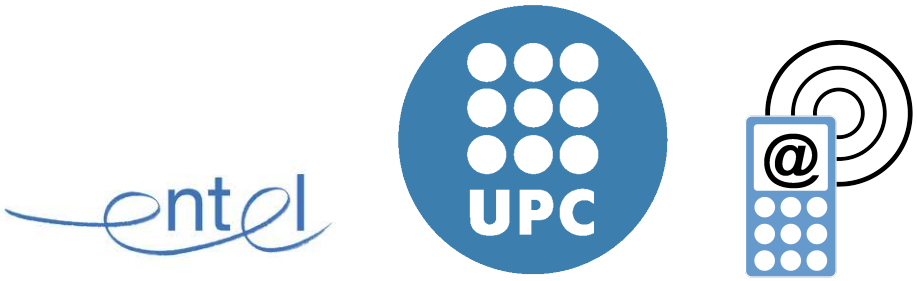
Contributions to Presence-Based Systems for Deploying Ubiquitous Communication Services



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

6

Capacity Demands of Inter-domain Traffic Optimizations on the IMS Network Servers

The presence service is considered as one of the enablers of the ubiquitous services that NGNs are expected to provide. Thus, the presence service plays a key role in the IMS, which is the foundation of NGNs. This provides an all-IP approach for the support and integration of multimedia services that will be accessed independently of the terminal device and access network. We refer the reader to Section 2.4 for further information on the IMS and NGNs. As explained in Section 2.9.4, the presence service may inject an unbearable amount of traffic into network servers because of periodic subscription refreshes and dissemination of presence information. This is a critical issue in the IMS, in which all the SIP messages initiated by and sent to IMS terminals traverse several centralized servers. 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 CSCF in the IMS. The performance analysis in [201] shows that SIP signaling traffic introduces long transmission delays on the UMTS network. 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 explained in Section 2.9.5, the presence

service's signaling traffic may overload and seriously damage networks servers when a large population of users in a domain subscribe to users in a different domain and vice versa. This scenario is referred to as presence federation. An analytical study [193] show that the traffic exchanged between two federated presence domains may reach 44 terabytes during 8 hours. This amount of traffic is calculated under the assumptions that 20 millions of users within a domain subscribe to 10 presentities in a federated domain each, and each presentity updates its presence information 6 times per hour. These assumptions are moderate considered that some presence applications such as IM are used in a planetary scale, measuring users in numbers of billions (see Section 2.1). Moreover, since a user's presence information may include dynamic information such the user's location, presence changes may occur at a rate higher than 6 changes per hour. A real presence service may therefore inject into federated domains much more presence traffic than that estimated in [193]. Section 5 analytically estimates the inter-domain presence traffic when some optimization techniques, such as Common Notify (CN) [194] and View Sharing (VS) [195], are applied. This section also proposes a novel strategy, Common Subscribe (CS), and its variation, Federated Common Subscribe (FCS), for optimizing presence traffic in federation scenarios. The reported results in Section 5.1 show that CS and FCS are more efficient at reducing inter-domain presence traffic than CN and VS. To further study the viability and performance of these strategies, this section discusses the impact of optimizing inter-domain traffic on the IMS. Given the stateful and centralized nature of the IMS network servers, reduction of its workload is crucial for increasing the scalability of presence applications and for preventing end-to-end delays.

Although Section 2.4.1 describes the IMS architecture, let us summarize it here for convenience. Figure 6.1 depicts the IMS architecture (IMS CN). 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. The Serving CSCF (S-CSCF) is the brain in the IMS signaling plane. This is a SIP server and SIP registrar that performs session control as well. The Interrogating CSCF (I-CSCF) receives SIP requests and routes them to the appropriate destination, an S-CSCF or an Application Server (AS). An AS is a SIP entity that hosts and executes

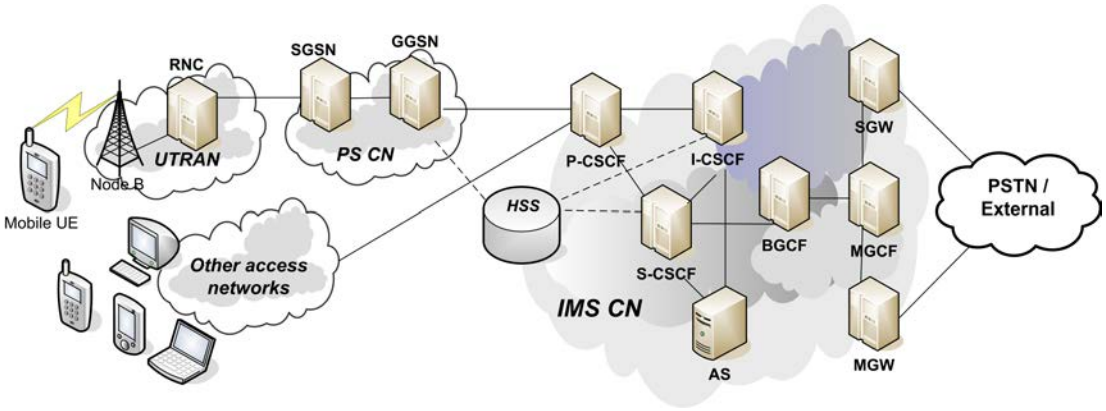


Figure 6.1: IMS architecture overview

services, such as the presence service. The MGCF, MGW, and SGW servers form the PSTN gateway.

Section 6.1 depicts the signaling flows involved in each optimization technique. Section 6.2 discusses the impact of these techniques on the IMS based on an analytical estimation of the workload at the CSCF.

6.1 IMS Signaling Flows

The following subsections depict the operation of the strategies for reducing inter-domain presence traffic and their signaling flows in the IMS. The sequence diagrams in Figures 6.2, 6.3, 6.4, and 6.5 provide a deep insight into the message load that each strategy injects into the IMS servers. In these diagrams, inter-domain messages are delimited by boxes that are marked with an indicator of the amount of messages (oval rectangle) and the type of messages (drop shadow rectangle). Servers within the watchers' domain are marked with 1 and servers within the presentities' domain with 2. Henceforth, we refer to the former as watcher or subscriber side domain, and to the latter as watched or notifier side domain.The dotted line arrows are the requests sent and received by the watchers (i.e., IMS terminals). The remaining messages triggered from these requests are omitted since they are intra-domain traffic. The subscriber side S-CSCF has to forward all the SUBSCRIBE messages sent by the watchers to the "module" containing the intelligence for optimizing inter-domain presence traffic. We assume that this intelligence is placed inside the subscriber side PS.

6.1.1 Common Notify

Rather than sending one NOTIFY message per presence subscription, CN proposes sending a single NOTIFY message to each watcher domain when a presentity's presence information changes, as described in Section 5.1.2. A common NOTIFY message carries the presentity's complete presence information and is targeted at all of his or her authorized watchers. Thus, the watcher domain is responsible for providing the watchers with the presence documents that they are allowed to see. To this end, the PS needs to know the privacy rules set by the presentity to these watchers. Section 5.1.2.1 describes the operation of privacy-filters subscriptions as the means for the watcher domain to obtain the presentities' privacy rules. When the first presence subscription to a presentity is established between two federated domains, the subscriber side PS should subscribe to the presentity's privacy-filters event. In addition, the watcher domain needs to know to which watchers each common NOTIFY message is targeted. Three alternatives are contemplated by [194]: Maintaining the list on the subscriber side PS, including the list in notifications and obtaining the list by subscribing to the presentity's winfo event [145]. The first alternative does not involve extra traffic but consumes memory at the subscriber side PS. The second alternative increases the size of the NOTIFY messages that are sent by the presenties' PS. Figure 6.2 depicts the message flows of CN configured with these two alternatives. Lastly, the third alternative involves a winfo subscription to each presentity; the message flows of CN with winfo subscriptions, therefore, result from adding these subscriptions to the diagram in 6.2 (i.e., a box like that for privacy-filters subscriptions).

6.1.2 View Sharing

This mechanism classifies the presentity's watchers according to the part of the presentity's presence they are authorized to see, as described in Section 5.1.3. It is referred to as the watcher's "view" on the presentity. 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 view, it is responsible for distributing this document to all the watchers who are allowed to see that view.

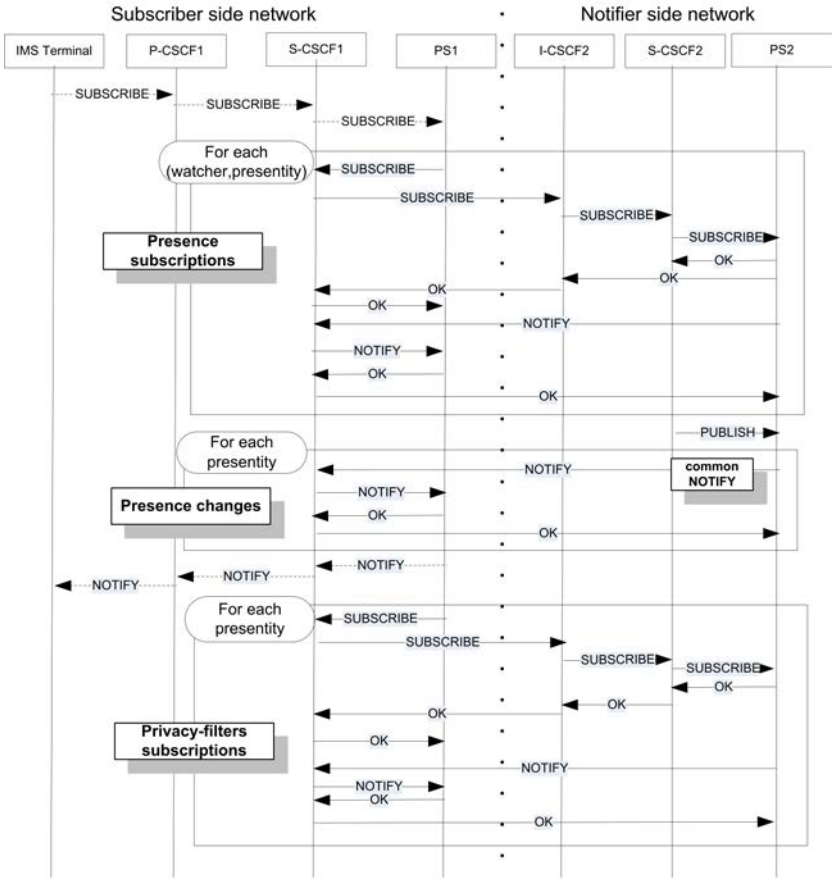


Figure 6.2: Common Notify's message flows

With VS, the watcher domain does not perform privacy filtering. Presentities' views are represented by Access Control Lists (ACLs), which are encoded by the XML scheme described in [195]. An ACL is basically a set of rules, each identifying a different view and containing the list of authorized watchers. When a watcher domain subscribes to a presentity, the presentity's ACL is attached to the resulting NOTIFY message. The content of this ACL depends on the type of trust established between both domains. Full trust means that all of the presentity's rules are sent. Thus, the complete association of watchers with views is disclosed and the watched domain will not know the full set of watchers actually subscribed. With partial trust, only the rule associated with the subscriber is sent. Thus, the presentity's PS only discloses the watchers who see the same view, but is still not aware of the subscribed watchers. With minimal trust, the ACL only contains the subscriber, and hence there is a backend subscription

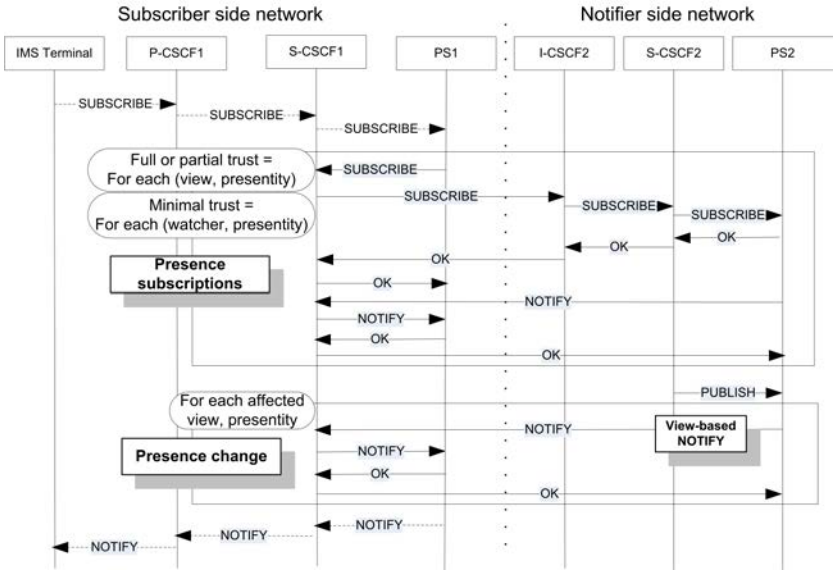


Figure 6.3: View Sharing's message flows

for each watcher. This lets the presentity's PS know about the watchers that are actually subscribed. 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 the other watchers. As regards the number of inter-domain presence subscriptions, the application of full or partial trust is more efficient, since there is only one subscription for each view. Figures 6.3 depicts the signaling traffic of VS.

6.1.3 Common Subscribe

CS creates a single subscription for each watched presentity between the federated domains, as described in Section 5.1.4. A common subscription to each presentity saves a great deal of the signaling traffic involved in multiple subscriptions (one per watcher with CN or one per view with VS). In turn, the list of watchers that are actually watching a presentity must be included in any common SUBSCRIBE message to the presentity. This lets the presentity's PS know about these watchers as well as performing authorization tasks. NOTIFY messages as a result of subscription requests must include the list of authorized watchers. Thus, these messages have a multipart structure with a presence document and a watcher list (encoded by the XML scheme described in [178]). Like CN, with CS, the PS does not notify a presence document to

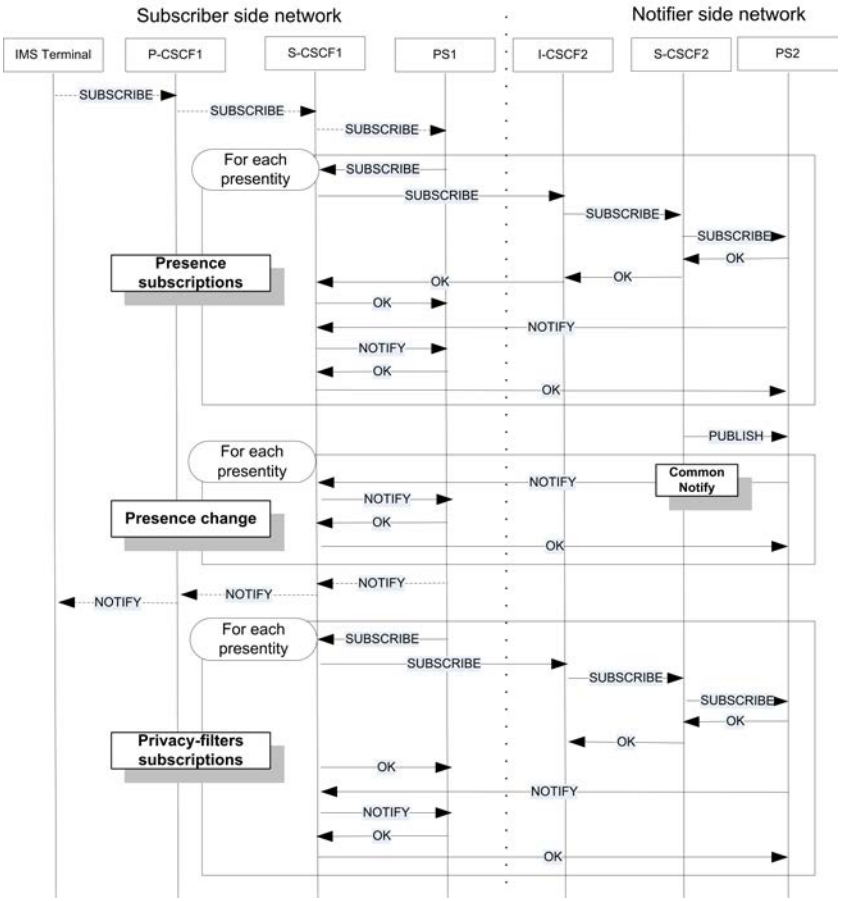


Figure 6.4: Common Subscribe's message flows

a single watcher, but rather to a set of watchers. The watcher domain therefore needs to know the presentity's privacy rules for the watchers at which the presence document is targeted. These privacy rules will be used to filter the complete presence information and so generate the presence document that each watcher is authorized to see. Thus, CS requires the watcher domain to subscribe to the presentities' privacy-filters events as described in 5.1.4.1. Figure 6.4 shows the flow of messages of CS in IMS.

6.1.4 Federated Common Subscribe

FCS is a variation of CS that unifies the presence and privacy-filters events into a composed event denominated as federated-presence, as described in 5.1.4. This event represents all the information about a presentity that a watcher domain is authorized to

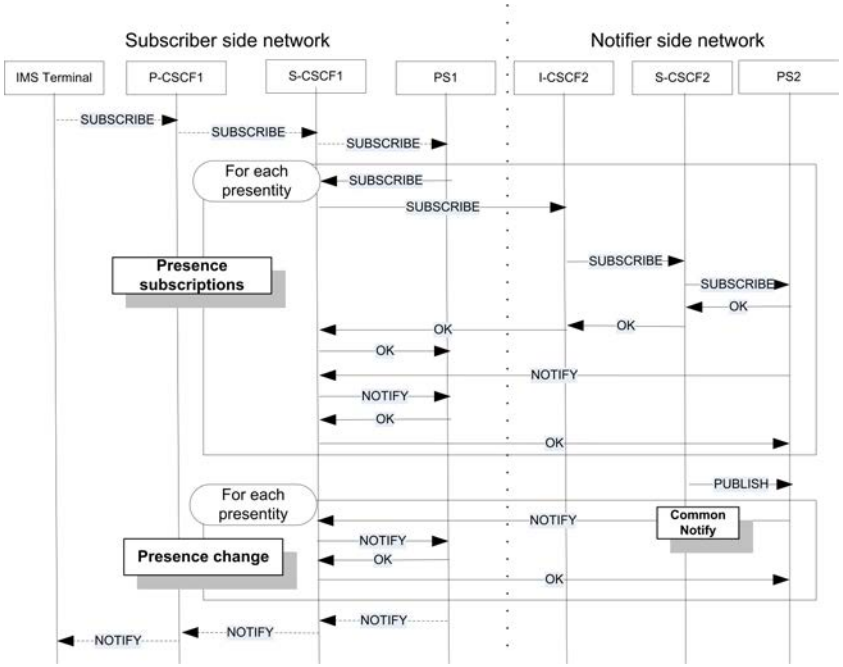


Figure 6.5: Federated Common Subscribe's message flows

see: his or her presence information and privacy rules for watchers within the watcher domain. Thus, in addition to the presence information and watcher list, NOTIFY messages resulting from subscription requests contain privacy rules. However, a change in a presentity's presence information or privacy filters only results in a notification of the information that has changed (i.e., a presence document or privacy rule). Figure 6.5 shows the flow of FCS messages in the IMS.

6.2 Impact of Traffic Optimization on the IMS CSCF

As depicted in Figures 6.2, 6.3, 6.4 and 6.5, the S-CSCF is the element most impacted by the presence service. This section discusses the workload that the inter-domain presence traffic optimizations inject into the IMS servers. This is an important QoS parameter for measuring the performance of each strategy, since an excessive workload at the CSCF may cause end-to-end delays. Moreover, Section 6.2.1 studies how privacy filtering can increase the workload at the network servers. Section 6.2.2 analyzes the impact of traffic optimizations on the network servers compared to the absence of any optimization.

Name	Description	Default average value
np	Total number of presentities	40000
wat	Total number of federated watchers per presentity	20
stime	Presence session time	8 hours
rtime	Subscription refresh interval	1 hour
nr	Total number of subscription refreshes (slife/sref)	20
pch	Number of presence changes per presentity	3 per hour
views	Number of views per presentity	1
watv	Number of watchers associated to a view (wat/views)	20 watchers
ip	Presence change's impact percentage on views	1

Table 6.1: Variables for estimating presence load in number of messages

We calculate the number of messages handled by the CSCF during a presence session, based on the message flows described in Section 6.1 and the variables in Table 6.1. We assume the presence federation scenario in which there are 40,000 presentities (*np* variable) in a domain (that is the notifier side domain). Every presentity is watched by 20 watchers on average in another domain. We assume the average session time to be 8 hours (*stime* variable). The subscription lifetime is 1 hour (*rtime* variable), which is the default value for presence subscriptions [120]. Thus, a watcher needs to resubscribe *stime/rtime* times in order to keep the subscription alive (*nr* variable). To tackle privacy filtering, we assume an average number of views per presentity that is given by the *views* variable. As described in Section 6.1.2, 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 [177], and henceforth we use the terms "presence view" and "privacy rule" indistinctly. We assume that the number of watchers associated to each view (*watv* variable) is uniformly distributed, and hence is calculated as *wat/views*. A presence change does not necessarily affect all the views, but only those that include the information that has changed. We therefore assume an impact percentage of presence changes on views (*ip* variable). This determines the number of views affected by each presence change, and hence the number of watchers that have to be notified. We classify presence traffic into the following two classes:

Presence update messages: This constitutes all the traffic for notifying the watcher domain of presence changes. In the case of CS, FCS, and CN, there is only

one presence notification anytime a presence change occurs. Thus, the amount of NOTIFY messages handled by the subscriber side S-CSCF and PS is $np * pch * stime$. In the case of VS, there are as many inter-domain notifications as different views. This means that the subscriber side S-CSCF and PS need to handle $np * pch * stime * views * ip$ NOTIFY messages.

