications. However, today's presence standards are lacking of fine-grained policy control that permits to customize presence information based on the requester. A user may wish to notify different, for example, activities based on the recipient (e.g., "in a meeting" for his boss and "out of town" for his customers). The use of presence information in enterprise environments, which was mentioned in Section 2.1, stresses the importance of fine-grained control of the disclosure of presence information. Enterprise policy authorization mechanisms should provide employees with the capability to control their presence information in a way consistent to their corporations' goals and privacy policies. Moreover, inter-domain federation introduces the non-trivial challenge of sharing user information and ensuring user-defined privacy policies on this information across different administrative domains. Federated domains should be able to authenticate each other and users within other federated domains. The authors of [64] discuss about the need of privacy policies in presence systems more deeply. In the frame of SIMPLE, presence authorization rules [177] allow specifying the pieces of presence information that is delivered to certain watchers. However, these rules do not permit to customize the value of presence attributes based on the watchers. This mechanism also allows automatically deciding about subscription requests. The mechanism to approve or deny subscriptions in real time is provided by the *winfo* event-package [145], which allows subscribing to SUBSCRIBE requests that are in place and waiting for approval, and the XML format for describing this kind of event [178]. The IETF GEOPRIV WG [136] works on extensions of PIDF for privacy rules that control the disclosure of location information.

2.9.3 User Customization

The ability to change the behavior of presence applications based on the user's needs, requirements, and circumstances over time will bring the success of these applications. To motivate the use of presence applications that handle rich presence information, users need to feel that they have the control on their presence information and communications. Users that are distrustful of service providers will be reluctant to use presence applications or will end up handing in a reduced set of their information. In our opinion, presence services should grant users control on privacy, communication, presence information sharing and subscriptions to others' presence information:

- Privacy. The user should be able to set rules to deliver the right presence information to the right watchers, as described in Section 2.9.2. These rules may change over time based on the user's context and circumstances. This includes the ability to customize the value of presence attributes depending on multiple factors, namely time, context and recipients. It would be also recommendable to provide the users with mechanisms to reject or accept subscription requests beyond the traditional blacklists and whitelists. For instance, a user may accept a subscription request if the requester works in the user's corporation, lives in his or her home city or shares any of his or her hobbies.

- Communications. The user should be able to set his preferences on how others communicate with him, which may depend on the time of the day and the requester's and user's context. This capability is known as call handling. Section 2.3.1 mentions some recent works that merge presence and call handling. However, most of them rely on predefined rules for handling the user's communications that the user can not modify. A few of them provides the user with tools to define his own rules about how to handle communications. However, these tools are not flexible and only include a very limited set of presence information and use cases. For example, LESS [47] is a scripting language that includes a few presence attributes for handling communications. The work [156] extends CPL, which is other scripting language for call handling, with a few presence attributes. The authors of [65] describe an IMS-based network architecture for managing presence information, which provides more flexible, user-centric preferences on user communications.

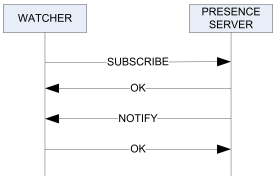


Figure 2.8: SIP presence subscription flow

- Handling of presence information . The user should be able to handle his presence information automatically and decide the presence sources that contribute to his presence at any moment. This would include rules such as "don't include presence from my personal phone during working hours", "don't include my location after work", "when I get my office, set my activity to working" and "During any event in my calendar, set my state as busy".

- Subscriptions to others' presence information. The user should be able to choose the circumstances under which he wishes to know about others' presence information. SIMPLE event notification filters [141] allows users to filter the presence information that they receive based on the subject of the information and the kind of change on the information (i.e., modification, addition or removal). However, we think that this filtering process should also take the recipient's circumstances into account. For instance, the user may change his notification filters over time (e.g., "let me know my boss' activity from 8.00 to 17.00") or based on his context (e.g., "let me know my friends' location when I get out of my office").

2.9.4 Scalability

Presence subscriptions must be refreshed periodically to prevent their lifetime from expiring, which would result in the elimination of their subscription state. A subscription's lifetime is restarted by sending a re-SUBSCRIBE message, which entails the exchange of four messages as shown in Figure 2.8. On the other hand, typically whenever a presentity changes its state, a NOTIFY message is sent to the watchers that are authorized to see the presence change, as shown in Figure 2.9. These operations make presence-based applications generate a great amount of traffic as the number of presentities, watchers and presence changes increases.

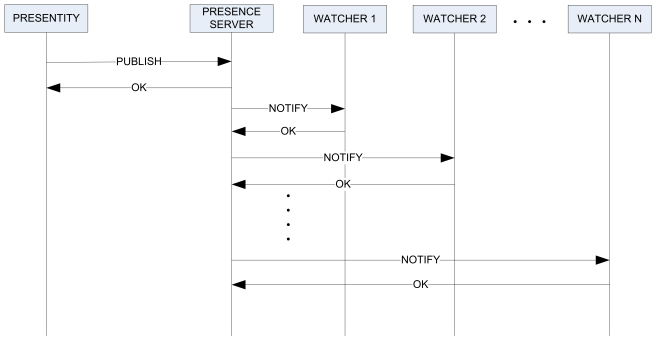


Figure 2.9: SIP presence publication flow

Presence traffic overload becomes even more harmful and critical in presence-based LBSs since frequent location updates have to be timely disseminated through PIDF-LO documents [38]. Even if presence updates and application traffic are discarded, signaling traffic for keeping presence subscriptions alive may be pretty considerable. Such an amount of signaling traffic may make the presence service unfeasible in the IMS because of the number of centralized servers that presence flows traverse, as explained in Section 2.4.1. Figures 2.10 and 2.11 show the messages that IMS servers need to exchange when a user subscribes to his or her resource list. When the RLS receives the subscription request (Figure 2.10), it subscribes to the contacts in the user's resource list. Figure 2.11 shows this process, which is repeated for each of the presentities in the resource list. The subscription request is forwarded via the S-CSCF in the RLS home network to the I-CSCF in the presentity's network. The I-CSCF queries the HSS to found out the S-CSCF that is allocated to the presentity and forwards the request to this S-CSCF. The authors of [179] conclude that presence traffic can constitute above 50% of the total traffic handled by the CSCF servers in IMS. This result is alarming given that IMS is thought of as the network infrastructure that will support NGNs. The performance analysis in [180] shows that SIP signaling traffic introduces long transmission delays on the UMTS network. They study the end-to-end delay that users of instant messengers perceive. This delay may reach so high values that IM could not be considered as an instantaneous service anymore. Approximately 70% of this delay is due to the network core and, hence, optimizing traffic on the radio access side is insufficient for providing multimedia services in real time.

Since presence information plays a key role in some applications that are used

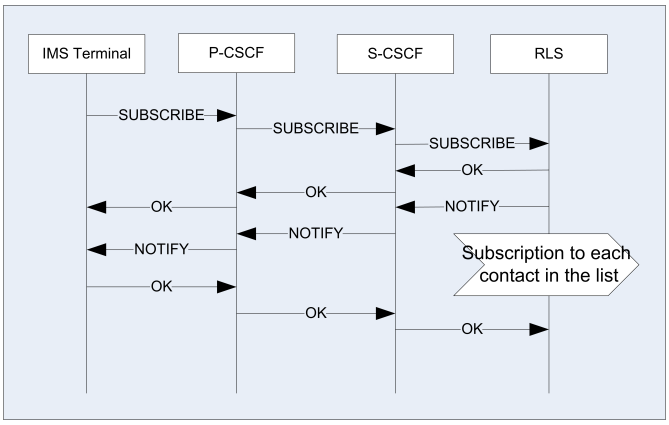


Figure 2.10: Watcher subscription in the IMS

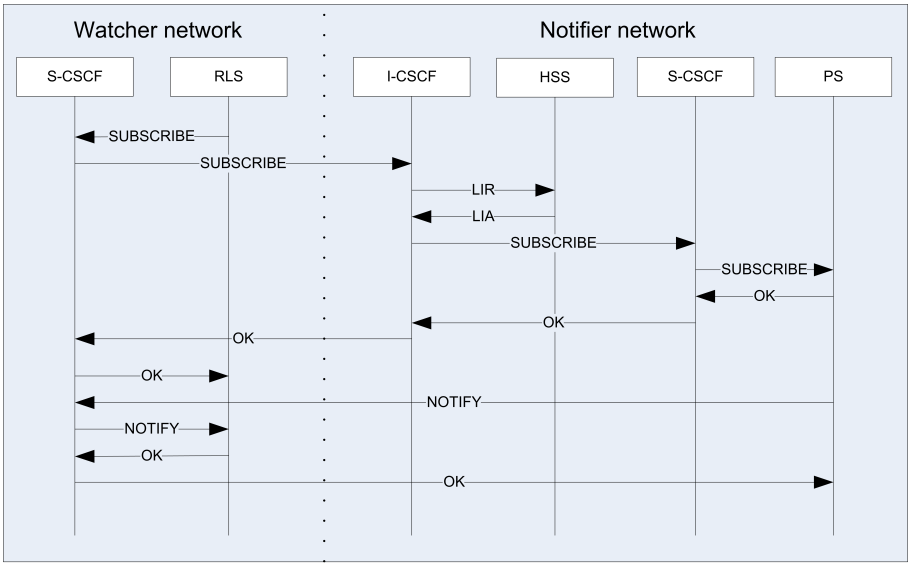


Figure 2.11: RLS subscription to a presentity in the IMS

worldwide, such as IM (see Section 2.1), presence traffic overload should be carefully considered. The authors of [12] study the traffic generated by MSN and AOL instant messengers within a corporation, and conclude that presence traffic constitutes the most part of this traffic. Obviously, it is necessary to reduce the number and size of messages that are sent through the network in SIP flows as well as the number of network elements through which the messages pass. Presence traffic should therefore be carefully studied in order to design and apply the most suitable optimization techniques. Presence notifications constitute a large part of the excessive traffic generated by presence applications [181][182]. In the SIMPLE framework, there are several techniques for reducing the presence subscription traffic as described in Section 2.7.4. These techniques include resource list subscriptions [121], partial notifications of presence [138], event notification filtering [140], notification rate control and conditional notifications [147]. Although SIMPLE allows each event package specification to define the maximum rate at which notifiers can send notifications, the notification rate is determined by the notifier rather than the subscriber. The Internet-Draft [183] allows a subscriber to set the maximum and minimum rate of event notifications generated by the notifier by means of two new event parameters in SUBSCRIBE messages: "max-rate" and "min-rate", respectively. Thus, the notifier generates notifications at a rate that is higher than the minimum rate and lower than the maximum rate. The maximum notification rate must be always greater than or equal to the minimum notification rate for a notifier.

Some research papers have addressed the optimization of presence protocols. On one hand, the notifier sends a notification when the time since the most recent notification exceeds the reciprocal value of the "min-rate" parameter regardless of the subscription's state information has changed. However, it may be combined with conditional notifications for suppressing the notification if no presence change has occurred. Moreover, notifications sent at the minimum rate may contain partial-state presence documents for notifying only the changes that have occurred from the last notification. On the other hand, the notifier ensures that a time longer than the reciprocal value of the "max-rate" parameter elapses between two consecutive notifications. An IETF Internet-Draft [184] defines new event filters for location information that are based on movement, speed, entering or exiting a region, changes in address labels and types of location. The authors also propose the mechanism in [183] to set up minimum and maximum times between two consecutive location notifications. In [185], two strategies

are described to reduce the number of messages in presence subscriptions. One allows subscribers to unsubscribe while keeping the RLS maintaining the subscription state for a certain time. The subscriber sends a SUBSCRIBE message with the header "expires" set to zero and a new option tag telling the time during which the RLS should keep subscribed to the presentities in the resource list. If the subscriber does not resubscribe during this time, the RLS eliminates the subscription. This optimization is intended at mobile applications, which are likely to get inactive during long periods. The other strategy proposed in [185] allows specifying a maximum notification interval between two notifications through the Ut interface defined by XML Configuration Access Protocol (XCAP). The authors of [186] propose maintaining hard-state presence subscriptions between end users and the IMS with the aim of saving presence traffic on the network access link. When a watcher subscribes to a presentity, a hard-state subscription is created between the end user and an AS. This does not require the end user to resubscribe for keeping the subscription alive. In turn, the AS creates a soft-state presence subscription to the presentity by sending periodic presence resubscriptions. An out-of-progress IETF Internet-Draft [187] addresses the temporal pause of presence notifications. The authors propose an extension to pause and un-pause notifications, and to perform pull requests within an established subscription dialog. This is achieved by setting a new header in SUBSCRIBE messages, whose possible values are "off", "on" and "once" for pausing, un-pausing and pulling, respectively. Although this work did not progress, we find it remarkable since no other authors have proposed a method for pausing and afterwards pulling presence notifications without removing the subscription presence state. Although the authors of [185] discuss about pausing notifications, they do not provide a method for triggering one-time notifications in pauses. Some queuing systems for controlling presence notification rate have emerged recently. The authors of [188] propose a delayed threshold in order not to notify watchers of presence changes immediately. When the PS receives a presence publication, it starts the delayed timer, and once this timer expires it sends the corresponding notifications. This allows aggregating presence changes that occur during the delayed timer, thereby saving notifications. This mechanism requires a different buffer for each presentity, which may be costly in large-scale presence services. Moreover, notifications are always delayed, which introduces unnecessary delays in notifying when the arriving publication rate is lower than the desirable output rate. The authors of [189] propose TNTC, which is a

token-bucket based mechanism for controlling the notification rate that the PS injects into the network core. This mechanism ensures that the maximum output rate is the token generating rate of the bucket. If the presence publication arrival rate is higher than the token generating rate, the tokens will be consumed at some point. When it happens, the arrival publications are queued until new tokens are generated. During this time, new publications replace older publications in the queue; thus, this mechanism saves presence notifications. The main difference between this mechanism and other notification control mechanisms [183][188] is that TNTC can change the maximum notification rate over time based on the publication arrival rate, while the others assume a predefined rate. Although the authors of both [188] and [189] are concerned about the probability of watchers accessing to consistent information, the notification rate, either static [188] or dynamic [189], is chosen just to meet the PS policy on output rate.

The PS is the intermediary in any presence publication and notification and, therefore, may easily become a bottleneck as the number of presentities, watchers and presence publications increases. Some authors have addressed the need to reduce the presence load at the PS. The authors of [190] propose a routing protocol for presence subscription requests that are exchanged between distributed PSs in the IMS. They claim that this protocol is useful in a kind of application denominated as Live Resource Finders (LRFs), which dynamically matches clients and service providers based on their presence information. The authors of [181] propose a queue system with vacation times for NOTIFY messages at the PS. Their goal is to provide the PS with more time to process other messages. The optimal value of the vacation time is calculated for preventing the queue from becoming full. Nevertheless, this system does not save presence traffic because notifications are not aggregated during the vacation times. A maximum notification rate is not ensured because, once the PS start processing the NOTIFY messages, it does not stop until the queue is empty. In [191], dropping presence publications selectively is proposed for reducing presence traffic load at PSs. The authors describe a queuing mechanism in which the publication arrival rates are deduced from a particular Markov chain. The research paper [192] tackles the scalability of IMP systems in terms of databases. The authors analyze the latency of several database architectures and recommend one based on subscriptions.

2.9.5 Presence Federation Scenarios

Although the optimization strategies mentioned in Section 2.9.4 reduce the number and size of the messages exchanged through presence subscriptions, they do not tackle the volume of subscriptions. They are not therefore efficient enough in large-scale presence federation scenarios where millions of users in a domain subscribe to millions of users in other federated domains. IM is the most evident example of presence-enabled application on a planetary scale, as described in Section 2.1. In such large-scale scenarios, SIMPLE generate a volume of subscriptions that may be unbearable, one for each different (watcher, presentity) pair. The authors of the IETF Internet-Draft [193] makes a valuable contribution towards measuring inter-domain presence traffic. They analytically estimate the number of messages and bytes exchanged between two federated presence domains in three scenarios with different levels of federation. The level of federation determines the number of cross-domain (watcher, presentity) pairs and the number of presence changes. The reported results show that the domains exchange 101.880, 152.820 and 44.046.000 megabytes during a session of 8 hours from lowest to highest level of federation. The authors show that presence traffic overload may become unbearable in large-scale scenarios. They also discuss about the complexity of the presence service and the optimization techniques that may help in increasing its scalability. Moreover, the authors describe a technique called dialog optimization, which basically consists in a federated RLS that keeps a single subscription between a watcher and its presentities in a particular federated domain. However, the estimation of this strategy's traffic [193] shows that it does not help in reducing presence traffic, and hence the authors discourage it. The authors of [194] describe a technique called Common Notify (CN) that consists in sending a single notification message to the watcher domain. This message contains the presentity's complete presence information and the watcher domain is in charge of providing watchers with the piece of presence that they are authorized to see. Although this optimization does not reduce the number of subscriptions, it does reduce the number of notifications drastically. In addition, the authors also describe batched notifications, which consist in aggregating the presence documents of multiple presentities in a single NOTIFY message. The authors of other IETF Internet-Draft [195] describe a mechanism called View Sharing (VS) that reduces the number of presence subscriptions between two federated domains. The number of

subscriptions is determined by the number of privacy rules that the presentities set up to their watchers. Unfortunately, the aforementioned Internet-Drafts [194] [195] do not provide any traffic estimation and are no longer active in the IETF.

2.9.6 Wireless Communications

As described in Section 2.9.4, presence applications are likely to generate a large amount of signaling traffic. This overload may restrain the use of presence applications on mobile devices with limited processing resources and battery life. Mobile presence applications need to handle two kinds of traffic: presence subscriptions and presence publications. In the SIMPLE framework, subscription traffic includes traffic for handling presence (about a presentity [120] or a resource list [121]) and winfo events [145]. These subscriptions involve periodic subscription refreshes and notifications, which are likely to involve a great amount of signaling traffic. Section 2.9.4 mentions some proposals for optimizing subscription-related presence traffic as, for example, [121][138][140][183][147] and [184].

Besides subscription-related traffic, limiting the number of presence publications may be crucial in mobile presence-enabled LBSs. SIMPLE-compliant LBSs publish the presentities' location information encoded by the XML-based format PIDF-LO [38]. Since the presentities' location information is likely to change very frequently, such frequent publications may involve harmful consequences: PS overload, large amounts of notifications on the network core, and overconsumption of radio access bandwidth and wireless user devices' battery life. In order to reduce the size of presence publications, SIMPLE defined partial presence information [137] to allow presentities to publish only the changes that have occurred from the last publication rather than their complete presence information. SigComp [142] and its presence-specific dictionary [143] can be used to compress PUBLISH messages. The OMA Presence SIMPLE defines the SOURCE-THROTTLE-PUBLISH parameter, which indicates a minimum passive interval that must elapse between two presence publications. This parameter is configured in client devices via Over-The-Air (OTA) provisioning. OMA also introduces Presence Network Agents (PNAs) [196] into operator networks for saving presence publications. These agents publish presence information that can be deduced from the network on behalf of presentities. Unfortunately, there still is much information that cannot be extracted from network services such as person-related information (e.g., mood and

willingness). The patent [197] proposes a solution for limiting the number of presence publications based on certain conditions that are set to the presentity. In the research community, to the best of our knowledge, only the authors of [65] tackle the need to reduce presence traffic due to publications. They proposed an IETF Internet-Draft [198] that defines a new event package for allowing presentities to be up to date with their PSs' presence requirements. Thus, presentities only publish the presence attributes needed by their PSs, rather than their complete presence information. This Internet-Draft expired without further progress, probably due to the overhead introduced by a new kind of subscription between the presentity and the PS.

Besides traffic optimization in wireless networks, mobile presence applications face limited user devices, multiple access technologies and unreliable communication channels. Thus, there is a number of challenging issues that should be considered carefully when treating presence traffic:

- Providing confidentiality and security may add a considerable traffic overhead, and hence a tradeoff between security and efficiency needs to be found according to each real-world scenario (i.e., security and traffic optimization policies, network traffic status, user devices' capabilities, etc.).

- Global connectivity involves mobile applications interacting with Internet-designed presence applications. As Internet applications are not concerned about traffic overload, they may generate an amount of traffic that is excessive for mobile applications. Thus, network servers should drop such an excessive traffic off when interacting with mobile users.

- Presence information of mobile users may be obtained from third-parties such as network operators and service providers. Thus, presence aggregation and reasoning play a key role in mobile presence applications.

- Mobile users can be connected to heterogeneous systems, that is, devices with different capabilities and multiple access technologies. Device characteristics include information related to software (e.g., supported libraries and operating system), hardware (e.g., CPU speed, display and memory), and communication link (e.g., bandwidth). Traffic optimization should be performed according to the user device's characteristics and network congestion status.

- Users may switch between terminal devices quite frequently while they are logged on a presence application. For instance, a user may connect to her mobile phone in the early morning, switch to a Personal Computer (PC) at her office and occasionally connect to her laptop. Mobile presence applications need be able to support session mobility by reconstructing application session and aggregating presence from multiple sessions.

2.9.7 Differentiated Treatment and Consistency of Presence Attributes

The diverse nature of presence information reinforces the need for differentiated treatment for presence attributes. Presence information may be composed of a diversity of information such as location, profile, personal, device, and service information. However, not all the information is needed by watchers at any given time. Watchers are, therefore, likely to have different preferences about what presence attributes are more important and when these attributes are needed. For example, a watcher may be very interested in location information, while mood or activities may be insignificant. Presence information was conceived to be useful for watchers to establish communications with presentities. Therefore, the watchers' needs for presence attributes should be taken into account to only request the needed information from presentities so as to unnecessary traffic. Limiting the rate at which watchers are notified for the sake of optimization traffic may result in watcher applications keeping obsolete information. This may make the presence service useless, since its success is actually due to the instantaneous knowledge of presence changes. Watcher applications or the users themselves may take inadequate decisions or assumptions based on wrong presence information. Thus, the watchers' needs on notification rate and information consistency should be considered when delaying presence notifications. Such needs may vary from some presence attributes to others.

The authors of [65] point out the lack of strategies for configuring requirements of urgency, cost, rate, and accuracy in both notification filters and presence sources. Facing this issue, they propose a hierarchical structure of servers within the IMS. One of the major functions of these servers is to balance source publications based on the needs of watchers. The authors highlight the need to allow watchers to subscribe to only the presence subsets of interest for saving unnecessary presence traffic. Moreover, they proposed an IETF Internet- Draft [198], which defines a new event package that

allows presentities to subscribe to their PSs requirements on urgency, rate, cost, etc. Thus, presentities should publish presence changes according to PS requirements. This draft expired without any further progress probably because the benefits of this new subscription do not make up for the overload and complexity it causes. Other work [66] introduces the arguments for which presence attributes should be treated differently depending on their importance. The authors propose decay functions to describe how the accuracy of presence attributes decreases over time in order to prevent watchers from retaining obsolete values. This work emphasizes the need for different decay functions depending on the presence attributes: each attribute has a different nature and hence it changes at a different rate. For example, a user normally publishes changes in basic personal information (such as state or activity) much more frequently than changes in the properties of his or her device. If the decay function of a presence attribute drops below its threshold value, this means that the probability that this attribute has changed is too high and is no longer reliable. If this happens, the PS should re-calculate the presence information that could be affected by a change in the attribute. If the presence attribute is binary (i.e., only two values are possible), its new value can be calculated automatically. Otherwise, the PS must somehow retrieve the exact value of the attribute, which is out of the scope of this work. This strategy is intended to prevent PSs from maintaining obsolete values of presence attributes, whose publications have somehow been lost in the network or when the rate of publication is being limited. Unfortunately, SIMPLE does not provide any mechanism for pulling presence publications, as described in Section 2.9.8.

As described in Section 2.9.4, an IETF Internet-Draft [183] defines two new event parameters, namely “min-rate” and “max-rate”, for subscribers to specify the desired minimum and maximum notification rate. Thus, the notifier generates notifications at a rate that is higher than the minimum rate and lower than the maximum rate. Given that there can only be a single minimum and maximum rate per subscription, all the presence attributes associated with a subscription are notified at the same rate. Since the minimum rate triggers a notification when the time since the last notification exceeds the reciprocal value of “min-rate”, this mechanism may generate much unnecessary traffic if conditional notifications and partial presence documents are not supported (see Section 2.7.4). In order to avoid the subscriber keeping obsolete information for too long, the subscriber should set the “min-rate” and “max-rate” headers according

to the rates desired for the most important presence attributes. This is however inefficient under certain circumstances. The notifications generated at the minimum rate may include presence attributes that change more frequently than this rate but are not important. A low maximum rate may delaying in notifying the subscriber of presence attributes that change much more rapidly than this rate for too long. This inefficient behavior is specially evident for RLS subscriptions since a resource lists presence information is composed of the presence attributes of all the presentities in the list. Event notification filtering [140] has been defined for subscribers to express what information presence notifications should include and when notifications should be delivered. The IETF Internet-Draft [184] defines new event filters for location information that are based on movement, speed, entering or exiting a region, changes in address labels and types of location. The authors also propose the mechanism in [183] to set up minimum and maximum times between two consecutive location notifications. However, as mentioned above, these times are applied to the entire presence information, which is especially inadequate for LBSs. Geographical coordinates change very frequently, and hence if they were relevant for the watcher, a low minimum and maximum rate would be required. In this case, all the presence information would be notified at so low rates. Even when geographical coordinates are not important, a low minimum rate may be set because of other presence attributes that are more urgent. This would involve notifying location changes at such a low rate although it was not necessary.