Subscription maintenance messages: This is all the traffic for creating, refreshing and terminating subscriptions. In the case of CN and VS with minimal trust, there is one inter-domain presence subscription per watcher, per presentity. VS with full or partial trust creates a presence subscription for each view. With CS and FCS, there is only one subscription per presentity. In addition, CN and CS need to handle one privacy-filters subscription per presentity. CN may need to handle one winfo subscription per presentity for obtaining the list of subscribed watchers. Every subscription request triggers multiple flows of messages that traverse the IMS servers in both the watcher and notifier domains. Moreover, the number and type of subscriptions are different for each optimization technique. Thus, to provide an intuitive comparison of the optimization techniques' subscription maintenance traffic, we use Tables 6.2 and 6.3. Let us define a server's incoming flow as a sequence of SIP messages of a particular type that the server receives. Table 6.2 shows the types of incoming flow that the maintenance of subscriptions causes for each IMS server. The subscriber side servers are marked with 1 while the notifier side servers with 2, as in Section 6.1. The types of flow are the same for all the optimization strategies, but not their magnitudes. The number of messages involved in incoming flows, which we refer to as flow weight, varies with the strategies. Table 6.3 shows each strategy's flow weight. A strategy flow weight is the same for all the types of incoming flow at any server, except for the CN strategy with winfo subscriptions[2]. For instance, with FCS, the subscriber side PS needs to handle a total of $2 * np * (wat + nr)$ messages (half NOTIFY messages and half OK for SUBSCRIBE messages).

Figure 6.6 shows the total number of presence messages that each IMS server would have to handle under the presence scenario defined by the variables in Table 6.1. The

[2]Both the NOTIFY and OK for NOTIFY incoming flows has $np*(wat-1)$ extra messages because of notifications of new watchers through winfo subscriptions

IMS server	Types of incoming flows			
	SUBSCRIBE	NOTIFY	OK for SUBSCRIBE	OK for NOTIFY
S-CSCF1	X	X	X	X
PS1		X	X	
S-CSCF2	X		X	
PS2	X			X

Table 6.2: Types of incoming flows involved in subscription maintenance

Technique	flow weight
CN	$np * (wat * (1 + nr) + (1 + nr))$
CN with winfo subs.	$np * (wat * (1 + nr) + 2 * (1 + nr))^2$
VS full or partial trust	$np * views * (1 + nr)$
VS minimal trust	$np * wat * (1 + nr)$
CS	$np * (wat + 2 * nr + 1)$
FCS	$np * (wat + nr)$

Table 6.3: Incoming flow weight for each optimization strategy

inter-domain presence traffic optimizations presented in Section 6.1 are shown on the X-axis. *CN* is the number of messages with CN when the watcher list is obtained by either maintaining the list on the PS or including the list in notifications. *CN-winfo* is the number of messages when CN uses winfo subscriptions for obtaining the watcher lists. Thus, *CN-winfo* involves more messages than *CN*. *VS-full* means VS with full or partial trust, while *VS-min* is VS with minimal trust. The subscriber side servers are marked with 1 and the notifier side servers with 2. We do not include the P-CSCF, since the traffic that it receives is intra-domain; neither do we include the notifier side I-CSCF because its workload is the same as that at the notifier side S-CSCF. We assume that the presentities do not set up privacy rules to any watcher and hence, privacy filtering is not performed. Thus, there is only one view that all the watchers can see and therefore, the impact percentage is one (i.e., any change affects the view). It is obvious that the S-CSCF is the most overloaded server and that a great part of its load is a result of notification messages. This is due to the fact that all the NOTIFY messages sent and received by the subscriber side PS traverse the S-CSCF.

Figure 6.7 shows the number of SIP messages per second that the subscriber side S-CSCF needs to handle during the session. One may observe that the S-CSCF has to handle a high number of inter-domain messages, which may require a considerable part

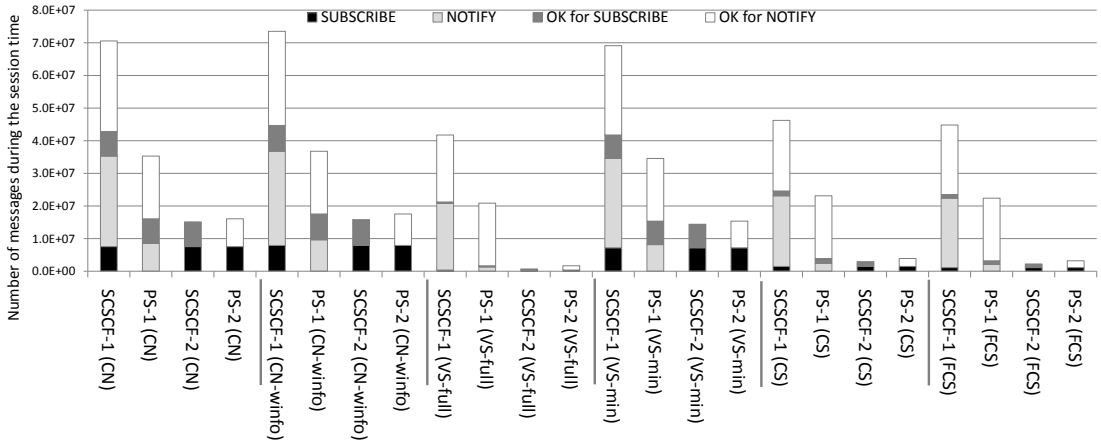


Figure 6.6: Number of presence messages when no privacy filtering is performed

of its capacity. The processing requirements at the S-CSCF vary approximately between 1500 and 2500 SIP messages per second. VS with full or partial trust is the strategy that load the network servers least. However, we do not consider privacy filtering, and hence there is a single presence subscription per presentity. In Section 5.1.5.3, we show that privacy filtering may seriously increase the VS traffic. Section 6.2.1 discusses the impact of privacy filtering on the network servers when VS is used. VS with minimal trust is the strategy that introduces the largest number of messages. However, this is the only way for VS to let the presentities' PS know about the list of watchers that are actually subscribed. Neither partial or full trust provide the presentities' PS with this information. CS generates more messages than VS with partial or full trust for two reasons: presence resubscriptions for new watchers and privacy-filters subscriptions. The first reason is due to the fact that CS always ensures that the presentities' PS has knowledge about the set of watchers actually subscribed. FCS is a variation of CS that avoids subscribing to privacy-filters events. However, its load is still slightly higher than that of VS because of the presence resubscriptions for new watchers. VS with minimal trust is very similar to CN, but the former does not need to establish privacy-filters subscriptions. This is the reason why the load on the S-CSCF with *VS-min* is slightly lower than that with *CN*. However, VS generates as many notifications per presence change as privacy rules created by the presentity, rather than one. As the number of privacy rules increases, CN could be preferable to VS with minimal trust. Regarding CN, the use of winfo subscriptions (*CN-winfo*) increases the load on the IMS servers.

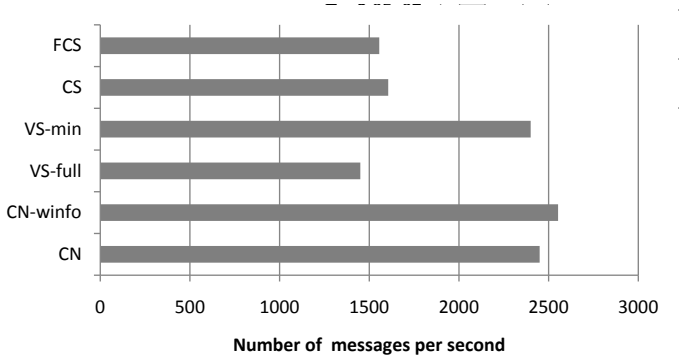


Figure 6.7: Number of inter-domain messages per second at the subscriber side S-CSCF

Another alternative for obtaining the list of watchers to which the notifications are targeted is to include this list in the notifications themselves. Section 5.1.5.4 studies the performance of these two alternatives. The reported results in this section show that winfo subscriptions save around 7% of the second alternative's traffic in the case of high activity (from 5 to 10 presence changes per hour). However, even in this case, the impact of winfo subscriptions on the servers' workload may not compensate for this saving of inter-domain traffic.

6.2.1 Privacy Filtering

Section 6.2 shows that VS with partial or full trust is the optimization technique that least impacts the IMS servers when no privacy filtering is performed. After VS, FCS, and CS are the strategies that involve the fewest number of messages. This section studies the effect of privacy filtering on VS, and compares it to that on FCS. This comparison is also applicable to CS since its mode of operation and involved load is very similar to FCS. Figure 6.8 shows the amount of messages that VS and FCS generate to each IMS server. With VS, the number of incoming messages at the servers increases with the number of views linearly. However, the number of messages generated by FCS does not vary, since there is a single subscription regardless of the number of views. We see that FCS causes less overload than VS from two presence views per presentity, that is, when privacy filtering is performed. We assume that presence changes affect all the presence views. However, a privacy rule many not include the information that has changed, and may therefore not be affected by the change. In Section 6.2, we define the impact percentage as the average percentage of views that are affected

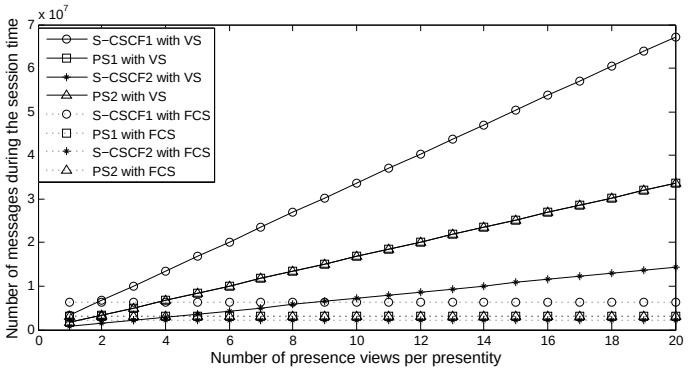


Figure 6.8: Number of presence messages as the number of views per presentity increases

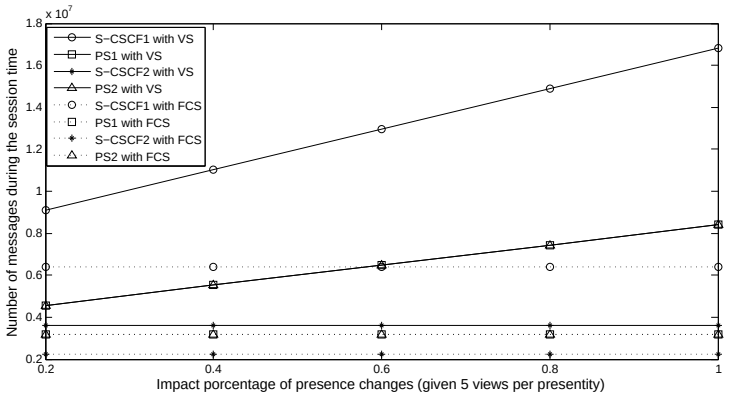


Figure 6.9: Number of presence messages as the impact percentage increases

by each presence change. Figure 6.9 shows the load of VS and FCS as the impact percentage increases, given that the presentities create five privacy rules. VS sends as many NOTIFY messages as the number of views affected by the presence change. Thus, its load increases with the impact percentage. FCS only sends a single NOTIFY message, regardless of the number of views or the impact percentage. It can be seen that even when only one view is affected, FCS involves less message load for each network server than VS.

6.2.2 Impact of an Application Server for Traffic Optimization on the IMS

Although traffic optimization is a necessary step to make the IMS presence service viable, this may make a profound impact on the S-CSCF. Section 6.1 only considers inter-domain traffic, and intra-domain traffic is left out of consideration. We consider

intra-domain traffic as the traffic that is sent and received by IMS terminals, depicted by the dotted line arrows in Figures 6.2, 6.3, 6.4, and 6.5. To optimize inter-domain presence traffic, an AS needs to be added in the IMS architecture for performing the necessary intelligence . This requires the subscriber side S-CSCF to redirect the SUBSCRIBE messages sent by IMS terminals to the AS, and to act as an intermediary in any transaction with this server. Traffic optimization therefore requires some of the CSCF capacity. Section 6.1 assumes that the intelligence for traffic optimization is placed into the subscriber side PS. It is reasonable, however, to collocate this intermediary with the subscriber side Resource List Server (RLS) (see Section 2.7). This is a well-known concept in presence frameworks that is designed to drastically reduce the number of messages between watchers and the network. In the SIMPLE normal mode of operation, watchers subscribe to their presentities directly, and hence there are as many subscriptions established on the access network as the number of presentities for each watcher. Henceforth, we refer to this operation mode as Direct Subscriptions (DS). With the RLS, watchers subscribe to their RLSs instead of their presentities, and so the number of presence subscriptions on the access network is reduced to one for each watcher. In turn, the RLS is responsible for subscribing to the watchers' presentities on the watchers' behalf as well as notifying them of the presentities' presence changes. Figures 6.10 and 6.11 depict the signaling flows with DS and when watchers use the RLS for optimizing intra-domain traffic, respectively. It is obvious that the RLS reduces considerably the number of messages sent over the access link. However, the RLS causes more load for the S-CSCF because it is the intermediary in any communication with the RLS, as shown in Figure 6.11. Although Section 6.1 assumes that all the optimization strategies require an AS for optimizing presence subscriptions, CN supports a mode of operation that is less harmful to the S-CSCF. When the list of watchers at which the common notifications are targeted is attached to the notifications themselves, the subscriber side PS does not need to maintain the watcher lists or subscribe to winfo subscriptions for obtaining these lists. Thus, the subscriber side PS does not need to be an intermediary in the watchers' presence subscriptions. In this case, the signaling flows introduced by CN would be almost equal to those in Figure 6.10. The only difference is that there is a single common notification at any time when a presentity changes its presence, rather than one per watcher. Common notifications are sent to the subscriber side PS, which notifies each watcher, as shown in Figure 6.2.

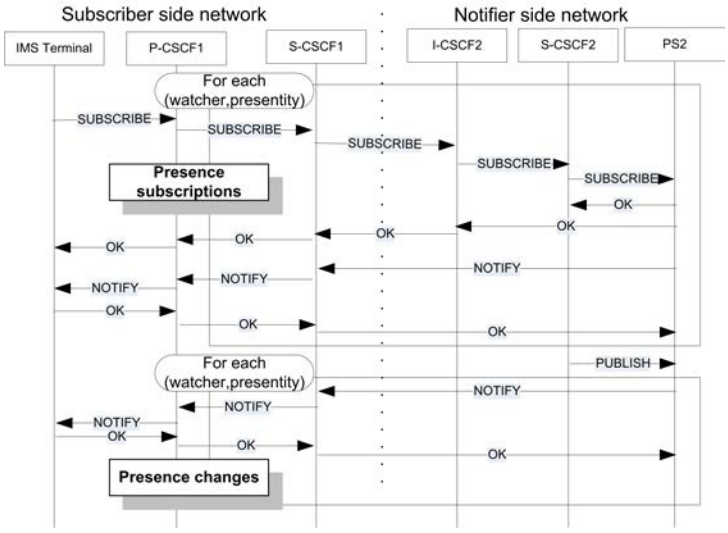


Figure 6.10: Signaling flows of presence subscriptions without any optimization

This section discusses how the workload at the S-CSCF can be affected by using an AS, either the PS or RLS, for optimizing presence traffic. Figure 6.12 shows the number of inter- and intra-domain messages per second handled by the subscriber side S-CSCF during a session. We take into account both intra-domain and inter-domain traffic. We calculate the message load when presence traffic is not optimized at all, and thereby watchers subscribe directly without any optimization technique (i.e., *DS*); when intra-domain traffic is optimized by an RLS (i.e., *RLS*); when CN is applied without an AS for optimizing inter-domain traffic (i.e., *CN-WA*); when FCS optimizes inter-domain presence traffic (i.e., *FCS-DS*), and when FCS is collocated with the RLS for optimizing intra-domain traffic too (i.e., *FCS-RLS*). We study CN because it is the only strategy that does not require an intermediary in the watchers' presence subscriptions, as described previously. Among the other strategies (i.e., those that need an AS), we focus on FCS because this is the one that impacts the S-CSCF the least, as shown in Section 6.2. To calculate the number of intra-domain messages, we need to estimate the population of watchers in the subscriber side domain. Let L_w be the average size of the watchers' resource list and nw the number of watchers. Thus $np * wat = nw * L_w$ must satisfy since $np*wat$ determines the number of inter-domain presence subscriptions and the RLS establishes L_w subscriptions on behalf of each watcher. Let us assume that the watchers watch 10 presentities on average; that is $L_w = 10$. We therefore deduce that

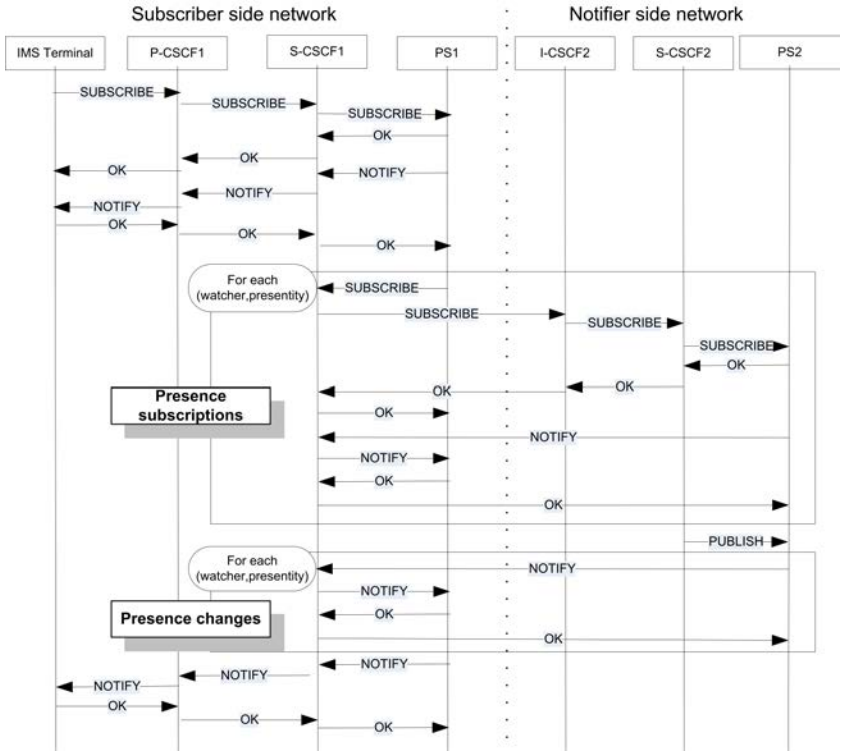


Figure 6.11: Signaling flows of presence subscriptions with an RLS

the number of watchers (nw) is equal to 80,000. Thus, the RLS and DS will involve 80,000 and 800,000 presence subscriptions on the access network, respectively. Thus, the S-CSCF needs to handle the traffic generated by 80,000 subscriptions with *RLS* and *FCS-RLS*, and 800,000 subscriptions with *DS*, *CN-WA* and *FCS-DS*, in addition to the inter-domain traffic.

It can be seen that the RLS seriously increases the capacity demands on the S-CSCF. This is due to the fact that the S-CSCF is the intermediary in any communication of the the RLS with the watchers and the notifier side PS. Thus, the RLS by itself (i.e., not combined with any optimization) is very harmful to the CSCF. It should be carefully studied whether the saving of presence traffic achieved by the RLS compensates for its impact on the IMS servers. Likewise, all the strategies for optimizing inter-domain presence traffic increase the workload at the S-CSCF. This is because they require an AS that processes the subscription requests sent by the watchers and the notification requests sent by the notifier side PS. Compared to the FCS strategy in Figure 6.7, one may observe that intra-domain traffic increases the workload at the S-CSCF by

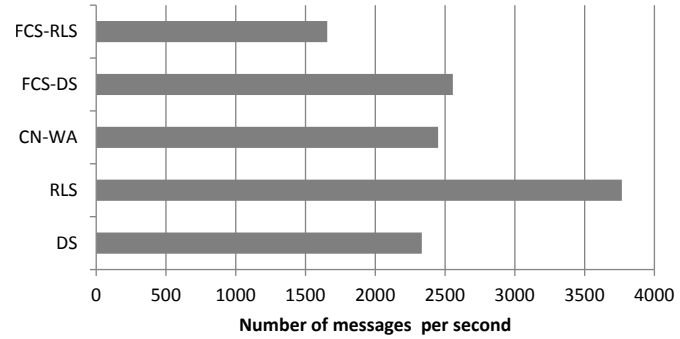


Figure 6.12: Number of inter- and intra-domain SIP messages/second at the subscriber side S-CSCF

approximately 1000 messages per second. This increase is the same for the remaining strategies in Figure 6.7, since all except CN-winfo need an AS. The CN-winfo strategy in Figure 6.12 is configured in such a way that the subscription requests sent by the watchers are not forwarded to any AS. This decreases the workload at the S-CSCF caused by the CS-winfo strategy in Figure 6.7. One may observe that, when intra-domain messages are taken into account, the CN-winfo strategy injects fewer messages into the S-CSCF than the FCS strategy (and therefore the other studied strategies). However, reducing the number of inter-domain and intra-domain presence subscriptions at the same time is the alternative with the least impact. Figure 6.12 shows that the combination of the RLS with the FCS strategy even reduces the processing requirements of the presence service on the S-CSCF. The workload at the S-CSCF without any optimization (i.e. w/o RLS) is approximately 2300 messages per second. An RLS that performs the FCS strategy reduces this load up to approximately 1600 messages per second. An RLS applying the CS strategy, or applying the VS strategy with partial or full trust if no privacy filtering is performed, would achieve a similar reduction.