2.9.8 Pull vs. Push Models for Presence Updates

Presence systems usually adopt a push model in which presentities proactively inform their PSs of any change in their presence information. Likewise, notifiers send subscribers asynchronous notifications for letting them know about presence changes instantaneously. However, under some circumstances, pausing and un-pausing publications and notifications may be very convenient to reduce presence traffic. It happens when the watcher only needs the presence information occasionally, or well when presence changes are infrequent and the watcher can estimate when these changes occur. In these cases, pulling presence changes would probably generate less traffic than maintaining a subscription, since it generates much signaling traffic as described in Section 2.9.4. Currently, SIMPLE does not provide any means to this end. SIMPLE does inherently permits to know a subscription's resource state information instantaneously since

every SUBSCRIBE request makes the notifier send a complete state notification. Out of subscription state, SIMPLE also permits to fetch resource state information from a notifier by sending a SUBSCRIBE message with its expiration interval set to zero. However, this SUBSCRIBE-based pull method presents two important shortcomings. First, subscribers are still not capable to pull particular presence attributes instead of the complete presence information. Second, whenever the notifier receives a pull request, it needs to fetch the resource state information again, send it to the requester and, after that, eliminate all the state information. This behavior is not efficient if pull requests are periodic and may introduce delays. A watcher should therefore be able to pause and un-pause notifications while the notifier maintain the resource's information state up-to-date. There is not however any efficient and complete mechanism for pulling information and pausing notifications in SIMPLE. Only the authors of the IETF Internet Draft [187] address this problem. They define a new header in SUBSCRIBE messages, whose possible values are "off", "on" and "once" for pausing, un-pausing and pulling notifications, respectively. Although this draft did not progress, to the best of our knowledge, there have not been more similar proposals. The drawback of this solution is the lack of fine-grained treatment for presence attributes.

As regards presence sources, SIMPLE does not provide any pull model for presence publications and no other researchers have addressed this topic. Nevertheless, pausing and pulling presence publications may reduce much presence traffic in some scenarios. A presentity's presence publications may be paused when its presence information is not useful for other entities as, for example, when all the watchers are offline. Temporally pausing publications of location information may reduce much presence traffic when frequent updates are not needed. For instance, a user in a meeting is not supposed to change his or her location before the meeting finishes. Then, a PS may be aware of the user's schedule through his o her calendar, and pause location updates until his or her meeting finishes if traffic optimization was necessary. Attribute-based pulling mechanisms are necessary to provide enough flexibility and information consistency. For instance, a PS may need to receive any change in person-related information while pausing publications about device information that usually does not change. Then, the PS may pull device information occasionally to verify that this information has not changed. Moreover, A PS may need to know some of the presentity's presence

attributes immediately due to some information loss or some watcher needing these attributes immediately.

2.9.9 Behavior of Presence Applications' Users

The nature of presence information is very diverse; it may be used by a wide range of applications with different requirements and patterns of use. Thus, there is a severe lack of formal models of the behavior of presence applications' users. However, user behavior should be analyzed for designing efficient and scalable presence applications. Knowing patterns of behavior allows saving presence updates when they are least needed. For instance, studies about instant messengers [13][12][8] provide statistics showing that users usually communicate with a low number of their buddies (between one or five people). It is therefore reasonable to think that users only need frequent presence updates from a few buddies, while the rest of buddies' presence could be delivered less frequently. Unfortunately, to date, there is not statistics about presence changes in real-world applications because of the unpredictable behavior of these applications' users.

Almost no authors have embarked on modeling presence changes formally. To the best of our knowledge, only two research works [182][191] model presence changes through Markov chains. In these models, each state of a Markov chain represents a different combination of the values of the user's presence attributes. The authors of [182] describe a Markovian model for users' online and offline times in presence applications. In addition to the fact that only a presence attribute is analyzed, this study is limited in several aspects. The authors assume that online and offline times at night and day are independent from each other, which is not necessarily true in many real-world applications. The authors of [191] model presence changes as a set of hops between presence states, independently of presence attributes. Both models [182][191] do not provide realistic behaviors of presence users and seem to be designed to facilitate mathematical calculations. Moreover, both of them rely on Markovian stationary distributions, and hence state probabilities are time-independent. This assumption is not valid in the vast majority of presence applications, in which the probability of users having a particular combination of attribute values changes over time.

3

Filters for Fine-Grained Notification Control

Presence information is a broad concept that includes a wide variety of presence attributes (see Section 2.7.3). A presentity's watchers may only be interested in a subset of its presence information or they may have different levels of interest in its presence attributes, as described in Section 2.9.7. It is even more probable for resource list watchers because a resource list's state information is composed by presence information about all the presentities included in the list (see Section 2.7). SIMPLE takes the watchers' needs into account for filtering the content of presence notifications [140]. Although notification filters can be applied to any kind of subscription, they were conceived for reducing the size of resource list notifications. Neither SIMPLE nor other researchers are concerned about the subscriber's needs on presence attributes when limiting the rate of presence notifications. SIMPLE considers a single maximum notification rate for all the presence information associated with a subscription [110]. The authors of [183] describe a mechanism that allows watchers to specify the desirable maximum and minimum notification rate when subscribing. All the presence attributes that compose a resource's state information are therefore notified at the same rate set by either the notifier itself [110] or the subscriber [183]. This approach may be very inefficient if the watchers have different consistency requirements on presence attributes. In this case, the notifier should not delay in notifying a watcher of the most important attributes too much (i.e, the attributes on which consistency is most required). If the notification rate is set to inform the subscriber about the most important attributes frequently

enough, this rate will be high. Thus, presence attributes that are not very necessary to the subscriber but change frequently would generate unnecessary traffic. If the notification rate is set to perform update aggregation by delaying notifications as much as possible, this rate will be low. Thus, the important attributes that change frequently would not be updated frequently enough. As described in Section 2.9.7, the watchers' information consistency requirements should be taken into account when controlling the rate of notifications in order not to introduce inapproppiate notification delays.

A pull model approach may save much presence traffic under some circumstances, as described in Section 2.9.8. The vast majority of presence models use a push approach to inform watchers of their presentities' presence information. This means that a watcher is notified every time its presentities' presence information changes. On the contrary, in a pull model, no proactive notifications are sent but the watcher pulls the presence information when necessary. Generally, the push model is more suitable for presence systems. However, when traffic optimization is needed, the pull model may be more efficient than the push one in some cases. When the watcher only needs to know the presence information occasionally, maintaining a presence subscription is useless and introduce unnecessary overhead. When presence changes are not frequent and the watcher can estimate when these changes occur approximately, instantaneous notifications do not compensate for the overhead introduced by a presence subscription. Pull requests may be efficiently combined with pauses in notifications. For example, when the user is not active (e.g., the mobile presence application is in background, the PC is locked, etc.), presence notifications may be paused, and un-paused once the user become active back. Temporally stopping notifications of presence attributes that change frequently, such as geographical coordinates, may save much unnecessary presence traffic. As described in Section 2.9.8, SIMPLE does not provide any mechanism for pausing notifications and pulling information when necessary.

We enhance SIMPLE notification filters for supporting fine-grained rate control of presence notifications, and pulling and pausing notifications of particular presence attributes. Section 3.1 introduces the concept of multi-throttling, and Section 3.2 proposes some extensions of the XML schema document for SIMPLE notification filters. Some conclusions are given in Section 3.3.

3.1 Multi-Throttling

We denominate multi-throttling as the use of multiple minimum intervals for controlling the maximum rate at which the presence attributes that compose a presentity's presence information are notified. A minimum interval is a passive time that has to elapse between two consecutive notifications. All the presence changes that occur during this interval are aggregated and sent all at once when the interval elapses. Multi-throttling is therefore an effective strategy for reducing the number of update messages that are sent. This multi-rate mechanism reduces the number of times that non-important presence attributes are notified while keeping higher notification rates for more relevant attributes. Given the diversity of information that resource lists may enclose (Appendix B gives an example), this mechanism greatly would reduce presence traffic between watchers and their RLSs. For instance, it is reasonable that, during working hours, workmates' presence information is more relevant than that of friends. Thus, an RLS could set a long minimum interval to relatives while leaving workmates out of rate control (i.e., their presence updates are received instantaneously).

Figure 3.1 depicts what happens when a minimum interval elapses and full- and partial-state notifications are sent [138]. The left-hand side shows the presence changes that occur at the notifier during the set interval. The right-hand side shows the single update message that would be sent to the subscriber with throttling. During the throttling interval, all the changes are aggregated, and once the interval expires they are sent into a partial- or full-state document. A full-state notification contains the complete presence information. A partial-state notification only includes the presence changes that have occurred during the interval are sent. Thus, partial-state notification is more efficient when consecutive changes in the same attributes occur.

With multi-throttling, in order to reduce the number of message updates as much as possible, it is convenient to include any change in the entire presence information whenever a notification is going to be sent. However, a minimum interval may be a means of withholding some resource state information due to some privacy policy. For instance, a notifier may not want to notify a subscriber of location information so frequent that the subscriber can figure out the complete route taken by the subscribed presentity. Thus, the notifier may apply a long minimum interval to the presentity's location information for the subscriber. Such a minimum interval is mandatory in the

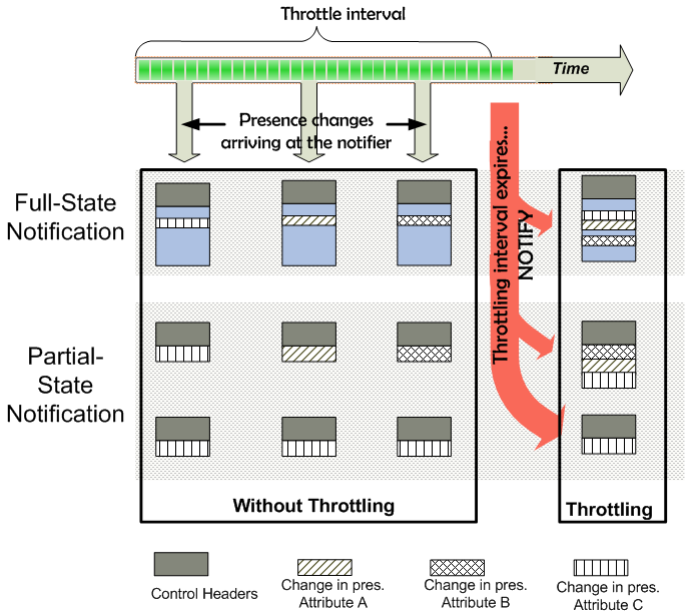


Figure 3.1: Aggregation of presence changes when a throttling interval expires

sense that the location information should not be aggregated into any other notification that is sent before this interval elapses. We classify control rate intervals based on their goal, as shown below.

$$\text{Control rate interval} \begin{cases} \text{Traffic optimization: } \begin{matrix}\text{Non-mandatory} \\ \text{minimum interval}\end{matrix} \\ \text{Other policy} \begin{cases} \text{Mandatory minimum interval (e.g., privacy)} \\ \text{Maximum interval (e.g., consistency)} \end{cases} \end{cases}$$

3.2 Enhanced XML Schema for Notification Filters

Notification filters express the information is of interest (i.e., content of notifications) and when this information should be notified (i.e., trigger conditions). The XML schema for notification filters [141] states that a filter can contain one or more *trigger* elements as well as one *what* element. The former describes the trigger conditions that must be satisfied to send a notification and the latter the content of this notification. Any filter must contain either a non-empty *trigger* or *what* element. If a filter does not contain any trigger condition, any change in the resource state information described by the *what* element triggers a notification. If there are more than one trigger conditions,

all of them must be satisfied to send a notification. In the case that the *what* element is absent, the notifications include all the resource state information. The schema [141] defines three kinds of trigger conditions: *changed*, *added* and *removed*. Appendix C shows an example notification filter document.

We consider notification filters as a tool for not only expressing watchers' preferences but also optimizing presence traffic. To this end, the XML schema document in [141] is enhanced for controlling the timing of notifications of presence attributes. We add four new types of trigger conditions: max-interval, min-interval, once and never. These new conditions allow (1) controlling notification rate, (2) temporally pausing and un-pausing notifications and (3) triggering one-time notifications (i.e., pull requests) of presence attributes. Below, the proposed XML elements are shown, which are defined within the complex type "TriggerType". We propose the new XML namespace urn:ietf:params:xml:ns:simple-filter:timing for this extension. Sections 3.2.1 and 3.2.2 describe the proposed elements, and Section 3.2.3 presents some general rules for these elements and shows an example document.

```
<xs:complexType name=‘‘TriggerType’’>
       <xs:sequence>
       <xs:element name=‘‘changed’’ type=‘‘ChangedType’’
                          minOccurs=‘‘0’’ maxOccurs=‘‘unbounded’’/>
       <xs:element name=‘‘added’’ type=‘‘xs:string’’
                          minOccurs=‘‘0’’ maxOccurs=‘‘unbounded’’/>
       <xs:element name=‘‘removed’’ type=‘‘xs:string’’
                          minOccurs=‘‘0’’ maxOccurs=‘‘unbounded’’/>
       <xs:any namespace=‘‘##other’’ processContents=‘‘lax’’
                         minOccurs=‘‘0’’ maxOccurs=‘‘unbounded’’/>

       <xs:element name=‘‘max-interval’’ type=‘‘MaxIntervalType’’
                          minOccurs=‘‘0’’ maxOccurs=‘‘1’’/>
       <xs:element name=‘‘min-interval’’ type=‘‘MinIntervalType’’
                          minOccurs=‘‘0’’ maxOccurs=‘‘1’’/>
       <xs:element name=‘‘once’’ type=‘‘empty’’
                  minOccurs=‘‘0’’ maxOccurs=‘‘1’’/>
       <xs:element name=‘‘never’’ type=‘‘empty’’
                          minOccurs=‘‘0’’ maxOccurs=‘‘1’’/>
       </xs:sequence>
 </xs:complexType>

<xs:complexType name=‘‘MinIntervalType’’ type=‘‘xs:nonNegativeInteger’’>
        <xs:attribute name=‘‘mandatory’’ type=‘‘xs:boolean’’ use=‘‘optional’’>
        <xs:attribute name=‘‘unit’’ type=‘‘xs:string’’ use=‘‘optional’’>
</xs:complexType>
```

```
<xs:complexType name=‘‘MaxIntervalType’’ type=‘‘xs:nonNegativeInteger’’>
        <xs:attribute name=‘‘unit’’ type=‘‘xs:string’’ use=‘‘optional’’>
</xs:complexType>
```

3.2.1 Min-Interval and Max-Interval Trigger Conditions

We define the *min-interval* and *max-interval* trigger types for limiting the number of notifications and ensuring notifications, respectively. The former means the minimum passive time that has to elapse between two consecutive notifications of the associated presence information. The latter means the maximum time that can elapse between two consecutive notifications of the associated presence information. These two concepts were also introduced by the authors of [183]. However, they apply a single minimum and maximum interval to the whole presence information, which we find less helpful and efficient. On the contrary, we allow the subscriber to play with multiple maximum and minimum intervals for different pieces of presence information. The proposed trigger types naturally implement multi-rate control because trigger conditions can be bound to particular pieces of a resource's state information.

If a notifier receives a notification filter that contains a max-interval trigger, it sets a timer to the specified time interval and restarts it whenever it expires. When a timeout occurs, if no notification of the state information described by the filter's *what* element was sent since the last timeout, a notification is triggered. If the notifier receives a notification filter that contains a *min-interval* condition, no notifications of the state information described by the filter's *what* element are allowed during the specified time interval. Therefore, whenever a notification of this information is sent, the notifier sets a timer to the specified time interval. Any presence change that occurs before the timer expires is aggregated. Once the timer expires the aggregated presence changes are notified all at once. If the *min-interval* trigger condition includes an attribute "mandatory", it means that the interval is mandatory. As described in Section 3.1, a minimum interval is mandatory when the associated presence information can not be attached to a notification of other presence information that is sent before this interval expires.

If a subscriber wishes to include a *max-interval* or *min-interval* trigger condition for controlling a set P of the subscription's state information, it should perform as follows. If P contains the complete resource state information and a filter without a

what element already exists, the trigger is added to this filter. Otherwise, a new filter with no *what* element but does include the trigger should be created. If P is a subset of the resource state information and a filter contains a *what* element that only includes P, the trigger is added to this filter. Otherwise, a new filter with the trigger and a *what* element that only contains P is created.

3.2.2 Never and Once Trigger Conditions

We define the *never* trigger condition for disabling notifications of some presence information. A notifier must never notify the information described by the *what* element of any filter that has a *never* condition. If a filter containing a *never* condition does not include a *what* element, notifications of changes in the entire presence information are paused. If the watcher wishes to un-pause notifications, it only needs to remove the filter. As [140] describes, the watcher should remove the filter by setting its remove attribute to true and including the updated filter with a new identifier.

When notifications of some presence information are disabled, the watcher may need sporadic information updates without enabling notifications back. Although a re-subscription request can be used to make the notifier send a notification, such a notification includes the complete presence information as stated in [110]. Conditional notifications [147] were defined for suppressing the full-state notifications due to subscription refreshes when the presence information has not changed from the last notification. This strategy defines a Suppress-If-Match header to be added to re-subscription requests, which corresponds to the full-state resource information. When a notifier sees this header in a re-subscription request, it checks its value against the local tag of the resource information. In case of matching, no change occurred from the last notification, and hence no notification is sent to the subscriber. Otherwise, a full-state notification is sent to the subscriber. We define the *once* trigger condition to be combined with conditional notifications for forcing the notifier to only notify some piece of presence information. This allows disabling full-state notification completely and pulling the presence attributes of interest when necessary. The presence of a notification filter containing a *once* trigger modifies the notifier behavior defined in [147]. When a notifier sees a Suppress-If-Match header in a re-subscription request that contains a filter with a *once* trigger element, it ignores this header and immediately notifies the information contained in the filter's *what* element. If the *what* element is not present, all the

resource information is notified. This notification only contains this information and is sent regardless of whether this information has changed from the last time it was notified. After notifying, any filter with a *once* condition is eliminated; these filters are not stored since they only have effect once.

A filter that contains either a *once* or *never* trigger should not include other types of trigger. However, a filter with a *once* trigger can contain the same *what* element as other filter with a *never* trigger. This actually enables a pull model for presence notifications. Moreover, *once* conditions can also trigger one-time notifications of information that has not been paused previously. This allows pulling certain presence attributes, probably due to information lost or unexpected requirements, while keeping notifications for refreshes disabled.

3.2.3 General Rules and Example Document

A notifier and subscriber handling the proposed trigger elements in notification filters should follow the general instructions in [140] except for the following considerations. The *what* elements of all the notification filters associated with a resource determine the view of the resource's presence information that is notified to the subscriber. Thus, full-state notifications contain the information determined by the *what* elements that are not included in any filter with a *never* trigger. A filter containing either a *never* or *once* trigger should not include any other kind of trigger. If a subscription request includes a filter with a *once* trigger, the notifier replies with the information determined by the filter's *what* element. Note that this kind of trigger only should be present when conditional notifications are applied, as described in Section 3.2.2. When a conditional subscription request does not include a filter with a *once* trigger, the notifier follows the procedures in [147] for determining whether to send a notification or not. A notification filter can combine control rate conditions, namely *min-interval* and *max-interval* triggers, with the *changed*, *added* and *removed* conditions specified in [141]. *Changed*, *added* and *removed* trigger conditions determine the presence changes of interest when they are combined with a *min-interval* trigger. This trigger determine the rate at which the changes of interest are notified. *Max-interval* triggers do not modify the effect of *changed*, *added* and *removed* triggers, since notifications of changes, additions and removes are notified instantaneously.

An example document is shown below. The watcher is interested in the presentity's entire presence information. However, due to traffic optimization, updates in location information are required only every 30 minutes. The user-input element is defined in [23] as the usage state of the service and device based on human user input (e.g., keyboard, pointing device or voice). This element can assume one of two values, namely active and idle. It is expected that this element can change very frequently, especially for mobile devices. In our use case, the watcher is not very interested in this information, and hence notifications of the user-input element are paused until traffic optimization is no longer necessary.

```
<?xml version="1.0" encoding="UTF-8"?>
   <filter-set xmlns="urn:ietf:params:xml:ns:simple-filter">
     <ns-bindings>
       <ns-binding prefix="pidf" urn="urn:ietf:params:xml:ns:pidf"/>
       <ns-binding prefix="rpid" urn="urn:ietf:params:xml:ns:pidf:rpid"/>
       <ns-binding prefix="gp" urn="urn:ietf:params:xml:ns:pidf:geopriv10"/>
       <ns-binding prefix="dm" urn="urn:ietf:params:xml:ns:pidf:data-model"/>
       <ns-binding prefix="e1" urn="urn:ietf:params:xml:ns:pidf:rpid:usecase1"/>
     </ns-bindings>

     <filter id="8439" uri="sip:bob@example.com">
       <what>
         <include>
           /pidf:presence/dm:device/gp:geopriv
         </include>
       </what>
       <trigger>
         <min-interval>30</min-interval>
       </trigger>
     </filter>
     <filter id="8439" uri="sip:bob@example.com">
       <what>
         <include>
           /pidf:presence
         </include>
       </what>
     </filter>
     <filter id="5681" uri="sip:bob@example.com">
       <what>
         <include>
           /pidf:presence/dm:person/rpid:user-input
         </include>
       </what>
       <trigger>
         </never>
       </trigger>
    </filter>
```

```
</filter-set>
```

3.3 Conclusions

We proposed an extension of the XML scheme for SIMPLE notification filters, which consists in new triggers conditions for i) controlling the notification rate of presence attributes, 2) pausing and un-pausing notifications of presence attributes and 3) triggering notifications of presence attributes. This extension is aimed at facilitating the differentiated treatment of presence attributes in presence notifications. Such a differentiated treatment is necessary when controlling the rate of notifications is applied for traffic optimization and the watchers have different consistency requirements on presence attributes. Notification rate control is already being considered for SIMPLE. However, it is applied to all the state information associated with a resource, and hence this information (e.g, a resource list) is notified at the same rate. To overcome this shortcoming, we include control rate conditions in notification filters, which allows fine-grained multi-rate control of notifications. We allow a minimum and a maximum interval between two consecutive notifications of a subset of presence information. No notifications can be triggered before the minimum interval expires. As regards the maximum interval, if no notification was sent during this interval, a notification is triggered regardless whether or not the information changed. We also define trigger conditions for pausing, un-pausing and triggering notifications, which allows pulling subsets of presence information while keeping notifications paused. These new trigger conditions are enablers of pull approaches to obtain presence information, which may be useful to optimize presence traffic in some circumstances. When applying throttling for traffic optimization, we should avoid watchers from perceiving delays. When a watcher perceives a delay, it means that it access obsolete values of presence attributes because they are not updated frequently enough. This can lead the watcher to undesirable behaviors and, therefore, a tradeoff between traffic optimization and the watcher's needs should be found. Our strategy helps in finding such a tradeoff since multiple minimum and maximum intervals can be established for a resource.

The proposed notification filters are specially interesting in location-based systems. In these systems, location updates occur very frequently and are timely spread over

the network. Our approach permits to control the rate of location updates and to pause, un-pause and pull them when necessary Regardless of the remaining presence information. Nevertheless, we are conscious that the proposed attribute-based notification rate control presents scalability issues and adds some complexity to the PS. The PS needs to handle multiple maximum notification rates for each watcher subscribed to a presentity. As the number of watchers and presentities increases, this approach becomes unfeasible. Moreover, as stated in [140], processing of notification filters may require a considerable amount of computation. Thus, we envision notification filters to be used in small-scale scenarios such as corporations. On the other hand, the proposed filters may be applied to presentities in a distributed manner. Each presentity may handle the rates associated to its presence information independently, thereby solving the scalability issue. Section 4 discusses this strategy and its efficiency at reducing presence traffic on the access network.

4

Optimization of Presence Publication Traffic: Proposal, Mathematical Model and Performance Estimation

Presence publication triggers end-to-end notification flows to notify each of the presentity's watchers. Let's repeat here as Figure 4.1 the illustration of notification flows for a publication given as Figure 2.9 for convenience.

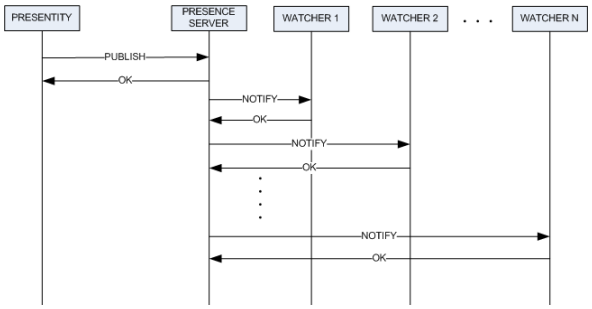


Figure 4.1: Publication and resulting notifications flows

A publication involves messages being exchanged by the presentity, its PS, the watchers' RLSs and finally the watchers themselves. Although Figure 4.1 does not show the watchers' RLSs for the sake of simplicity, the watchers normally do not subscribe to their presentities directly. Instead, they subscribe to their RLSs, which in turn

subscribe to each of the watchers' presenties, as described in Section 2.7. Thus, a presentity's presence publication results in (2+4*N) messages, assumed that N watchers are subscribed to the presentity. As the number of presentities, watchers and presence publications grows up, the impact of presence publications on network latency and servers can be severe. Taking into account that large-scale federated presence services are envisioned in NGNs (see Sections 2.4 and 2.9.5), easing the impact of presence publications on the presence service is necessary. In radio access networks, the effect of frequent presence publications may be especially dramatic on low-bandwidth links and wireless devices with limited battery life and processing capabilities. Furthermore, presence information has become a general concept that embraces all kind of context about not only users but also places, objects and any relevant entity to communication applications. In the near future, presence information will be enriched with diverse data generated by calendar applications, social networks, sensor networks, social television, pervasive gaming, etc. Such rich presence information is encoded by XML, which is very verbose and therefore constitutes a heavy burden for devices and links with scarce resources.