6.3 Conclusions

Presence information is regarded as a key tool to personalize and adapt next-generation services. However, the scalability of large-scale presence-enabled applications is challenging because of the large amounts of signaling traffic that presence services generate. SIMPLE is the SIP extension for presence and instant messaging that has become the de facto protocol for presence services. To date, the proposals for reducing inter-domain

presence traffic, Common Notify (CN) and View Sharing (VS), have been discontinued in the IETF. However, the need to make presence federation scenarios scalable still remains for the future large-scale converged networks. The fact that SIMPLE, the standard protocol for presence in IMS, is no longer involved in reducing inter-domain presence traffic should alarm both the industry and academia. Common Subscribe (CS) and Federated Common Subscribe (FCS) drastically reduce the number of inter-domain presence subscriptions up to one per presentity, regardless of the number of subscribed watchers, as decribed in 5.1. CS and FCS are more efficient at reducing inter-domain presence traffic than CN and VS (see Section 5.1).

We estimated the processing requirements in the number of messages that CN, VS, CS and FCS add to the IMS servers. High processing requirements can worsen the IMS scalability and introduce end-to-end delays. They are therefore an important parameter to consider when choosing an optimization strategy apart from the reduction of presence traffic between the federated domains. The reported results show that the presence service adds relevant capacity demands on the S-CSCF. The introduction of an AS that optimizes presence traffic into the IMS considerably increases this server's load. This is because the S-CSCF is the intermediary in any communication with the AS. All the inter-domain presence traffic optimizations increase the S-CSCF load as compared to when no optimizations are applied. The common optimization of access network traffic, the RLS, almost doubles the S-CSCF workload without any optimization. The RLS is therefore strongly discouraged if it is not combined with other optimization to reduce its overload. However, if the RLS applies the FCS or CS strategies for optimizing inter-domain presence traffic, it decreases the S-CSCF message load when presence traffic is not optimized by around 29%. Thus, these optimization strategies combined with the RLS are not only helpful in optimizing inter-domain presence traffic but also ease the impact of the presence service on the CSCF. However, it does not necessarily mean that the intra-domain presence traffic is reduced. This depends on a number of factors, as described in Section 7. Regarding CN, the subscriber PS needs to obtain the watcher list to which any common notification is targeted by one of three alternatives: 1) the subscriber side PS maintains the list, 2) the subscriber side PS subscribes to winfo events and 3) the notifier side PS includes the list in the notifications. The third alternative has the important advantage that it does not require an AS acting as an intermediary in the watchers' presence subscriptions. Thus, this alternative injects

fewer messages into the S-CSCF than the other optimization strategies without being combined with the RLS. The second alternative increases the S-CSCF workload because of winfo subscriptions. We therefore recommend the use of the third alternative in CN. When no privacy filtering is performed, VS with partial or full trust is the strategy that least increases the CSCF' workload. This means that all the watchers of a presentity are allowed to see the same piece of the presentity's presence information. This may not be true in many scenarios. Moreover, VS with partial or full trust does not allow the presentities' PS to know about the set of watchers that are actually subscribed, which may not be acceptable by the presentities' domain. When privacy filtering is performed, FCS is the strategy that places the lowest capacity demands on the IMS servers, closely followed by CS.

We therefore conclude that the FCS strategy is a good alternative for reducing inter-domain presence traffic. This should be implemented by the subscriber side RLS in order to reduce the capacity impact on the S-CSCF. To better study the efficiency and viability of CS and FCS, our next step is to implement these strategies for evaluating their complexity, performance, and processing demands in real scenarios. We envision that the impact of FCS and CS on the AS memory resources will be fairly lower than that of regular SIP subscriptions, since these strategies reduce drastically the number of inter-domain subscriptions. Regarding CPU usage, CS and FCS require some extra computation at the AS for handling common subscriptions. We will therefore study the computational cost of CS and FCS in detail, thereby concluding whether or not their implementation at large-scale operator networks is viable.

7

SIP/SIMPLE Resource List Server: Optimization or Burden for Presence Systems?

SIMPLE relies on event subscriptions for disseminating context information to subscribed services or users. Event subscriptions generate a considerable amount of signaling traffic that is restraining the widespread adoption of presence-enabled services, as described in Section 2.9.4. On one hand, the SIMPLE subscription model requires sending periodic messages to prevent subscriptions from expiring. On the other hand, any presence change triggers end-to-end notification flows that involve end-user applications and multiple network nodes exchanging numerous messages. Presence traffic overload is specially harmful and critical in mobile applications because of low-bandwidth links and end-user devices with limited battery life and processing capabilities. Moreover, the fact that SIMPLE encodes presence information by XML, which is textual and verbose, constitutes a heavy burden for mobile devices. User location is a primordial parameter in many pervasive and proactive applications. Although presence is seen as a tool for implementing next-generation LBSs, timely dissemination of frequent location updates is still challenging in presence systems. Both industry and academia have concern about the need to reduce presence traffic for making presence services scalable and viable. Sections 2.9.4 and 2.9.6 describe some techniques for optimizing presence traffic that have emerged to date. One of the most popular optimizations of presence traffic specified by the SIMPLE WG is the RLS [121], which dramatically reduces the

number of messages that are sent on the user's access link. Let us reiterate here the description about the RLS given in Section 2.7 for convenience. Although watchers can subscribe to each of their presentities directly, typically they subscribe to their RLSs. This allows watchers to subscribe to their presentities through a single SUBSCRIBE message. This message contains a URI that identifies the watcher's resource list, in which each resource represents a presentity. When an RLS receives a subscription request, it subscribes to each resource on the list, and when any resource changes it sends an RLMI document [121] to the subscriber. An RLMI document is a list that contains zero or more resource items, each one for a presentity. A resource item points to the resource's presence document in the message's body, which has a Multipart structure [203]. Appendix B shows an example of RLMI document. The sequence of notification is as follows: the presentity sends a PUBLISH message to its PS. Then, the PS sends a NOTIFY message, which contains a PIDF document, to the RLS of each watcher that is authorized to see the published presence change. Each of these RLSs, in turn, sends a NOTIFY message that contains an RLMI document, which includes the notified PIDF document, to its subscribed watcher. PSs and RLSs are called notifiers because they are in charge of notifying RLSs and watchers of presence changes, respectively.

Although the RLS is widely adopted to reduce presence traffic in mobile presence applications, its performance has not been studied in depth yet. Only the authors of [193] estimate inter-domain presence traffic when a federated RLS is used to reduce traffic on the network core. Section 5 explains this work in more detail. The authors conclude that a federated RLS only helps in reducing presence traffic when it is combined with other techniques such as conditional notifications. Even so, a federated RLS may be discouraged because it increases the complexity of some parts of the system such as subscription state, interlinkage and notifications. As the authors state, further work is necessary to analyze and optimize presence traffic. Section 5 compares a federated RLS with other strategies for reducing inter-domain presence traffic. The results reported in this section show that a federated RLS always involve more traffic than the other studied strategies. Furthermore, when conditional notifications are applied, a federated RLS introduces more traffic rather than optimizing it. Section 6 discusses the impact of an RLS on the IMS and concludes that this impact may be severe if the RLS is not optimized. The reduction in presence traffic on the network access achieved by the RLS may not compensate for this impact.

Since the afore-mentioned studies reveal that an RLS may not be as efficient as expected and may even be harmful to the network servers, this section analyses the efficiency of an RLS at reducing traffic on the network access in detail. This section also studies the parameters that affect the performance of the RLS and gives some guidelines on its usage. Section 7.1 presents the mathematical formulas that calculate the traffic in bytes generated by an RLS, with and without two optimizations: conditional notifications and partial-state documents. Section 7.2 discusses the results of these formulas in a particular scenario. Lastly, Section 7.3 summarizes the main findings of this study.

7.1 Calculation of RLS Traffic on the Access Link

We follow the methodology described in Section 5 for estimating the presence traffic that is sent over the wireless link between an end-user and the network with or without an RLS. Presence traffic is given in number of bytes through a set of formulas, which are based on the following assumptions. Messages are classified 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. A subscription's traffic is calculated as the sum of the initial, steady state and termination messages during a session. The reported formulas are functions of the variables in Table 7.1. We assume that the average subscription lifetime is 8 hours (*slife* variable), and the average refresh interval to keep subscriptions alive is 1 hour (*sref* 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 variable *mpb* is the size of a boundary in Multipart documents. The *rlitem* variable is the size of each resource item (i.e, a presentity's identity) of RLMI documents. Appendix B shows an example of RLMI document. The *rlroot* variable is the size of the root XML elements that contain resource items in RLMI documents. The value of the *rlroot* and *rlitem* variables has been deduced from [121]. The *psdoc* variable is the size of partial-state presence documents, which has been deduced from the examples in [139]. In Table 7.1, we have marked the variables whose values vary for

Name	Description	Average value
slife	Subscription lifetime	8 hours
sref	Subscription refresh interval	1 hour
pres*	Number of presentities per watcher	20
pch*	Number of presence changes per presentity	3 per hour
sub	SUBSCRIBE message size	450 bytes
sok	size of 200 OK for SUBSCRIBE message	370 bytes
not	NOTIFY message size	500 bytes
nok	size 200 OK for NOTIFY message	370 bytes
mpb	Size of a boundary in Multipart bodies	144 bytes
rlroot	Size of RLMI documents' root	144 bytes
rlitem	Size of RLMI documents' resource items	144 bytes
doc*	size of full-state presence documents	1000 bytes
psdoc	size of partial-state presence documents	400 bytes

Table 7.1: Variables for estimating the RLS traffic

the analysis described in Section 7.2 with an asterisk: the number of presentities (*pres* variable), the number of presence changes per presentity (*pch* variable) and the size of presence documents (*doc* variable). Initially, we have assumed that a user watches 20 presentities that make three presence changes per hour on average. We have also assumed presence documents of 1000 bytes. These assumptions are quite moderate if we consider always-on users with rich presence information that may contain geographical coordinates, contact information, activities, mood, status, device and service capabilities, etc. Appendix B shows an example presence document with this kind of information.

The mathematical formulas below are based on the variables in Table 7.1 and calculate the presence traffic on the user access link 1) without an RLS and 2) with an RLS, during a presence session. These two methods can be combined with conditional notifications and partial-state documents. When conditional notifications are used, subscription refreshes and terminations do not trigger presence notifications. Partial-state presence documents only contain the changes that have occurred from the last notification rather than the complete resource information. Thus, when presence changes are notified in partial-state documents, the *change* variable takes the *psdoc* variable instead of *doc*. Moreover, the RLS can notify partial-state RLMI documents, which only contain the presentities whose presence has changed rather than the complete

list of presentities. Both methods' total number of bytes is given by the sum of four variables based on the above-mentioned message classification: The *initial* variable is the number of bytes of initial messages. The *termination* variable counts the number of bytes of termination messages. The variables *change* and *refresh* are the number of bytes of the messages involved in presence changes and subscription refreshes (i.e., steady state messages), respectively. For the *change* variable, we subtract 2 from the number of presence changes because the presence changes because the user gets online and goes offline are not counted as steady state traffic.

1. Direct subscription: The user subscribes to each of his or her presentities directly.
   $initial = pres * (sub + sok + not + doc + nok)$
   $change = pres * (pch - 2) * (not + doc + nok)$
   $refresh = pres * (slife/sref - 1) * (sub + sok + not + doc + nok)$
   $termination = pres * (sub + sok + not + doc + nok)$

   If conditional notifications are applied, the refresh and termination messages are:
   $refresh = pres * (slife/sref - 1) * (sub + sok)$
   $termination = pres * (sub + sok)$

2. RLS: The user subscribes to his or her RLS, which in turn subscribes to each of his or her presentities.
   $initial = (sub + sok + not + rlroot + pres * (rlitem + mpb + doc) + nok)$
   $change = pres * (pch - 2) * (not + rlroot + pres * (rlitem + mpb + doc) + nok)$
   $refresh = (slife/sref - 1) * (sub + sok + not + rlroot + pres * (rlitem + mpb + doc) + nok)$
   $termination = (sub + sok + not + rlroot + pres * (rlitem + mpb + doc) + nok)$

   If partial-state RLMI documents are used, the change messages are:
   $change = pres * (pch - 2) * (not + rlroot + (rlitem + mpb + doc) + nok)$

   If conditional notifications are applied, the refresh and termination messages are:
   $refresh = (slife/sref - 1) * (sub + sok)$
   $termination = sub + sok$

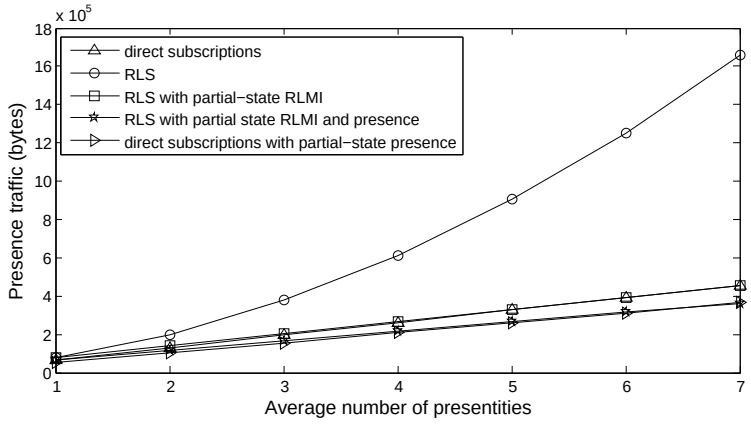


Figure 7.1: Presence traffic with and without RLS, and partial-state documents

7.2 Estimation of RLS Traffic on the Access Link

Based on the mathematical formulas presented in Section 7.1, Figure 7.1 shows the presence traffic that is generated when the user employs direct subscriptions and an RLS, with and without conditional notifications and partial-state documents. A non-optimized RLS is clearly the least efficient technique. The performance of non-optimized direct subscriptions and an RLS optimized with partial-state RLMI documents is very similar, and hence direct subscriptions are recommended for the sake of simplicity. Only when the RLS combines partial-state presence and RLMI documents, it is preferable to non-optimized direct subscriptions. If all the watchers notify partial-state presence documents, direct subscriptions are still recommended because of their simplicity. However, guaranteeing that all the watchers notify partial-state presence may not be possible in real scenarios. On the contrary, an RLS is a centralized server that can ensure the implementation of optimizations on the user's access link more easily.

Figure 7.2 compares non-optimized direct subscriptions with an RLS that notifies partial-state RLMI documents. It can be seen that the performance of the RLS worsens as the number of presence changes increases. Direct subscriptions are preferable to an RLS from 4 presence changes per hour. The number of presentities affect more seriously direct subscriptions than the RLS.

Figure 7.3 shows the number of presentities up to which non-optimized direct subscriptions are preferable to an RLS. The absence of Y coordinate for an X coordinate

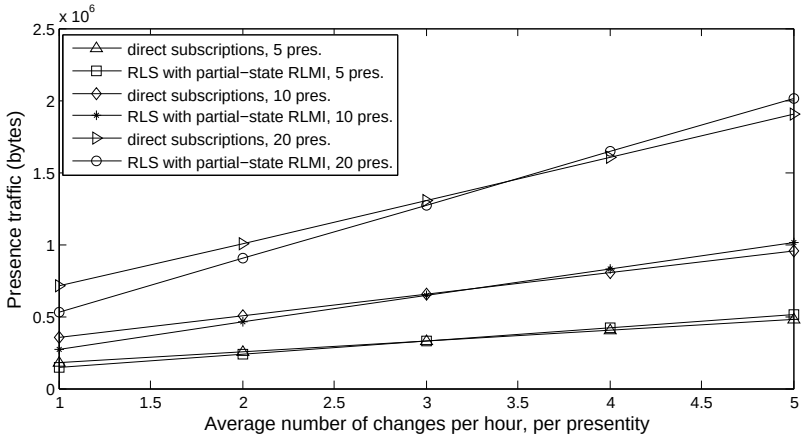


Figure 7.2: Traffic of an RLS with partial-state RLMI documents, and direct subscriptions

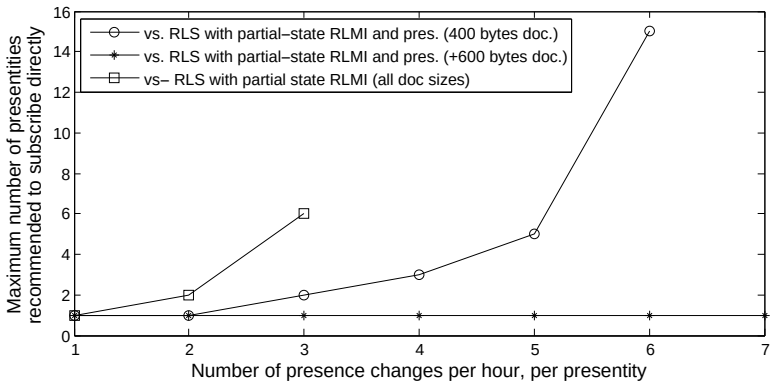


Figure 7.3: Maximum number of presentities recommended to use direct subscriptions

means that direct subscriptions are always preferable (i.e., the maximum number of presentities is infinite). When the RLS does not implement partial-state presence documents and presentities change their presence 3 times per hour, direct subscriptions are recommended up to 6 presentities. If the frequency of change is higher, direct subscriptions are always recommended regardless of the document size. When the RLS notifies partial-state presence and the size of presence documents is larger than 600 bytes, direct subscriptions are discouraged. However, when there are more than 6 changes per hour and the document size is small (400 bytes), direct subscriptions are always recommended. These conditions may be satisfied in LBSs that only include geographical coordinates into presence documents (which usually take less than 400 bytes).

As an RLS makes it easier to implement optimization techniques, let us assume that

the watchers do not optimize presence traffic but the RLS does implement conditional notifications and partial-state RLMI documents. We refer to conditional notifications as Notify Optimization (NO) for distinguishing between this technique and Common Notify (CN), which is an strategy studied in Section 5. Figure 7.4 compares both alternatives' presence traffic with 5, 10 and 20 presentities as the number of presence changes increases. One may observe that the amount of traffic saved by the RLS is not significative and may not compensate for its complexity. Even, when the 5, 10, and 20 presentities change their presence more than 4, 5, and 6 times per hour, respectively, non-optimized direct subscriptions are more efficient than the optimized RLS. Now, let us assume that the watchers do optimize presence traffic by means of conditional notifications and partial-state notifications. Figure 7.5 compares the presence traffic generated by such optimized direct subscriptions and a likewise optimized RLS. It also shows the direct subscriptions' traffic when the watchers only implement partial-state presence. It can be seen that the optimized direct subscriptions are more efficient than the optimized RLS from 3 changes per hour. Direct subscriptions are more efficient as the number of presence changes increases. They save 6.6% and 22.8% of the RLS traffic with 3 and 10 presence changes per hour, respectively. If the watchers do not implement conditional notifications but they do implement partial-state documents and there are more than 6 hours per hour, direct subscriptions are still preferable. However, non-optimized direct subscriptions always generate more traffic than the optimized RLS. In this case, the RLS saves 69.4% and 29% of the direct subscriptions traffic with 1 and 10 changes per hour, respectively.

7.3 Conclusions

RLS is a widely-adopted optimization of presence traffic in mobile applications that importantly reduces the number of presence messages on access links. However, this server introduces complexity and generates much larger presence documents, which may not compensate for the saving of messages. Network administrators should carefully study three parameters before configuring watcher applications to subscribe to an RLS: average number of presentities, number of presence changes and size of presence documents. We estimated the access link's presence traffic when a user subscribes to both an RLS and his or her presentities directly. We studied these two alternatives with

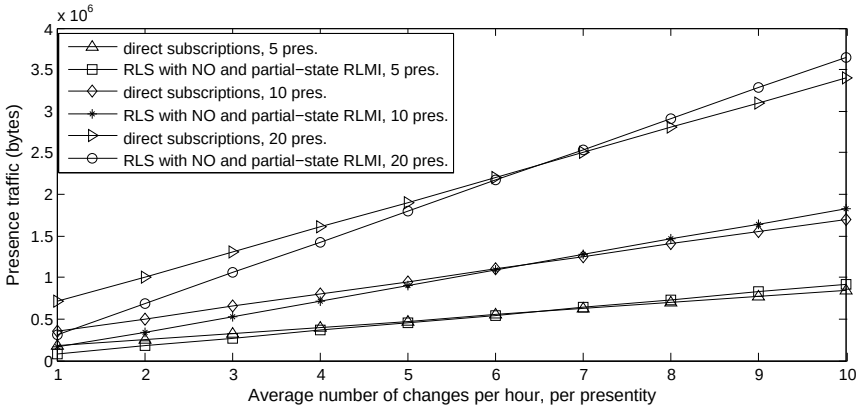


Figure 7.4: Traffic of direct subscriptions, and an RLS with partial-state RLMI documents and conditional notifications (NO)

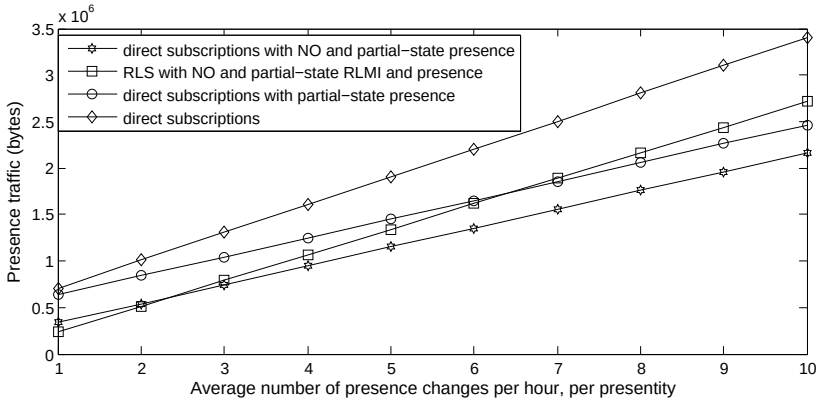


Figure 7.5: Traffic of optimized direct subscriptions, and an RLS with partial-state presence documents and conditional notifications (NO)

and without two optimizations: partial-state documents and conditional notifications. The reported results revealed that the RLS is not efficient at reducing presence traffic. An RLS without any optimization is strongly discouraged. The performance of the RLS decreases dramatically as the number of presence changes increases. Partial-state presence documents ease the impact of presence changes but having an RLS handle each presentity's partial-state changes involves some computational cost. Even when the RLS is optimized with the above-mentioned strategies, subscribing to presentities directly is more efficient under many circumstances. If there is a guarantee that all the watchers implement conditional notifications and partial-state documents, direct subscriptions are always more efficient than an optimized RLS excepting when presen-

tities only change their presence once per hour. Even when direct subscriptions are not optimized at all, if presence changes occur frequently and presence documents are small, which is probable in some LBSs, direct subscriptions are more efficient than an optimized RLS.