Section 2.9.6 introduces some SIMPLE optimizations that may be used for reducing presence traffic on the network access link. Most of these optimizations reduce subscription-related traffic rather than the number and size of presence publications. Presence subscriptions pave the way for traffic optimization because they involve periodic refresh messages, which are used by subscribers and notifiers to interoperate. On the contrary, publications are totally asynchronous and not bound to any state. This makes the optimization of presence publications especially difficult, and hence there are very few optimizations of presence publications. Partial presence information [137] and presence compression [142] reduce the size of presence documents rather than the number of publications. Thus, the harmful notification flows depicted in Figure 4.1 are still triggered. These flows may involve harmful consequences: PS overload, large amounts of notifications on the network core, and overconsumption of radio access bandwidth and the battery life of wireless devices. Some optimizations of subscription traffic, such as event filtering [140] and notification rate control [183] (see Section 2.7.4) help in impeding the notification flows due to presence publications. A PS can filter the content and control the rate of the presence notifications generated as a result of a publication

in order to save traffic on the network core. Equally, an RLS can perform so for reducing the number and size of RLMI notifications to resource list subscribers on the access network link. Although these strategies decrease the presence traffic for notifications, presence publications are not yet optimized, and hence processing resources at PSs and at user devices, in addition to access link bandwidth, may still be over-consumed.

Limiting the number of presence publications is specially crucial in mobile presence-enabled LBSs since the presentities' location information is likely to change very frequently. To reduce the number of presence publications on the network access link, OMA has defined Presence Network Agents (PNAs) [196], which publish presence information that can be deduced from the network on behalf of presentities. Unfortunately, there still is much information that cannot be extracted from network services such as person-related information. For instance, the presentity's GPS coordinates have to be published by the end device's UA, which is probable to involve many publications over the air interface. In the research community, almost no authors have tackled the presence traffic that is due to publications. To the best of our knowledge, only the work [65] addresses the need to limit the rate of presence publications in the IMS. The authors propose a hierarchical structure of PSs, called Context Mediators, which are specialized in managing particular kinds of presence. A Context Mediator only communicates with the presence sources that publish the kind of information that it manages. These servers require advanced intelligence in order to perform presence aggregation and balance the source publications with the needs of watchers. However, the impact of multiple PSs on the IMS CSCF capacity may be severe since all the messages sent and received by each PS need to be processed by the S-CSCF. Unfortunately, the authors do not address the viability and impact of their proposal. Moreover, the authors proposed an IETF Internet-Draft [198] that defines a new event package for allowing presentities to be up to date with their PSs' presence requirements. Thus, presentities only publish the presence attributes needed by their PSs rather than their complete presence information. Although this event package was specified to reduce the number and size of presence publications, a new kind of subscription between the presentity and its PS does not seem to be a good approach. Probably, the maintenance of this subscription generates more traffic than presence publications and it was the reason why this draft expired without further progress.

On the other hand, a pull model for presence publications may be helpful in reducing presence traffic on the access network link under some circumstances, as described in Section 2.9.8. Instantaneous presence publications are not efficient and useful if the watchers only need to know the presentity's presence information occasionally. In this case, it would be more efficient to have the presentity not publishing presence changes and the PS pulling presence changes periodically. As the needs of the PS and watchers may change over time, it would be desirable to provide the PS with a mechanism for pausing and un-pausing publications automatically.

As described in Section 2.9.7, the importance that watchers give to presence attributes should be considered when optimizing presence traffic. A presentity's presence information may be composed by a wide range of presence attributes of different nature. Some presence attributes, such as activity or location information, change more rapidly than others, such as contact information or service capabilities. Moreover, watcher applications generally are not interested in the full set of presence attributes. Instead, they have different rate and accuracy restrictions on presence attributes. Due to the fact that presence attributes change their values at different rates, the authors of [66] define decay functions of subsets of presence information that allow the PS to be aware of when some presence attributes are very probable to have changed their values. When a decay function reaches a threshold value, the PS should somehow retrieve the latest value of the presence information associated with the function. The PS may be missing this latest value due to network losses or some publication rate control at the presentity. Thus, the PS should be capable to pull this value if necessary. Although this work does not tackle traffic optimization, they address the diverse nature of presence information and the need to maintain information consistency between presentities and watchers. The above-mentioned Internet-Draft [198] propose a new kind of subscription for presentities to subscribe to their PSs' QoS requirements on presence information (e.g., urgency, rate ad cost). Thus, presentities are able to publish presence attributes according to these requirements.

In order to optimize presence publications and tackle the afore-mentioned issues, we propose applying the notification filters described in Section 3 to presentities. These filters, henceforth called publication filters, can determine the presence changes that trigger publications, the content of such publications and when to send them. The proposed publication filtering enables i) fine-grained control of publication rate and

ii) pausing and un-pausing publications of presence attributes. We propose attribute-based publication rates to limit the number of publications while satisfying the watchers' preferences on presence update rate. We also allow forcing presentities to publish some presence attributes at minimum rates, which ensures that these attributes do not become obsolete at watcher applications. We analytically estimate the traffic rate generated by the proposed publication rate control mechanism. To this end, we model the behavior of some presence applications' users through a continuous-time Markov process. Moreover, we propose a strategy for dynamically adapting a presentity's maximum publication rate to the frequency at which its presence information changes, which we refer to as sojourn-based rate. This strategy avoids setting too low publication rates, which may lead to information inconsistency issues at the watchers.

Section 4.1 describes our proposal for implementing publication filtering. Section 4.2 discusses a mathematical model for estimating presence traffic rate when throttling publications, and studies the performance of this strategy for a particular use case. Section 4.3 presents sojourn-based rates and studies their performance. Finally, Section 4.4 gives some conclusions.

4.1 Publication Filters for Presence Sources

We propose setting presentities to apply publication filters that are defined by the XML schema document described in Section 3.2. This XML schema is an extension of that for SIMPLE event notification filters [141]. This allows the PS to perform dynamic fine-grained control of its presentities' presence publications by means of:

- Minimum and maximum publication rates of presence attributes.
- Pausing and un-pausing publications of presence attributes.
- Pulling presence attributes when publications are paused
- Triggering instantaneous publications of presence attributes
- Filtering the content of presence publications

Section 4.1.1 describes the use of minimum and maximum rates for throttling publications. Section 4.1.2 explains a pull model for presence publications. Section 4.1.3 discusses how to implement publication filtering.

4.1.1 Multi-rate Control of Publications

Limiting the number of presence publications is advisable for reducing presence traffic in scenarios where presence changes occur frequently. The most obvious scenario is a presentity publishing his or her geographical coordinates whenever they change. Setting a passive interval between two publications, which we refer to as throttling, avoids floods of presence publications. Presentities may be configured with such a passive time statically or via OTA provisioning through the SOURCE-THROTTLE-PUBLISH parameter defined by OMA. However, the maximum delays in presence updates that are acceptable by watchers besides traffic optimization should be taken into account. Thus, a presentity's passive interval should be set to a value that ensures that publications of the attributes that are most urgently needed by watchers are published frequently enough. This involves publishing changes in a presentity's entire presence information at the highest rate required by watchers. This single interval is especially inadequate in LBSs in which presentities' geographical coordinates are crucial and therefore require high rates. In presence systems that do not need location updates frequently, this throttling technique may even be more inefficient because of a high minimum rate due to some other important information. In this case, changes in geographical coordinates would trigger presence publications at such high rate although it is unnecessary. On the other hand, some location systems may require periodic location updates, even if the location information has not changed or contains uncertainty, for working appropriately. If we set a maximum interval that cannot elapse without publications of location, whenever no location publications occurred during this interval, a publication should be sent. If this publication is full-state, much unnecessary information will be published at the same minimum rate.

Apart from traffic optimization, throttling could be a means of ensuring privacy policies about device-specific presence information such as location information. Let us consider a corporation that provides its employees with a presence application that runs on laptops and smart phones. This presence application publishes the employees' presence information including geographical coordinates during working hours. This corporation has different privacy policies on the employees' location based on their position. Let us assume that any update in the location of the directors' laptops is required instantaneously by the system. However, the location of the directors' phones

is more protected and should be spread every 1 hour as minimum. Presence multi-rate control constitutes a safe and neat solution in this scenario. The corporation's PS may set the proper minimum publication rates of location for each employee's device (e.g., one hour for director phones and none for director laptops). In this way, user devices only disclose their location to their PSs according to the set intervals and PSs are free of handling low-level authorization rules every time a location update is received. The minimum interval for directors should be mandatory. The information associated to a mandatory interval can not be attached to any publication of other information that is sent before this interval expires, as described in Section 3.1.

We can tackle the afore-mentioned issues by using the XML schema described in Section 3 for fine-grained multi-rate control. The *min-interval* and *max-interval* trigger conditions allows associating the presence information included by a filter with a minimum and maximum interval between two consecutive publications, respectively. We refer the reader to section 3.2.1 for more information about these trigger conditions.

4.1.2 Pull Model for Publications

Currently, PSs are not capable of pausing presence publications when it is needed to reduce presence traffic. However, pausing and un-pausing publications may be very convenient to reduce presence traffic on wireless links. A presentity's presence publications could be paused when its presence information is not useful for other entities as, for example, when all its watchers are offline. Temporally pausing publications in location information could reduce much presence traffic when frequent updates are not needed. For instance, a user in a meeting is not supposed to change location before the meeting finishes. Then, a PS may be aware of the user's schedule through his calendar and pause location updates until his meeting finishes if traffic optimization was necessary. If the meeting's end time is unknown, the PS may throttle location updates at low rates until a significant location change occurs. Moreover, there is no means of requesting presentities to publish their presence information (or a subset of it) immediately. This is because presence systems always adopt a push model in which presentities proactively inform their PSs of any change in their presence information. However, a PS may need to know a presentity's presence information immediately due to some information loss or some watcher in urgent need of certain presence attributes. Pulling presence information may be conveniently combined with publication pauses.

For example, a PS may prefer to receive any change in person-related information but to pause publications about device information that usually does not change. The PS may pull device information occasionally for verifying that this information has not changed.

We can pause publications by using the filters described in Section 3 to presentities. These filters can include *never* conditions for pausing, and *once* conditions for triggering publications of presence attributes. We refer the reader to Section 3.2.2 for more information about these trigger conditions.

4.1.3 Implementation

We propose the PUBLISH method to be used by the PS for informing presentities about publication filters. Thus, the PS is considered as a presentity whose presence information consists in publication filters. Although this approach does not correspond to the PS role (i.e., notifier), we think it is the SIMPLE-based most efficient and simplest means of configuring presentities. Figure 2 outlines the big picture of this strategy. The presentity's watchers send their notification filters to their RLSs, as described in [140]. The RLSs attach these notification filters to the SUBSCRIBE requests sent to the presentity. An RLS may modify the watcher's notification filter or even create a new one based on some administrative policy, such as traffic optimization. However, the information sent to the watcher should always be according to his or her notification filter. The presentity's PS should find a tradeoff between the requirements of all of the presentity's watchers, create publication filters accordingly and publish them to the presentity. Finding this tradeoff may be an arduous task, if not impossible. However, the presence of watchers themselves is a sign of interest in being notified and can significantly help in choosing suitable publication filters. States such as online, busy and idle are indicators of the watcher's interest in communicating and therefore in getting presence information about their presentities. For example, if all of the presentity's watchers are offline, generally there is no need to publish or, at least, publications should be sent much less frequently. In this case, the PS may send the presentity a publication filter to pause publications of any presence attribute or set a long minimum interval between two consecutive publications. In general, the greater the number of online watchers, the higher the probability of the presentity's presence information being useful. Even conversation history may be useful to deduce the watchers' interest

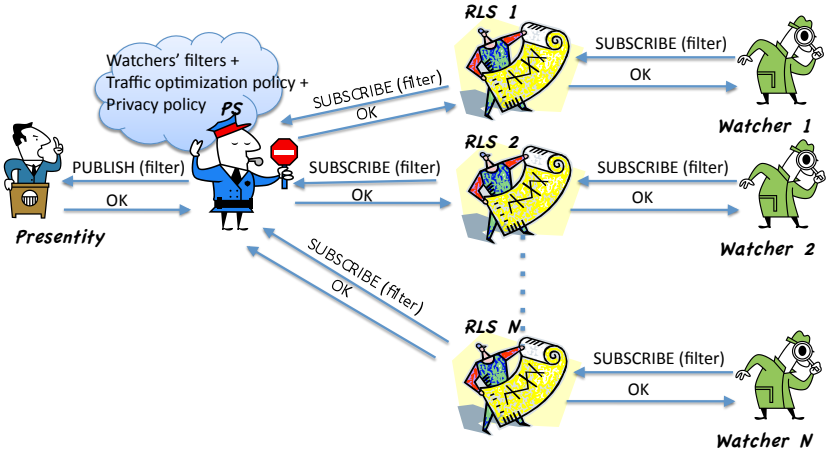


Figure 4.2: Publication filtering

in a presentity's presence. The more frequently the watcher communicates with the presentity, the greater the probability of this watcher being interested in the presentity's presence information. Therefore, the PS may prioritize the watchers' notification filters based on the probability of each one being actually watching the presentity. PSs could choose more restrictive publication filters than those of watchers when optimizing presence publications is crucial. For instance, if some presence attributes change too frequently, a PS may decide to request them from the presentity at lower rates. The presence of a presentity is a significant parameter to consider for reducing the number of location updates. During working hours, for example, the user usually stays indoors (i.e., in his or her office), and the PS could therefore request location updates at a very low rate because location increments will be considerably small. When the user finishes working, the PS could request presence updates at a higher rate.

Publication filtering follows the same funtional rules as those for notification filters [140] excepting that filters are borne by PUBLISH rather than SUBSCRIBE requests. Whenever a PS publication filter for a presentity changes, it sends a PUBLISH message that contains the updated filter to the presentity. To this end, the PS should remove the filter that changed by setting its remove attribute to true and including the updated filter with a new identifier, as described in [140]. Moreover, The PS and the presentity should follow the general rules for handling the proposed extensions described in Section 3.2.3.

4.2 Mathematical Analysis of Publication Rate Control

This section mathematically estimates the performance of throttling publications through a single and multiple maximum rates. Section 4.2.1 discusses about modeling presence changes through state diagrams. Section 4.2.2 proposes using such state diagrams for modeling presence changes probabilistically by means of Markov chains. This section also summarizes how to calculate the time-dependent state probability distribution of Markov processes. Sections 4.2.3 and 4.2.4 mathematically analyze the byte rate with single- and multi-throttling through the Markovian properties. Based on this analysis, Section 4.2.5 analytically estimates the traffic rate after throttling intervals for a particular use case.

4.2.1 Modeling Presence Changes as State Diagrams

While a user is connected to a presence application, this user takes the role of presentity and his presence information goes through a series of states. Each state reflects a different status of the presence information, and the sequence of states over time is determined by the changes that the user (or even other authorized agents) makes in his presence. This type of system can easily be represented by state diagrams: each possible presence status is a state in the diagram. The states are connected by direct arcs, which represent the actions that alter the presence. If the presentity's presence information is made up of A_N attributes, where each attribute a_k can take r_k possible values, the representative state diagram will have $\prod_{k=1}^{A_N} r_k$ states. The number of states will be smaller if there are incompatibilities between presence attributes because two or more values cannot coexist. If we assume that presence changes occur one after the other, the maximum number of arcs for each state in the diagram would be $\sum_{j=1}^{A_N}(r_j-1)$. In contrast, if we assume that a presentity is able to change several attributes at the same time, each state could be connected to all the others, and hence the maximum number of arcs in the diagram would be $N\prod_{k=1}^{A_N}(r_k-1)$. The presence diagram for a presentity, therefore, contains its maximum number of arcs when there is no restriction on the possible presence changes.

An illustration is given in Figure 4.3. This example presence information consists of three presence attributes: audio (A), moving (M) and sphere (SPH). The first attribute

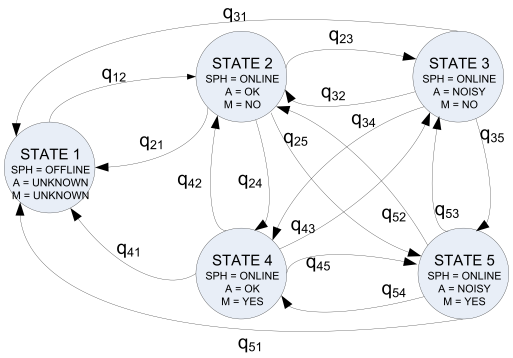


Figure 4.3: A state diagram for changes in three particular presence attributes

indicates whether the level of noise at the presentity's location is acceptable or inappropriate for incoming voice calls through the OK and noisy value, respectively. The moving attribute indicates whether the user is moving so fast that he or she is likely to be using a mode of transport, such as a motorcycle or a car. The sphere attribute refers to the role of the presentity and only has two possible values: online (i.e., the presentty is logged onto the application) and offline (i.e., the presentity is logged off the application). As shown in Figure 4.3, the presentity's presence information goes through a series of states. Each presence state is a particular combination of attribute values and represents the status of the user's presence information at a given moment. The arrows indicate transitions between states, which are due to changes in presence attributes. For example, in Figure 3, the presence information goes from state 2 to 3 when the audio attribute changes from OK to noisy. Compared to other models of presence changes [191][182], the transition graph in Figure 4.3 is more adaptable to other scenarios since all the states except state 1 are connected. Thus, one could create other presence change patterns by setting some transition rates to zero.

4.2.2 Presence Information Probability Distribution

The graphical representation of the dynamics of presence information discussed in Section 4.2.1 closely matches the transition diagrams of Markov chains. Changes in presence information can be analyzed through a Markovian model only if the times at which presence changes occur follow the exponential distribution, and hence the system is memoryless [199]. This means that the probability of a state in the chain moving

to another only depends on the present state and is independent of the chain's past history. The past history is completely summarized by the present state, which is known as the Markov property [199]. To satisfy this property, we analyze presence changes over short periods of time, such as working, having lunch and watching a movie, in which changes occur according to particular patterns. In such short periods of time, it is reasonable to assume that the time until the next presence change is exponentially distributed. Thus, the current status of the user's presence information summarizes all the previous changes in this information. For example, call duration that determines busy and idle states is an illustrative example that satisfies this assumption. We are however conscious that some presence systems may not satisfy this assumption. Let us consider a user that is driving at a constant speed on a non-congested road and publishes his location at a constant rate. This is a periodic distribution and therefore cannot be modeled as a Markov chain. Nevertheless, exponential distributions are more pessimistic than periodic distributions, and hence assuming the former implies working on the worst case.

We perform a non-steady analysis about the probability distribution of a Markov chain that models changes in some presence information. Compared to other steady-state analysis [182][191], this approach is closer to the nature of presence applications, since presence changes are dependent on time and can occur at any instant of time. Let n denote the number of states in the chain, each representing a particular status of the presence information. Transition rates q_{ij} represent the rates at which the chain moves from one state to another, that is, the rates at which presence changes occur. Let Q be the transition rate matrix or infinitesimal generator of the chain, as shown below:

$$Q = \begin{pmatrix} \sum_{j \neq 1} -q_{1j} & q_{12} & \cdots & q_{1n} \\ q_{12} & \sum_{j \neq 2} -q_{2j} & \cdots & q_{2n} \\ \vdots & \vdots & \ddots & \cdots \\ q_{1n} & q_{2n} & \cdots & \sum_{j \neq n} -q_{nj} \end{pmatrix} \tag{4.1}$$

Let $X(t)$ denote the state in which the Markov chain is found at time epoch t. Setting

$$p_k^x(t) = P\{X(t) = k\}, 1 \leq k \leq n \tag{4.2}$$

From the Markov theory [199], the chain's state probability distribution matrix $P^x(t)$ can be expressed as (4.3). The general solution of this equation can be written as (4.4), where λ_i is an eigenvalue of the matrix Q, Z_i is the eigenvector that corresponds to λ_i, and C is the 1-by-n constant coefficient matrix. The matrix C is calculated to satisfy $I_0 = C\Omega$, where Ω is the n-by-n matrix of the eigenvectors $Z_i, i = 1, .., n$, and I_0 is the scalar vector that summarizes the initial state of the Markov chain.

$$\frac{dP^x(t)}{dt} = P^x(t)Q \tag{4.3}$$

$$p_j^x(t) = \sum_{i=1}^{n} C[1,i]Z_i[j]e^{-\lambda_i t} \tag{4.4}$$

4.2.3 Mathematical Analysis of Byte Rate with Single-Throttling

This section analyzes the probability that a presence change in the presentity's complete presence information has occurred during a throttling interval, that is, the probability of publishing after this interval. When a throttling interval expires, a presence change has occurred if the Markov chain's state is different from that in which the chain was at the beginning of this interval. Let us define the change probability when a throttling interval φ has expired at time t_φ, $cp_{st}(t_\varphi)$ as shown below:

$$cp_{st}(t_\varphi) = \sum_{i=1}^{n} \sum_{j=1(j\neq i)}^{n} P\{X(t_\varphi) = j | X(t_\varphi - \varphi) = i\} \tag{4.5}$$

Stated differently, $cp_{st}(t_\varphi)$ is the probability of finding the chain in a given state at the beginning of the throttling interval and in a different state when this interval ends. Expression (4.5) disregards the possible intermediate transitions between the states at the beginning and at the end of the throttling interval. The intermediate states are summarized by the end state, and this is actually the advantage of throttling: intermediate state changes do not need to be published. In Expression (4.2), $p_i^x(t)$ is the probability of the Markov chain being in state i at time t, which is built on the scalar vector I_0 that summarizes the initial state of the Markov chain. The Markovian memoryless property states that the future probabilistic behavior of Markov chains only depends on the current state of the chain, regardless of how the chain reached that state. Therefore, in Expression (4.5), the condition $X(t_\varphi - \varphi) = i$ can be summarized by setting state i as the initial state for $X(\varphi) = j$. Let $X_i(t)$ denote the state in which

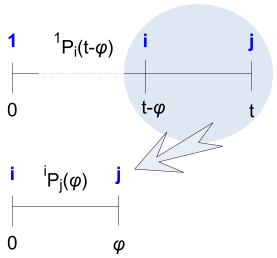


Figure 4.4: Markovian property outline

the Markov chain is found at time t, assuming that the chain started in state i. We can therefore rewrite Expression (4.5) as (4.6), assuming that the Markov chain always starts in state 1 at $t = 0$. This assumption is reasonable in presence applications since users initially have an offline status. Figure 4.4 outlines the idea behind the Expression (4.6).

$$\begin{aligned} cp_{st}(t_\varphi) &= \sum_{i=1}^{n} \sum_{j=1(j\neq i)}^{n} P\{X_i(\varphi) = j | X_1(t_\varphi - \varphi) = i\} \\ &= \sum_{i=1}^{n} \sum_{j=1(j\neq i)}^{n} P\{X_i(\varphi) = j\} P\{X_1(t_\varphi - \varphi) = i\} \end{aligned} \tag{4.6}$$

The variable $X_k(t)$ only relies on the fact that state k was reached because this state summarizes all the past history. Thus, the probability distributions of $X_i(t)$ and $X_j(t)$ in Expression (4.6) are independent and can therefore be multiplied. Setting

$$^{i}p_k(t) = P\{X_i(t) = k\}, 1 \leq i \leq n, 1 \leq k \leq n \tag{4.7}$$

where $^{i}p_k(t)$ can be deduced by setting the initial vector I_0 to state i, as described in Section 4.2.2. Using (4.7), we can rewrite (4.6) as follows:

$$cp_{st}(t_\varphi) = \sum_{i=1}^{n} {}^{1}p_i(t_\varphi - \varphi) \sum_{j=1(j\neq i)}^{n} {}^{i}p_j(\varphi) \tag{4.8}$$

From Expression (4.8), we can estimate the byte rate during a throttle interval as shown in (4.9), assuming that $B_{i,j}$ is the average size of the presence document that needs to be sent when the presence information changes from state i to j.

$$br_{st}(t_\varphi) = \frac{\sum_{i=1}^{n} {}^{1}p_i(t_\varphi - \varphi) \sum_{j=1, i\neq j}^{n} {}^{i}p_j(\varphi) B_{ij}}{\varphi} \tag{4.9}$$

4.2.4 Mathematical Analysis of Byte Rate with Multi-Throttling

Expression (4.8) gives the probability of presence changes occurring during a throttling interval when a single time interval is applied to the whole presence information. This expression therefore works on the assumption that any presence change triggers a presence publication when the throttling timer expires. However, when we apply multi-throttling, we associate pieces of presence information with different throttling intervals. Thus, when a throttling interval expires, only the changes in the piece of presence information associated with this interval trigger a presence publication. Thus, Expression (4.8) does not therefore work for calculating the change probability when multi-throttling is applied; below, we address how to calculate this probability.

As described in Section 3.1, when a publication is going to be sent, all the changes pending for publication are included into the publication commonly. This is a common practice for reducing the overhead of multiple messages, which we refer to as non-forced multi-throttling. However, when a minimum throttling interval for a piece of presence information is mandatory, the publication of this information can only be sent when this interval expires. We refer to this strategy as forced multi-throttling.

In the case of forced multi-throttling, the change probability of the presence attributes associated with a throttling interval can be deduced from Expression (4.8) by limiting states i and j to those that change these attributes. Let us assume the set TR_f to be composed of all the transitions q_{ij} in the Markov chain that change the value of the presence attributes associated with a forced throttling interval of φ time units. The change probability of these attributes when the throttling interval expires at time t_φ, $cp_{fmt}(t_\varphi)$, and the corresponding byte rate, $br_{fmt}(t_\varphi)$, is given by Expressions (4.10) and (4.11).

$$cp_{fmt}(t_\varphi) = \sum_{i=k:(k,x)\in TR_f}^{1} p_i(t_\varphi - \varphi) \sum_{j=l:(i,l)\in TR_f}^{i} p_j(\varphi) \tag{4.10}$$

$$br_{fmt}(t_\varphi) = \frac{\sum_{i=k:(k,x)\in TR_f}^{1} p_i(t_\varphi - \varphi) \sum_{j=l:(i,l)\in TR_f}^{i} p_j(\varphi) B_{ij}}{\varphi} \tag{4.11}$$

4. OPTIMIZATION OF PRESENCE PUBLICATION TRAFFIC: PROPOSAL, MATHEMATICAL MODEL AND PERFORMANCE ESTIMATION

In the case of non-forced multi-throttling, any change is aggregated when a publication is going to be sent, and hence further considerations are necessary. Below, we present an algorithm for calculating the change probability when a non-forced throttling interval φ expires at time epoch t_φ, $cp_{mt}(t_\varphi)$, and the corresponding byte rate, $br_{mt}(t_\varphi)$. A Markov chain's state represents a particular configuration of attribute values, and hence we denominate it as presence state. Whenever a new value of an attribute is published, the PS watches a new state of the Markov chain that models the presentity's presence information. Thus, a publication of some presence information means a publication of a presence state.

ATT = { set of attributes whose throttling interval expired at t_φ}
TR_{nf}={set of transitions q_{ij} that change the value of any of the attributes in ATT}
t_{last}= last time that a throttling timer expired or 0 if no timeout before t_φ
For each state k
    $PLP(k,t_\varphi) = PLP(k,t_{last}) *^k P_k(t_\varphi - t_{last})$ (a)
    $PPS(k,t_\varphi) = 0$
    For each state i connected to k through a transition q_{ik}
        If($q_{ik} \notin TR_{nf}$)
            $PLP(i,t_\varphi) = PLP(i,t_\varphi) + PLP(i,t_{last})^i p_k(t_\varphi - t_{last})$ (b)
        else
            $PPS(k,t_\varphi) = PPS(k,t_\varphi) + PLP(i,t_{last})^i p_k(t_\varphi - t_{last})$
            $Bytes(t_\varphi) = Bytes(t_\varphi) + PLP(i,t_{last})^i P_k(t_\varphi - t_{last}) B_{ik}$
            $PLP(k,t_\varphi) = PLP(k,t_\varphi) + PLP(i,t_{last})^i p_k(t_\varphi - t_{last})$ (c)
        endIf
    endFor
endFor
$cp_{mt}(t_\varphi) = \sum_e PPS(e,t_\varphi)$
$br_{mt}(t_\varphi) = \frac{Bytes(t_\varphi)}{\varphi}$

When one or more throttling intervals expire at a time epoch t_φ, the change probability or, in other words, the probability of publishing a presence change, is calculated and stored in the variable $cp_{mt}(t_\varphi)$. The variable $PPS(e,t)$ (Probability of Publishing State) provides the probability of publishing the presence state e at time t. The variable $PLP(e,t)$ (Probability of Last Published) provides the probability of the presence state e being the last one that was published until t, and hence the state that is being watched by the PS. The variable ATT contains the presence attributes associated with the throttling intervals that expired at t_φ. The variable TR_{nf} is the set of all the transitions in the Markov chain that change the value of any attribute in ATT. We store

the probability of publishing the end state of a transition in TR_{nf} in the PPS variable. This is the probability of having published the transition's origin state the last time a throttling interval expired (PLP variable) and having transited to the transition's end state. This transition probability is shown in (4.7). Thus, the change probability $cp_{mt}(t_\varphi)$ is the sum of the PPS probabilities for the end states of the transitions in TR_{nf}. The probability of each state e being the last published (PLP variable) when a throttling timeout occurs is the sum of the following three probabilities, whose letter identifies the corresponding line in the above algorithm:

(a) No transition probability. The probability that state e was the last published and afterwards the Markov chain has not transited.

(b) Non-notifying transition probability. The probability that state e was the last published and the Markov chain has transited to the other state e', which does not change the value of any attribute associated with the throttling timeout.

(c) Notifying transition probability. The probability that the Markov chain has transited to state e, which has changed the value of some attribute associated with the throttling timeout.

The variable $br_{mt}(t_\varphi)$ is the byte rate during the non-forced throttling interval φ that expires at time epoch t_φ. In other words, this is the rate byte during $t_\varphi - t_{last}$ time units. Other kind of measure is the total byte rate since the presence application starts at $t = 0$ until the throttling interval φ expires at time t_φ. Let us order all the interval timeouts that occur from $t = 0$ until t_φ increasingly: the first timeout is t_1, the second t_2 and so on, up to the last, t_φ, which is t_n. Thus, the total byte rate until t_φ is calculated by Expression (4.12), where the variable $Bytes(t_k)$ is computed as described in the algorithm above.

$$totalbr_{mt}(t_n) = \frac{\sum_{k=1}^{n} Bytes(t_k)}{t_n} \tag{4.12}$$

4.2.5 Byte Rate Estimation

This section estimates the byte rate generated by a presence application that throttles publications of the presence attributes in Figure 4.3. The presence application cannot therefore send two consecutive presence publications in an interval shorter than a

throttling interval. The presence application aggregates any presence change that occurs during this throttling interval. Once the interval expires, all the presence changes are published by means of a single PUBLISH message. To estimate the presence application's byte rate over time, we assume that the application sets a timer to the throttling interval. Whenever this timer expires, the application checks out whether any presence change has occurred since the last timeout. In this case, the application sends a PUBLISH message that contains the presence changes to the PS. This assumption allows estimating the byte rate generated by the presence application during some session time by using the mathematical formulas provided in Sections 4.2.3 and 4.2.4. We assume that the presence information is sent to the PS through partial-state documents [139]. This means that only the presence attributes that have changed since the last publication are sent. We assume partial-state presence documents of from 350 to 590 bytes depending on the presence attributes that are included. Below, an example of a partial-state presence document that publishes the *noisy* attribute is shown:

```
<?xml version="1.0" encoding=‘‘UTF-8’’?>
<p:pidf-diff
        xmlns=‘‘urn:ietf:params:xml:ns:pidf’’
        xmlns:p=‘‘urn:ietf:params:xml:ns:pidf-diff’’
        xmlns:r=‘‘urn:ietf:params:xml:ns:pidf:rpid’’
        xmlns:d=‘‘urn:ietf:params:xml:ns:pidf:data-model’’
        entity=‘‘pres:someone@example.com’’
        version=‘‘568’’>
        <p:replace sel=‘‘presence/person/place-is/audio/text()’’>
   <r:noisy/>
         </p:replace>
</p:pidf-diff>
```

We consider a session time of three hours, which is sufficient for the chain to reach its stationary state. To perform the mathematical calculations introduced in Sections 4.2.3 and 4.2.3, we need to assign a value to each transition rate $q_{i,j}$ of the Markov chain in Figure 4.3. Unfortunately, to date, there are not statistics about presence changes in real-world presence applications. Section 4.2.5.1 discusses an alternative method for estimating the transition rates for a particular use case. Sections 4.2.5.2 and 4.2.5.3 discuss the results obtained for this use case with single- and multi-throttling, respectively.

4.2.5.1 Use Case: a Technical Employee

The presence information described by the Markov chain in Figure 4.3 consists of three presence attributes, namely sphere, audio an moving. The rhythm of presence changes is determined by the transition rates q_{ij}, which mean the velocity at which the chain transits between states. Ideally, these transition rates should be based on testbeds with reliable patterns of state changes. The ideal sources of testbeds are logs of real-world presence applications. However, there are not real testbeds or statistics about the behavior of presence applications to date because of the unpredictable behaviour of their users and the diverse nature of presence applications. We adopt an alternative way of estimating the value of transition rates. The average time that a Markov chain spends in a particular state is inversely proportional to the sum of the state's outgoing transition rates. We therefore can assume a reasonable time for the chain to transit from one presence attribute's value to another, and deduce the corresponding transition rate as the inverse of this time. To this end, we consider a particular scenario which serves as reference for estimating the average time that each presence attribute takes to change value. We assume that the user (i.e., the presentity) is connected to her smartphone that has two sensors; one of them acts as a sound level meter and the other as an accelerometer. These two sensors pick up information about the ambient acoustic conditions and movements, such as inclination, vibration and shock that the user is experiencing. The presence application that is running on the user's device retrieves the information received by the sensors and modifies the presence attributes accordingly. We also assume that the application automatically changes the value of the sphere attribute to "online" and "offline" when the user logs on and logs off, respectively. We assume that the presentity is a technical employee responsible for maintaining some kinds of electrical machine (e.g. washing machines, refrigerators, etc.) supplied by her company. Therefore, this user visits various homes, offices, shops, etc., for offering technical support throughout the day. The employee's supervisors are subscribed to her presence information, and are therefore notified of the employee's sphere, audio and moving attributes. We assume that the employee's attributes take the average times in Table 4.1 to change value, which are, in our opinion, reasonable for the described use case. Table 4.1 shows these average times and the corresponding transition rates based on the states in Figure 4.3. For instance, when the employee is

Transition	Time (mins)	Rate
q_{12}	10	0.1
q_{21}	360	0.0028
q_{31}	360	0.0028
q_{41}	360	0.0028
q_{51}	360	0.0028
q_{23}	20	0.05
q_{24}	30	0.033
q_{25}	30	0.033
q_{32}	20	0.05
q_{34}	20	0.05
q_{35}	20	0.05
q_{42}	15	0.0667
q_{43}	20	0.05
q_{45}	5	0.2
q_{52}	10	0.1
q_{53}	15	0.0667
q_{54}	15	0.0667

Table 4.1: Assumed times and transition rates

on her way to a new client's place, it is probable that the ambient acoustic becomes noisy. Thus, we assume an average time of 5 minutes for the chain to move from state 4 to 5, and hence the corresponding transition rate is 0.2. Figure 4.5 shows the Markov chain with the assumed transition rates.

4.2.5.2 Results with Single-Throttling

Figure 4.6 shows the probability of presence changes occurring during each throttling interval when time intervals of 1, 5, 10, 15 and 30 minutes are used. The throttling interval of 6 seconds simulates the case of no throttling since this value is close to zero. These probabilities are given by (4.8) and, stated differently, they mean the probability of the presence application sending a PUBLISH message every time a throttling timer expires. Figure 4.7 shows the total number of bytes sent every time the throttling interval expires, which is given by the numerator in (4.9). We are mainly interested in the byte rate per minute generated by presence applications because it is a clear signal of the level of presence traffic on the network, which is given by (4.9). This formula depends on two parameters: the probability of publishing presence changes and the time

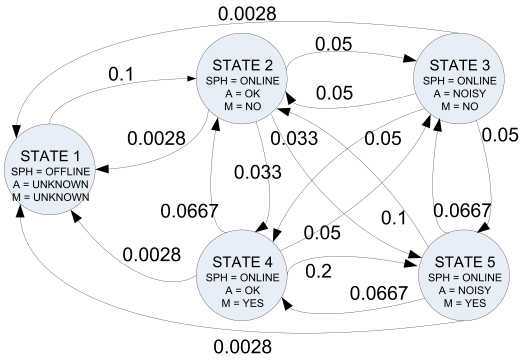


Figure 4.5: Markov chain for three particular presence attributes

interval during which publications are not allowed. Ideally, the publication probability should be decreased and the throttling time should be increased for reducing the byte rate. However, these two parameters are correlated; the longer the throttling interval, the higher the probability of changes occurring during this interval. Since the maximum value of the publication probability is the unit, the throttling interval is the most determining factor for reducing the byte rate. Figure 4.12 shows the estimated byte rate during each throttling interval. The byte rate for a throttling interval of 1 minute is very close to the case of no throttling (i.e., that of 6 seconds). Thus, the reduction in presence traffic achieved by using such a short interval does not compensate for implementing throttling. It is not therefore recommended to apply throttling with an interval equal to or shorter than 1 minute. The longer the throttling interval, the lower the byte rate injected into the network. The byte rate is reduced by approximately 41%, 61%, 71% and 85% when throttling intervals of 5, 10, 15 and 30 minutes are applied, respectively. However, the time during which the PS and the watchers keep inconsistent information increases with the throttling interval. A trade-off between traffic optimization and information consistency should be found. Let us assume the average delay in publishing a presence change to be half the throttling interval. In the case of a throttling time of 30 minutes, its average delay would be 15 minutes. It is probable that the PS or some watchers find this delay excessive. However, a throttling time of 15 minutes offers a shorter average delay of 7 minutes while its traffic reduction is similar to that achieved with 30 minutes. The watchers' and the PS' needs about the frequency of presence updates and traffic load may change over time. Thus, the PS

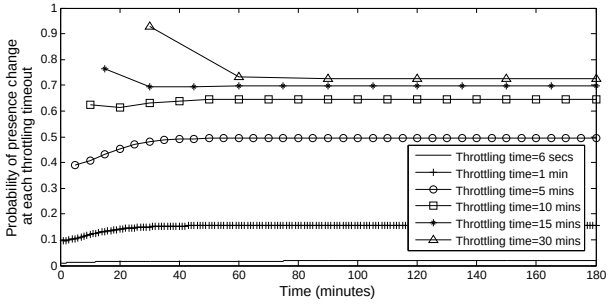


Figure 4.6: Probability of change at each throttling timeout

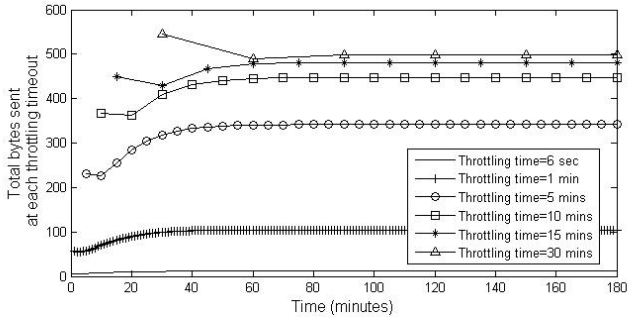


Figure 4.7: Bytes sent at each throttling timeout

and the client presence applications should be sufficiently intelligent to properly change the throttling interval.

4.2.5.3 Results with Multi-Throttling

Let us assume that each presence attribute of the user's presence information described in Section 4.2.5.1 (i.e., sphere, audio an moving attributes) is associated to a different throttling interval. The efficiency of multi-throttling depends on the interval length set for the presence attributes that change most frequently. In the best-case scenario, the watchers have no strict requirements on the consistency of these attributes. Thus, the presentity can delay in publishing these attributes through long throttling intervals. In the worst-case scenario, the watchers need to be updated with these attributes quite frequently, and hence only short throttling intervals should be used. The assignation of throttling intervals may not only be based on the watchers' requirements but also on the PS policies such as traffic congestion. Figure 4.9 shows the change probability

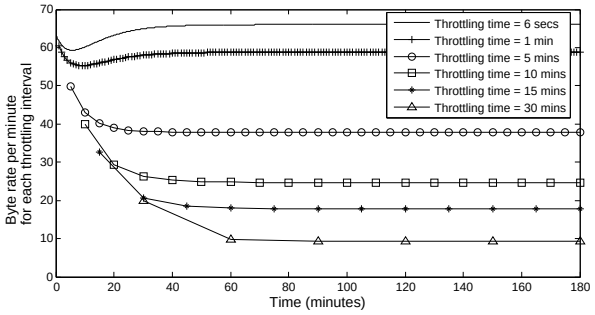


Figure 4.8: Rate of bytes during each throttling interval

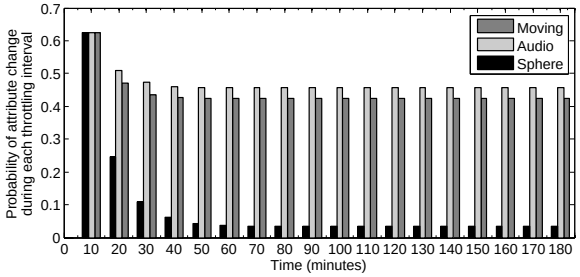


Figure 4.9: Probability of each attribute having changed after throttling timeouts

of each presence attribute every time a throttling interval of 10 minutes expires, which is calculated as described in Section 4.2.4. This helps us to ascertain the frequency of change of each presence attribute. The sphere attribute is much less dynamic than the others. The audio attribute is the one that changes most frequently although it is closely followed by the moving attribute.

To study the efficiency of multi-throttling, let us consider several scenarios of the use case described in Section 4.2.5.1 that associate the presence attributes with different levels of urgency:

1. The employee's supervisors are not interested in the employee's audio or moving conditions; they only wish to know whether she is online, and hence able to communicate. Thus, it is reasonable to assign a throttling interval of 10 minutes to the employee's sphere attribute, and one of 30 minutes to the audio and moving attributes.

2. The employee's supervisors need to know the most available employees at any time

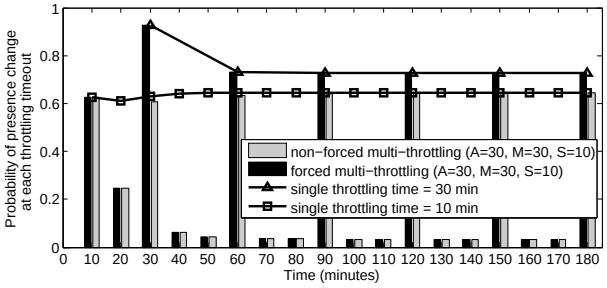


Figure 4.10: Probability of presence change after each throttling interval

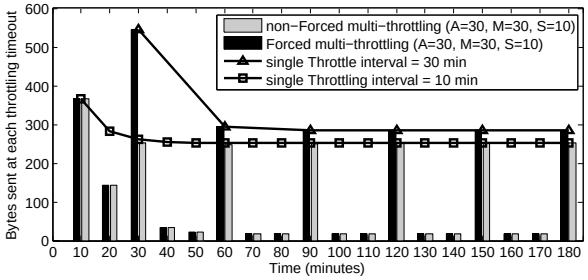


Figure 4.11: Bytes sent at each throttling interval

in order to assign tasks as quickly as possible. Thus, the employee's audio and moving conditions are important because they determine her capacity to maintain high-quality voice calls. It is therefore reasonable to avoid delaying the audio and moving attributes, and hence no throttling is applied to these attributes. On the contrary, it makes sense to apply a throttling interval of 10 minutes to the sphere attribute.

3. The employee's supervisors are mainly interested in knowing whether the employee is using some means of transport because this indicates that she is headed for the next customer. However, they do not pay regard for the employee's ambient noise. Thus, it is reasonable to assign a throttling interval of 5 minutes, 30 minutes and 10 minutes to the moving, audio and sphere attributes, respectively.

Since the audio and moving attributes are the ones that change most frequently, the first and second scenario are the best and worst cases for traffic optimization, respectively. In the second scenario, no throttling is applied to the audio and moving

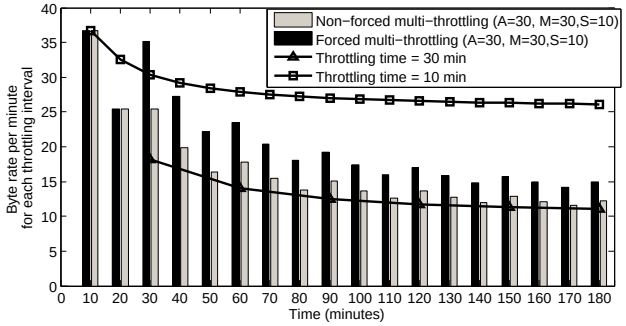


Figure 4.12: Rate of bytes during each throttling interval (A=30, M=30, S=10)

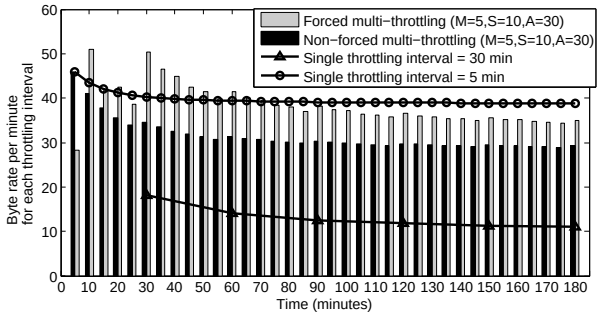


Figure 4.13: Rate of bytes during each throttling interval (A=30, M=5, S=10)

attributes. This means that any change in these attributes is published without delay. This involves publishing the sphere attribute without delay too. This is due to the fact that this attribute's state transitions change the value of the audio and moving attributes (see Figure 4.3). Thus, the byte rate for the second scenario is the same as that when no throttling is applied. Figure 4.8 simulates this case with an interval of 6 seconds. In the first scenario, multi-throttling seems to be an efficient alternative. Otherwise, the presence application would have to choose between traffic optimization and satisfying the watchers' urgency requirements. For the sake of traffic optimization, a long throttle interval would probably involve the watchers keeping obsolete information for too long. For the sake of information consistency, a short throttling interval would involve a high number of publications of the audio and moving attributes, which are not of interest to the supervisors and change frequently. Thus, multi-throttling may be used to achieve a tradeoff between traffic optimization and information consistency. Figure 4.10 shows the probability of sending a PUBLISH message every time a

throttling interval expires with non-forced and forced multi-throttling intervals. We are mainly interested in non-forced multi-throttling because it is applicable to the vast majority of presence applications. The triangle-marked and square-marked lines represent the publication probability when a single throttling interval of 30 and 10 minutes is used, respectively. The former would be reasonable for a single-throttling strategy that prioritize traffic optimization and the latter would be aimed at information consistency. It can be seen that the change probability in non-forced multi-throttling is much lower than the two single-throttling strategies at the expiration times that are not multiples of 30. This is because, with single-throttling, the change probability at each expiration time is mainly increased by changes in the audio and moving attributes. With multi-throttling, only the sphere attribute is checked every 10 minutes, and hence the change probability is lower. When the audio and moving attributes are checked, the change probability increases significantly. However, this probability with non-forced multi-throttling is always lower than that with single-throttling. This is because whenever the sphere attribute is published any change in the other two attributes is attached. In contrast, forced multi-throttling does not aggregate presence changes, and hence the change probability every 30 minutes is equal to that with a single throttling interval of 30 minutes.

Figure 4.11 shows the bytes sent every time a throttling interval expires. This figure shows the same pattern as that in Figure 4.10 because it is the result of multiplying the change probabilities by the number of bytes that would be sent to publish each change. It can be seen that the number of bytes at any expiration time is always lower than that of the two single-throttling strategies. Figure 4.12 shows the total byte rate per minute at each throttling timeout, which is calculated by (4.12). As time passes, it can be seen that the byte rate with non-forced multi-throttling is almost equal to that with a single throttling interval of 30 minutes, which is approximately 10 bytes per minute. Thus, a rate similar to the best case (i.e., the longest interval) is achieved while the required update frequency for the sphere attribute is maintained. Note that multi-throttling decreases the byte rate when a single throttling interval of 10 minutes is used by around 54%. Figure 4.13 shows the byte rate per minute at every throttling timeout for the third scenario described above. It can be seen that the required update frequency for the moving attribute is achieved. Moreover, this saves approximately 27% of the traffic that is generated when a single interval of 10 minutes is used.

4.3 Sojourn-Based Publication Rates

Ensuring that a minimum time elapses between two consecutive publications can certainly reduce presence traffic on the radio access link, as showed in Section 2.4. However, choosing this minimum time, which we refer to as throttling interval, is crucial. As the throttling interval increases, more presence changes may be aggregated during this interval, and so more presence load would be saved. On the other hand, the longer the interval, the lower the publication rate. Although long throttling intervals are preferable for traffic optimization, presence information that changes frequently should not be published at such low rates. Otherwise, watchers would keep obsolete presence information for too long and publications would even turn out inefficient. If the presence information changes much more rapidly than the publication rate, the short time during which watchers see valid information does not compensate for the traffic generated to publish such information. Moreover, the nature of presence information is very diverse; some presence attributes such as geographical coordinates change frequently, while others such as profile information rarely change. Furthermore, some presence attributes may be more relevant than others to the watchers. Thus, the watchers' needs on information consistency may vary from one presence attribute to another. For instance, a watcher application may need timely notifications of the presentity's luminosity and noise conditions for call handling, while it may occasionally prefer a notification of the presentity's activities. Thus, the application of a single throttling interval to the presentity's complete presence information may not be efficient or suitable in many circumstances. In some cases, a long throttling interval for traffic optimization may retain presence attributes that change rapidly and are relevant to watchers. Watchers keeping the wrong values of crucial information may involve harmful consequences. In other cases, a short throttling interval may result in many publications of presence attributes that change rapidly but are not relevant to watchers. This would involve more updates of these attributes than necessary, thereby wasting bandwidth and processing resources.