8

Queueing System and Adaptive QoS Mechanism for Controlling the Rate of Presence Publications and Notifications

Presence protocols normally adopt a push approach because of the asynchronous nature of presence changes. This means that watchers are notified of presence changes, rather than querying the presence service periodically. Such a timely dissemination of presence changes entails much signaling traffic, and therefore scalability issues in presence-enabled applications, as described in Section 2.9.4. Even if presence updates are discarded, signaling traffic for keeping presence subscriptions alive may be quite considerable. The authors of [179] analytically estimate presence traffic, thereby concluding that this traffic may account for more than 50% of the total traffic handled by the CSCF in the IMS. The IMS, which is the foundation of NGNs, evolves mobile operators towards an all IP technology for the support of advanced multimedia services (see Section 2.4). The performance analysis in [201] shows that SIP signaling traffic introduces long transmission delays on the UMTS network. Much of this delay is due to excessive traffic on the network core, and hence optimizing this traffic is necessary for providing multimedia services in real time.

Section 2.9.4 introduces some proposals for reducing subscription-related presence traffic. Since presence notifications constitute a large part of the excessive traffic gener-

ated by presence applications [181][182], most of these proposals tackle the number of NOTIFY messages. As described in Section 2.7.4, conditional notifications [147], notification filtering [140], partial-state presence documents [139] and resource list subscriptions (i.e., RLS) [121] are proposals that have emerged from the IETF SIMPLE WG. SIMPLE allows notifiers to limit the rate of notifications [110] but does not mandate any particular rate control mechanism. The authors of [183] describe a mechanism that allows watchers to specify the desirable maximum rate when subscribing. The drawback of this mechanism is that it requires handling a different notification buffer for each watcher, which may not be scalable in large-scale presence services. Some authors have proposed queuing systems for controlling the presence notification rate 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, and hence it save some 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 introduce 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 so as to save presence notifications. The main difference between this mechanism and the above-mentioned 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. This means that the watchers' needs on information consistency are not considered when delaying notifications.

As described in Section 2.9.5, scalability becomes even more critical in large-scale presence federation scenarios, where millions of users in a domain subscribe to millions of users in other federated domains. In these scenarios, SIMPLE generates millions of inter-domain subscriptions, a different one for each (watcher, presentity) pair. Section 2.9.5 shows that large-scale presence services may inject dozens of terabytes into the network core during 8 hours. This amount does not include access link traffic and is calculated under quite moderate assumptions. In such scenarios, optimizing the number and size of notifications sent through presence subscriptions is not enough. Reducing the number of inter-domain subscriptions is necessary for scaling down inter-domain traffic. Section 5 proposes the optimization technique Common Subscribe (CS) for reducing the number of subscriptions between two federated domains to one for each watched presentity. Section 5.1.5 estimates the presence traffic generated when this optimization, among others, is applied. This shows that CS saves 80% of the presence traffic of the studied use case, resulting more efficient than other studied inter-domain traffic optimizations. Nevertheless, even with CS or other optimization techniques, the amount of presence notifications injected into federated domains may be harmful.

On the other hand, each presence publication normally involves the PS notifying all the watchers authorized to see the publication. Since the PS is the intermediary in any presence publication and notification, presence publications may demand a considerable part of the PS capacity. As described in Section 2.9.6, presence publications also may overconsume the wireless user devices' battery life and processing resources, as well as the radio access bandwidth, specially in LBSs. Section 2.4 shows that applying a minimum throttling time between two consecutive publications can dramatically reduce presence traffic on the access link. The reported results show that this strategy saves around 41%, 61%, and 71% of presence traffic with minimum intervals of 5, 10, and 15 minutes, respectively. However, controlling the rate of presence publications and notifications may result in watcher applications keeping obsolete information, as described in Section 2.9.7. This may make the presence service useless since its success is actually due to the instantaneous knowledge of presence changes; presence applications by its very nature have strict real-time constrains. Watcher applications or the users themselves may take inadequate decisions or assumptions based on wrong presence information. Thus, Section 4.3 proposes adapting the presentities' publication rates according to a probabilistic model of changes in their presence information.

This kind of rate is referred to as sojourn-based rate. This strategy allows optimizing presence traffic on the network access link while avoiding too long publication delays.

This section proposes a novel queuing mechanism for controlling the rate of both notifications and publications in presence federation scenarios. This mechanism provides: a) optimization of access link and network core traffic in a correlated way, b) avoidance of inconsistency between the presentities' latest presence information and the information kept by watcher domains, and c) guarantee that the PS performance is not degraded. This queueing mechanism overcomes the limitations of sojourn-based rates. Calculating this kind of interval is not viable without a Markov chain that models presence changes. Moreover, a sojourn-based interval is based on a delay factor that represents how long a presence change publication should be delayed. Section 4.3 does not tackle how this delay factor can be deduced. Section 4.3.1 assumes the delay factor to be half of the maximum delay in notifications acceptable by watchers. This assumption is however too simplistic. The PS plays the leading role in notifying watchers, and hence it introduces the better part of the total time taken to notify watchers of a presence change. The PS may even delay notifying presence changes with some rate control mechanism [110].

Section 8.1 discusses about the design of the proposed queuing system. Sections 8.2, 8.3, and 8.4 describe a mathematical model for this system. Section 8.5 describes an algorithm for dynamically setting the proposed queuing system's parameters based on some QoS conditions. This section also evaluates this algorithm's performance under different circumstances. Section 8.6 addresses a limitation of this algorithm by combining it with sojourn-based intervals. Lastly, Section 8.7 presents the main findings of this study.

8.1 Design

Figure 8.1 shows the PS operation modules involved in receiving presence publications and notifying watchers. White boxes represent basic tasks that are always present at any PS. Presence publications arrive to the Publication Receiver (PR) module, which performs presentity authorization and acknowledges them. The Notification Generation (NG) module is responsible for generating the notifications that need to be sent out. In Figure 8.1, 'Nxy' denotes a notification from a presentity 'x' to a watcher

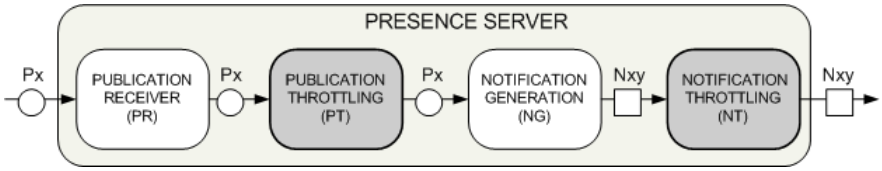


Figure 8.1: Presence Server components

'y'. In federation scenarios optimized with CS, when a presentity publishes a presence change, only one notification is sent to each watcher domain. We consider a watcher domain as any federated domain to which one or more of the presentity's watchers belong. Thus, in Figure 8.1, 'y' denotes a watcher domain rather than an individual. To limit the rate at which notifications are sent out, the PS may apply a control mechanism to either publications or notifications, that is, the PT or NT module in Figure 8.1, respectively. Some authors propose throttling publications [189][188], while others notifications [183]. The main advantage of the former is that presence publications can be aggregated at the PT module. This may reduce the number of publications that go through the notification process, thereby optimizing the PS processing resources. However, throttling notifications enable watcher-based notification control to be conducted, which is reasonable since watchers (i.e., in the presented study, watcher domains) may wish to be notified at different rates.

Figure 8.2 shows the proposed queuing system, which is capable of controlling the rate of both notifications and publications. The PS performs fine-grained notification rate control by adapting to the needs of each watcher domain. The NT module therefore consists of a notification buffer for each watcher domain, each controlled by a different maximum notification rate θ_w. The PR module only queues arriving presence publications and forwards them to the NT module. We omit the process of generating notifications for the sake of simplicity. As mentioned previously, throttling presence publications is recommended to prevent notifications from being generated. We delegate this process to the presentities themselves, which we denominate as source-throttling. Section 4.1.1 discusses about limiting the rate of publications. The presented approach prevents not only presence notifications in a more scalable way but also presence publications on the access link, which is beneficial for wireless presence applications. The system is designed to provide a rate control of publications and notifications adapted

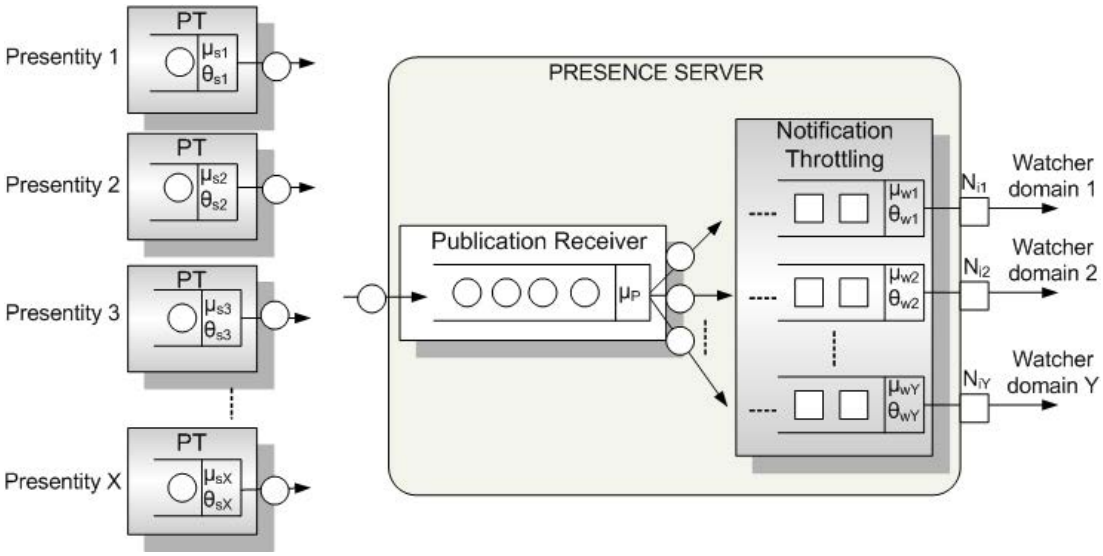


Figure 8.2: Proposed queuing system

to presentities and watcher domains while keeping the PS scalable. Each presentity throttles its own publications by means of a queue of size 1. The PS only handles one publication queue and as many notification queues as the number of watcher domains subscribed to any of the presentities under the PS control. The maximum size of the publication queue is the total number of presentities. The maximum size of a watcher domain's notification queue is the number of presentities that notify this domain. There can only be one message for each presentity in any queue. When a presentity's message arrives to a queue and finds that there is an older message of this presentity waiting in the queue, the arriving message replaces the older one.

The level of traffic optimization is determined by the maximum rates at the presentities and at the NT module. The lower the rates, the more presence traffic that is saved. However, the notification delay increases as these rates decrease. Thus, publication and notification rates should be carefully selected in order to find a tradeoff between traffic optimization and presence information consistency. Section 8.5 proposes an algorithm for dynamically setting these rates while ensuring a maximum delay in notifying the watcher domains.

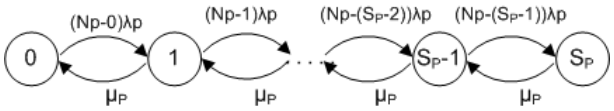


Figure 8.3: State transition diagram of the publication queue

8.2 Analytical Modeling of Publication Receiver

Let us assume that each presentity changes its presence information at times that are exponentially distributed with mean λ_p^{-1}. Thus, the presence change process of each presentity conforms to Poisson distribution with mean λ_p. Let us assume that the presence change processes of different presentities are independent of each other, and the total number of presentities is N_p. As the sum of N independent Poisson variables with the same mean λ is a Poisson variable with mean $N\lambda$ [199], we can conclude the arrival process of PUBLISH messages at the PS conforms to Poisson distribution with mean λ_P, as shown in (8.1):

$$\lambda_P = N_p \lambda_p \tag{8.1}$$

Therefore, the PR module can be modeled as an $M/M/1/S_P$ queuing system, which we refer to as the publication queue. Service times are exponentially distributed with mean μ_P^{-1} and PUBLISH messages are served by a single server on an FCFS (First Come First Served) basis. The queue capacity is denoted by S_P. Let $N_P(t)$ be the number of PUBLISH messages waiting in the queue. $N_P(t)$ is increased by one unit when a PUBLISH message is put into the queue and decreased by one when a PUBLISH message is processed by the server. Since the Poisson process prohibits the possibility of having more than one arrival in Δt, and the exponential service time ensures that there is at most one departure in Δt, $N_P(t)$ is a birth-death process because it can only go to its neighboring states, $(N_P(t)+1)$ or $(N_P(t)-1)$ in a time interval Δt [199]. Figure 8.3 shows the state transition graph of $N_P(t)$.

An arriving PUBLISH message is queued only if there is not other message in the queue that belongs to the same presentity; otherwise, the arriving message replaces the older one. Since the presence change process of each presentity is independent, each presentity has the same probability of owning a message that is waiting in the queue. Thus, the probability that a PUBLISH message arrives in the queue and does not find

a waiting message that belongs to its presentity is $1 - \frac{N_P(t)}{N_p}$. The effective PUBLISH arrival rate at the PS is therefore $(1 - \frac{N_P(t)}{N_p})\lambda_P$. Substituting (8.1) into the previous expression, we obtain the effective PUBLISH arrival rate, as shown in (8.2).

$$\lambda_P(t) = (N_p - N_P(t))\lambda_p \tag{8.2}$$

From the theory of Markov chains [199], it follows that $\{N_P(t), t \geq 0\}$ has a unique equilibrium distribution. Let $p_k(t)$ denote the probability of k messages in the queue, as shown in (8.3). The limitation of $p_k(t)$ as t goes to inifinity is given by (8.4).

$$p_k(t) = P\{N_P(t) = k\}, 0 \leq k \leq S_P \tag{8.3}$$

$$p_k = \lim_{t\to\infty} p_k(t), 0 \leq k \leq S_P \tag{8.4}$$

From Figure 8.3, we obtain the following equilibrium state transition equations:

$$p_0\lambda_0 = p_1\mu_P \tag{8.5}$$

$$p_k(\lambda_k + \mu_P) = p_{k-1}\lambda_{k-1} + p_{k+1}\mu_P, 0 < k < S_P \tag{8.6}$$

$$\sum_{k=0}^{S_P} p_k = 1 \tag{8.7}$$

where λ_k is defined as shown below:

$$\lambda_k = (N_p - k)\lambda_p, 0 \leq k \leq S_P \tag{8.8}$$

This linear equations array (8.5)-(8.7) can be solved recursively, which leads to the following solution:

$$p_k = \begin{cases} \frac{1}{1+\upsilon} & k = 0 \\ \frac{\prod_{i=0}^{k-1}(N_p-k)}{1+\upsilon}(\frac{\lambda_p}{\mu_P})^k & 0 < k \leq S_P \end{cases} \tag{8.9}$$

where $\upsilon = \sum_{j=0}^{S_P-1}\prod_{i=0}^{j}(N_p - i)(\frac{\lambda_p}{\mu_P})^{j+1}$

8.2.1 Loss Probability of PUBLISH Messages

The publication queue loss probability, LP_P, is the probability that a PUBLISH arrives at the PS and is discarded because the queue is full. It happens when an arriving PUBLISH message finds that the queue is full and none of the S_P PUBLISH messages that are waiting in the queue belongs to its presentity. Let $a_k(t)$ be the probability of the state $\{N_P(t) = k\}$ as seen by an arriving PUBLISH message. From the PASTA property [199], for Poisson processes, the probability of the state as seen by an arriving customer $a_k(t)$ is the same as the probability of the state as seen by an outside observer, that is, $p_k(t)$. However, the publication queue input process λ_P is time-dependent as shown in (8.2), and hence the PASTA property does not hold. We define a_k as:

$$a_k = \lim_{t\to\infty} a_k(t), 0 \leq k \leq S_P \tag{8.10}$$

Since a_k is the probability of the queue having k waiting messages at the time a new PUBLISH message arrives, it can be calculated as the proportion of arrivals that occur when the queue has k messages, as shown in (8.11).

$$a_k = \frac{\lambda_k p_k}{\sum_{i=0}^{S_P} \lambda_i p_i}, 0 \leq k \leq S_P \tag{8.11}$$

Substituting (8.8) into (8.11), we obtain:

$$a_k = \frac{(N_p - k)p_k}{\sum_{i=0}^{S_P}(N_p - i)p_i}, 0 \leq k \leq S_P \tag{8.12}$$

As mentioned previously, the loss probability LP_P is the probability that an arriving PUBLISH message finds the queue full, that is, a_{S_P}:

$$LP_P = a_{S_P} = \frac{(N_p - S_P)p_{S_P}}{\sum_{i=0}^{S_P}(N_p - i)p_i} \tag{8.13}$$

Note that when S_P is equal to N_p, $LP_P = 0$, and hence the system is lossless. In this case, the publication queue can accommodate any arriving PUBLISH message. The queue capacity S_P therefore does not need to be larger than N_p, that is the number of presentities.

8.2.2 Average Length of the Publication Queue

Let us define L_P as the number of PUBLISH messages waiting in the queue, and hence $P\{L_P = k\} = p_k, 0 \leq k \leq S_P$. Therefore, the average publication queue length is:

$$\overline{L_P} = E[L_P] = \sum_{i=1}^{S_P} i p_i \tag{8.14}$$

8.2.3 Average Waiting Time of PUBLISH Messages

From Little's theorem [199], the average number of customers ($\overline{L_P}$) in a steady-state queueing system is the product of the average arrival rate of customers entering the system ($\overline{\lambda_P}$) and the average time a customer spends in that system ($\overline{W_P}$). Thus, the average waiting time of a PUBLISH message can be calculated as follows:

$$\overline{W_P} = \frac{\overline{L_P}}{\overline{\lambda_P}} = \frac{\sum_{i=1}^{S_P} i p_i}{\sum_{i=0}^{S_P} (N_p - i)\lambda_p p_i} \tag{8.15}$$

8.2.4 Mathematical Analysis

This section studies the correctness of the mathematical model for the publication queue described previously. We assume the publication queue to be lossless, which means that its length is equal to the number of presentities ($N_p = S_P$), for investigating the effects of different parameters on its performance. Some results below rely on traffic intensity, which is a measure of the congestion of queuing systems. Traffic intensity is defined by the number of arrivals per unit time over the number of departures per unit time [199]. Thus, let ρ_P be the publication queue traffic intensity is:

$$\rho_P = \frac{N_p \lambda_p}{\mu_P} \tag{8.16}$$

Figure 8.4 shows the publication queue average waiting time $\overline{W_P}$ as each presentity's PUBLISH arrival rate λ_p increases, for different number of presentities N_p. The increase in λ_p results in more messages waiting in the queue, and hence $\overline{W_P}$ increases. When λ_p is small, $\overline{W_P}$ increases significantly and, when λ_p gets larger, the increase in $\overline{W_P}$ decreases. We explain this phenomenon as follows. At the beginning the queue length increases rapidly because there are few messages in the queue. When the queue length increases, the arriving PUBLISH message will probably replace the older one waiting

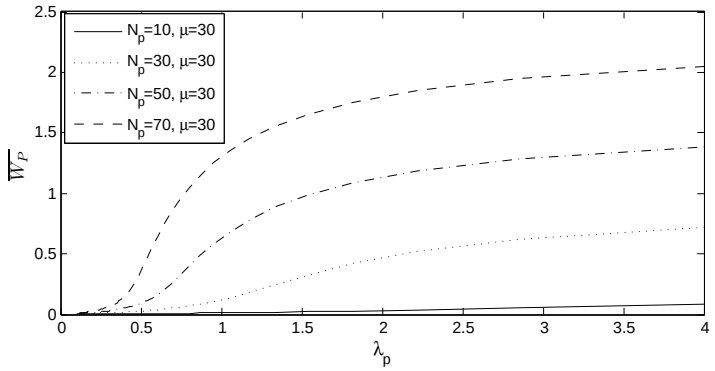


Figure 8.4: Effect of λ_p on $\overline{W_P}$

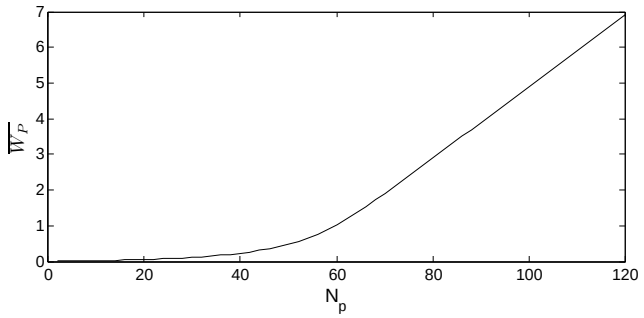


Figure 8.5: Effect of N_p on $\overline{W_P}$ ($\lambda_p = 0.2$, $\mu_P = 10$)

in the queue. As λ_p increases, the increasing rate of $\overline{W_P}$ slows down and gets closer to the maximum average waiting time, which occurs when the arrival PUBLISH message finds $(N_p - 1)$ messages waiting in the queue. The increase in N_p intensifies the effect of λ_p on $\overline{W_P}$, since this increase results in more messages waiting in the queue (i.e., the total arrival process of PUBLISH messages at the queue in $N_p\lambda_p$).