This section studies how to calculate throttling intervals when changes in presence information can probabilistically be modeled, as described in Section 4.2.2. We contemplate both the dynamics of presence information and the watchers' requirements

on information consistency. We propose calculating the throttling interval for the presence information pi over time by adding the average time that this information remains without change $\mathbb{T}_{pi}(t)$ and a delay factor ω. We refer to this interval as the presence information's sojourn-based interval (SBI), which is shown in (4.13). Thus, the presence information's sojourn-based rate (SBR) is SBI^{-1}. The presence information pi may be a presentity's complete presence information or a subset of it (i.e., one or more presence attributes).

$$SBI_{pi}(t) = \mathbb{T}_{pi}(t) + \omega \tag{4.13}$$

The delay factor (ω) for the presence information pi means the amount of time that publications of such information can be delayed. We assume that the PS somehow can deduce this factor based on some policies such as traffic optimization or the watcher's requirements on information consistency. The permanence time of the presence information pi ($\mathbb{T}_{pi}(t)$) changes over time and is calculated from a Markov chain. This approach relies on the assumption that certain users share the same pattern of changes in their presence information and can therefore be modeled through the same Markov chain. As instance, users within a corporation may be grouped by their responsibilities (e.g., managers, secretaries, messengers, etc.) since users with the same responsibility are likely to behave similarly. The PS therefore only needs to handle a state machine for each different pattern of change rather than for each user, which makes the presented approach computationally feasible.

The average time that a Markov chain spends in a state, which is the state's sojourn time, is given by the inverse of the sum of the state's outgoing transition rates [199]. Thus, we can approximate the average time that the presence information pi remains without change, that is, the permanence time of pi, $\mathbb{T}_{pi}(t)$ over time as (4.14).

$$\mathbb{T}_{pi}(t) = \frac{1}{\Phi_{pi}(t)} \tag{4.14}$$

where $\Phi_{pi}(t)$ is the transition probability of the presence information pi. This probability can be deduced from the Markov chain's distribution probability $P^x(t)$ (see Section 4.2.2) and transition rates. Let OS_{pi} be the set of origin states of transitions that change the value of the presence information pi. Let $ES_{j,pi}$ be the set of end states of transitions that depart from state j and change the value of the presence information

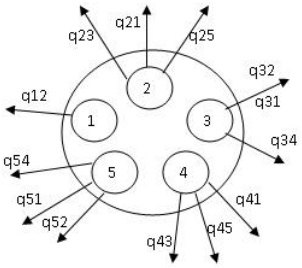


Figure 4.14: SC for the audio attribute

pi. We define the state of change for the presence information pi, SC_{pi}, as a logical state that is made up of all the states in OS_{pi} and the outgoing transition rates from each state j in this set to the states in its corresponding $ES_{j,pi}$. From the state of change SC_{pi}, we can estimate $\Phi_{pi}(t)$, as shown in (4.15). Since state probabilities change over time, the probability of the states in OS_{pi} making a transition, which is the numerator of $\Phi_{pi}(t)$, needs to be weighted with the probability of the chain being at any of these states, which is the denominator of $\Phi_{pi}(t)$.

$$\Phi_{pi}(t) = \frac{\sum_{i \in OS_{pi}} \left[p_i^x(t) * \sum_{k \in ES_{i,pi}} q_{i,k} \right]}{\sum_{i \in OS_{pi}} p_i^x(t)}, i, k = 1...n \tag{4.15}$$

As an example, Figure 4.14 shows the state of change for the audio attribute in Figure 4.3, and Expression (4.16) shows its transition probability.

$$\begin{aligned} \Phi_{audio}(t) = & \frac{\begin{array}{c} p_1^x(t)q_{12} + p_2^x(t)(q_{21} + q_{23} + q_{25}) \\ + p_3^x(t)(q_{31} + q_{32} + q_{34}) + p_4^x(t)(q_{41} \\ + q_{43} + q_{45}) + p_5^x(t)(q_{51} + q_{52} + q_{54}) \end{array}}{p_1^x(t) + p_2^x(t) + p_3^x(t) + p_4^x(t) + p_5^x(t)} \end{aligned} \tag{4.16}$$

4.3.1 Byte Rate Estimation

This section estimates the bytes rate generated by throttling publications of the presence attributes in Figure 4.3 with each attribute's SBI. Section 4.2.4 describes an algorithm for analytically estimating the byte rate that is generated by throttling presence publications with multiple intervals over time, which is based on Expression (4.9). We use this algorithm with SBIs for estimating the byte rate of presence traffic during

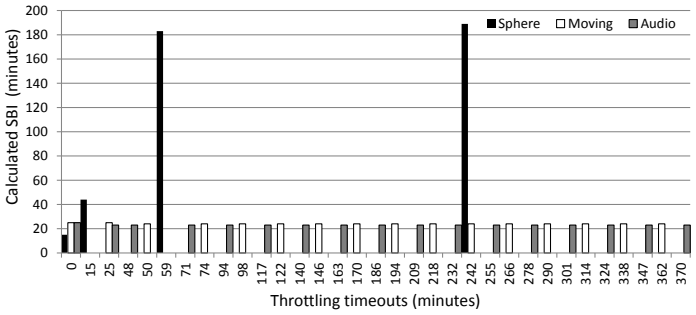


Figure 4.15: SBIs calculated at each throttling timeout over time

some session time. This algorithm assumes that there is a recursive timer for each subset of presence information whose publications are controlled. Each timer ensures a minimum time between two consecutive publications of the presence information associated. We assume that whenever the timer associated with some presence information expires (i.e., throttling timeout), the SBI of this information is recalculated. Initially, each attribute's throttling interval is its SBI at $t = 0$. This study is based on the particular use case described in Section 4.2.5.1, which is a technical employee. Regarding each presence attribute's delay factor, we consider it as half of the maximum delay in receiving a notification acceptable by watchers. Let us assume that the technical employee's watchers are willing to accept a maximum delay of 30 minutes for the audio and moving attributes, and a maximum delay of 10 minutes for the sphere attribute. Therefore, the PS sets a delay factor of 15 minutes for the audio and moving attributes, and other of 5 minutes for the sphere attribute.

Figure 4.15 shows the SBI that is calculated for each presence attribute at throttling timeouts. The X-axis shows the times at which timeouts occur. Each X coordinate for a presence attribute is therefore the sum of the previous X coordinate for this attribute (i.e., the time at which the last timeout occurred) and its corresponding Y coordinate (i.e., throttling interval that was calculated at the last timeout). The assumed transition rates in Table 4.1 describe a user who gets online quickly, and after that a long time goes by until he or she goes offline. Thus, the sphere attribute's SBI increases rapidly over the first 50 minutes until reaching a high stationary value. The audio attribute is the fastest to change, closely followed by the moving attribute.

Figure 4.16 shows the probability of publishing each presence attribute at each

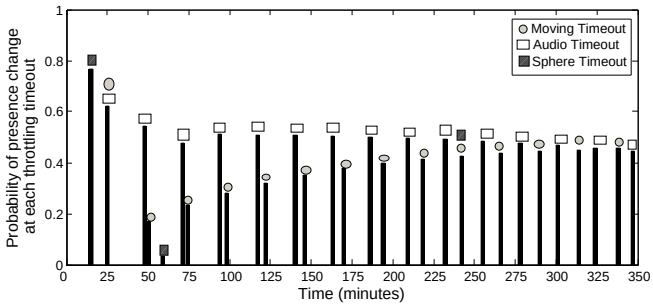


Figure 4.16: Probability of presence change at each throttling timeout with SBIs

timeout when the throttle intervals in Figure 4.15 are used. We have marked every timeout with an identifier of the presence attributes associated with the expired interval. The three presence attributes are very likely to change during the first 25 minutes. They have the same transition probability at the beginning (see Figure 4.3). However, the sphere attribute's throttling interval is shorter than that for the audio and moving attributes because of its shorter average delay. After a while, the audio attribute is a little more dynamic than the moving attribute; thus, the throttling intervals for the former become slightly shorter than those for the latter. When a presence attribute is going to be published, the remaining attributes that changed since the last publication are included into the publication. This is the reason why the moving attribute's change probability is lower than that of the audio attribute. The sphere attribute's change probability is quite low after the first minutes, and hence this attribute is checked very few times during the session time.

Figures 4.17 and 4.18 compares the change probability and byte rate at each throttling timeout, respectively, of sojourn-based and static throttling intervals. In the case of static intervals, the PS is not aware of the dynamics of presence information, and hence, for the sake of traffic optimization, it sets each attribute's throttling interval to its assumed maximum delay (i.e., 30 minutes for the audio and moving attributes, and 10 minutes for the sphere attribute). Since the audio and moving attributes are only checked every 30 minutes and these attributes change frequently, the probability that the watchers keep wrong values of these attributes during the better part of this interval is high. On the other hand, throttling with SBIs adapts to the dynamics of presence information efficiently. It applies throttling intervals to the sphere attribute

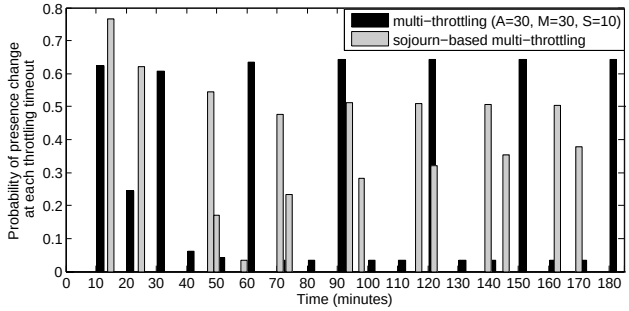


Figure 4.17: Probability of change at each timeout with static and sojourn-based intervals

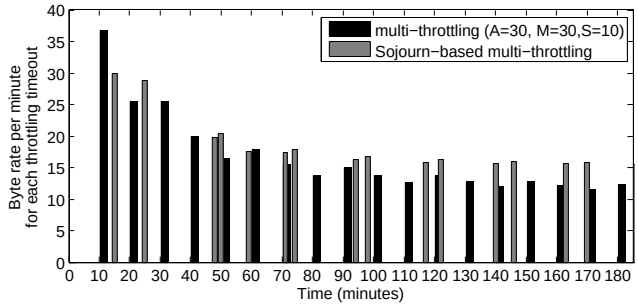


Figure 4.18: Rate of bytes during throttling static and sojourn-based intervals

that are much longer than 10 minutes because this attribute rarely changes. The audio and moving attributes' SBIs are shorter than 30 minutes because these attributes change more frequently. This prevents watchers from keeping obsolete information for too long. Figure 4.18 shows that the traffic rate with SBIs is slightly higher than with static intervals. This is a small price to pay for adapting to the presence information's rate of change, which ensures that watchers are updated with sufficient frequency when optimizing traffic.

4.4 Conclusions

We presented the need to reduce the great amount of presence traffic that presence applications generate for making the presence service scalable in NGNs. Presence publications play a relevant role in presence traffic overload since each publication involves sending and receiving messages by all the components of a presence system: the presentity that makes the change, its PS, its watchers' RLSs, and finally these watchers. Thus,

optimizing presence publications would be very effecient at reducing the overload on the network and user devices. We proposed applying SIMPLE event notification filters to presence sources, which we call publication filtering. We proposed the SIP PUBLISH method for implementing publication filtering. Publication filers let presentities know about what information is important and when it should be published. Publication filtering prevents publishing information that is not necessary, and ensures that the necessary information is published according to certain threshold conditions. Moreover, controlling the rate of presence publications is necessary for deploying resource-efficient mobile presence applications, specially in LBSs. We, therefore, proposed using the extended XML schema for notification filters described in Section 3. This schema permits to control the presentity's publication rate in a fine-grained manner by setting multiple maximum rates for the presentity's presence information. This schema also allows pausing, un-pausing and requesting publications of presence attributes. Such a fine-grained control of presence publications is convenient since the nature of presence attributes is diverse (i.e., they change at different rates) and watchers may have different consistency needs on presence attributes. Thus, attribute-based rate control mechanisms are necessary to optimize traffic efficiently while satisfying the watchers' needs.

We modeled presence changes through a continuous-time Markov chain, and probabilistically estimated the traffic rate when throttling publications with a single minimum interval (i.e., single-throttling) and multiple intervals (i.e., multi-throttling). We gave a guideline about how to model presence changes through Markov chains, and took an example model for estimating presence traffic. The presented mathematical model is valuable because, to the best of our knowledge, there are not other mathematical models of presence changes that are as general as it. The behavior of presence users differs a great deal because the nature of presence applications is very diverse. Even, the presence information of a particular application's users usually does not follow any particular pattern; the actions that users take and affect their presence information (e.g., modifying personal state, mood, activities, location) are highly subjective and depend on temporary circumstances. These issues make the analysis of presence systems specially difficult.

We presented the mathematical formulas that calculate the probability of presence changes occurring during each throttling interval with single- and multi-throttling.

From these formulas, we derived the total number and rate of bytes sent on the network access link because of presence publications during a presence application session. Regarding single-throttling, we analyzed throttling intervals of 5, 10, 15, and 30 minutes, which reduce the presence traffic by 41%, 61%, 71%, and 85%, respectively. Our analysis validated some reasonable fundamentals: It is not recommended to apply a very short throttling interval because it does not save presence publications and increase the complexity of end devices. Moreover, the longer the throttling interval, the lower the presence traffic rate. However, the delay perceived by the watchers increases with the throttling interval length. It is therefore necessary to find a tradeoff between the watchers' information consistency and the saving of presence traffic. Multi-throttling can actually help in finding such a tradeoff since it consists in setting attributes-based throttling intervals. Multi-throttling makes it possible to set short intervals for the most urgent information while the remaining information is regulated by longer intervals. Thus, this technique may save publications of information that is not urgent or important to the PS or watchers. We described an algorithm for calculating the change probability of presence attributes with multi-throttling. Thus, we calculated the presence traffic rate when multi-throttling is applied to different scenarios. The efficiency of multi-throttling depends on the level of importance that is set for the presence attributes that change most frequently. In the best case, the attributes that change the most are not important, and hence can be associated with long throttling intervals. In the worst case, these attributes are the most important, and hence they have to be notified at short intervals.

A throttling interval length should be chosen carefully because presence attributes that change frequently should not be published at low rates. Otherwise, watchers may keep obsolete presence information for too long and publications would even turn out to be inefficient. If the presence information changes much more rapidly than the publication rate, the short time during which watchers see valid information does not compensate for the traffic generated to publish such information. We proposed Sojourn-Based Intervals (SBIs) for limiting the rate at which presentities publish presence information while avoiding excessive delays in publishing. We probabilistically compared the traffic rate generated by throttling publications with SBIs and fixed intervals. While the former change over time for adapting to the presence attributes' change frequency, the latter are predefined, static intervals. The reported traffic rate

shows that the application of SBIs increases the traffic rate generated by fixed intervals slightly (by approximately 17%). This increase is due to the fact that SBIs are shorter than the fixed intervals so as to publish the presence information at a rate that is close to the change frequency of this information. This shows that SBIs can adapt well to presence change patterns.

4. OPTIMIZATION OF PRESENCE PUBLICATION TRAFFIC: PROPOSAL, MATHEMATICAL MODEL AND PERFORMANCE ESTIMATION

5

Strategies for Reducing Inter-domain Presence Traffic: a Performance Analysis and Novel Proposal

Future presence-based applications are expected to be used worldwide across different administrative domains. Moreover, there is a strong indication that presence will be a key enabler of the convergent services supported by the future NGNs, as described in Section 2.4. There will be no administrative or technological barriers in these networks for communicating with users from different domains. The foundation for such a new generation of communication networks is the IMS. This system evolves mobile operators towards an all IP technology for the support and integration of advanced multimedia services. We refer the reader to Section 2.4 for information on NGN and IMS. The service of presence forms part of the IMS specification, and plays an increasingly important role in existing and emerging multimedia services. As explained in Section 2.9.4, the IMS presence service is particularly challenging because of its constant flows of signaling traffic, which are likely to impact the IMS centralized servers severely. Presence subscriptions have to be refreshed periodically to prevent their lifetime from expiring, which would result in their elimination. A subscription's lifetime is refreshed by sending a re-SUBSCRIBE message, which entails the exchange of four messages. On the other hand, whenever a presentity changes its state, a NOTIFY message is sent

to all the watchers that are authorized to see the presence change. These operations cause presence-based applications to generate a great amount of traffic as the number of presentities, watchers and presence changes increases. Presence traffic overload becomes even more harmful and critical in LBSs, since frequent location updates have to be timely disseminated through XML documents [38]. The authors of [200] discuss important scalability issues of the IMS. The authors of [179] analytically estimate presence traffic, and thereby conclude that this traffic may account for more than 50% of the total traffic handled by the IMS CSCF. The performance analysis in [201] shows that SIP signaling traffic introduces long transmission delays on the UMTS network. The authors study the end-to-end delay that IM users perceive. The reported results show that this delay may reach so high values that IM can no longer be regarded as an instantaneous service. Much of this delay is due to the network core, and hence optimizing traffic on the radio access side is insufficient for providing multimedia services in real time.

As mentioned in Section 2.7.4, in the SIMPLE framework, there are three techniques that subscribers can apply for reducing inter-domain presence traffic on the network core: partial notifications of presence [138], event notification filtering [140], notification rate control [183] and conditional notifications [147]. Section 2.9.4 describes other research works that may be used to reduce the number and size of presence notifications. However, all these strategies do not deal with the volume of subscriptions, and hence they are not efficient enough in large-scale presence federation scenarios. As described in Section 2.9.5, scalability becomes critical in these scenarios where millions of users in a domain subscribe to millions of users in other federated domains. Presence federation simply refers to the interconnection of different domains for sharing presence information. When two or more watchers in a domain are interested in a particular presentity within a federated domain, the watchers' domain creates a different subscription to this presentity for each watcher. Each of these subscriptions is refreshed and kept up to date on the presentity's presence independently. Whenever the presentity's presence information changes, the federated domain sends a different presence document to each subscribed watcher regardless whether or not these documents contain the same information. Some proposals for reducing inter-domain presence traffic in federation scenarios have been proposed as IETF Internert-Drafts to date, namely dialog optimization [193], Common Notify (CN) [194] and View Sharing (VS) [195]. Notwithstanding

these proposals have been discontinued, the need to ease the impact of inter-domain presence traffic remains to achieve scalable presence services as pointed out in Section 2.9.5. Dialog optimization basically consists in a federated RLS that maintains a single subscription between a watcher and all its presentities within a particular federated domain. Although this technique was thought of for reducing presence traffic, the authors conclude that this technique is not adequate for this end based on a mathematical analysis of its traffic. The authors of [193] analytically estimate inter-domain presence traffic when this optimization and conditional notifications are applied by the federated domains. Only when the dialog optimization technique is combined with conditional notifications, the inter-domain presence traffic is reduced. In two of the scenarios considered by the authors, this combination reduces the bytes exchanged by the federated domains by approximately 22%. These scenarios are "Widely distributed inter-domain presence" and "Intra-domain peering". For further information on them, we refer the reader to [193]. In the last scenario considered, which is "Large network peering", the reduction of presence traffic drops to 8%. This result is demotivating since this scenario represents a large-scale presence federation. Moreover, the authors state that dialog optimization worsens the scalability of the overall system, since it increases the complexity of subscription state, interlinkage and notifications. Dialog optimization is therefore no longer considered as an alternative for reducing inter-domain presence traffic. Nevertheless, the authors make a substantial contribution towards analyzing SIMPLE inter-domain presence load. They conclude that the scalability of presence systems is far from being trivial from several perspectives: number of messages, network bandwidth, state management and CPU load. Moreover, they stress that not all the possible optimizations for reducing presence traffic have been done yet, and that further work must be done by the IETF in order to provide better scalability. CN basically consists in sending a single presence notification to the federated domain, which is therefore in charge of generating a different notification for each watcher within this domain. VS proposes establishing a different inter-domain presence subscription for each view of the presentity's presence information that the watchers are authorized to see. The authors of both CN and VS do not provide any mathematical analysis or experimental result about these techniques' efficiency. Thus, we do not know the causes by which these works were discontinued.

5. STRATEGIES FOR REDUCING INTER-DOMAIN PRESENCE TRAFFIC: A PERFORMANCE ANALYSIS AND NOVEL PROPOSAL

No other alternatives for optimizing inter-domain presence traffic in large-scale federation scenarios have emerged apart form the aforementioned strategies. The authors of [202] apply CN and batched notifications [194] to a service differentiation scenario. Batched notifications consist in aggregating the presence documents of multiple presentities in a single NOTIFY message. The authors state that the optimization techniques tend to delay notifications. Thus, they do not apply optimization techniques to gold customers, while the notifications to silver and copper customers are optimized by CN and batched notifications, respectively. Batched notifications do delay presence notifications in order to aggregate changes from multiple presentities. However, the authors do not justify the assumption that CN introduces delay. On the contrary, it is reasonable to assume that a common NOTIFY message is generated more quickly for two reasons: First, the PS does not need to perform privacy filtering, and secondly only one NOTIFY message per presentity needs to be sent. In fact, the authors conclude that the lower gold delay is achieved by a customer-based priority mechanism at the PS rather than the lack of optimizations. While the authors present the first implementation of these strategies, they do not seem to address some implementation details. Specially, they seem not to tackle the process of privacy filtering at the subscriber side PS. CN requires this PS to obtain the presentities' privacy rules in order to generate the presence documents that its watchers are authorized to see. This process may increase the inter-domain presence traffic, and hence it should be taken into account when measuring the performance of CN.

Our contribution in the frame of inter-domain presence traffic optimization in large-scale scenarios is as follows. Section 5.1 studies and analizes the performance of dialog optimization, CN and VS. We define the formulas that calculate the number of bytes per session exchanged between two federated domains that apply these strategies. We also describe in detail the assumptions about the operation of each technique that we had to make for the presented analysis. Such assumptions give an idea of the complexity of each strategy and pave the way for other researchers on the subject. We tackle the various parameters that affect the performance of these proposals. As well, we give guidelines on which approach to choose based on these parameters. Our goal is to figure out (1) how complicated the optimization of inter-domain presence may get to be; (2) how the studied strategies perform; (3) what parameters network administrators should take into account for optimizing inter-domain presence traffic; and last, but not

least, (4) what the sensitivity of each technique to these parameters is. Section 5.1 also proposes a new strategy for reducing inter-domain presence traffic. We compare this strategy's efficiency with that of the afore-mentioned strategies. Section 5.2 enhances the proposed strategy for disclosing as few privacy rules as possible, which in turn involves a considerable reduction of presence traffic. Finally, Section 5.3 presents the main contributions of this study.

5.1 Inter-Domain Presence Traffic Estimation and Sensitivity Analysis

As described previously, SIMPLE involves much overload in large-scale federation scenarios because it generates a different presence subscription for each watcher. Thus, the PSs at the subscriber- and notifier-side domains are extremely likely to maintain numerous presence subscriptions to the same presentity. We refer to watcher intersection as the probability that watchers in a domain subscribe to the same presentities. The optimization techniques CN and VS consider watcher intersection for reducing the number of inter-domain presence notifications and subscriptions. We propose other watcher-intersection-aware optimization strategy, which we refer to as Common Subscribe (CS), that reduces the number of presence subscriptions to the maximum. We analytically estimate the presence traffic generated by these three strategies and carry out a deep comparison between them. Section 5.1.1 explains the method and general assumptions behind all the traffic calculations presented in this section. Sections 5.1.2 and 5.1.3 describe CN and VS, respectively, while Section 5.1.4 explains our proposal for reducing inter-domain presence traffic, that is, CS. Section 5.1.5 shows the results of the traffic calculations, and evaluates the various factors and implementation decisions that affect the performance of the studied strategies.

5.1.1 Methodology and Assumptions

We follow the methodology in [193] for estimating the presence traffic between two federated domains. Since the traffic generated by the dialog optimization technique is calculated in [193], we do not tackle it in the presented analysis. We only compare the results about this technique presented in [193] with those presented in the following sections. Figure 5.1 outlines this strategy for the sake of clarification. This consists in an

RLS (see Section 2.7) that handles the watchers' lists of presentities within the federated domain. In Figure 5.1, a watcher in DomainB, Alice, is interested in obtaining presence information about Bob, John and Anna in DomainA. Alice sends a single SUBSCRIBE message that contains an URI identifying her list of federated presentities in DomainA, rather than creating three independent subscriptions.