Figure 8.5 plots the effect of N_p on $\overline{W_P}$ when $\lambda_p = 0.2$ and $\mu_P = 10$. It can be seen that $\overline{W_P}$ continuously increases as N_p increases. This occurs because we set $S_P = N_p$, and hence when S_P increases more messages are placed into the queue and $\overline{W_P}$ therefore increases. When $N_p > 50$, the traffic intensity ρ_P, which is shown in (8.16), is higher than 1, and hence the average PUBLISH arrival rate exceeds the average service time. Thus, the increasing rate of $\overline{W_P}$ augments rapidly with high values of N_p.

Figure 8.6 plots the effect of λ_p on the average publication queue length $\overline{L_P}$ with different values of N_p. The number of messages that arrive to the queue increases

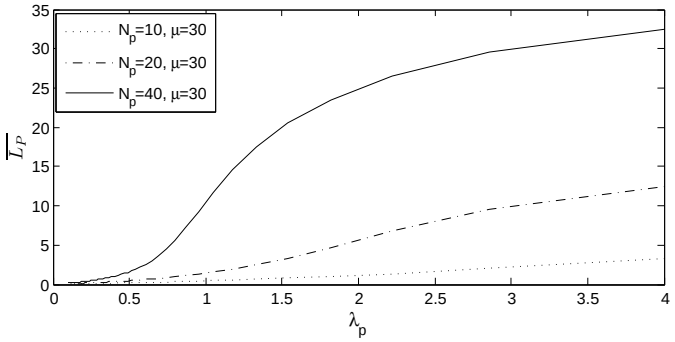


Figure 8.6: Effect of λ_p on $\overline{L_P}$

with λ_p, which leads to the increase in $\overline{L_P}$. When λ_p is small, $\overline{L_P}$ increases rapidly since there are few messages in the queue and an arrival PUBLISH message is highly probable to be queued. When $\overline{L_P}$ gets larger, there are more messages in the queue and an arrival PUBLISH message is more probable to replace an older one waiting in the queue. As λ_p increases, $\overline{L_P}$ gets closer to its maximum length. The increase in N_p intensifies the effect of λ_p on $\overline{L_P}$, since this increase results in more messages waiting in the queue and S_P is equal to N_p.

Figure 8.7 shows the distribution of L_P with different values of intensity traffic ρ_P and $N_p = 20$. When $\rho_P \leq 1$, L_P tends to be zero. When $\rho_P \geq 1$, the arrival rate of PUBLISH messages ($N_p\lambda_p$) is greater than the service rate (μ_P). Thus, L_P increases with ρ_P. Figure 8.7 helps in estimating the minimum queue length necessary to guarantee that the publication queue is lossless. For instance, when $\rho_P = 1.5$, the probability that the queue length is higher than 14 tends to be zero. Thus, setting the queue length S_P to 14 is sufficient to keep the message loss probability under a certain threshold while optimizing the PS resources.

Figure 8.8 plots the effect of S_P on the publication queue loss probability LP_P when $N_p = 20$. As S_P increases, LP_P decreases because more messages can be accommodated. When $S_P = 20$, the publication queue is lossless and therefore LP_P is zero. LP_P is higher for higher values of λ_p, since it involves more messages arriving at the queue.

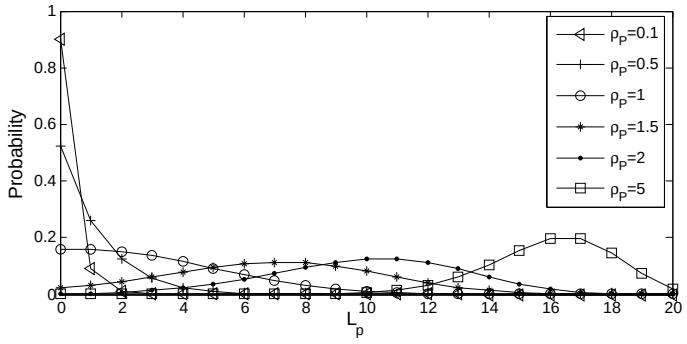


Figure 8.7: Distribution of L_P with different values of ρ_P

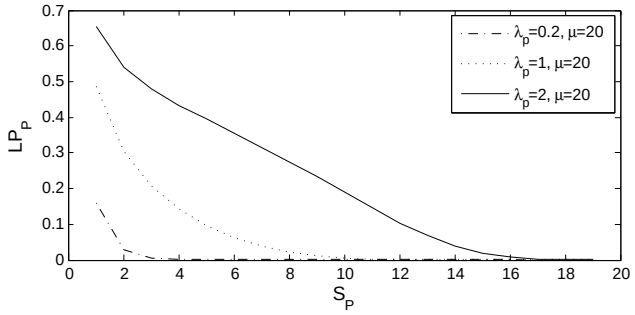


Figure 8.8: Effect of S_P on LP_P

8.3 Analytical Modeling of Notification Throttling

The TN module performs watcher-domain-based notification throttling by setting a minimum passive time between any two consecutive notifications that are sent to each watcher domain. When the PS sends a NOTIFY message to a watcher domain, it starts a throttling timer, and during this interval any other arriving NOTIFY message to the watcher domain is queued. We study the publication queue and the notification queue independently rather than in tandem. Although these queues are dependent since the notification queue input is the publication queue output, we perform an approximate mathematical analysis for the sake of simplicity. This assumption greatly simplifies the mathematical analysis of the overall queuing system. When a presentity's PUBLISH message is processed by the publication queue, a NOTIFY message is generated to the watcher domains subscribed to the presentity. Thus, the number of NOTIFY messages inserted into a watcher domain's notification queue depends on the number of

presentities to which this domain is subscribed and the rate at which these presentities send PUBLISH messages. Let $\delta_w \in [0,1]$ denote the percentage of the total number of presentities to which the domain w is subscribed. We refer to δ_w as the subscription factor of the domain w. Thus, $N_p\delta_w$ is the number of presentities that notify the watcher domain w. Let $N_w(t)$ be the number of NOTIFY messages waiting in the notification queue of the watcher domain w at time epoch t. As described in Section 8.2, the arrival process of PUBLISH messages of each presentity conforms to Poisson distribution with mean λ_p. Thus, we assume that the arrival process of NOTIFY messages targeted at the watcher domain w conforms to Poisson distribution with mean λ_w, as shown in (8.17):

$$\lambda_w = N_p\delta_w\lambda_p \tag{8.17}$$

There can only be one NOTIFY message waiting in a notification queue for each presentity. When an arriving NOTIFY message finds an older message in the queue that belongs to its presentity, the arriving message replaces the older one. We therefore deduce the effective arriving rate $\lambda_w(t)$ of NOTIFY messages at a notification queue as in Section 8.2, thereby obtaining:

$$\lambda_w(t) = (N_p\delta_w - N_w(t))\lambda_p \tag{8.18}$$

The notification queue of any watcher domain w is described by four parameters: the service rate (μ_w); the throttling time (θ_w^{-1}); the queue capacity (S_w), and the subscription factor (δ_w). Service times are exponentially distributed with mean μ_w^-1. NOTIFY messages are served by a single server on an FCFS basis. Figure 8.9 shows an approximate state transition graph for a notification queue. Whenever the server processes a NOTIFY message, it goes on vacation during the passive time θ_w^{-1}. This ensures that the notification output rate for the watcher domain is never higher than θ_w. Let $B_w(t) = 1$ and $B_w(t) = 0$ denote the events that the server is busy and on throttling vacation at time t, respectively. Setting

$$p_{1,k}^w(t) = P\{N_w(t) = k, B_w(t) = 1\}, 0 \le k \le S_w \tag{8.19}$$

$$p_{0,k}^w(t) = P\{N_w(t) = k, B_w(t) = 0\}, 0 \le k \le S_w \tag{8.20}$$

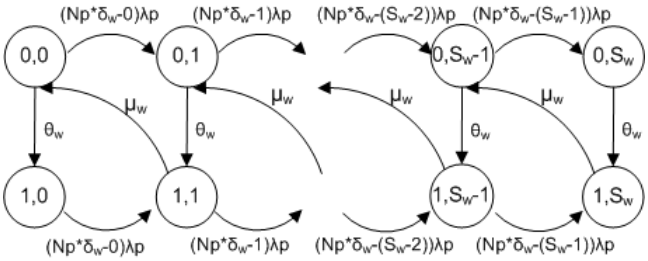


Figure 8.9: State transition diagram of a notification queue

From the Markov theory [199], it follows that $\{N_w(t), B_w(t), t \geq 0\}$ has a unique steady-state probability distribution. Setting

$$p_{1,k}^w = \lim_{t\to\infty} p_{1,k}^w(t), 0 \leq k \leq S_w \tag{8.21}$$

$$p_{0,k}^w = \lim_{t\to\infty} p_{0,k}^w(t), 0 \leq k \leq S_w \tag{8.22}$$

From Figure 8.9, we obtain the following state transition equations:

$$p_{0,0}^w(\lambda_0^w + \theta_w) = p_{1,1}^w \mu_w \tag{8.23}$$

$$p_{0,k}^w(\lambda_k^w + \theta_w) = p_{0,k-1}^w \lambda_{k-1}^w + p_{1,k+1}^w \mu_w, 0 \leq k \leq S_w \tag{8.24}$$

$$p_{1,0}^w \lambda_0^w = p_{0,0}^w \theta_w \tag{8.25}$$

$$p_{1,k}^w(\lambda_k^w + \mu_w) = p_{1,k-1}^w \lambda_{k-1}^w + p_{0,k}^w \theta_w, 0 \leq k \leq S_w \tag{8.26}$$

$$\sum_{k=0}^{S_w} p_{0,k}^w + p_{1,k}^w = 1 \tag{8.27}$$

where $\lambda_k^w = (N_p\delta_w - k)\lambda_p$. This linear equations array (8.23)-(8.27) cannot be solved by recursive substitutions. Therefore, we use the tool *Matlab* to obtain the state probabilities $p_{1,k}^w$ and $p_{0,k}^w, k = 0, 1, ..., S_w$.

8.3.1 Loss Probability of NOTIFY Messages

The notification queue loss probability, LP_w, is the probability that a NOTIFY message arrives at the notification queue of the watcher domain w and is discarded because the queue is full. This happens when an arriving NOTIFY message finds that the queue is

full and none of the S_w NOTIFY messages that are waiting in the queue belongs to its presentity. We can deduce LP_w as in Section 8.2.1, thereby obtaining:

$$LP_w = \frac{(N_p\delta_w - S_w)(p^w_{0,S_w} + p^w_{1,S_w})}{\sum_{i=0}^{S_w}(N_p\delta_w - i)(p^w_{0,i} + p^w_{1,i})} \tag{8.28}$$

When S_w is equal to $N_p\delta_w$, $LP_w = 0$, and hence the system is lossless. The queue capacity S_w therefore does not need to be larger than $N_p\delta_w$, that is, the number of presentities that notify the watcher domain.

8.3.2 Average Length of the Notification Queue

Let us define L_w as the number of NOTIFY messages waiting in the notification queue of the watcher domain w. Thus, $P\{L_w = k\} = p^w_{0,k} + p^w_{1,k}, 0 \leq k \leq S_w$. Therefore, the average notification queue length is:

$$\overline{L_w} = E[L_w] = \sum_{i=1}^{S_w} i(p^w_{0,i} + p^w_{1,i}) \tag{8.29}$$

8.3.3 Average Waiting Time of NOTIFY Messages

From Little's theorem [199], the average number of customers ($\overline{L_w}$) in a steady-state queueing system is the product of the average arrival rate of customers entering the system ($\overline{\lambda_w}$) and the average time a customer spends in that system ($\overline{W_w}$). Thus, the average waiting time of a NOTIFY message in the watcher domain's notification queue can be calculated as follows:

$$\overline{W_w} = \frac{\overline{L_w}}{\overline{\lambda_w}} = \frac{\sum_{i=1}^{S_w} i(p^w_{0,i} + p^w_{1,i})}{\sum_{i=0}^{S_w}(N_p\delta_w - i)\lambda_p(p^w_{0,i} + p^w_{1,i})} \tag{8.30}$$

8.3.4 Mathematical Analysis

This section studies the correctness of the mathematical model for a watcher domain's notification queue described previously. We assume that the notification queue for a watcher domain is lossless, which means that its capacity is equal to the number of presentities that notify this domain ($N_p\delta_w = S_w$). We investigate the parameters that affect this queue performance, namely λ_p, N_p, δ_w, and θ_w. In this analysis, we consider traffic intensity, which is the number of arrivals per unit time over the number of

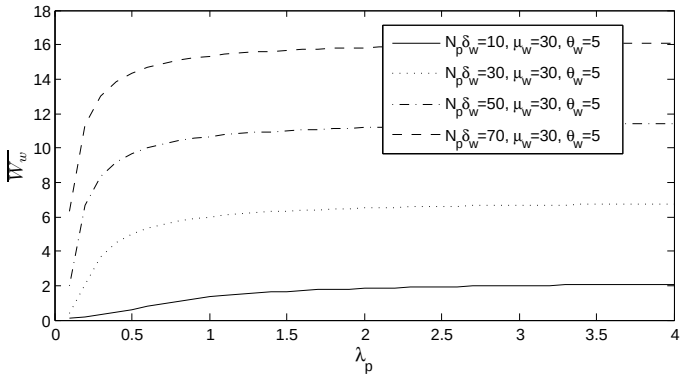


Figure 8.10: Effect of λ_p on $\overline{W_w}$

departures per unit time [199]. Thus, let ρ_w be the notification queue traffic intensity, as shown below:

$$\rho_w = \frac{N_p \delta_w \lambda_p}{\mu_w} \tag{8.31}$$

Figure 8.10 plots the effect of λ_p on $\overline{W_w}$. The increase in λ_p results in more messages waiting in the queue, and hence $\overline{W_w}$ increases. One may notice that $\overline{W_w}$ increases rapidly when $\lambda_p < 0.5$. When $\lambda_p > 0.5$, the increasing rate of $\overline{W_w}$ decreases towards an stationary value. We explain this phenomenon as follows. When λ_p is small, the queue length increases rapidly since there are few messages in the queue, and hence an arriving message is not probable to replace an older one. Moreover, the high rate at which $\overline{W_w}$ increases is due to the fact that any time a notification is processed the queue goes to a passive time θ_w^{-1}, during which notifications are not processed. Thus, $\overline{W_w}$ rapidly gets close to its maximum value, which occurs when the arriving notification finds $(N_p\delta_w - 1)$ messages in the queue. Obviously, θ_w is expected to affect $\overline{W_w}$ dramatically.

Figure 8.11 shows the effect of θ_w on $\overline{W_w}$. It can be seen that $\overline{W_w}$ is inversely proportional to θ_w. We explain this phenomenon as follows. Since θ_w is the rate at which the server moves out from passive intervals in which no messages are served, the increase in θ_w reduces the time during which messages are queued without being processed. Thus, the increase in θ_w leads to the decrease in the notification queue length, thereby decreasing $\overline{W_w}$. One may notice that the effect of θ_w on $\overline{W_w}$ is considerable when its value ranges from 0.5 to 2. This range means that any time when the server

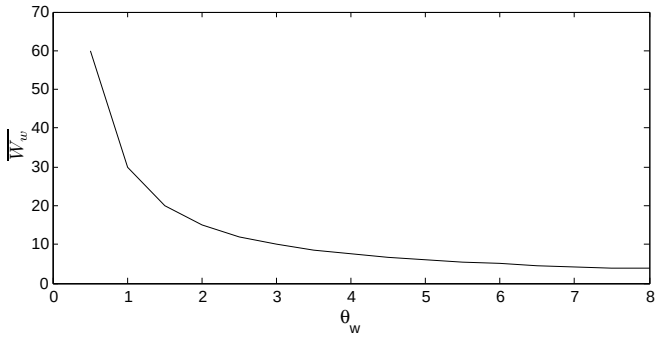


Figure 8.11: Effect of θ_w on $\overline{W_w}$ ($\mu_w = 30$, $\lambda_p = 1$, $N_p\delta_w = 30$)

sends a message to the watcher domain, it waits from 2 minutes to 30 seconds, respectively, to process the next message. Such times are very long and lead to a dramatic increase in the queue length, which results in high values of $\overline{W_w}$. When $\theta_w > 2$, the passive time θ_w^{-1} is smaller and fewer messages are therefore queued. As θ_w increases, $\overline{W_w}$ gets closer to its minimum value, which occurs when θ_w is so high that no messages are queued during the passive intervals.

Figure 8.12 plots the effect of δ_w on $\overline{W_w}$. We assume $\mu_w = 10$, $\lambda_p = 0.2$, $N_p = 120$ and $\theta_w = 8$. As δ_w is the percentage of presentities that notify the watcher domain and we set the system to be lossless ($S_w = N_p\delta_w$), the queue length increases with δ_w, which leads to the continuous increase in W_w. It can be seen than, when $\delta_w < 0.2$, the increasing rate of $\overline{W_w}$ is lower. We explain this phenomenon as follows. As δ_w increases, the arrival rate of notifications at the queue increases. However, the service rate μ_w remains the same, which leads to the rapid increase in the queue length, and hence $\overline{W_w}$ reaches its maximum value (i.e., when the arriving notification finds $(N_p\delta_w - 1)$ messages in the queue).

Figure 8.13 shows the effect of λ_p on $\overline{L_w}$. The increase in λ_p leads to the increase in $\overline{L_w}$ because more messages arrives at the queue. It can be seen that $\overline{L_w}$ rapidly gets close to its maximum value, that is the queue size ($S_w = N_p\delta_w$). It is due to the passive interval times θ_w^{-1} during which no messages are processed by the server. $\overline{L_w}$ increases rapidly when $\lambda_p < 0.5$ because there are fewer messages in the queue, and hence an arriving notification is less probable to replace an older one. As $\overline{L_w}$ increases, the probability of an arriving notification replacing an older notification waiting in the queue is higher, and hence the increasing rate of $\overline{L_w}$ decreases.

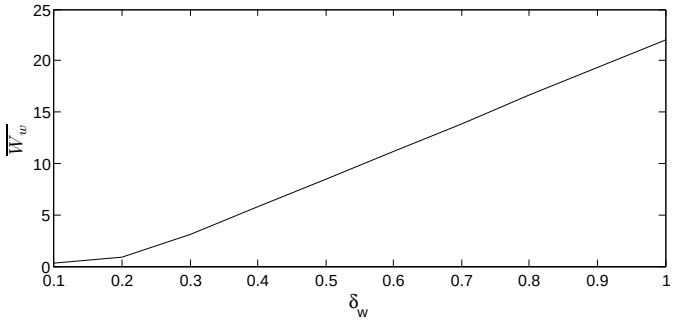


Figure 8.12: Effect of δ_w on $\overline{W_w}$ ($\mu_w = 10$, $\lambda_p = 0.2$, $N_p = 120$, $\theta_w = 8$)

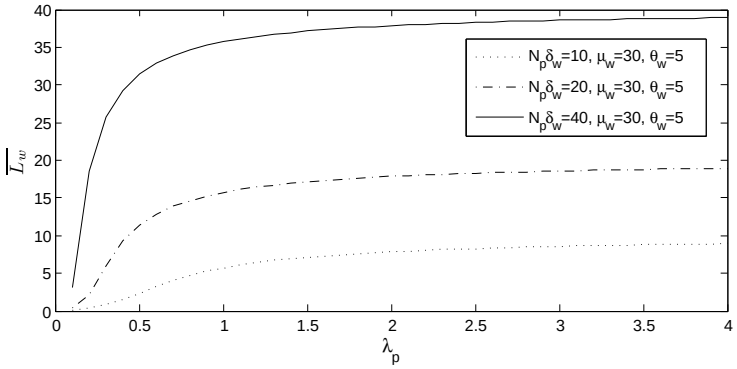


Figure 8.13: Effect of λ_p on $\overline{L_w}$

Figure 8.14 shows the effect of θ_w on $\overline{L_w}$. The increase in θ_w leads to the decrease in $\overline{L_w}$ because the server spends less time in passive intervals, and hence fewer messages are queued. This decrease is slow, since the server goes to a passive time of θ_w^{-1} unit times whenever a message is processed.

Figure 8.15 shows the probability distribution of L_w with different values of traffic intensity ρ_w. The probability of L_w being large increases with ρ_w, since the increase in ρ_w means that the arrival rate of notifications becomes higher in proportion to the service rate. It can be seen that, when $\rho_w > 0.5$, the probability of L_w is distributed among the highest values. Even, when $\rho_w = 5$, the most probable value of L_w is 20, which means that the queue is full. We explain this phenomenon with θ_w. The passive times θ_w^{-1} involve queuing many messages, and hence L_w increases with ρ_w quickly. Figure 8.16 shows the probability distribution of L_w when θ_w is set to 10. Compared to Figure 8.15, it can be seen how the probability distribution of L_w moves towards lower

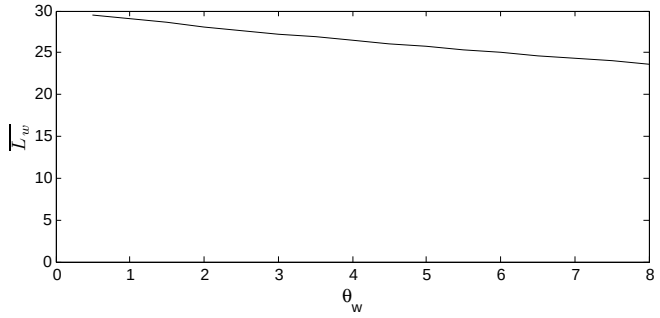


Figure 8.14: Effect of θ_w on $\overline{L_w}$

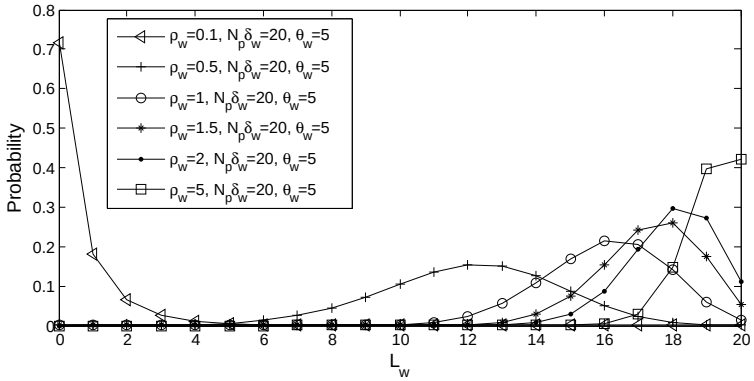


Figure 8.15: Distribution of L_w with different values of traffic intensity ρ_w

values. Thus, θ_w is a determining factor on L_w. This fact can be noticed in Figure 8.17, which shows the distribution of L_w with $\rho_w = 1$ and different values of θ_w.

Figure 8.18 shows the effect of S_w on the notification queue loss probability LP_w, with different values of λ_p. LP_w decreases as S_w increases, since more messages can be queued. When $S_w = 20$ the queue is lossless, and hence $LP_w = 0$. LP_w is higher for higher values of λ_p, since more notifications arrive at the queue. It can be seen that, when $\lambda_p = 1$ and $\lambda_p = 2$, LP_w is high and only decreases with high values of S_w. These two cases involve a traffic intensity ρ_2 of 1 and 2, respectively. In Figure 8.15, it can be seen that the probability of L_w is distributed among high values for $\rho_w = 1$ and $\rho_w = 2$. Thus, LP_w is high if S_w is not close to these values since the queue cannot accommodate most of the arriving messages.

Figure 8.19 shows LP_w as S_w increases, with different values of θ_w. It can be seen that LP_w decreases with higher values of θ_w. It is due to the fact that L_w decreases as

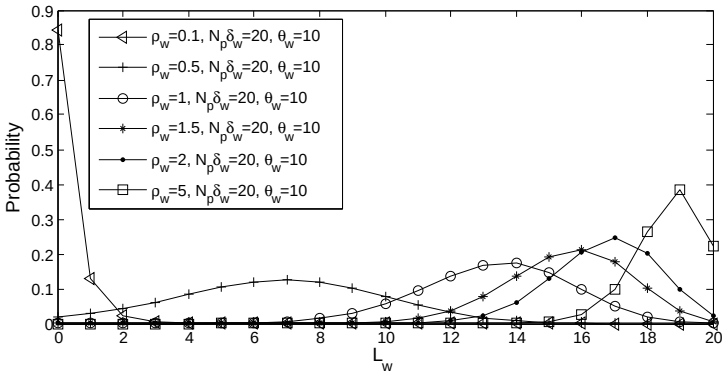


Figure 8.16: Distribution of L_w with different values of ρ_w and $\theta_w = 10$

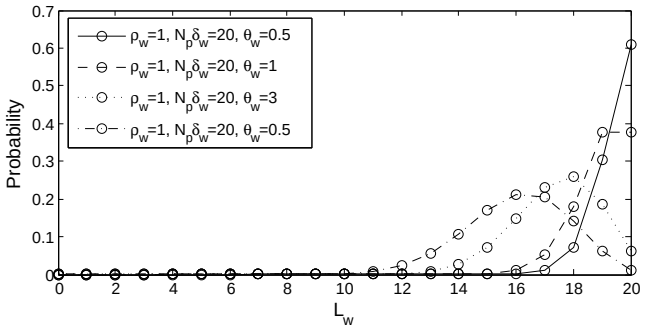


Figure 8.17: Distribution of L_w with $\rho_w = 1$ and different values of θ_w

θ_w increases, and hence more messages can be queued.

8.4 Analytical Modeling of Publication Throttling

As previously described, the presence change process of each presentity conforms to Poisson distribution with mean λ_p. The PS may set a presentity to throttle its publications by setting a minimum time between two consecutive publications, that is, source-throttling. Let us assume that PUBLISH message service times at the presentity are exponentially distributed with mean μ_s. Let θ_s^{-1} be the throttling time for the presentity. Thus, source-throttling at the presentity can be modeled through the queuing system described in Section 8.3 by setting $N_p = 1$, $S_w = 1$, $\mu_w = \mu_s$, $\theta_w = \theta_s$ and $\delta_w = 1$. We refer to this system as the presentity's source queue. Thus, the average waiting time that the presentity takes to publish a presence change $\overline{W_s}$ can be

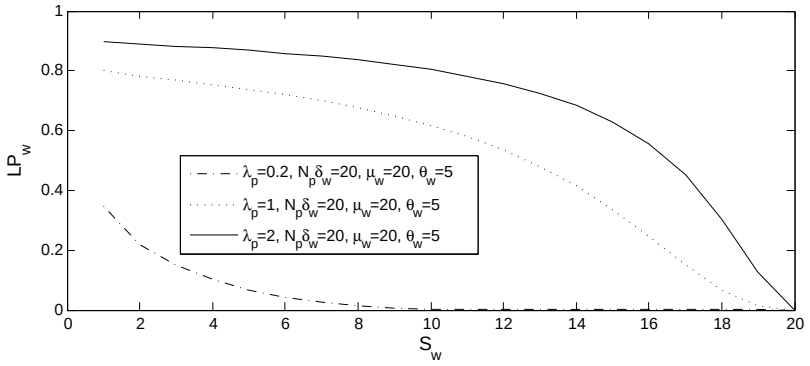


Figure 8.18: Effect of S_w on LP_w, with different values of λ_p

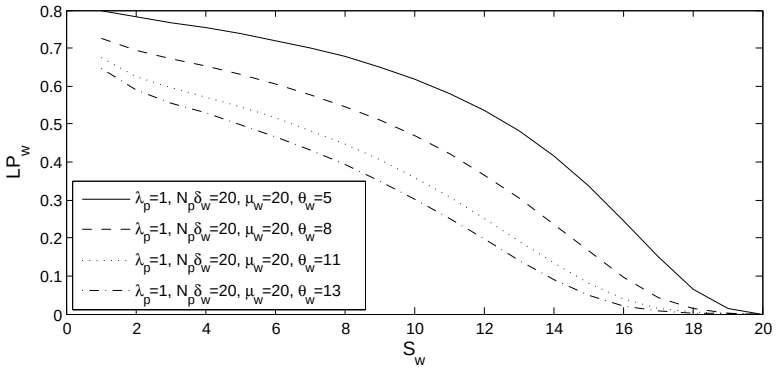


Figure 8.19: Effect of S_w on LP_w, with different values of θ_w

calculated as described in Section 8.3.3. When source-throttling is not applied, the presentity does not queue publications, and $\overline{W_s}$ is therefore equal to the average service time μ_s^{-1}.

8.5 Adaptive Control Algorithm

Since watcher domains may have different traffic capacities and policies on traffic optimization, we assume that they are able to let the PS know about their preferred maximum notification rates. Dropping off the number of notifications could however involve watchers keeping obsolete information, thereby perceiving notification delays. Therefore, we also assume that watcher domains are able to let the PS know about their preferred maximum notification delays. We consider the notification time for a

watcher domain as the total time that elapses from the occurrence of a presence change until the corresponding notification to the domain is sent. This is the time that watchers within the domain are keeping obsolete information. Let NT_w denote the average notification time for the domain w. For the sake of simplicity, we analyze notification queues independently of the publication queue in Section 8.3. We therefore approximate the average time taken by the PS to notify a watcher domain of a presence change as the sum of the average time spent by the PUBLISH message at the publication queue plus the average time spent by the corresponding NOTIFY message at the watcher domain's notification queue. Moreover, presentities may control the rate at which they send PUBLISH messages to the PS. Thus, we calculate NT_w as $\overline{W_s} + \overline{W_P} + \overline{W_w}$ (see Sections 8.4, 8.2.3 and 8.3.3). In addition, presence traffic overload at the PS should be carefully taken into consideration in order not to waste communication and processing resources. A PS under saturation conditions would drop messages, thereby wasting network and server resources and degrading the overall presence system performance. We propose an adaptive algorithm for satisfying the following input QoS parameters:

- α: The notifier domain's maximum probability of PUBLISH message loss, that is, $LP_P \leq \alpha$.
- β: The notifier domain's maximum probability of NOTIFY message loss, that is, $LP_w \leq \beta$.
- $maxDelay_w$: Delay in notifications acceptable by the watcher domain w, that is, $NT_w < maxDelay_w$.
- θ_w: The watcher domain's preferred maximum notification rate.
- θ_{wmax}: The watcher domain's threshold notification rate for traffic optimization. This means that the watcher domain is willing to increase θ_w up to θ_{wmax} with the aim of keeping the notification time under $maxDelay$. Thus, when $\theta_{wmax} \leq \theta_w$, it means that the watcher domain gives priority to traffic optimization.

Although keeping a maximum notification rate θ_w and a maximum notification delay $maxDelay_w$ at the same time may seem contradictory conditions, they can certainly be combined. The former determines a minimum time between any two consecutive notifications, while the latter is the maximum time that the notifier domain should

N_p	Number of presentities
λ_p	Average rate of presence changes per presentity
δ_w	The watcher domain's subscription factor
μ_s	Source queue service rate
μ_P	Publication queue service rate
μ_w	Notification queue service rate

Table 8.1: Variables that describe the queuing system

hold back a presence change notification. In addition to the QoS parameters above, the control algorithm receives the parameters in Table 8.1, which describe the system scenario and were defined in Sections 8.2 and 8.3. From the above-mentioned input parameters, the algorithm returns the following outputs:

- S_w: minimum size of the watcher domain's notification queue that ensures $LP_w \leq \beta$.
- S_P: minimum size of the publication queue that ensures $LP_P \leq \alpha$.
- θ_{wn}: Maximum notification rate for the watcher domain's notification queue (i.e., notification throttling).
- θ_{ws}: Maximum publication rate for the presentities that notify the watcher domain (i.e., source throttling).

We propose a strategy for always ensuring the PS QoS conditions on message loss probability (i.e., α and β). Moreover, the algorithm finds the most suitable notification rate based on the maximum rate and delay preferred by the domain (i.e., θ_w and $maxDelay_w$). When it is possible, the algorithm finds the minimum rate at which presentities can throttle PUBLISH messages while meeting the watcher domain's preferences. Below, the control algorithm in pseudo-code summarizing this strategy is shown.

S_P=$MinSize_P(LP_P < \alpha)$
T_P= $W_P(S_P)$
S_w=$MinSize_w(\theta_w,\ LP_w < \beta)$
T_w=$W_w(S_w,\ \theta_w)$
T_s=$\frac{1}{\mu_s}$
$NT_w = T_s + T_P + T_n$

$\theta_{wn} = \theta_w$
$\theta_{ws} = 0$
IF($NT_w > maxDelay_w$ AND $\theta_{wmax} \neq 0$) THEN
  $notTime = maxDelay_w - T_s - T_P$
  Find minimum θ that satisfies
    1. $\theta_w < \theta \leq \theta_{wmax}$
    2. $W_w(S_w, \theta) \leq notTime$
  IF(θ is not found) THEN $\theta_{wn} = \theta_{wmax}$
  ELSE $\theta_{wn} = \theta$ END
  S_w=$MinSize_w(\theta_{wn}, LP_w < \beta)$
ELSE IF($NT_w < maxDelay_w$)
  $sourceTime = maxDelay - T_P - T_w$
  Find minimum θ_{ws} that satisfies
    1. $W_s(\theta_{ws}) \leq sourceTime$
END

This algorithm should be applied to each watcher domain w for obtaining its θ_{wn} and θ_{ws}. Thus, the PS can set each watcher domain's maximum notification rate to θ_{wn} and the publication rates of the presentities that notify the domain to θ_{ws}. We assume that μ_s is the same for all the presentities. Note that presentities are likely to notify more than one watcher domain. A presentity's publication rate should therefore be the maximum θ_s obtained from applying the control algorithm to each of the domains subscribed to the presentity. This ensures that the maximum delay preferred by each of these domains is satisfied. The publication queue size S_P needs only to be calculated once since there is only one publication queue. The algorithm first finds the S_p and S_w that satisfy the conditions $LP_p \leq \alpha$ and $LP_w \leq \beta$ by invoking $MinSize_P$ and $MinSize_w$, respectively. These conditions are strict because they ensure that the PS does not get saturated. The functions W_P and W_w calculate the average waiting time of the publication and notification queue, respectively. Ensuring $LP_w \leq \beta$ may require a large queue size S_w, since the larger the queue size the lower the loss probability. However, the queue waiting time increases with the queue size. Thus, ensuring the condition $LP_w \leq \beta$ may result in the average notification time being longer than the preferred maximum delay, that is, $NT_w > maxDelay_w$. In this case, if the watcher domain prefers to sacrifice traffic optimization for information consistency (i.e., $\theta_{wmax} > \theta_w$), the algorithm tries to find a minimum notification rate between θ_w and θ_{wmax} that meets this condition. If such a rate is not found, the algorithm sets θ_{wn} to θ_{wmax}. Initially, the average notification time NT_w is calculated assuming

that no source-throttling is applied ($T_s = \mu_s^{-1}$). If NT_w meets the notification delay condition, that is, $NT_w < maxDelay_w$, the algorithm finds the minimum publication rate θ_{ws} that meets this condition. The function W_s calculates the average waiting time of PUBLISH messages at the source queues.

8.5.1 Performance Evaluation

This section studies the performance of the proposed adaptive control algorithm on a watcher domain, as well as the correctness of the mathematical models described in Sections 8.2 and 8.3. We set the scenario variables as follows: $N_p = 100$, $\delta_w = 0.1$, $\lambda_p = 1$, $\mu_s = 1$, $\mu_P = 100$ and $\mu_w = 10$. We assume the QoS conditions a) $LP_P \leq 0.2$, b) $LP_w \leq 0.2$, c) $NT_w < 5$, and d) $\theta_{wmax} = 10$.

The algorithm's outputs under different preferred maximum notification rates θ_w are shown in Figures 8.20 and 8.21. Figure 8.20 shows the values of S_P and S_w that satisfy the above-mentioned conditions. S_P is constant and the same for all the watcher domains. S_w decreases as θ_w increases. The higher θ_w, the shorter the minimum time that has to elapse between two consecutive notifications θ_w^{-1}. Thus, $\overline{L_w}$ decreases, and so does LP_w, thereby requiring a lower S_w to satisfy condition b). Figure 8.21 shows the output rates for the notification queue and the presentities that notify the watcher domain. When $\theta_w < 2$, condition c) cannot be met with this rate. The algorithm therefore sets the minimum θ_{wn} that meets this condition, which is $\theta_{wn} = 2$. In this case, source-throttling is not performed in order to keep meeting condition c), and hence $\theta_{ws} = 0$. When $\theta_w \geq 2$, notifications can be throttled with θ_w satisfying condition c). Thus, $\theta_{wn} = \theta_w$ and source-throttling is applied. The publication rate θ_{ws} decreases as the notification rate θ_{wn} increases. This is because $\overline{W_w}$ decreases, and hence presentities can hold publications back for longer while satisfying condition c).

Figure 8.22 compares the average waiting times of PUBLISH messages at the presentities (i.e., source queues) and the publication queue, and NOTIFY messages at the notification queue. With $\theta_w < 2$, $\overline{W_w}$ and $\overline{W_s}$ are constant because $\theta_{wn} = 2$ and $\theta_{ws} = 0$. When $\theta_w \geq 2$, $\theta_{wn} = \theta_w$, and hence $\overline{W_w}$ decreases with θ_w. Thus, $\overline{W_s}$ increases with θ_w. In Figure 8.22, the stepped increases and decreases are due to the decrease in S_w (the lower S_w, the lower $\overline{W_w}$).

Figure 8.23 shows the effect of the preferred maximum notification rate on LP_P and LP_w. LP_w is constant when $\theta_w < 2$ because $\theta_{wn} = 2$. When $\theta_w \geq 2$, $\theta_{wn} = \theta_w$, and

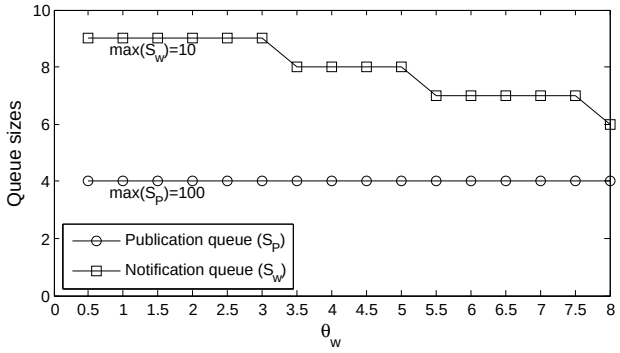


Figure 8.20: Minimum queue sizes over preferred notification rate

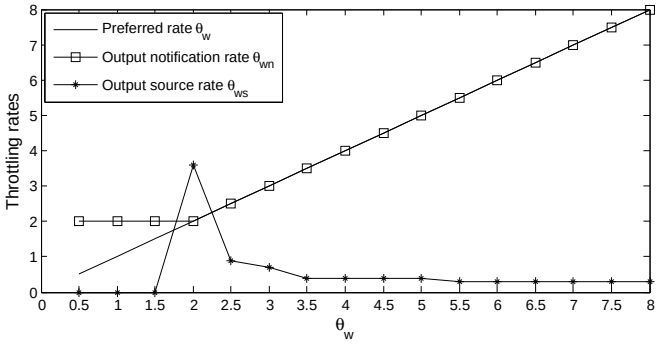


Figure 8.21: Output throttling rates over preferred notification rate

hence LP_w decreases as θ_w increases. The stepped increases are due to the decrease in S_w. For instance, when $2 \leq \theta_w \leq 3$, $S_w = 9$. As θ_w increases, there are fewer NOTIFY messages in the queue, and hence LP_w decreases. When $\theta_w = 3.5$, S_w is decreased by one, which increases LP_w. As S_w becomes smaller, more increments in θ_w are necessary to decrease S_w by one while meeting condition b).

Figure 8.24 compares the average waiting times when the watcher domain sets $\theta_{wmax} = 0$, which means that traffic optimization is given priority. In this case, the algorithm always sets θ_{wn} to θ_w, rather than increasing θ_{wn} for meeting condition c). It can be seen that when $\theta_w < 2$, condition c) cannot be met. The total average time is lower than $maxDelay_w$ when $\theta_w \geq 2$. From this value, Figures 8.22 and 8.24 are the same.

Figures 8.25 and 8.26 show the notification rates and the average waiting times, respectively, with different values of $maxDelay_w$. We assume the above-mentioned

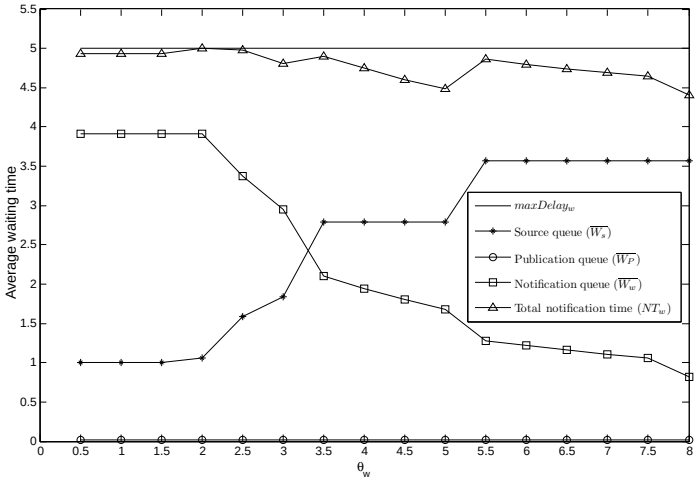


Figure 8.22: Waiting times over preferred notification rate

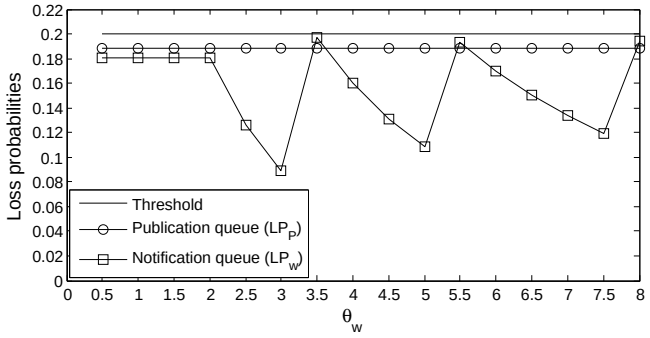


Figure 8.23: Queue loss probabilities over preferred notification rate

input parameters with the exception of $\theta_w = 2$ and $\theta_{wmax} = 5$. When $maxDelay_w \leq 2.5$, no notification rate lower than or equal to θ_{wmax} meeting condition c) can be found. Thus, θ_{wn} is set to the maximum θ_{wmax}. When $maxDelay_w > 2.5$, θ_{wn} can be decreased towards θ_w while satisfying condition c), as shown in Figure 8.25. The preferred notification rate $\theta_w = 2$ can only be satisfied when $maxDelay_w$ reaches the value 5. In this case, presence publication can be throttled ($\theta_{ws} \neq 0$). In Figure 8.26, one may observe that $\overline{W_w}$ increases when $maxDelay_w \geq 3$. This is due to the decrease in θ_{wn}. In this Figure, the stepped increase at $maxDelay_w = 4$ results from the increase in S_w, whichf is equal to 8 when $maxDelay_w < 4$ and 9 when $maxDelay_w \geq 4$. This phenomenon is explained as follows. The increase in $maxDelay_w$ results in the decrease

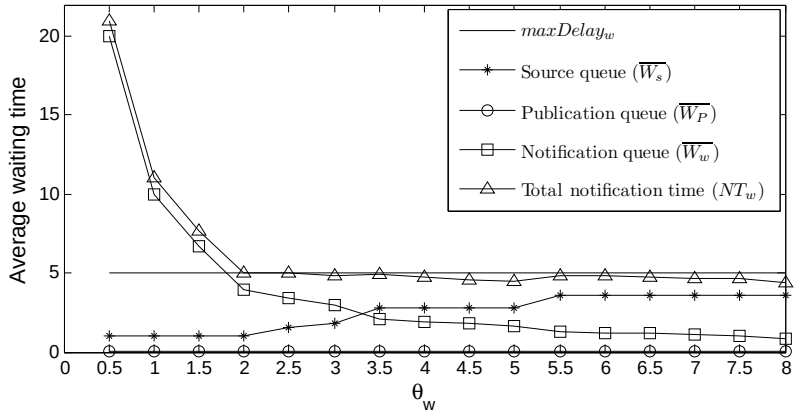


Figure 8.24: Waiting times over preferred notification rate with $\theta_{wmax} = 0$

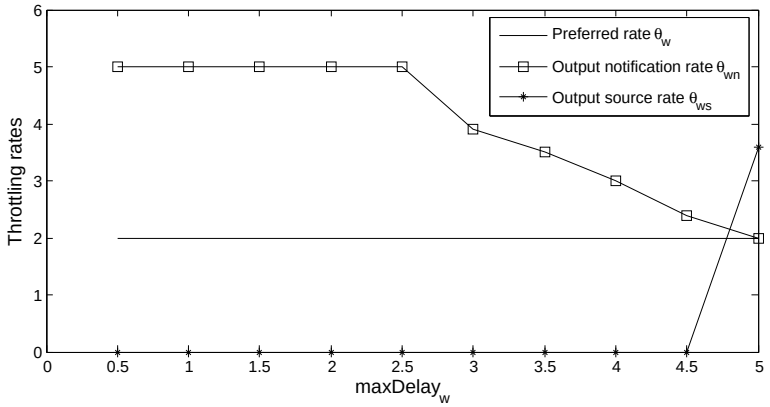


Figure 8.25: Output rates over preferred maximum notification delay

in θ_{wn} for traffic optimization. In turn, decreasing θ_{wn} involves a higher number of NOTIFY messages in the queue, and hence LP_w increases. Thus, S_w is increased at $maxDelay_w = 4$ in order to meet condition b), thereby increasing $\overline{W_w}$.