Throughout the following sections, we refer to the domain that sends presence subscription requests as subscriber or watcher domain (i.e., DomainB in Figure 5.1), and to the domain that receives these subscriptions as notifier or watched domain (i.e., DomainA in Figure 5.1). We classify presence messages into three groups. The initial messages are those in the initial phase of establishing a subscription. The steady state messages are exchanged in the time that elapses between the initial subscription and the termination of the subscription. They contain the notifications due to state changes and subscription refreshes. Finally, the termination messages are those in the termination phase of the subscription. Throughout this paper, we show a number of mathematical formulas that calculate the traffic in bytes generated by the SIP subscriptions involved in each optimization strategy. A subscription's traffic is calculated as the sum of the initial, steady and termination messages during the session time. We only show the final formulas in the following sections. Appendix F shows the complete formulation of the SIP subscriptions involved in the studied strategies. The presented formulas are functions of the constants and variables in Table 5.1. The *wat* variable counts the number of watchers who are subscribed to a particular presentity, which means the intersection of watchers for that presentity. The other two variables are *pch* and *pres*, which respectively mean the frequency at which presentities change their presence and the total number of presentities. Section 5.1.5 assign these variables default values for analyzing presence traffic. The remaining elements in this table are constants. We assume the average subscription lifetime is 8 hours, and the average refresh interval to keep subscriptions alive is 1 hour. The constants *sub*, *sok*, *not* and *nok* are the sizes of subscription-related SIP messages; their values have been taken from [110]. The constant *doc* is the average size of presence documents that contain person-related, device and location information. We assumed presence documents of 3000 bytes, which is a moderate assumption if device-related and location information are included into these documents, as pointed out by the authors of [193]. Appendix B shows an example presence document that contains this kind of information. The constant *mpb* is the size

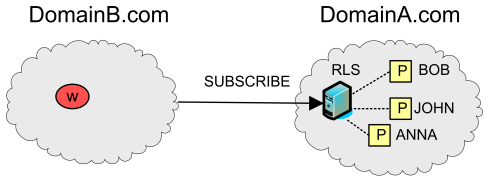


Figure 5.1: Sketch for dialog optimization

Name	Description	Average value
slife	Subscription lifetime	8 hours
pch	Presence state changes / hour	*variable*
sref	Subscription refresh interval / hour	1
wat	Number of watchers per presentity within the subscriber domain	*variable*
pres	Number of presentities within the notifier domain	*variable*
sub	SUBSCRIBE message size	450 bytes
sok	200 OK for SUBSCRIBE message size	370 bytes
not	NOTIFY message size	500 bytes
nok	200 OK for NOTIFY message size	370 bytes
doc	size of presence documents	3,000 bytes
wlit	size of *watcher* elements in watcher lists	160 bytes
mpb	Size of boundaries in Multipart bodies[1]	144 bytes
wlenv	Size of XML envelopes in watcher lists	144 bytes
aclenv	Size of XML envelopes in ACL lists	186 bytes
aclrl	Size of rules in ACL lists	22 bytes
aclmb	size of *member* elements in ACL lists	38 bytes
pfwat	Percentage of watchers to which presentities apply privacy rules	50%
pfdoc	Size of privacy authorization documents	1,000 bytes

Table 5.1: Constants used to estimate federated presence traffic

of a boundary in Multipart documents [203]. The constant *pfdoc* is the size of privacy authorization documents (so-called privacy filters), which we have deduced from the examples in [177] and [204]. Appendix D shows an example privacy filter document. The constants *wlit* and *wlenv* are used to calculate the watcher list size in CN and CS, and their values have been deduced from [178]. Appendix E shows an example of watcher information. Lastly, the constants *aclenv*, *aclrl* and *aclmb* are used to calculate the size of Access Control Lists (ACL), which describe the views of the presentities' presence information in VS. We have considered the examples in [195] for choosing the values of these variables.

Privacy filtering is a determining factor in the efficiency of any technique for optimizing presence traffic. In the normal mode of operation of privacy filtering [204], when the PS is going to notify a watcher of a presentity's presence information, it determines the set of privacy rules that match the watcher. The combined rules are applied to the presentity's presence information, thereby generating the presence document that the watcher is permitted to watch. Presence privacy rules or policies [177] (also known as authorization rules or policies) specify what presence information can be given to which watchers. A privacy rule contains conditions, which determine under what circumstances the rule is to be applied, actions, which indicate what actions the PS has to take, and transformations, which specify the visibility the watcher is granted. Appendix D shows an example privacy authorization document. With watcher-intersection-based strategies, the PS notifies a presence document to a set of watchers rather than a single watcher. Thus, this type of strategy needs more complicated operation modes for privacy filtering than were previously required. Privacy filtering operates in a different way for each of the optimization techniques and therefore is explained in Sections 5.1.2.1, 5.1.3.1, and 5.1.4.1. Nevertheless, we have some general assumptions for all them. We assume that all the privacy rules (i.e., *rule* XML element in Appendix D) that match a particular watcher are contained in a single document, which is called the watcher's privacy rules document or privacy filter. Thus, if privacy filtering is applied to a watcher, the watcher is associated with a privacy rules document (or privacy filter) that contains all the rules that concern the watcher. We assume that each privacy filter contains a single watcher for simplifying calculations. Privacy filters are therefore individual and unique. Moreover, we assume that the privacy filters have already been created when watchers subscribe to presentities and do not change during the session time.

[1]Multipart messages combine different sets of data in a single body. Each "body part" is preceded by an encapsulation boundary

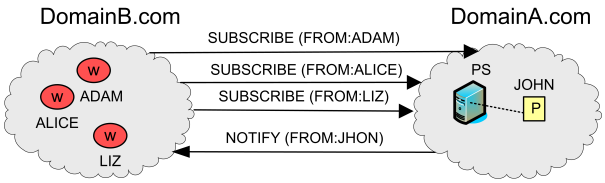


Figure 5.2: Sketch for the Common Notify strategy

5.1.2 Common Notify

5.1.2.1 Overview

With CN [194], when a presentity changes its presence information, the PS sends a single NOTIFY message to all the watchers within the same domain that are subscribed to the presentity. This single NOTIFY message carries a presence document that contains the presentity's complete presence information. Thus, the watcher domain is responsible for providing watchers with the presence documents that they are allowed to see. This domain needs therefore to know each watcher's privacy rules, which are set in the presentity's domain. Figure 5.2 is a straightforward illustration of this strategy. Three watchers in DomainB are interested in John's presence information and wish to subscribe to him. The PS in DomainA receives three requests from DomainB to watch John, and creates a separate subscription for each authorized watcher. When John changes his presence information, the PS sends a single NOTIFY message to Domain B rather than three separate messages through each subscription. This strategy eliminates the need to send individual NOTIFY messages from the domainA's PS to each watcher within DomainB. A positive feature of CN is that it does not need the watcher domain's PS to work as a proxy for any subscription request. In very large scale domains, this feature could be helpful to save overload at the servers. Section 6 discusses about how the studied strategies overload the PS and other network servers. As described in [194], implementing CN involves resolving two main issues:

Knowledge of watchers to whom the common NOTIFY is targeted: When a watcher domain receives a common NOTIFY, it needs to know to which watchers this message is targeted. There are three different ways to obtain the watcher list: (1) *maintaining the list on the subscriber side PS*, (2) *including the list in notifies* and (3) *obtaining the list by subscribing to a winfo package*. In the first

one, the watcher domain's PS creates and constantly updates the list of watchers who are watching the presentity's presence. This means that all SUBSCRIBE messages must go from watchers to presentities through the watcher domain's PS. In the second way, the common NOTIFY messages include the list of watchers to whom these messages are sent. Lastly, with winfo event subscriptions, the watcher domain's PS subscribes to the presentity's winfo event package [145]. Thus, whenever a new watcher subscribes to the presentity, a watcher list document is notified to the watcher's PS. The last two ways follow the format described in [178] for watcher lists; Appendix E shows an example watcher list document.

Knowledge of privacy rules: Privacy filtering has to be performed to provide watchers with the presence documents generated according to presentities' privacy rules. There are two ways to define privacy filtering: domain-based and watcher-based privacy filters. The former sets the same privacy filter for all the watchers in the same domain and, therefore, privacy filtering can be performed by the presentity's PS. The latter defines privacy filters based on particular watchers, and hence privacy filtering must be applied by the watcher domains. With watcher-based filters, a presentity's complete presence state information and privacy filters must be sent to the watcher domain. As the technical report [205] describes, the watcher domains may subscribe to a new event package that would represent the privacy filters associated with a presentity. The presentity's PS would handle this event package and generate a NOTIFY message whenever a privacy filter changed. NOTIFY messages may contain privacy rules themselves (in a full- or partial-state document) or a URL that points to a privacy rules document handled by an XCAP server [206]. We denominate this new event package as "privacy-filters", and assume that notifications of this event carry privacy rule documents. Since the authors of [205] do not give any detail about this new event package implementation, we need to take some design decisions. We assume that if a watcher does not have any associated privacy rule, it means that the watcher is allowed to see the presentity's complete presence. Thus, only the watchers which are restricted to see a subset of the presentity's presence have privacy rules associated. Moreover, we consider two options for subscribing to a privacy-filters event:

1. Basic privacy-filters subscriptions: When a watcher domain subscribes to a presentity's privacy-filters event, the presentity's PS notifies all the privacy rules related to any watcher within this domain, regardless of whether the watcher is actually active, that is, watching the presentity. Since privacy filters are unlikely to change often, once the subscriber downloads the rules, almost no notifications will be received normally.
2. List-based privacy-filters subscriptions: The SUBSCRIBE messages sent by the watcher domain include the list of watchers who are interested in obtaining the presentity's presence (i.e., the watchers to who the privacy rules will be applied). Thus, the presentity's PS only notifies the privacy rules related to the watchers on this list. The drawback of this option is the amount of re-SUBSCRIBE messages that are sent for obtaining new watchers' privacy rules. Whenever a watcher of a particular presentity becomes active (i.e., starts watching), the presentity's privacy rules for this watcher must be downloaded. Moreover, the watchers must use their PS as gateway to send their presence subscription requests because the PS has to know the watchers that are really watching the presentity.

5.1.2.2 Traffic Calculation

Table 5.2 shows the different alternatives for implementing CN, and the types of subscription that these alternatives involve. When watcher-based filtering is applied, these alternatives assume basic privacy-filters subscriptions, which are described above. This is because the basic method for privacy-filters subscriptions is expected to generate less traffic than the list-based method. Nevertheless, we include the traffic calculation for the list-based method below because a comparison between the two methods is given in Section 5.1.5.3. Note that the mark *(*WLN)* indicates that the watcher list is included in NOTIFY messages. The traffic for each alternative is calculated as the sum of all the involved events' traffic. The formula that results of this sum is omitted because of their extensive length. The following points show the formulas that calculate each event's traffic in bytes.

- Subscriptions to presence events

ID	Description	Presence event	Privacy-filters event	winfo event
CN1	*Domain-based privacy filters, WLs in notifies*	YES (*WLN)	NO	NO
CN2	*Domain-based privacy filters, WL with winfo subscription*	YES	NO	YES
CN3	*Domain-based privacy filters, WL in watchers' PSs*	YES	NO	NO
CN4	*Watcher-based privacy filters, WLs in notifies*	YES (*WLN)	YES	NO
CN5	*Watcher-based privacy filters, WL with winfo subscription*	YES	YES	YES
CN6	*Watcher-based privacy filters, WL in watchers' PSs*	YES	YES	NO

Table 5.2: Types of events involved in CN based on its configuration

Calculating this type of traffic depends on the means for obtaining the watcher lists. As described in Section 5.1.2.1, there are two possibilities that involve traffic:

1. Including the list in the NOTIFY messages: $pres * wat * (sub + sok + not + doc + nok) * (\frac{slife}{sref} + 1) + (pch * slife - 2) * pres * (not + mpb + doc + mpb + wlenv + \frac{wat}{2} * wlit + nok)$.

   We assume that, by the time a presence change occurs, the average number of active watchers is half the total number of watchers. This assumption determines the size of the watcher list included in each presence notification.

2. Obtaining the list by subscribing to a winfo event: $pres * wat * (sub + sok + not + doc + nok) * (\frac{slife}{sref} + 1) + (pch * slife - 2) * pres * (not + doc + nok)$.

- Subscriptions to winfo events: $pres * (sub + sok + not + wlenv + nok) * (\frac{slife}{sref} + 1) + pres * wlit * \frac{wat}{2} * (\frac{slife}{sref} + 2 + wat) + pres * (wat - 1) * (not + wlenv + nok)$

  We have made the following assumptions in these calculations. The PS notifies the watcher domain of the watcher list whenever a new watcher subscribes to the presentity. Winfo notifications are full-state, which mean the watcher lists contain all the watchers that are active at the notification time. By the time

the watcher domain resubscribes to keep a winfo subscription alive, the average number of watchers subscribed is half the total number of watchers.

- Subscriptions to privacy-filters events: $pres*(sub+sok+not+\frac{pfwat*wat}{100}*(mpb+pfdoc)+nok)*(\frac{slife}{sref}+1)$
- List-based subscriptions to privacy-filters events: $pres*((sub+wlenv+sok+not+nok+\frac{\frac{pfwat*wat}{100}+1}{2}*(mpb+pfdoc))*(\frac{slife}{sref}+wat)+wlit*\frac{wat}{2}*(\frac{slife}{sref}+2+wat)-2*mpb-pfdoc)$

  We assume that at anytime when the subscription state is refreshed, the number of watchers that are active and have an associated privacy filter is equal to the average of one plus *pfwat* per cent of the total number of watchers (note that as minimum one watcher is already active when refreshes occur). This number determines the number of privacy filters that the NOTIFY messages for refreshes contain.

5.1.3 View Sharing

5.1.3.1 Overview

VS classifies the presentity's watchers based on the part of the presentity's presence that they are authorized to see. It is referred as the watcher's "view" on the presentity. A view is a particular sequence of presence documents that come about as a consequence of a particular authorization and privacy policy. Two watchers who share the same view will always receive the same presence document when the presentity's presence changes. The key idea in VS is that the watcher domain handles a single subscription for the watchers that share a particular view of the presentity's presence. Whenever the watcher domain is notified of a new presence document associated with a particular view, it is responsible for distributing this document to all the watchers who are allowed to see that view. Contrary to CN, with VS, there is no need to obtain the watcher lists and the privacy filters. No privacy filters exchange is a significant advantage from the point of view of security, and constitutes the main benefit of VS. Figure 5.3 outlines this strategy. The presentity John is watched by three watchers in DomainB, and hence the DomainB's PS may manage a maximum of three different views (i.e., three different subscriptions). This is the case depicted in Figure 5.3.

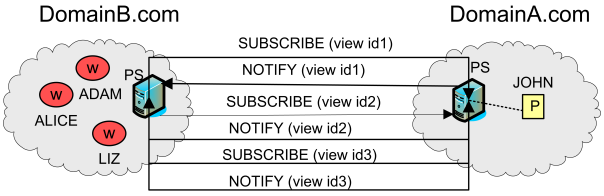


Figure 5.3: Sketch for the View Sharing strategy

The first time that the watcher domain's PS receives a SUBSCRIBE message to a presentity, it resends this message to the presentity. The presentity's domain replies with a NOTIFY message that contains an ACL whose XML scheme follows the specifications in [195]. In the ACL document, each view is represented by a *rule* element and identified by a unique ID attribute. Each view contains a set of *member* elements that contain the URIs of the watchers authorized to see the view. An ACL document's content depends on the level of trust the notifier domain grants to the watcher domain. As described in [195], there are three levels of trust:

Full trust: The ACL documents include all the watchers in the requester domain, associated with their respective views. There is only one backend subscription from the watcher domain to the PS for each view. The PS discloses the complete association of watchers with views, and does not know the full set of watchers currently subscribed.

Partial trust: A watcher's ACL document includes the watcher and all the other watchers that are authorized to see the view associated with the watcher. As with full trust, there is only a backend subscription between the watcher domain and the PS for each view, and the PS does not know the full set of watchers currently subscribed. However, the PS only discloses the watchers who see the same view, not the full set of them.

Minimal trust: A watcher's ACL document only includes the watcher, and there is therefore a backend subscription for each watcher. However, if multiple watchers share a particular view, the presence changes are sent through one of the subscriptions and the watcher domain distributes the changes to all of the other watchers. The PS never discloses the list of authorized watchers or their views and has full knowledge of the watchers actually subscribed.

ID	Description	Presence event
VS1	*Domain-based privacy filtering and full trust*	YES
VS2	*Domain-based privacy filtering and partial trust*	YES
VS3	*Domain-based privacy filtering and minimal trust*	YES
VS4	*Watcher-based privacy filtering and full trust*	YES
VS5	*Watcher-based privacy filtering and partial trust*	YES
VS6	*Watcher-based privacy filtering and minimal trust*	YES

Table 5.3: Types of events involved in VS based on its configuration

In the case of minimal trust, VS applies the idea behind CN to the scope of views. A presence subscription is established for each watcher, but when a presence change occurs, a single NOTIFY message is sent per view. The watcher domain only knows a watcher's view when this watcher becomes active and then a watcher subscription is created. However, with partial and full trust, the watcher domain gets to know all the watchers associated with a view, regardless whether they are active or not. In the case of partial trust, only active views (i.e., views that are actually being observed by some watcher) are disclosed, while full trust gives all the views in. These two methods create a subscription and a watcher list for each view. When the watcher PS receives a SUBSCRIBE message to a presentity, it checks the available ACLs and determines whether the user is associated with some active view (i.e., subscription). In this case, the watcher is added to the subscription's watcher list. Otherwise, the SUBSCRIBE message is forwarded to the presentity and, if the request is successful, a subscription state and a watcher list, which contains the watcher, is created for the view. Regarding state notifications, the NOTIFY messages for refreshes must include the most recent ACL and presence document, while the NOTIFY messages for presence changes only include the presence document.

5.1.3.2 Traffic Calculation

Table 5.3 shows the possible alternatives for VS, which only involve the presence event and drastically differentiate from each other based on the type of trust. The following points give the formulas that estimate each type of trust's traffic in bytes. In these formulas, the percentage of watchers with privacy restrictions is given by the variable *pfwat* (see Table 5.1). The domain-based privacy filtering is done when *pfwat* is set to

0. Thus, the calculations below, from up to down, represent the VS3, VS2, and VS1 cases in Table 5.3 when this variable is 0; otherwise, the VS6, VS5, and VS4 cases.

- Minimal trust (VS3 and VS6): $pres * wat * (sub + sok + not + mpb + doc + mpb + aclenv + aclrl + aclmb + nok) * (\frac{slife}{sref} + 1) + (pch * slife - 2) * pres * \frac{\frac{pfwat*wat}{100}+1}{2} * (not + doc + nok)$.

- Partial trust (VS2 and VS5): $pres*((sub+sok+not+mpb+doc+mpb+aclenv+aclrl+(wat-\frac{pfwat*wat}{100}*aclmb+nok))+\frac{pfwat*wat}{100}*(sub+sok+not+mpb+doc+mpb+aclenv+aclrl+aclmb+nok))*(\frac{slife}{sref}+1)+(pch*slife-2)*pres*\frac{\frac{pfwat*wat}{100}+1}{/2}*(not+doc+nok)$

- Full trust (VS1 and VS4): $pres*(sub+sok+not+mpb+doc+mpb+aclenv+aclrl+(wat-\frac{pfwat*wat}{100})*aclmb+\frac{pfwat*wat}{100}*(aclrl+aclmb)+nok)*(\frac{pfwat*wat}{100}+1)*(\frac{slife}{sref}+1)+(pch*slife-2)*pres*\frac{\frac{pfwat*wat}{100}+1}{2}*(not+doc+nok)$

The calculations above require doing the following assumptions. The watchers with no restrictions on presence, that is, $(100 - pfwat)\%$ of watchers, are added to a default view that includes the full presence information. The presentity's PS knows all the possible watchers, and hence they are always associated with a specific view or the default one. Whenever a presence change occurs, it affects half of the total views, and hence every presence change is notified through half of the total number of subscriptions. Each view is associated with a single watcher, and hence each watcher to which privacy filtering is applied has a different view of the presentity's presence. With minimal trust, the watcher domain maintains a view for each watcher, and hence the number of ACL documents that it handles is *wat*. With full trust, the watcher domain handles a single ACL document that contains information about all the presentity's views. This ACL document has a different rule element for *pfwat* per cent of the watchers. Each of these rule elements contains a single member element with the URI of the watcher that is authorized to see the view. Another rule element is the default view, which contains the remaining watchers. With partial trust, the watcher domain handles a different ACL document for each view, which includes a single rule and a single member element. It also handles an ACL document for the default view, which includes a single member element and as many member elements as $\frac{(100-pfwat)wat}{100}$. Therefore, the number of ACL documents that it handles is $1 + \frac{pfwat*wat}{100}$.

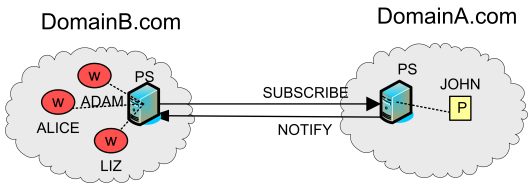


Figure 5.4: Sketch for the Common Subscribe strategy

5.1.4 Common Subscribe

5.1.4.1 Overview

CS is our proposal for reducing inter-domain presence traffic. This strategy creates a single subscription for each presentity between the subscriber and notifier domains (see Figure 5.4). This single subscription saves much signaling traffic for managing multiple watcher subscriptions for each presentity. CN maintains a different subscription between the subscriber and notifier domains for each (watcher,presentity) pair. This means that a great deal of traffic is generated for refreshing subscriptions. However, these subscriptions do not serve to send presence notifications. They are simply means of letting the notifier domain know and authorize the watchers that are actually watching each presentity.

In CS, the watcher domain must obtain the presentities' privacy filters for generating the presence documents that watchers are authorized to see. The domains only manage one subscription per presentity, which is associated with all the watchers interested in the presentity. Therefore, the subscriber domain has to somehow let the notifier domain know about the watchers that are actually watching some presentity. Sending this list in the body of SUBSCRIBE messages is a reasonable solution. It requires the watcher domain's PS to work as a proxy for all the SUBSCRIBE messages sent by watchers within this domain. When this PS receives a subscription request to a presentity, it sends a common SUBSCRIBE message to the presentity. This message's body contains the presentity's watcher list that includes the new requester. When the presentity's PS receives a common SUBSCRIBE message, it examines the watcher list in this message and accordingly updates the common subscription's watcher list. The PS adds any new watcher, as long as it is authorized to see the presentity's presence, to the list and removes the missing watchers. Then, the updated watcher list is included in the

body of the NOTIFY message that is sent in response to the subscription request. This message has a multipart body in which one part is the presence document and the other part is the watcher list. Both the subscriber and notifier domains know the watchers to which presence notifications are directed and no other techniques (e.g., those in CN) are necessary. Should a presentity revoke the authorization of a watcher to see its presence, the presentity's PS sends a NOTIFY message that includes the watcher list without the rejected watcher. Regarding privacy filtering, the watcher domain subscribes to the presentities' privacy-filters events for getting their filters, which was described in Section 5.1.2.1.

Each presentity is associated with two subscriptions for each watcher domain: one to presence information and other to privacy filters. Let us consider these two events as the two parts of a more general event that represents all the information about a presentity that a watcher domain is authorized to see. We refer to this event as "federated-presence". When a presentity' PS receives a federated-presence subscription request, if the requester domain is authorized, the PS generates a subscription status to the presentity's presence and privacy-filters events for the requester domain. The PS, therefore, sends both the presence document and the privacy filters in the NOTIFY message as a result of a subscription request. Nevertheless, a change in the presentity's presence or privacy filter information only results in notifying the information that changed (the presence document or the filter). Federated-presence subscriptions save the SUBSCRIBE, OK and NOTIFY messages for subscribing, refreshing and terminating privacy-filters subscriptions. We denominate this variation of CS as Federated Common Subscribe (FCS).

As described in Section 2.7.4, conditional notifications [147] avoid sending the NOTIFY messages as a result of re-subscription requests when no presence changes have occurred from the last notification. With CS, if these NOTIFY messages were saved, the subscriber would not be able to know whether or not the watched domain authorized the list of watchers given by the re-SUBSCRIBE message. The NOTIFY messages can not be saved since they carry the list of authorized watchers. However, conditional notifications can still be combined with CS. The purpose of this optimization is to save the resource state information that is sent in response to subscription refreshes. A presentity's resource state information is the presentity's presence; watcher list information is control data. With CS, conditional notifications must not be applied to the watcher

ID	Description	presence event	privacy-filters event	federated-presence event
CS1	*Domain-based privacy filters*	YES	NO	NO
CS2	*Watcher-based privacy filters*	YES	YES	NO
CS3	*Watcher-based privacy filters*	NO	NO	YES

Table 5.4: Types of events involved in CS based on its configuration

list information. Therefore, this optimization does not save sending the NOTIFY messages for refreshes but just avoids including the presentity's presence document in these messages when it is unnecessary.

5.1.4.2 Traffic Calculation

Table 5.4 shows the alternatives for implementing the CS strategy and their involved events. Each alternative's total traffic is the sum of the associated event subscriptions' traffic. The formula that results of this sum is omitted because of its extensive length. The following points show the formulas that calculate each event's traffic in bytes. The calculation for the privacy-filter event's traffic is omitted since it was already presented in Section 5.1.2.2.

- Common subscriptions to presence event: $pres * (sub + sok + 2 * wlenv + not + doc + nok + 2 * mpb) * (\frac{slife}{sref} + wat) + pres * wlit * wat * (\frac{slife}{sref} + wat + 2) + (pch * slife - 2) * pres * (not + doc + nok)$

  The above formula is based on the following assumptions. The first SUBSCRIBE message that is sent for creating a common subscription contains a single watcher (i.e., the first to send a subscription request). By the time the session finishes and therefore the subscription is terminated, all the watchers are active. The watcher list in common SUBSCRIBE and NOTIFY messages is full-state (i.e., contain all the watchers associated with the common subscription). When the watcher domain refreshes a common subscription, half of the total number of watchers are active on average.

- Common subscriptions to federated-presence event: $pres*(sub+sok+2*wlenv+not+doc+nok+2*mpb+\frac{pfwat*wat}{100}*(mpb+pfdoc))*(\frac{slife}{sref}+wat)+pres*wlit*wat*(\frac{slife}{sref}+wat+2)+(pch*slife-2)*pres*(not+doc+nok)$

This kind of event subscription operates in the same way than presence subscriptions excepting that the privacy rules are sent in any notification. Thus, all the assumptions done above are valid for this event too.