Figures 8.27 and 8.28 show the effect of λ_p on the average queue waiting times and the output rates, respectively. We set $maxDelay = 5$, $\theta_w = 3$ and $\theta_{wmax} = 4$. When $\lambda_p < 2$, θ_w can be applied to meet condition c), and hence $\theta_{wn} = \theta_w$. Thus, source-throttling is applied and $\overline{W_s}$ is therefore higher than its service time, that is, 1. $\overline{W_w}$ increases with λ_p, and hence $\overline{W_s}$ decreases (i.e., θ_{ws} increases) to keep meeting condition c). When $\lambda_p \geq 2$, θ_{wn} is increased above θ_w in order to meet condition c). Thus, source-throttling can not be applied and $\overline{W_s}$ is equal to the service time, that is,

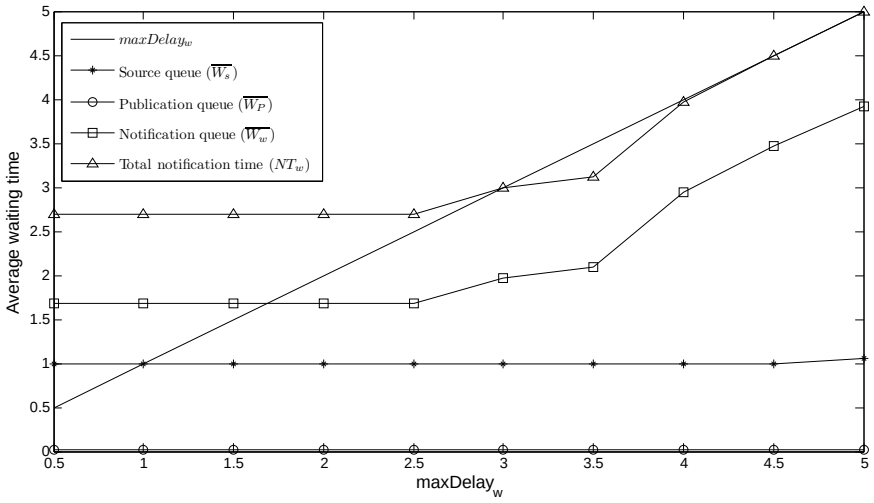


Figure 8.26: Waiting times over preferred maximum notification delay

1. $\overline{W_P}$ increases with λ_p because more messages are put into the queue.

Figure 8.29 compares the average waiting times without the proposed adaptive algorithm. In this case, θ_{wn} is always equal to 3 and source-throttling is not performed. One may observe that with low arrival rates, the average total notification time is much lower than the preferred maximum delay. However, network access link traffic is not saved by controlling the rate of presence publications. When the arrival rate increases, $\overline{W_P}$ and $\overline{W_w}$ increase and condition c) is not satisfied. Moreover, in Figure 8.29, we assume the queues to be lossless (i.e., $S_w = 10$ and $S_P = 100$). Otherwise, since the system does not change the queue sizes dynamically, LP_P and LP_w may become too high as λ_p increases.

8.6 Use of Sojourn-Based Intervals

The adaptive control algorithm presented in Section 8.5 sets the minimum publication and notification rates that ensure that the average notification time for a watcher domain is shorter than a maximum delay set by this domain. This algorithm does not consider how frequently the presentities' presence information changes. If the maximum rate of publication is much lower than the rate at which the presence information changes, publishing this information becomes inefficient; the time during which the watchers see valid information is too short. As described in Section 4.3, sojourn-based

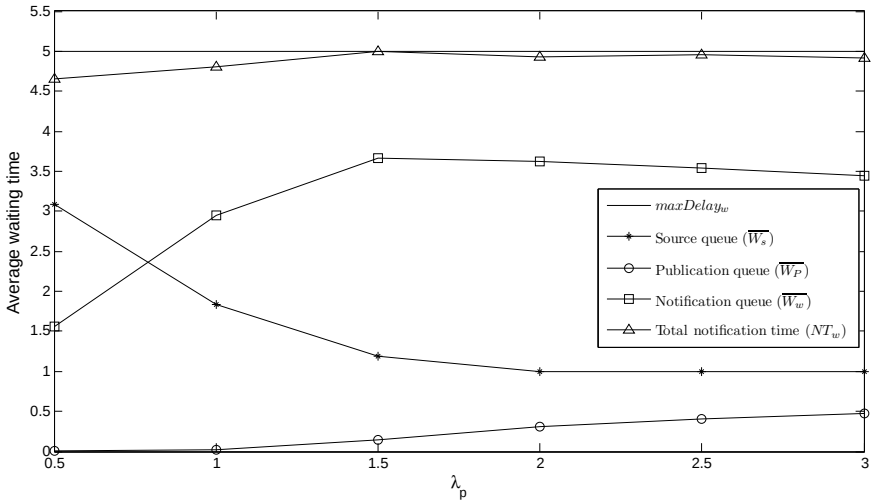


Figure 8.27: Waiting times over PUBLISH arrival rate of each presentity

intervals (SBIs) are calculated according to the change frequency of the presentities' presence information, which is modeled through the Markov theory. SBIs may be combined with the adaptive algorithm for preventing the PS from setting publication rates that are too low or too high. When the control algorithm finds that source-throttling can be performed, it compares the calculated throttling rate θ_{ws} with the presence information transition probability Φ (see Section 4.3). If $\theta_{ws} << \Phi$, the algorithm sets θ_{ws} to the presence information's SBR (see Section 4.3). If $\theta_{ws} >> \Phi$, presence changes occur at such a low rate that throttling publications with θ_{ws} is not likely to aggregate any presence change. Since θ_{ws} is the minimum throttling rate ensuring that publications are not delayed too long, this rate cannot be decreased. Thus, source-throttling with θ_{ws} is inefficient and the algorithm decides not to apply it.

8.7 Conclusions

We tackled the need to optimize presence traffic on the network access link as well as on the network core. We proposed a queuing system for optimizing presence traffic in presence federation scenarios. This system is designed to be scalable and adapt itself to the QoS needs of both the PS and watcher domains. To this end, the system requires that the PS only sends a presence notification to each watcher domain per presentity.

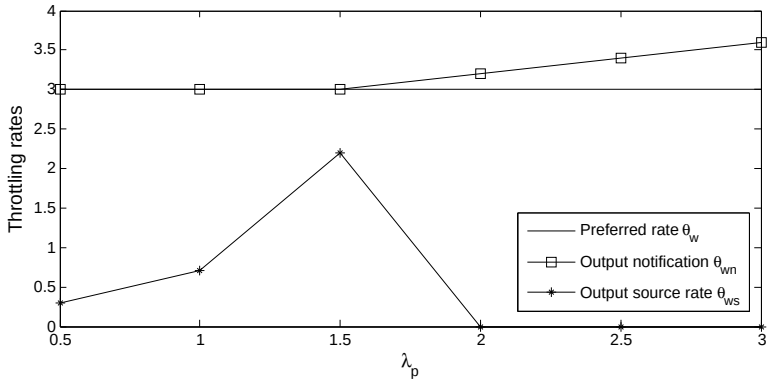


Figure 8.28: Output rates over PUBLISH arrival rate of each presentity

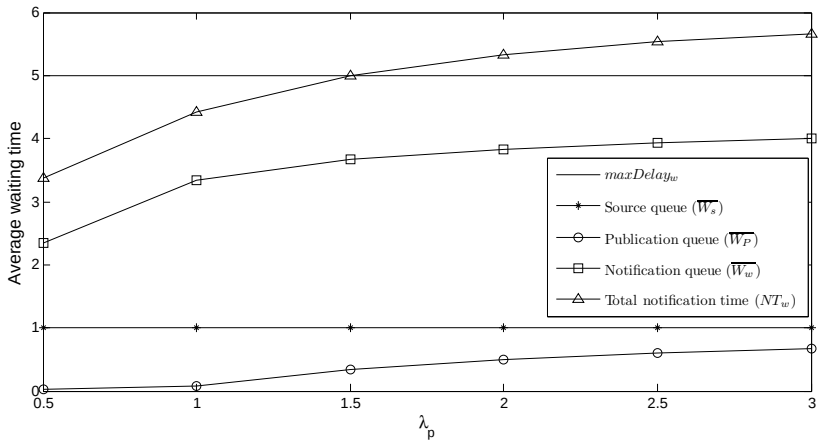


Figure 8.29: Waiting times over PUBLISH arrival rate of each presentity without the adaptive algorithm

We proposed an adaptive algorithm for changing publication and notification rates based on the watcher domains' needs as regards traffic optimization and information consistency. This algorithm ensures certain maximum message loss probabilities at the PS and a maximum delay in notifying each watcher domain, while limiting the rate of presence publications and notifications. We mathematically modeled the proposed queuing system and obtained its main probability features. Based on this mathematical model, we studied the performance of the proposed adaptive algorithm with different QoS parameters. This algorithm effectively adapts the notification rate for a watcher domain and the publication rates of the users to which the domain is subscribed with

the aim of ensuring that presence change notifications are not delayed longer than a maximum delay set by the domain. We also discussed how this algorithm can use sojourn-based intervals (SBIs) for ensuring that presence publications are not throttled inefficiently.

8. QUEUEING SYSTEM AND ADAPTIVE QOS MECHANISM FOR CONTROLLING THE RATE OF PRESENCE PUBLICATIONS AND NOTIFICATIONS

9

Personal Proxy

To deploy large-scale, intelligent presence-enabled applications, platforms for collecting, processing and disseminating presence information in a scalable and efficient way are necessary, as explained in Section 2.8. Such platforms should comply a standard presence protocol for the sake of interoperability. As well, user-centric personalization, privacy and extensibility are key aspects of the future presence-enabled ubiquitous applications, as described in Sections 2.3.2 and 2.9.3. The flexibility in these aspects offered by the underlying presence platform is key to the success of these applications. To date, no presence platform with these characteristics has emerged. The IMS presence service involves serious scalability issues as described in Section 2.9.4. Many authors have worked on platforms for handling context information. However, they do not consider presence information or are not concern about scalability and traffic optimization. Section 2.8 mentions some of these works. The authors of [150] propose a RESTful web service for providing a lighter-weight presence service. However, HTTP is not suitable for subscriptions and presents a number of limitations difficult to overcome. Subscriptions are simulated with persistent connections and chunked encoding. No resource list subscriptions are defined and the mapping between addresses of the form "user@domain" and URLs is not evident.

We propose a decentralized architecture for presence management in which each user owns a Personal Proxy (PP) that handles her presence information. The nature of this architecture provides a great flexibility to converge multimedia protocols in a single point and tightly personalize user communications and privacy. This is a logical functionality that may be included in a home gateway or even a femtocell as a value-

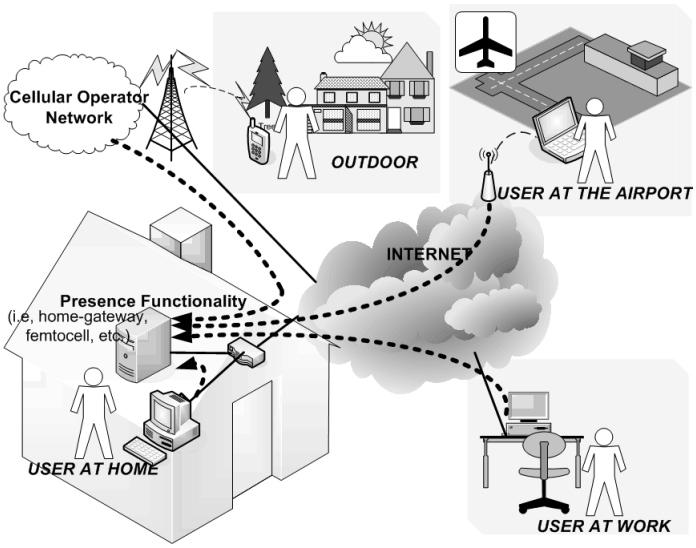


Figure 9.1: PP scenario

added service for managing presence information and optimizing user communications in a distributed way. Figure 9.1 depicts the common scenario for PP, which is the user's home or office. PP however may be applied to other scenarios where a third-party, such as an operator, provides the user with its PP and administrates it.

Section 9.1 discusses the design of the proposed solution while Section 9.2 addresses its software architecture. Sections 9.3 and 9.4 describe the PP techniques for optimizing presence and HTTP traffic, respectively. These sections also present some insights into these techniques' efficiency at reducing traffic. Lastly, Section 9.2.4 explains the implementation and current state of PP and Section 9.5 gives some conclusions.

9.1 Design

PP is a SIMPLE-based presence service that applies user-created fine-grained rules about the content of presence documents according to the users' preferences on privacy and communication means. PP is capable to optimize presence traffic if necessary, as for example when the network access link's bandwidth is low. Apart from presence handling, the proposed solution allows centralizing the user access to certain Internet services, such as HTTP or VoIP, in a single point. Merging presence handling with some Internet services in distributed point brings out scalability and flexibility in user-

based customization of these services. Sections 9.1.1 and 9.1.2 describe the PP presence functionality and multimedia aspects, respectively.

9.1.1 Presence PP

Contrary to centralized PSs, the proposed approach is fully distributed, and hence its performance does not degrade with the number of users. A user's PP will interact with other users' PPs as well as other SIMPLE-compliant presence services, as depicted in Figure 9.2. A user's PP recollects the presence information that comes from all of the user's terminal devices, aggregates this information and provides a single point to accessing this information. This provides device-independent, always-on presence information as well as a greater security and flexibility. The PP can let other users know about the user's presence at any moment, even when the user is not available on any device. Since PP is a fully distributed solution, it allows implementing context-aware fine-grained rules about privacy and presence handling for each user. Presence handling should be done in a user-centric manner since users have different preferences on privacy and communications, and these preferences may change over time based on situational or temporal context. Users may need different behaviors from presence-based applications based on their circumstances. Some users might be interested in knowing as much as possible about the presence of all of their buddies. Other users however might be more concerned about announcing their presence while the accuracy level of their buddies' presence is less important to them. The distributed nature of PP makes user-centric personalization scalable and therefore possible. PP is designed to reduce the presence traffic between the end user and the network as much as possible. It is specially useful when the user is connected to an access link with limited bandwidth. Section 9.3 describes how PP reduces presence traffic.

PP is a SIP/SIMPLE-compliant PS and RLS that offers a high-level API to applications for accessing its functionality. This agnostic-operator, standard and user-centric solution achieves interoperability and may motivate users to use presence applications. Presence information may include private information such as appointments, buddies, schedule, activities and so on; thus, users may not trust service providers to keep this information undisclosed. PP stores the user's presence information at his or her place, and ensures user-defined privacy rules about what information is sent out. This may

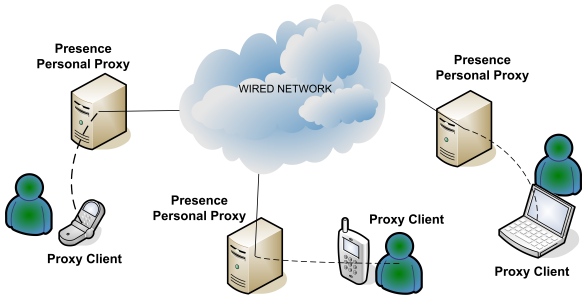


Figure 9.2: Interaction between PPs

make the user feel the presence service more familiar to her, thereby giving more information in to it. PP is a middleware composed of a client software installed on the user's devices and a server part, that is, a proxy at the user's place, as described in Section 9.2. Most of the provided functionality is carried out by the proxy for keeping user devices as simple and resource-efficient as possible. The proxy retrieves a minimum set of presence from the user's devices (i.e., that most dynamic and personal to the user such as status, nickname and mood) and enriches this presence before notifying it. This approach allows lighter terminal devices because they do not need to store and process the majority of presence information. To date, the user's presence information is enriched with end devices' hardware and service capabilities, which are discovered by means of a detection mechanism [208].

9.1.2 Multimedia PP

The distributed nature of PP allows the synthesis of SIP and other Internet protocols, such as HTTP and RTSP, on a single and scalable point. As described in Section 2.4.1, such a synthesis is necessary for the success of FMC. PP may implement a SIP Registrar and Proxy for managing multimedia sessions. User devices would register their addresses in the Registrar and use their PPs as gateways to initiate any SIP session. If a PP received a request to initiate a session from the outside, it would link this session with the most suitable device on which the user is available. PP would therefore act as a bridge between the user and any communication requester.

PP provides the user with a single invariable address and hides the real user device's physical address. This offers the benefit of security and mobility privacy, as well as the opportunity to build improved services. A PP may be connected to the PSTN network

and act as a gateway between IP-connected and PSTN users. Mobile IP integration is other advantageous possibility, which would allow the user to change network domain without incurring connection losses. Integrating a Home Agent (HA) into PP allows knowing the user's Care-of-Address (CoA) at any moment. This saves the user from unregistering the old IP address and registering the new one at any time when he or she changes network. When an IP address change occurs, presence applications normally unsubscribe and subscribe back to the events of interest. This generates unnecessary traffic if the user only changes the network interface rather than the device. The proxy would prevent this inefficient behavior because it could deregister the user's old address and resend any message to the user's CoA automatically.

9.2 Software Architecture: a Middleware-Based Approach

The proposed platform for handling presence consists in a middleware that acts as a dedicated proxy for each user. This middleware has been designed to reduce the complexity of user devices and optimize the use of access networks. This optimization is achieved because of the collaboration between the client and server parts of the middleware. The client part needs to be installed on the user's devices and the server part is the user's proxy that acts as his or her SIMPLE-based presence service. We refer the reader to Section 2.7 for a detailed description of SIMPLE. A middleware approach allows implementing non-standard solutions between the users' devices and the proxy for minimizing the traffic between them. This middleware makes it possible to decrease the amount of presence information exchanged with user devices as far as possible. Since some traffic optimizations consume processing resources at user devices, a tradeoff between traffic optimization and the costs of reducing traffic (in terms of processing resources and battery life) should be found. To this end, the knowledge about the user devices' characteristics is crucial. The middleware handles the following kinds of presence information:

- Personal information that is closely related to the user such as mood, activities, willingness to communicate, ambient conditions, profiles, personal addresses, localization, and so on.

- Information about the services on which the user is available, and the end devices that support these services. For example, information about a service may include content types accepted by the service as well as the hardware and software characteristics of the devices on which the service is running.

- Resource list information, which includes presence information about all the contacts (i.e., resources in the SIMPLE terminology) in which the user is interested.

- User presence rules that allow the user to build her model about data privacy and communications. These rules are split into several groups: request admission rules determine the communication types that the user is willing to establish with other users. Presence publication rules determine the presence information that will be given in to watchers. Lastly, white and black list rules automatically accept or reject, respectively, unknown entities that are requesting to watch the user's presence.

Figure 9.3 shows a layered architecture of the presence middleware. This is composed of two logical layers: the Management layer and the SIP/SIMPLE layer. The former contains the intelligence needed to process and manage presence information and the latter is in charge of receiving and sending SIP/SIMPLE messages. The modules of both layers are explained briefly in Sections 9.2.1 and 9.2.2. Figure 9.4 shows the communication flows between the client and server parts of the middleware. The client middleware, which we refer to as Client Presence Middleware (CPM) is more minimalist than the server part and therefore suitable for mobile client devices with scarce processing and memory resources. This middleware offers an API (the two-color rhombus in Figure 9.4) for any application on the user device to access the PP functionality in a transparent way. The server middleware, which we refer to as Server Presence Middleware (SPM), is the proxy placed in the user's place. This stores all the user's presence in a context repository named Context Manager in Figure 9.4.