5.1.5 Estimation and Analysis of Presence Traffic

We estimate the presence traffic exchanged between two federated domains when CN, VS, and CS are applied for reducing network traffic. We use the mathematical formulas given in Sections 5.1.2.2, 5.1.3.2, and 5.1.4.2 to assess the performance of these strategies. These formulas give us each strategy's traffic in bytes, and are based on the variables and constants in Table 5.1. We take the value in Table 5.1 from a presence scenario described in [193]. This scenario has 40,000 presentities (*pres* variable) that are watched by 20 watchers (*wat* variable). Presentities change their presence at a rate of three changes per hour (*pch* variable) on average. As the authors of [193] state, these assumptions are pretty moderate compared to the statistics about real systems that they provide. As described previously, other presence scenarios are considered in [193], which represent higher levels of federation. We assume the afore-mentioned scenario becuase it represents a moderate level of federation, and hence the result of applying traffic optimizations to it can be extrapolated to larger systems. In the research report [207], we show that the qualitative results of each strategy are the same on all the studied presence scenarios. It is important to highlight that, although we find the above-mentioned assumptions reasonable, we do not set out to analyze specific presence systems or make rigorous general models of the behavior of presence systems. This would be an arduous, if not impossible, task since presence users do not follow any particular statistical pattern that could be applied to all of them.

Figure 5.5 shows the traffic generated with the dialog optimization strategy (RLS), the case of no optimizations (BASIC), and the various configurations in CN (CN1-CN2), CS (CS1-CS3), and VS (VS1-VS6) that are shown in Tables 5.2, 5.4, and 5.3, respectively. In the case of the dialog optimization strategy, we assume that the RLS sends partial-state RMLI documents for notifying watchers of the presentities' presence. Conditional notifications are not applied to any strategy; this optimization is analyzed in Section 5.1.5.1. Figure 5.5 shows that dialog optimization is by far the least scalable strategy. Whenever a presence change occurs, the federated RLS notifies a different RLMI document to each of the presentity's watchers. However, the other

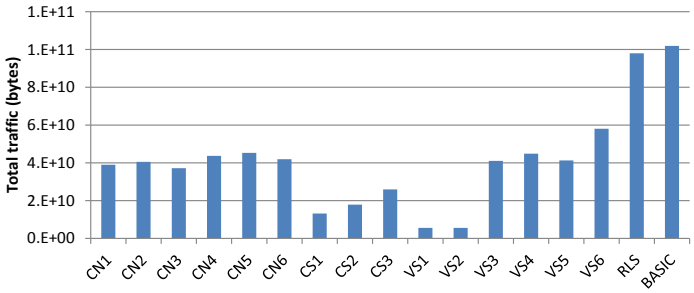


Figure 5.5: Presence traffic of all the strategies

three strategies reduce the number of notifications per presence change to one in the case of CN and CS, and to the number of different views in the case of VS. In addition, these strategies do not introduce overhead for RLMI-related data in the notifications.

Cases CN1-CN3, CS1 and VS1-VS3 deal with domain-based privacy filtering. This means that all the watchers in a domain that are subscribed to a particular presentity have the same privacy rules, and can therefore see exactly the same presence documents. In this case, when VS is applied with partial or full trust (cases VS1 and VS2) there is a single subscription between the watcher domain and the presentity, as happens with CS. These two cases do not inform the presentity's PS of the watchers who are currently watching the presentity; the PS only knows one of them (i.e., the first one to subscribe). However CN and CS are designed to always provide the presentity's domain with full knowledge about watchers. If such knowledge is not needed by the watched domain, the VS strategy with partial or full trust is strongly recommended. Both cases generate the same amount of traffic since there is a single view per domain. Then, VS saves 58% and 85% of the traffic in CS and CN, respectively. However, if the presentity's PS needs to know the full set of watchers, the VS strategy is only permitted in the case of minimal trust (case VS3). In this case, the VS traffic is multiplied by more than sevenfold and exceeds the CN traffic. The most efficient strategy is CS, which reduces VS3 traffic by 68% and CN traffic by 65-68%. CN reduces VS3 traffic by between 1,2% and 9,4% depending on its configuration.

Cases CN4-CN6, CS2, CS3, and VS4-VS6 deal with watcher-based privacy filtering, which is applied to *pfwat*% of watchers. CS is doubtless the most efficient strategy at reducing presence traffic between domains. Case CS2 reduces the VS traffic by 60%, 57% and 69% with full, partial, and minimal trust, respectively. Moreover, this case

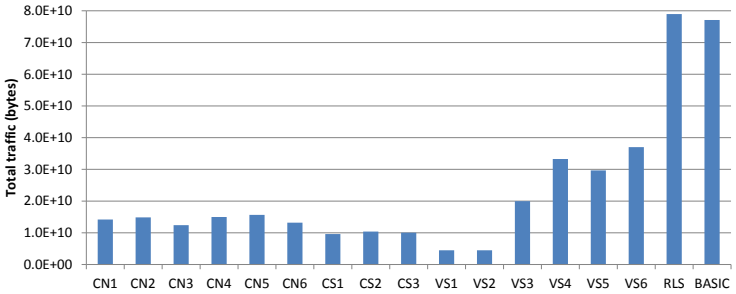


Figure 5.6: Presence traffic of all the strategies when NO is applied

reduces the CN traffic by around 57-60% based on the configuration of CN. Case CS3 (i.e., federated-presence subscriptions) involves more traffic than CS2 because it sends both presence information and privacy rules in any NOTIFY message. Section 5.1.5.1 shows how conditional notifications improve the FCS efficiency. Case CN4 introduces fewer bytes than case CN5 because the former adds the watcher list to the NOTIFY messages, rather than using the winfo event package. Section 5.1.5.4 discusses these two methods in depth. Case CN6 is obviously the most efficient configuration for CN since no traffic is involved in obtaining the watcher lists. Despite that Figure 5.5 shows that CS is very efficient, our assumptions to calculate its traffic are pessimistic. Whenever a watcher becomes active, a SUBSCRIBE message with a watcher list containing the new watcher is sent. However, the watcher domain may have a throttling mechanism for aggregating new watchers' requests in a single SUBSCRIBE message, which may be specially useful in rush hours. Moreover, we count the SUBSCRIBE messages for refreshing and adding new watchers separately. However, in a real system, the SUBSCRIBE messages for new watchers would also refresh the subscriptions by restarting their expiration timer. Thus, the watcher domain would need to send fewer re-SUBSCRIBE messages to refresh the subscriptions.

The following sections analyze a number of parameters that affect the performance of CN, VS, and CS. Section 5.1.5.1 analyzes the effect of conditional notifications on these strategies. Section 5.1.5.2 studies the performance of VS. Section 5.1.5.3 studies how the strategies are affected by privacy filtering, specially with privacy-filters subscriptions. Finally, Section 5.1.5.4 tackles the management of watcher lists in CN.

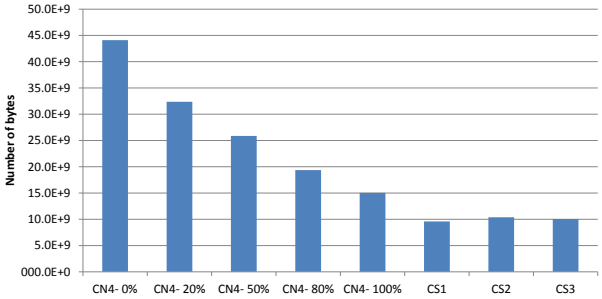


Figure 5.7: Presence traffic of CN when different percentages of watchers support NO

5.1.5.1 Conditional Notifications

Conditional notifications [147] suppress the unnecessary notifications sent in response to the SUBSCRIBE requests for keeping the subscriptions alive. The traffic estimation in [193] show that this technique is very efficient at reducing subscription traffic. Henceforth, we refer to this technique as NOTIFY optimization (NO) for differentiating its acronym from that of the Common Notify strategy (i.e., CN). Figure 5.6 shows the result of applying this technique to the traffic in Figure 5.5. The presence traffic in both figures follows the same pattern. Using domain-based filtering, the best strategy is VS (with partial or full trust), followed by CS. However, in the case of watcher-based filtering, the most efficient strategy is CS, followed by CN. Dialog optimization (RLS case) is the least efficient strategy; this even generates more traffic than the BASIC case in which no strategies for reducing traffic are applied. This is because we assume that, in the BASIC case, either all the watchers implement NO or all the subscription requests go through the watcher domain's PS that implements NO. Ensuring that all the watcher apply NO may however be more difficult to ensure in real scenarios. With FCS (CS3 case), the NOTIFY messages due to refreshes carry privacy filters in addition to presence documents, and hence NO saves around 61% of its traffic. NO is very effective in CN and VS with minimal trust because these strategies handle a different presence subscription to a particular presentity for each watcher. Therefore, saving the notifications for refreshing all the subscriptions reduces inter-domain traffic significantly. With CN, when the watcher domain's PS does not work as a proxy, this reduction is not achieved unless all the watchers implement NO, which may not a realistic situation as mentioned previously. The federated RLS, CS, and VS strategies however always need

the watcher domain's PS to work as a proxy, which allows implementing NO between this PS and the notifier domain's PS. With CN, the number of watchers that apply NO should therefore be carefully considered in their presence subscriptions. Figure 5.7 shows the CN4 traffic when different percentages of watchers implement NO. As expected, CN is only nearly as efficient as CS if all the watchers support NO. For example, if we take an optimistic case in which 80% of watchers implement NO, it can be seen that the CN traffic is almost twice that of CS.

5.1.5.2 View Sharing

Figure 5.5 shows that VS is not as efficient as expected. Case VS6 implements minimal trust which offers full knowledge about watchers to the notifier domain. This case is the least efficient; it involves more bytes than any configuration of CN. Like CN, minimal trust maintains one presence subscription per watcher, per presentity. However, case VS6 introduces more overload because (1) a single presence change may result in notifying more than one subscription and (2) the NOTIFY messages for refreshes contain ACL documents. If the notifier domain does not need to be aware of the watchers subscribed to the presentity, it is preferable to apply partial trust (VS5) to full trust (VS4) because the former notifies smaller ACL documents. Case VS5 saves approximately 8% of the VS4 traffic. Although Figure 5.5 shows that the VS traffic with partial trust is hardly lower than the CN traffic, a number of issues concerning the performance of VS should be considered. Its performance may be seriously affected by changes in privacy filters. For our calculations, we assume that privacy filters do not change once they are created. This assumption is optimistic since changes in privacy filters may involve updating watcher lists, removing the subscriptions associated with views that have become invalid, and creating subscriptions for new views. In addition, we assume that all the possible watchers are already known by the presentity's PS and, therefore, included in ACLs. This assumption is also optimistic in partial and full trust since it avoids SUBSCRIBE messages that would be necessary for obtaining the views that unknown watchers are authorized to see.

With partial trust (VS5) or full trust (VS4), the federated domains must handle as many presence subscriptions as the number of views associated with watchers. Thus, these techniques generate more traffic as the number of views increases. With minimal trust (VS3 and VS6), although the number of subscriptions is always equal to the

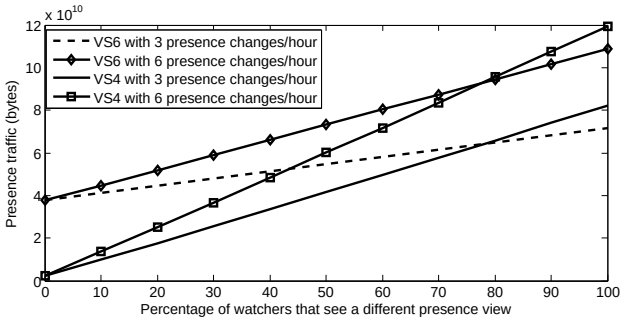


Figure 5.8: Effect of the number of views on VS traffic

number of watchers, the number of views does affect the amount of NOTIFY messages for changes to send. The number of notifications per presence change is equal to the number of views affected by the change. We assume that each presence change affects half of the number of views. The variable *pfwat* means the percentage of watchers to which privacy filtering is applied and is set to 50% for our calculations. Each of these watchers has a different privacy filter and therefore watches a different view of the presentity's presence (through a different presence subscription). Figure 5.8 shows how VS with full trust (VS4) and minimal trust (VS6) are affected by the increase in views. Partial trust is omitted because its performance on view growth is very similar to that of full trust. The increase in views obviously has a more harmful effect on full trust, and when 80% of watchers have a different view associated, the application of minimal trust is more efficient than partial trust. The number of presence changes significantly increases the traffic of both cases.

5.1.5.3 Privacy Filtering

Figure 5.9 shows the effect of the increase in views on VS4, CN4 and CS2 when three and six presence changes occur per hour. The number of views for a presentiy is equal to the number of privacy filters created by this presentity. It can be seen that VS involves more traffic than the other techniques as the number of views increases. The average number of presence changes also drastically affects VS. The CS and CN traffic does not significantly increase as the number of filters grows, since these strategies handle a constant number of presence subscriptions regardless of the number of privacy filters. These strategies do not worsen as the number of presence changes increases

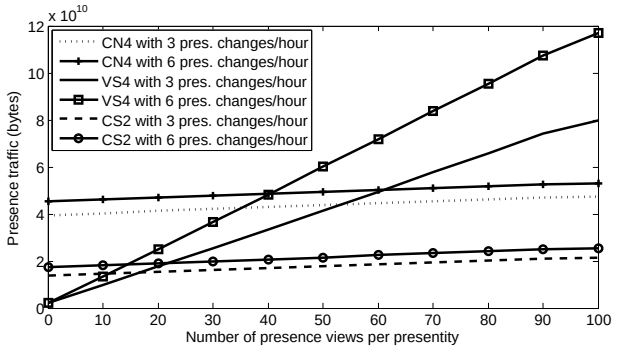


Figure 5.9: Traffic of CN, VS, and CS over the number of views

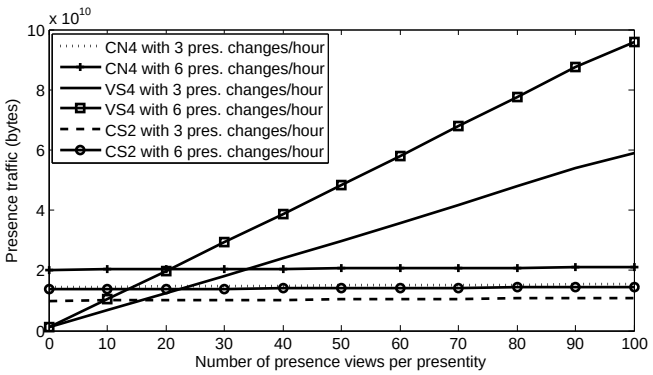


Figure 5.10: Traffic of VS, CN, and CS combined with NO over the number of views

because the number of presence notifications does not depend on the number of views (i.e., notifications are only sent through one presence subscription). Figure 5.10 is the result of applying NO to the traffic in Figure 5.9. It can be seen that the difference between VS and the other two strategies becomes even greater when this optimization is applied.

Figure 5.11 shows the thresholds from which the case VS4 starts to generate more bytes than CN4 and CS2 based on the percentage of the presentity's watchers that watch a different view. Stated differently, this figure shows the maximum percentage of watchers watching a different view per presentity that is advisable to use VS (i.e., VS generates fewer bytes than CN and CS up to that percentage). Since these percentages refer to a total of 20 watchers, the numbers on the columns denote the advisable maximum number of views (i.e., privacy filters). VS is only preferable to CS when

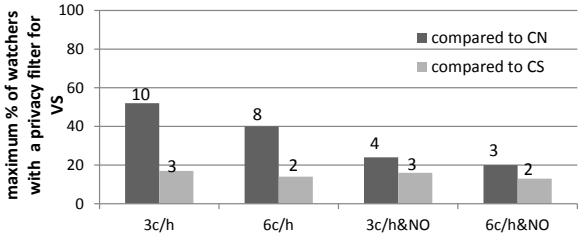


Figure 5.11: Maximum percentage of watchers seeing a different presence view up to which VS4 is recommended

there are almost no privacy filters per presentity (three or two in case of three and six presence changes/hour, respectively). The choice between VS and CN very much depends on whether the latter applies NO or not. Without this optimization, VS is more efficient than CN up to ten and eight different views in the case of three and six changes/hour, respectively.

CN and CS need privacy-filters subscriptions to get the prensentities' privacy filters, which involves a considerable amount of traffic. One may notice this fact in Figure 5.5, specially in CN4 and CN1 cases. The single difference between these cases is that CN4 creates privacy-filters subscriptions; this is the reason why CN4 case generates more traffic. Since the CN and CS strategies handle privacy-filters subscriptions in the same manner, we take CN as a reference for analyzing this kind of traffic. Figure 5.12 shows the CN4 traffic as the number of privacy filters per presentity increases. This traffic is split into presence-related and filters-related traffic. The maximum number of filters per presentity is 20 because the presentities have 20 watchers. This figure also shows the CN4 traffic when NO is applied to privacy-filters subscriptions. In the best case, no privacy filters are created, and therefore there is only one default filter per presentity. In this ideal case, the traffic due to privacy-filters subscriptions constitutes 2.5% or, if NO is applied, 0.9% of the total traffic. In the worst case, the presentities create a specific privacy filter for each watcher (i.e., 20 filters per presentity) and the privacy-filters subscriptions' traffic represents 18.6% or if NO is applied, 3.1%, of the total traffic.

Presence event subscriptions involve less traffic in CS than CN, and hence the traffic related to privacy filters accounts for a higher proportion of traffic in CS. Privacy-filters-related traffic in CS (case CS2) accounts between 7.3% in the best case and 40.8% in

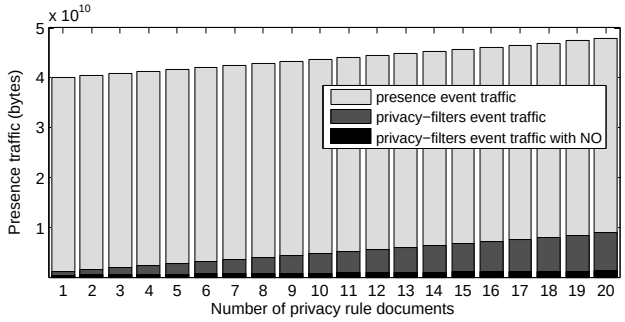


Figure 5.12: Privacy-filters and presence event traffic of CN

the worst case of the total traffic. When NO is applied, these percentages are reduced to 2.8% in the best case and 8.9% in the worst case.

The following sections discuss the effect of changes in privacy filters on privacy-filters subscriptions and the use of watchers lists in these subscriptions.

Changes in privacy rules We assume that presentities do not change their privacy filters after creating them. Although we believe that this is a reasonable assumption for many scenarios, it may not be true in others. Whenever a presentity changes a privacy rule, its PS notifies the subscriber domain of the privacy filters that are affected by the change. Figure 5.13 shows the effect of changes in privacy filters on privacy-filters subscriptions' traffic. Each presentity is subscribed by 20 watchers and therefore can create up to 20 different privacy filters. We consider the minimum case (one privacy filter per presentity), the average case (10 privacy filters per presentity) and the maximum case (20 privacy filters per presentity). Figure 5.13 shows that the increase in changes in privacy filters worsends the efficiency of privacy-filters subscriptions. Figure 5.14 shows the traffic related to privacy-filters and presence events in CN4 when each presentity creates 10 privacy filters. The sum of these two types of traffic is the total traffic in CN4. It can be seen that the traffic related to privacy filters becomes a greater part of the total traffic as the number of changes increases. Both Figures 5.13 and 5.14 assume that NO is applied to all the subscriptions.

List-based privacy-filters subscriptions As described in Section 5.1.2, there are two methods for subscribing to a presentity's privacy-filters event: list-based and basic subscriptions. The former includes the list of subscribed watchers in SUBSCRIBE

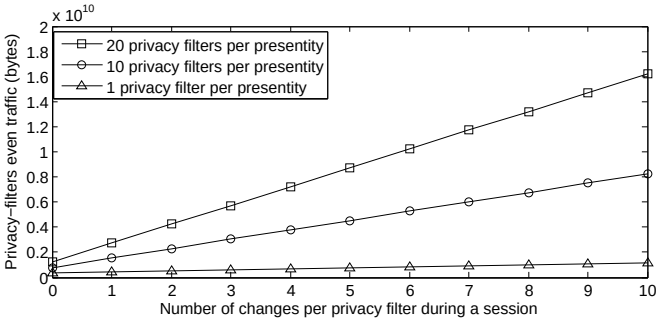


Figure 5.13: Privacy-filters event traffic over the number of changes in privacy filters

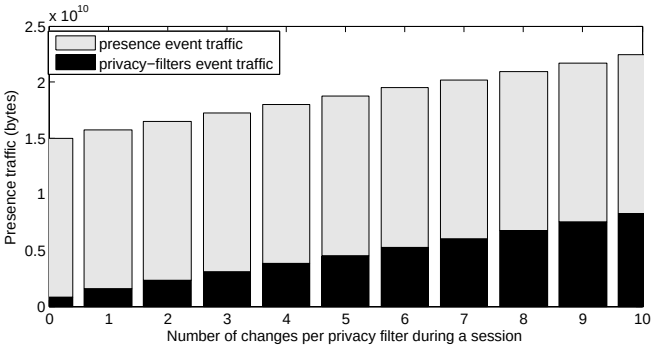


Figure 5.14: CN traffic over the number of changes in each privacy filter

messages for allowing the notifier to send only the privacy filters associated with the watchers on the list. The idea behind list-based subscriptions is to provide watcher domains with only the privacy rules that are actually useful. Basic subscriptions do not include a watcher list but just use standard SUBSCRIBE messages. Thus, the notifier sends all the privacy filters associated with any watcher within the subscriber domain whether or not the watcher is active (i.e., watching the presentity). Figure 5.15 shows the traffic for these two types of subscription with and without NO. In the case of list-based subscriptions, NO cannot save the NOTIFY messages in response to the re-SUBSCRIBE messages aimed at downloading new watchers' privacy filters. Thus, the subscriber must tell the notifier which re-SUBSCRIBE messages are sent for downloading privacy filters and which are just for refreshing the subscription. This can be easily done by removing the "Suppress-if-match" header from the SUBSCRIBE messages for downloading privacy filters. Section 2.7.4 outlines how conditional notifi-

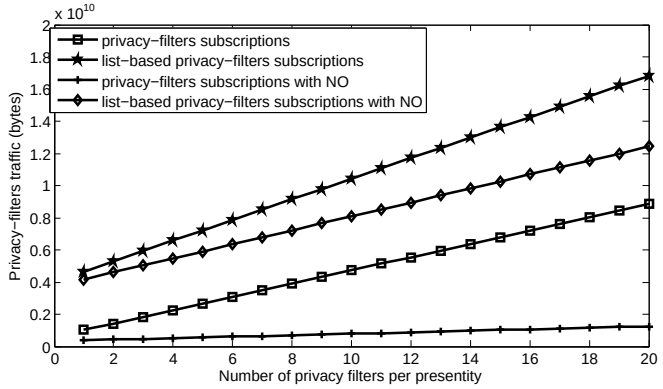


Figure 5.15: Privacy-filters event traffic with the basic and list-based methods over the number of privacy filters

cations work. Figure 5.15 shows that list-based subscriptions involve more traffic than the basic method, even with NO. This is because whenever a watcher becomes active, a re-SUBSCRIBE message is sent to download its privacy filter. However, with the basic method, the first SUBSCRIBE message to a presentity's privacy-filters event triggers a notification of all of the presentity's privacy filters. After that, only re-SUBSCRIBE messages for refreshes are sent. The drawback of this strategy is that the watcher domain may waste bandwidth and memory resources for privacy filters that will never be used.

5.1.5.4 Watcher List in CN

With CN, watchers subscribe to the presentities directly, and hence the watcher domain must ascertain which watchers are currently subscribed when it receives a common NOTIFY request. There are three possibilities, as described in Section 5.1.2.1: (1) maintaining the watcher list on the subscriber side PS; (2) including the watcher list in notifies (hereinafter, WLN); and (3) obtaining the list by subscribing to the presentities' winfo event. Concerning WLN, the authors of [194] state: "This has a disadvantage when the number of watchers from domain B is very large, every NOTIFY message increases in size proportionately". However, they do not mention anything about the performance of winfo subscriptions. Thus, readers may gain the initial impression that the WLN efficiency deteriorates as the number of watchers increases while winfo subscriptions do not. Figure 5.16 shows that such an impression is not true. As the

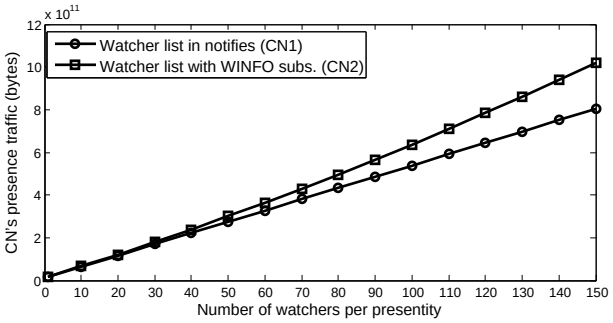


Figure 5.16: CN traffic with WLN and winfo subscriptions over the number of watchers

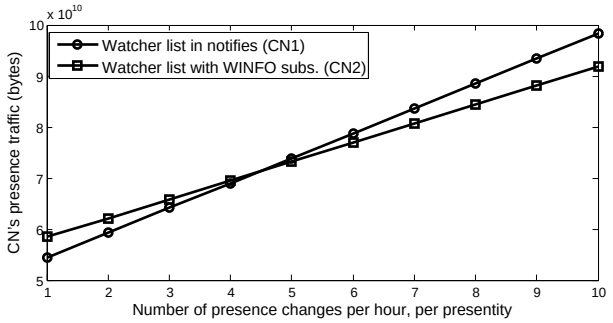


Figure 5.17: CN traffic with WLN and winfo subscriptions over the number of presence changes

number of watchers grows, winfo subscriptions generate more traffic than WLN. This is because whenever the watcher domain resubscribes to a presentity's winfo event, the complete list of watchers is notified. The difference between the two proposals is more noticeable with large watcher lists. For example, with 150 watchers per presentity, winfo subscriptions generate more than 120% of the WLN traffic. The performance of WLN is more affected by the number of presence changes than the number of watchers, as shown in Figure 5.17. This is due to the fact that any time a presence change occurs, the presentity's watcher list is included in the notification.

Figure 5.18 gives the maximum average number of presence changes up to which WLN generates less traffic; thus, it is preferable to include the watcher list in the NOTIFY messages (CN1) rather than subscribing to the winfo events (CN2) in CN. The maximum number of changes depends on the number of watchers per presentity. As the number of watchers increases, winfo subscriptions involve more bytes, and hence

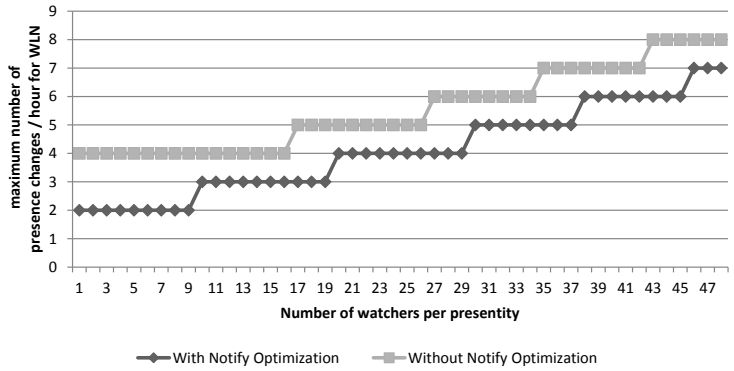


Figure 5.18: Recommended maximum number of presence changes per hour, per presentity, for using WLN instead of winfo subscriptions

WLN is preferable up to a higher number of presence changes. Without NO, when four presence changes occur each hour on average, WLN generates less traffic regardless of the number of watchers. If the presentities have a maximum of 16 watchers and four presence changes occur per hour on average, winfo subscriptions are more suitable. As the number of watcher increases, winfo subscriptions only perform better than WLN if the presentities are highly active. For instance, if there are 40 watchers per presentity, winfo subscriptions are more efficient as long as the average number of changes per hour is higher than seven. When NO is used, winfo subscriptions generate less traffic, and hence the average number of presence changes up to which WLN is recommended decreases.

5.2 CS and FCS Enhancement for Minimizing the Disclosure of Privacy Rules

Section 5.1.5 shows that Common Subscribe (CS) and Federated Common Subscribe (FCS) greatly help in decreasing the amount of bytes exchanged between two federated presence domains. However, these strategies require a greater level of trust between the federated domains since privacy filtering has to be performed by the subscriber side domain. Every presence change is notified by means of a common NOTIFY message, which contains the presentity's complete presence information. The subscriber domain needs therefore to generate the presence document that each watcher is allowed to see.

This is done by filtering the presence information with the presentity's privacy rules. This section discusses some slight variations of the normal operation of CS and FCS that minimize the exchange of privacy rules. A privacy rule [204] states the presence that one or more watchers are authorized to see. When a watcher is not active (i.e., is not actually subscribed and watching the presentity), no presence notifications are sent to the watcher. A privacy rule is, therefore, useless for the subscriber side domain while all of its associated watchers remain inactive. In this case, we say that the privacy rule is inactive. The proposed variations of CS and FCS only let the subscriber side PS obtain the privacy rules that are active. Our purpose is twofold: on one hand, these variations enhance the privacy of presentities to some extent, since inactive privacy rules are not disclosed; on the other hand, they save notifying unnecessary privacy rules, which probably reduces the overall traffic generated by FCS and CS. Below, the proposed variations are outlined.

With CS, when a domain subscribes to a presentity's presence event, afterwards it also subscribes to the presentity's privacy-filters event in order to obtain the presentity's privacy rules (see Section 5.1.4). This privacy-filters subscription notifies all the presentity's privacy rules for any watcher within the requester domain, regardless of whether the watchers are active. However, the presentity's PS always has knowledge about the active privacy rules, since it knows the watchers that are watching the presentity through the common subscription. This is because CS requires the subscriber side PS to include the watcher list in every presence subscription request. Therefore, we propose that the presentity's PS only notify the active privacy rules as a result of any privacy-filters subscription request. Henceforth, we refer to this as active-privacy-filters subscriptions. Furthermore, the PS should proactively notify a privacy rule whenever it becomes active. This happens when one of the privacy rule's watchers becomes active.

With FCS, the subscriber side PS does not subscribe to presence events but rather to federated-presence events, as described in Section 5.1.4. This means that any federated-presence notification sent as a result of a subscription request contains both the presentity's presence information and privacy rules. Like CS, the presentity's PS is aware of the active watchers because the requester domain attaches the watcher list to any subscription request. Thus, we propose that the presentity's PS only include the active privacy rules into any notification. Moreover, whenever a privacy rule becomes active, the PS should send a federated-presence notification containing that rule.

As shown in Section 5.1.5.3, the exchange of privacy rules may involve a considerable amount of traffic in CS and FCS. In the case of CS, it may account for approximately 40% of the total traffic when the presentities create a different privacy rule for each watcher. The following sections analyze privacy-filters-related traffic in more detail than Section 5.1, which is valuable to understand the variables that affect this traffic and therefore to optimize it. Section 5.2.1 presents the formulas for calculating the number of bytes that two federated domains need to exchange because of privacy rules. Section 5.2.2 estimates the amount of traffic related to privacy rules in a presence scenario.

5.2.1 Calculation of Traffic Related to Privacy Rules

This section gives the formulas for calculating the number of bytes involved in exchanging privacy rules, in both FCS and CS, with and without the variations described previously. We take the same approach as that in Section 5.1.1 for the presented mathematical formulas. The total traffic related to privacy rules during a presence session is split into three categories: initial, termination and steady-state traffic. Initial and termination traffic are due to the establishment and termination, respectively, of the presentities privacy-filters or FCS subscriptions. Steady-state traffic is all the traffic exchanged in the time elapsed between the initial subscription and the termination of the subscription. Since we assume that privacy rules do not change over time (and already exist when the session begins), steady state traffic only contains subscription refreshes. We assume the federation scenario described in Section 5.1.1, which is summarized in Table 5.5 for convenience.

This scenario consists of a total of 40,000 presentities (*pres* variable) that are watched by 20 watchers (*wat* variable). The variables *sub*, *sok*, *not* and *nok* are the sizes of subscription-related SIP messages, and their values have been taken from [110]. The average session time is 8 hours (*slife* variable). The subscription lifetime is 1 hour (*sref* variable), which is the default value for presence subscriptions [120]. The other variables in Table 5.5 are related to privacy filters. We assume an average number of views per presentity, which is given by the *views* variable. As described in Section 5.1.3, a view on a presentity's presence information is the subset of information that a set of watchers are authorized to see. Thus, a view is determined by a privacy rule, and henceforth we use the terms "presence view" and "privacy rule" indistinctly. We assume that

the number of watchers associated to each view is uniformly distributed and therefore calculated as *wat/views*. The constants *rule*, *rid* and *rsetb* are used to calculate the size of privacy rule documents and are deduced from the examples in [177] and [204]. Basically, a privacy rule document is a set of rules, each containing a list of authorized watchers and a set of elements that state what presence they can see. Appendix D shows an example privacy rule document. Each authorized watcher is indicated by an XML element *identity* (*rid* variable) and the granted presence information is determined by an XML element *transformations* (*rule* variable). The variable *rsetb* is the size of the XML data that wraps the rule set. We calculate the average size of the presentities' privacy rule document as $prds_i = rsetb + i * (rule + rid * (wat/views))$, where i is the number of privacy rules included in the document, and hence $1 \leq i \leq views$ is satisfied.

Below we present the formulas for calculating the number of bytes involved in the exchange of privacy rules, both in FCS and in privacy-filters subscriptions. In the normal mode of operation of CS and FCS, privacy rule documents always contain all the presentity's privacy rules regardless of whether the rules are active. Thus, the average size of these documents is $prds_{views}$. We assume that when the watcher domain refreshes a privacy-filters or federated-presence subscription, half the total number of privacy rules on average are active. Furthermore, by the time the session finishes, all the privacy rules are active. Notifications due to refreshes are full state and therefore contain all the active privacy rules.

5.2.1.1 Privacy-Filters Subscriptions in CS

The *initial*, *refreshes* and *termination* variables below count the initial, steady-state, and termination traffic, respectively.

- **Normal mode of operation**

$$initial = np * (sub + sok + not + prds_{views} + nok)$$
$$refreshes = (stime/sref - 1) * np * (sub + sok + not + prds_{views} + nok)$$
$$termination = np * (sub + sok + not + prds_{views} + nok)$$

Name	Description	Default Default average value
slife	Subscription lifetime	8 hours
sref	Subscription refresh interval / hour	1
wat	Total number of watchers per presentity within the subscriber domain	20
pres	Total number of presentities within the notifier domain	40,000
views	Number of views per presentity ($1 \leq views \leq wat$)	*variable*
aviews	Number of active views ($1 \leq aviews \leq views$)	*variable*
sub	SUBSCRIBE message size (bytes)	450 bytes
sok	200 OK for SUBSCRIBE message size	370 bytes
not	NOTIFY message size (bytes per presentity)	500 bytes
nok	200 OK for NOTIFY message size	370 bytes
rid	Average size of a rule's identity element in a privacy rule document	50 bytes
rule	Average size of a rule in a privacy rule document	600 bytes
rsetb	Average size of the ruleset boundary in a privacy rule document	200 bytes

Table 5.5: Variables for estimating the traffic related to privacy rules

- **Active-privacy-filters subscriptions**

  The *activations* variable calculates the traffic due to privacy rules that become active. These rules are notified through partial-state privacy rule documents that only contain the privacy rule that has become active.

$$initial = np * (sub + sok + not + prds_1 + nok)$$
$$refreshes = (stime/sref - 1) * np * (sub + sok + not + prds_{aviews/2} + nok)$$
$$termination = np * (sub + sok + not + prds_{aviews} + nok)$$
$$activations = np * (aviews - 1) * (not + prds_1 + nok)$$

5.2.1.2 Traffic Related to Privacy Rules in FCS

We only take into account the privacy rules contained in federated-presence notifications. Subscription request and response messages are not exclusively aimed at obtaining privacy rules, but also presence information. Thus, these messages are not considered as privacy-filters-related traffic. FCS generates extra subscription requests

for letting the presentity's PS know about the watchers that are actually subscribed; this involves notifying the presentity's privacy rules. The total number of federated-presence subscription requests that are sent to a presentity because of watchers that become active is $aviews * (wat/views)$. The *newwatchers* variable below calculates this traffic, which includes the initial traffic. The *refreshes* and *termination* variables count the steady state and termination traffic, respectively.

- **Normal mode of operation**

  $refreshes = np * ((stime/sref - 1)) * prds_{views}$
  $newwatchers = np * aviews * (wat/views) * prds_{views}$
  $termination = np * prds_{views}$

- **Awareness of the active privacy rules**
  The *activations* variable below counts the traffic due to the privacy rules that become active. Contrary to CS (Section 5.2.1.1), FCS does not establish a privacy-filters subscription to the presentity. Instead, the privacy rules, together with the presence information, constitute the presentity's federated-presence event. The presentity's PS is aware of when a watcher is subscribed to the presentity because it is added to the common subscription's watcher list by means of a re-subscription request. The first watcher among those associated with a privacy rule to be subscribed leads, for first time, to the inclusion of the privacy rule in a notification. Notifications triggered from subscription requests must be full-state [110] (i.e., all the resource state information is included). Thus, federated-presence notifications for subscription refreshes must include all the privacy rules that are active, as well as the presentity's presence document and watcher list. This is the reason for the summation in the *activations* variable. This variable takes account of the privacy rule notifications that are triggered the first time a watcher of each privacy rule becomes subscribed. However, the *newwatchers* variable counts the notifications for the remaining watchers that become active, and hence the first notification of each privacy rule is subtracted.

$$refreshes = (stime/slife - 1) * np * (prds_{aviews/2})$$
$$newwatchers = np * (aviews * (wat/views) - aviews) * prds_{aviews/2}$$
$$termination = pres * (prds_{aviews})$$
$$activations = np * \sum_{i=1}^{aviews} prds_i = np * (views * rsetb + (rule + (wat/views) * rid) * (aviews * (aviews + 1)/2))$$

5.2.2 Analysis of Traffic Related to Privacy Rules

We use the mathematical formulas presented in Section 5.2.1 for estimating the traffic that is due to privacy rules during a presence session. Figure 5.19 shows this traffic as the number of privacy rules increases (*views* variable in Table 5.5). We assume that all the privacy rules are active, and the *aviews* variable in Table 5.5 is therefore equal to *views*. It can be seen that even when all the rules are active, the proposed variations ("CS active-privacy-filters subscription traffic" and "FCS traffic related to active privacy rules" in Figure 5.19) save traffic. This is because they avoid notifying all the privacy rules once the subscription has been established. Instead, privacy rules are notified only when they become active. Figure 5.20 shows the amount of traffic related to privacy rules when there are rules that never become active. We assume that presentities create 5 privacy rules on average. Thus, up to 4 privacy rules may be rendered useless if their associated watchers are not subscribed during the session time. Note that there is at least one active rule per presentity; this is the one containing the first watcher that subscribes to the presentity and therefore leads to establish the presentity's common subscription. One may observe that the number of rules that are active does not affect privacy-filters subscriptions, since these subscriptions always notify the complete set of privacy rules. However, it does affect FCS. If a rule does not become active, it is because none of its associated watchers has subscribed to the presentity. This saves the subscriber side PS from refreshing the federated-presence subscription to add these watchers to the common subscription, as well as the resulting notifications. The proposed variations perform better as the number of inactive views increases. The FCS variation saves from 45% (when all the rules are active) up to 81% (when only one rule is active) of the traffic related to privacy rules in the regular FCS.

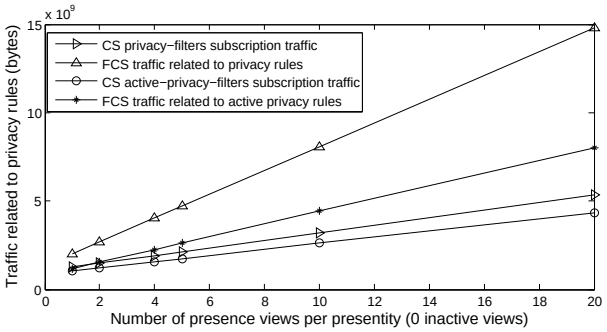


Figure 5.19: Traffic related to privacy rules without inactive rules

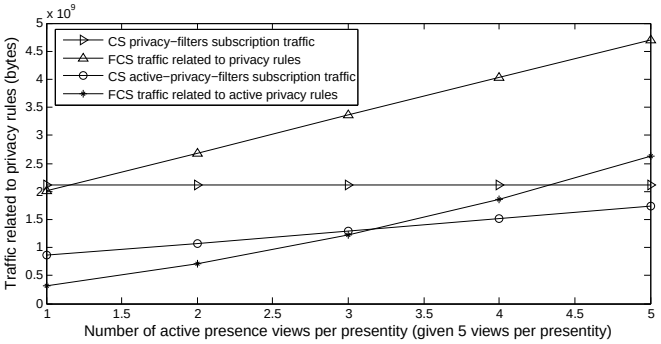


Figure 5.20: Traffic related to privacy rules with inactive rules

Active-privacy-filters subscriptions save from 18% (when all the rules are active) up to 60% (when only one rule is active) of the regular privacy-filters subscription traffic.

Conditional notifications [147] suppress the notifications that are sent in response to subscription refreshes when no changes have occurred from the last notification. Section 5.1.5.3 shows that this optimization can considerably reduce the amount of traffic related to privacy rules. In privacy-filters subscriptions, since we assume that the presentities do not modify their privacy rules, this optimization saves all the notifications except the first that notifies the complete set of privacy rules. In the case of FCS, however, the watcher domain cannot apply conditional notification to the SUBSCRIBE messages that are sent when a new watcher wishes to subscribe to a presentity. This is because the presentity's PS has to acknowledge the watcher list included in this message by sending a NOTIFY message in response to the request. Figure 5.21 shows the effect of conditional notifications on the traffic related to privacy rules. We assume that the presentities create an average of 5 privacy rules. Compared to Figure 5.20, it is clear that conditional notifications save a considerable amount of traffic. In the case

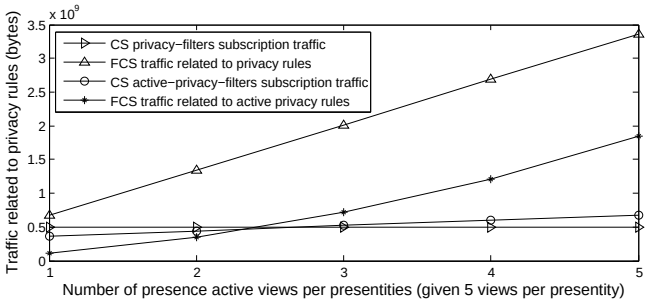


Figure 5.21: Traffic related to privacy rules with inactive rules and conditional notifications

of FCS and its proposed variation, conditional notifications reduce their traffic to the same degree. The reduction of traffic achieved by the proposed variation is therefore the same as that without conditional notifications (from 45% up to 82%). In the case of privacy-filters subscriptions, almost all the traffic of these subscriptions is due to subscription refreshes. Thus, conditional notifications reduce this traffic to such an extent that the regular privacy-filters subscription outperforms its proposed variation when many privacy rules are active. In particular, Figure 5.21 shows that active-privacy-filters subscriptions generate more traffic than privacy-filters subscriptions when more than half the privacy rules become active. This is because active-privacy-filters subscriptions trigger a notification anytime a privacy rule becomes active. On the other hand, privacy-filters subscriptions notify all the privacy rules at once and no more notifications are sent because conditional notifications are applied.

5.3 Conclusions

We described a number of alternatives for reducing inter-domain presence traffic and estimated how efficient these strategies are. Our motivation comes from an IETF Internet-Draft [193] that mainly studies how far the technique dialog optimization reduces presence traffic. This study shows that this technique is not efficient enough and highligthts the need to further work on strategies to reduce presence traffic with the aim of making presence services scalable.

We analyzed and studied the performance of two strategies that were proposed as IETF Internet-Drafts: CN and VS. Although these Internet-Drafts have been discontinued in the IETF, studying CN and VS is valuable to understand how complicated

	knowledge of subscribed watchers IS required	knowledge of subscribed watchers IS NOT required
watcher-based privacy filtering	CS (VS:57-69% CN:57-60%)	CS (VS:57-69% CN:57-60%)
domain-based privacy filtering	CS (VS:68% CN:65-68%)	VS with partial or full trust (CS:58% CN:85%)

Table 5.6: Most efficient strategy (the percentage of the other strategies' traffic saved in parenthesis)

the reduction of presence traffic is. Moreover, there is no more proposals in the SIMPLE framework for reducing inter-domain presence traffic. Thus, we proposed and studied a novel strategy called CS. We defined the formulas that calculate the number of bytes per session that are exchanged between two federated domains for the three aforementioned strategies. We also described the assumptions about the operation of each technique that we had to make in order to estimate presence traffic. These assumptions give an idea of the complexity of each strategy and may serve as a guideline for other researchers on the subject. We carefully considered the parameters that affect the efficiency of each strategy, made a comparison between the strategies and gave some indicators of the suitability of each strategy based on these parameters. Table 5.6 shows what of the three strategies is the most efficient at reducing presence traffic based on the reported results. This table is a function of the type of privacy filtering and whether or not the notifier domain needs to know the watchers actually subscribed to any presentity. This table also shows the percentage of the other strategies' traffic saved by the most efficient strategy in parenthesis. CS is considerably more efficient at reducing presence traffic than VS and CN. The only exception happens when the presentities set the same privacy rules for all the watchers in a particular domain (i.e., domain-based privacy filtering) and the presentities' domain does require to know the list of subscribed watchers. In this improbable case, it is preferable to apply VS with partial or full trust. However, when the presentities' domain needs to know the subscribed watchers, only VS with minimal trust is applicable and it always generates more traffic than CS and CN.

In the estimation of presence traffic, privacy filtering is an impacting factor to take into account. We assumed that each presentity applies a privacy filter to 50% of its watchers; Table 5.6 relies on this assumption. However, we also analyzed what

	CS	CN	VS
CS preferable to		always	$views \geq 16\%\ watchers$
CN preferable to	never		$views \geq 52\%\ watchers$
VS preferable to	$views < 16\%\ watchers$	$views < 52\%\ watchers$	

Table 5.7: Number of views up to which each strategy is preferable to the others

happens when presentities create different numbers of privacy filters. Since each privacy filter determines a different view of the presentity's presence information, Table 5.7 summarizes our findings in terms of views. This shows the maximum number of *views* (i.e., privacy filters) up to which it is recommended to use each strategy. This number is a function of the percentage of watchers to which a different privacy filter is applied and, therefore, watch a different view.

We proposed a variation of CS, which we refer to as FCS, that basically consists in aggregating presence and privacy rule information into a single event. FCS generates more traffic than CS excepting when conditional notifications are applied. When FCS is combined with conditional notifications, it saves 3% of the CS traffic. Conditional notifications actually help to reduce the presence traffic of any of the studied strategies. This doubles the efficiency of CN but requires all the watchers to either support this optimization or have the PS as a proxy for any subscription request. When VS and CN are combined with conditional notifications, the number of views up to which VS is preferable to CN drops to 24% of the watchers.

Below, we summarize other findings based on the reported results:

Dialog optimization: This strategy always generates much more traffic than the other strategies. If conditional notifications are applied, dialog optimization does not decrease but increases the presence overload on the network (i.e., not to apply any optimization generates less traffic).

VS: The operation and performance of VS is strongly affected by two parameters: the type of trust between the domains and the number of privacy filters. Partial trust always involves a smaller number of bytes than full and minimal trust. If the presentities' domain needs to know which watchers are currently subscribed, this domain must establish a minimal trust with the subscriber domain. In this case, VS is discouraged because it involves much more traffic than the other strategies.

The number of privacy filters determines the number of views and, therefore, the number of presence subscriptions in VS with partial and full trust. This is the reason why the efficiency of VS considerably drops as the number of views increases. Although minimal trust generally generates more traffic than partial or full trust, minimal trust is more efficient when there are numerous views. The reported results show that when 80% of the watchers have a different view, minimal trust is preferable. The increase in the number of presence changes has more harmful effects on VS than the other strategies. This is because a single presence change may involve notifying through more than one subscription (i.e., more than one view). Likewise, changes in the presentities' privacy filters may have disastrous effects on VS, since a single change may involve modifying, creating, or eliminating one or more presence subscriptions.

CN: This strategy's traffic is increased when two methods for obtaining the watcher lists are used: 1) the notifier domain adds the list to the body of NOTIFY messages and 2) the watcher domain subscribes to the presentities' winfo event. The increase in the number of watchers affects the latter more seriously than the former. The main parameter that affects the first method is the number of presence changes per presentity. We advise that the presentities' average activity be considered in making a choice between one of the two methods. In general, when presence changes occur very frequently, the second method is more efficient than the first.

Subscriptions to privacy filters account for a considerable part of CS and FCS traffic, and notify sensitive information (i.e., authorization rules set by presentities). Thus, we enhanced CS and FCS by reducing the number of privacy rules that are disclosed. We analyzed the variables that affect the traffic related to privacy filters in more detail. The proposed enhancements of CS and FCS rely on the fact that the presentities' PS always knows the set of watchers that are actually subscribed. Thus, this PS only notifies the privacy rules that contain at least one subscribed watcher rather than all the presentities' privacy rules. We say that a privacy rule containing any watcher that is actually subscribed is an active rule. We estimated the number of bytes exchanged between two federated domains due to privacy rules, during a presence session. The reported results show that the proposed enhancements lead to a considerable reduction

in this traffic, even when all the rules are active. The reduction of bytes is inversely proportional to the number of privacy rules that become active during the session. The enhanced FCS saves between 45% and 81% of the traffic related to privacy rules in the regular FCS. The enhanced CS saves between 18% and 60% of the privacy-filters subscriptions' traffic in the regular CS. We studied the effect of conditional notifications on our proposals. This optimization greatly reduces the traffic related to privacy rules and is, therefore, strongly recommended in both FCS and CS. The application of conditional notifications to the enhanced FCS is always recommended. However, in the case of CS, the reported results show that when more than half the presentities' privacy rules become active during the presence session, the enhanced CS with conditional notifications performs worse than the regular one with conditional notifications. Thus, the regular CS with conditional notifications is preferable to the enhanced CS.

On the basis of the presented study, we conclude that the proposed enhancement of FCS combined with conditional notifications is a good solution to save inter-domain presence traffic as far as possible. The main drawback of FCS is that the process of privacy filtering must be delegated to the subscriber side PS. Nevertheless, the fact that two domains exchange presence information and allow their users to communicate is an indication that some degree of trust relationship exists between them. Thus, delegating privacy filtering to the subscriber side domain would simply mean an extension of an existing trust. Likewise, VS needs some kind of trust relationship since the presentities' domain needs to trust the watcher domain to distribute the right views to the right watchers. Regarding the interoperability of privacy rules, PSs only need to exchange the rules encoded in the SIMPLE proposed standard for encoding authorization rules [177] regardless of their low-level implementations of privacy filtering.