9.2.1 Management Layer

In the SPM, the *HTTP Dispatcher* receives HTTP requests that contain user information such as configuration data, preferences or rules. This information is entered into the user device by some GUI and is sent to this module, which communicates

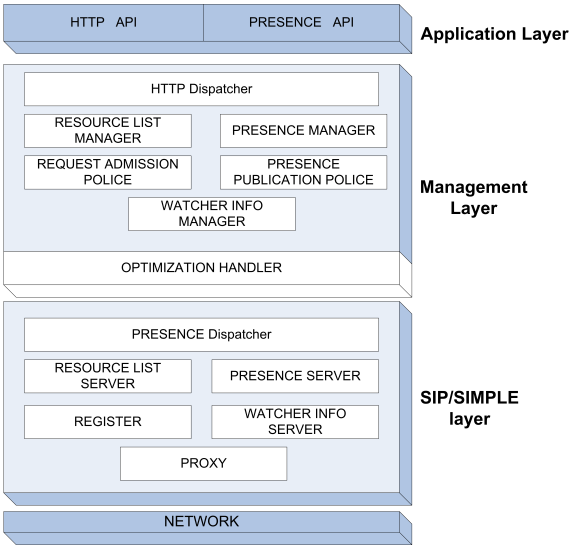


Figure 9.3: Logical modules of PP

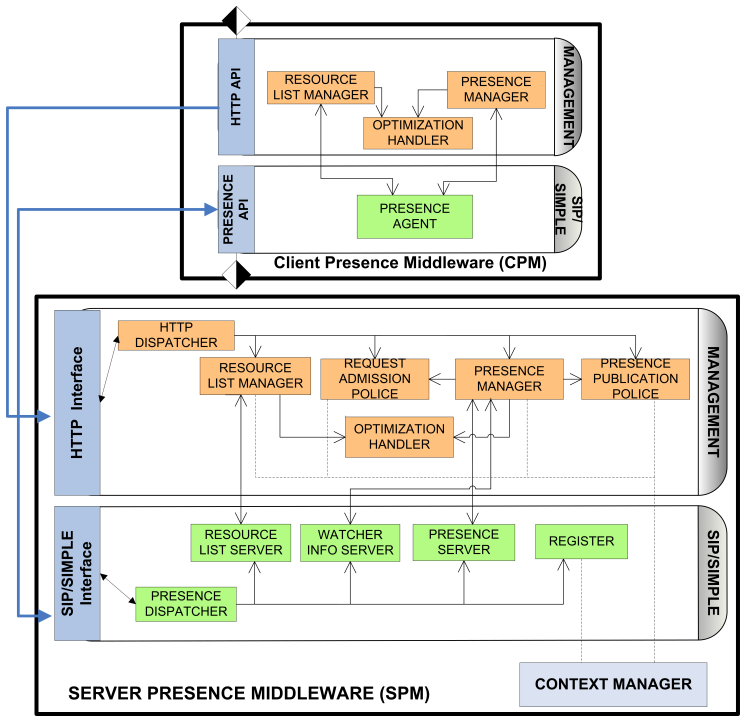


Figure 9.4: Interaction flows of the PP modules

with other modules at the same layer based on the type of information. The *Presence Manager* processes and aggregates the presence information published by the CPM and generates the presence documents that are sent to the user's watchers. This module communicates with the *Request Admission Police* and *Presence Publication Police* modules for generating the documents that each watcher is authorized to see based on request admission and presence publication rules, respectively. The presence information associated with the user's resource list is managed by the *Resource List Manager*. This module is responsible for processing and aggregating presence notifications from the contacts on the resource list in a single RLMI document. The complexity of the CPM is much lower than that of the SPM. The *Presence Manager* is in charge of controlling and updating presence information about the user. Thus, when the user's presence changes, this module is responsible for sending the change to the SPM through the *Presence Agent*. The *Resource List Manager* stores presence about the user's resource list and offers it to the applications. The optimization handlers of the SPM and CPM collaborate for performing the optimizations described in Section 9.3. The SPM optimization handler will contain the intelligence needed to decide what optimizations to apply based on the user's circumstances, the characteristics of the device on which the user is available and the access link congestion, for example.

9.2.2 SIP/SIMPLE Layer

This layer implements the SIMPLE framework. In the CPM, there is only one module, the *Presence Agent*, that carries out simple tasks related to the storage of the user's and her resource list's presence information. This module also registers the physical address of the user's devices in the SPM by means of REGISTER messages. Whenever the user's presence information changes, the *Presence Agent* publishes this information by sending a PUBLISH message to the SPM. Moreover, this module is notified by the SPM about the resource list's information. In the SPM, the *Presence Dispatcher* receives the SIP/SIMPLE requests sent by the CPM and forwards them to the corresponding module based on the type of request. The *Register* stores the user devices' physical addresses contained in REGISTER requests. This allows other modules in the SPM and external applications to send messages to the user. The *Resource List Server* is responsible for sending and receiving all SIMPLE messages related to the management and maintenance of the user's resource list. This module subscribes to each resource

by sending a SUBSCRIBE message and receives NOTIFY messages that update the resources' presence. The *Presence Server* module receives the PUBLISH requests that contain the user presence, and notify the user's watchers of this presence by means of NOTIFY messages. This module updates the state of each watcher subscription to the user according to its lifetime and the SUBSCRIBE messages received for refreshing its lifetime. Lastly, the *Watcher Info Server* notifies the user of the arrivals of new requests to watch his or her presence. This module waits the user's decision about whether or not to permit them to watch his or her presence. The *Presence Manager* interacts with the *Watcher Info Server* when it receives a SUBSCRIBE message from an unknown watcher. An unknown watcher is one that is not included in a black or white list. The *Watcher Info Server*, therefore, sends a NOTIFY message to the CPM for requesting for authorization. The CPM will indicate the user's decision by sending a PUBLISH message.

9.2.3 Presence Filtering

The user is capable to set policies about presence privacy and communication preferences via HTTP. These policies are determined by request admission and presence publication rules, as described in Section 9.2. These rules are managed by the *Request Admission Police* and *Presence Publication Police* modules, and are used at any time when the SPM is going to notify watchers of the user's presence information. The presence documents for watchers are created according to these rules.

Request admission rules restrict the communication types that other users can establish with the user. These rules internally determine what presence about services is going to be included in the user's presence documents sent to watchers. When a watcher receives the user's presence document, it communicates with the user by the services included in this document. Thus, notifying suitable presence documents permits to make watchers communicate with the user by certain means. Request admission rules are determined by a watcher (that is the future requester), and the user's circumstances. Some high-level example rules are "do not accept instant messages from team mates", "only accept video when I have leisure state", and "only accept audio from my boss when I am using my PDA7865 device". Figure 9.5 shows an example in which a user, Alfred, has two devices, a personal phone identified by nokia6280 and a work PDA identified by acerc510. At the top of this figure, Alfred's presence tree is shown,

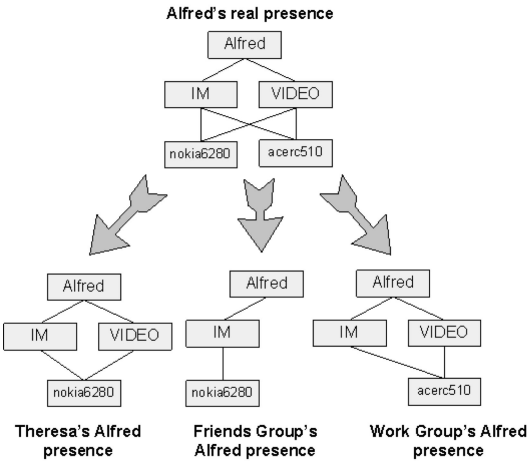


Figure 9.5: Outline of the the presence documents for different watchers

which indicates that his devices support instant messaging (IM) and video services. However, Alfred does not want to communicate with any person in his resource list by the same means. When he is working, he is willing to accept any type of call from his wife, Theresa, to his personal phone. Regarding school friends, he is only willing to accept instant messages sent to his personal phone. Work mates are only allowed to send instant messages and video to Alfred's acerc510 device. Figure 9.5 shows the subset of Alfred's presence tree sent to Theresa, school friends and work mates. Presence publication rules allow the user to indicate the privacy level of personal information in presence documents based on the watchers and his circumstances. Some examples are: "Not publish my personal presence to my work mates", "Not publish my vCard to my partners" and "Not publish my personal presence to any contact when I am connected to PDA987 and my activity is working".

9.2.4 Implementation and Stage

The developed presence middleware is fully based on the Java Platform: The SPM has been programmed using Java Platform Standard Edition (J2SE) and the CPM using the Java Platform Micro Edition (J2ME) with CLCD (Connected Limited Device Configuration) and MIDP (Mobile Information Device Profile) for mobile phones. The CPM has been developed on top of a SIPMethod Application Server (AS) that is a SIP AS runtime engine complaint with JSR-116 and JSR-289, which are SIP Servlet APIs. SIP Servlet API is a Java extension for SIP servers, which is similar in spirit

to the HTTP servlet API and offers many benefits such as performance, platform independence and high level abstractions. The JSR 180 (SIP API for J2ME) has been used to implement the SIP communication between CMP and SPM. This is a generic library that implements SIP. To date, the software layers described in Section 9.2 except those related to privacy and communication rules (see Section 9.2.3) have been implemented. The HTTP and presence optimization techniques described in Sections 9.4 and 9.3 except Inhibitor Subscribe have also been implemented. The PP APIs have not been provided yet.

9.3 Presence Traffic Optimization

PP is designed to decrease the complexity of user devices and utilize wireless links efficiently. The proxy decreases the amount of presence information sent and received by user devices as far as possible. It should be intelligent enough to use different optimization techniques based on the user's context. Traffic optimization is maximized due to the collaboration between the CPM installed on user devices and the SPM, that is, the proxy. Section 9.2 describes the PP architecture. This collaboration allows implementing non-standard solutions between user devices and the proxy in order to obtain a higher efficiency than that given by the SIMPLE framework by itself. Such non-standard solutions do not entail interoperability problems because all the traffic exchanged with external entities is fully standard. PP implements the following standard SIMPLE optimizations: RLS, partial notification, partial publication and event notification filtering. Section 2.7.4 describe these techniques, and we summarize them here for convenience. An RLS allows the user to subscribe to her contacts by sending a single subscription request. In turn, the RLS subscribes to each of the user's contacts and notify the user of RLMI documents when her contacts' presence has changed. Partial publication allows end devices to publish only the presence changes that have occurred from the last publication rather than the complete presence information. Likewise, partial notification allows notifiers to only include the resource's state information that has changed from the last notification rather than the user's complete state information. Thus, the proxy can send partial-state RLMI documents (i.e., only the resources whose presence information has changed are included into the list) and partial-state presence documents (i.e., only the resource's presence attributes that have changed are

notified). Event notification filtering allows filtering the content of RLMI documents. Event throttling limits the number of RLMI notifications sent by the PP by setting a minimum time interval between two consecutive notifications. PP aggregates any change in the presence information of the user's contacts that occurs during this interval. Once this interval has expired, all the changes are notified to the user by sending a single NOTIFY message. Moreover, PP encodes presence information by WBXml[4] [209] for reducing the size of presence documents. WBXml is a compact binary representation of XML that reduces the transmission size of XML documents by encoding XML tags as hexadecimal values. Below, the proposed non-standard techniques to be applied on the channel between the CMP and the proxy are listed:

- Presence Extension. End devices only handle the information that depends on the user's circumstances, such as location and mood. When an end device registers in its PP for first time, the proxy launches a procedure for detecting as much context information as possible. This procedure detects the device's hardware capabilities and the services that this device is capable to run. The proxy, therefore, aggregates this information to the user's presence document when it is going to be sent out.

- Empty NOTIFY [5]. The SIMPLE event notification process states that any subscription refresh triggers a full-state notification. Empty NOTIFY is a simple variation that consists in sending an empty NOTIFY message if no change is pending to be notified when a subscription is refreshed.

- Inhibitor Subscribe. Presence applications running on mobile devices often are placed into the background while the user is engaged in other activities such as voice calls. During these inactive periods, these applications are consuming battery and bandwidth because of the constant signaling traffic for updating presence and keeping presence subscriptions alive. To save such unnecessary traffic, the CPM sends a SUBSCRIBE message with its Expires header set to a negative number, which we refer to as Inhibitor SUBSCRIBE, when the user stops using its presence application. Then, the PP stops notifying the end device

[4]Since PP was developed in 2007, the 3WC has proposed a binary data format for XML, Efficient XML Interchange (EXI) [210]. Thus, in the future, PP will possibly be enhanced for using EXI rather than WBXml.

of the user's resource list. When the user returns to the application, the CPM sends a standard SUBSCRIBE message (i.e, one with an Expires header set to a positive number) to the proxy in order to restart the notification process.

- Source throttling. This technique limits the number of publications by setting the CPM with a minimum time interval that has to elapse between two consecutive publications. The CPM aggregates any change in the user's presence information that occurs during this interval. Once this interval has expired, all the changes are published by sending a single PUBLISH message. Section 4 discusses the benefits and applicability of this mechanism.

9.3.1 Analytical Estimation

This section presents an approximation of the PP efficiency at reducing presence traffic on the access network. Let us assume a user that is available on his mobile phone and has 10 contacts in his resource list. Let us assume that this user's contacts change their presence 5 times during the session, and change times follow a uniform distribution. The presence document associated with each contact is assumed to be of medium size (750 bytes). This document contains personal information (i.e., activity, place type, mood, state, display name and home page) and information about an available service (i.e., state and capabilities for audio, text, video and applications). Presence subscriptions' lifetime is 1 hour and the session is active from 8 a.m. to 10 p.m. Figure 9.6 shows the presence traffic sent over the wireless uplink and downlink due to the user's presence subscriptions. This traffic is exchanged between the user and the proxy throughout the day. This figure compares standard and proxy-based optimization techniques, which were described previously. The first group of columns from left (*Without RLS*) shows the case when the user does not use an RLS (i.e., she subscribes to each contact directly). The second group of columns from left (*Basic RLS*) shows the traffic when the user uses a standard non-optimized RLS. The third and forth groups of columns from left represent an standard RLS optimized by only partial RLMI notifications (*RLS with PRLMI*) and both partial RLMI and presence notifications (*RLS with PRLN and*

[5]Since PP was developed in 2007, the IETF have standardized conditional notifications through the RFC 5839 [147], which completely prevent notifiers from sending unnecessary NOTIFY messages. We will therefore integrate conditional notifications into PP in the future.

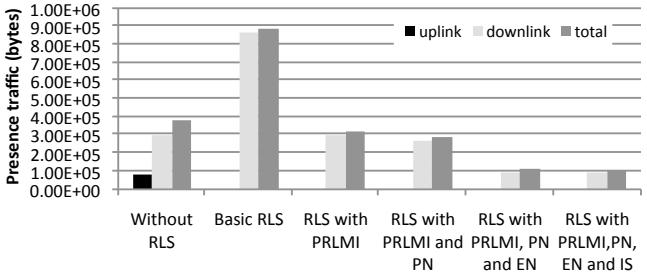


Figure 9.6: Presence traffic over the wireless link when a user uses a PP

PN), respectively. The last two groups of columns represent a PP that, in addition to the partial RLMI and presence notifications, uses the Empty NOTIFY technique (*RLS with PRLMI, PN and EN*), and the Inhibitor Subscribe technique (*RLS with PRLMI, PN, EN and IS*). For the Inhibitor Subscribe technique, we assume that the user uses a presence application on his mobile phone at different intervals throughout the day: from 8 to 9 a.m., from 11 a.m. to 1 p.m., from 6 to 7 p.m., and from 8 to 10 p.m. It can be seen that, although the use of an RLS reduces the traffic over the uplink, it increases the bytes sent over the downlink significantly. A non-optimized RLS increases the overall load on the communication channel drastically and therefore is not recommended. The majority of the RLS load is due to the full-state RLMI notifications sent at any time when a contact changes its presence state. Thus, partial-state RLMI notifications (i.e., *RLS with PRLMI*) reduce the RLS presence traffic by approximately 65%. In addition, when an RLS is combined with partial-state presence and RLMI documents (i.e., *RLS with PRLMI and PN*), the total presence traffic generated when no optimization is used (i.e., *Without RLS*) is reduced by about 25%. Although this reduction is relevant, it may not be enough in highly restricted networks or devices. The first two groups of columns from right show that the Empty NOTIFY and Inhibitor Subscribe techniques reduce the presence traffic generated when no optimization is used by approximately 74%. Proxy-based solutions can therefore optimize the presence traffic on wireless links significantly. The performance of Inhibitor Subscribe depends on the number and duration of intervals for which the user's presence application is active.

To calculate the presence traffic due to presence publications, we assume a user who publishes presence changes 15 times a day. We assume the presence document previously described, whose size is 750 bytes. We estimate presence traffic in the following scenarios: the user is connected to a standard PS without optimization strategies, the

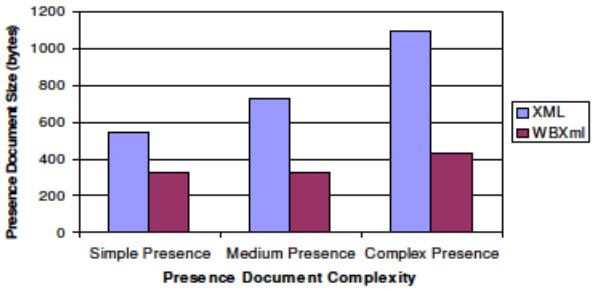


Figure 9.7: Size of presence documents encoded by XML and WBXML

user is connected to a PS that can process partial publications, and finally the user is connected to a PP that extends her presence. From the calculations done, we can conclude that partial publications reduce the total presence traffic generated by a standard PS by approximately 34%. When the PP applies presence extension, it reduces the presence traffic by approximately 20%. Since partial publications significantly reduce the uplink traffic caused by presence changes, it is highly recommended. Nevertheless, if the user device does not have enough processing resources, presence extension is a reasonable alternative because it does not require extra processing on the user devices.

We assume that presence documents are encoded by WBXml. Although WBXml is simple and not particularly flexible, it achieves a high performance when the larger part of the XML document consists of XML tags and attributes rather than textual content. As SIMPLE presence documents are rich in XML tags, WBXml may substantially reduce the size of presence documents. Figure 9.7 shows the size of several presence documents encoded by XML and WBXml. Each document has a certain complexity level according to its number of XML tags. The simple document contains the activity that the user is doing, the place type where the user is and his state, display name and mood, as well as basic information on a service on which he is available. The medium document contains the aforementioned information in addition to the user's home page and capabilities of the service available. Lastly, the complex document includes all the aforementioned information plus the SIP event packages supported, the available communication methods and the characteristics of the user device.

9.4 HTTP Traffic Optimization

Concentrating multimedia traffic in a single point that is close to the user provides much flexibility for applying techniques that improve the user's QoE. These techniques may be applied based on the user's preferences, his or her context or traffic optimization policies. To date, PP implements various techniques for reducing HTTP traffic and therefore decreasing the response time perceived by users while browsing. To this end, the user's devices need to be configured with the user's PP as HTTP proxy. HTTP optimization is specially useful when billing for data services is carried out by volume or the access link has limited bandwidth and is shared by many users. The server part of PP, which is the proxy as described in Section 9.2, implements an HTTP and DNS cache. An HTTP cache at the proxy may be very effective, since some HTTP requests sent by the user may be replied by the proxy locally, without querying the remote web server. DNS caching may also be efficient when the network capacity is restricted and shared by many users. In this case, the download time of web pages may increase due to name server resolutions. Caching DNS responses at the proxy allows replying some of the user's DNS requests locally without forwarding these requests to the original DNS servers. Moreover, the proxy compresses the textual content of HTTP responses for saving bandwidth on the user's downlink.

In addition to the above-mentioned techniques, the proxy and the client part of PP, which is called CPM as described in Section 9.2, implement a technique called HTTP header reduction collaboratively. This technique optimizes the size of HTTP requests on the user's uplink by removing the most static content negotiation headers of HTTP requests at the user side, and afterwards reconstructing them at the network side. This technique is based on the assumption that the proxy maintains a database of device profiles, which contains the characteristics of the user's devices. PP uses a sophisticated device capabilities detection and user profiling system [208]. This detection mechanisms basically executes some applets on the user device for automatically detecting its characteristics when the device registers in its PP for first time. Based on the detected characteristics, the proxy creates a local profile for the user device. If the device had an external UAProf profile, it would be compared with its local profile for enhancing the latter. To perform HTTP header reduction, the CPM needs intercept HTTP requests and collaborate with the proxy. When an HTTP request is to be sent,

the CPM removes any HTTP header that is dispensable for web browsing from the request (i.e., headers other than *Host* and dynamic headers such as those for cookies). In addition, the CPM modifies the *User-Agent* header to be the identifier of the end device's profile. Then, the CPM forwards the HTTP requests to the proxy, which regenerates the missing HTTP headers before forwarding this request to the original server. The headers of the HTTP requests sent to the original web servers have to be restored for letting these servers know about browser capabilities (i.e., content negotiation headers) and navigation information (i.e., cookies, referrer, host, etc.). Otherwise, web navigation, content negotiation or content adaptation may be degraded. A web server with minimal intelligence is capable to adapt an HTTP response's content according to its HTTP request's headers. Thus, the proxy may modify HTTP headers according to its privileged vision of the requester device's capabilities and access link congestion. Different HTTP header regeneration policies may therefore be applied. For example, some HTTP headers may be changed based on the cellular link's available bandwidth for making remote web servers deliver lighter contents, thereby mitigating the link congestion and the delay perceived by users.

9.4.1 Experimental Results

This section discusses the PP efficiency at reducing HTTP traffic. The PP functionality related to HTTP traffic optimization described previosly was transferred to the corporations Vodafone R&D and Fundació I2Cat through a research project. This project was aimed to optimize mesh networks connected to Internet via UMTS or GPRS. This allowed us to experimentally measure the response time of HTTP requests when a user browses the Web through his or her PP in a real scenario. This scenario is depicted by Figure 9.8 and described as follows. Mesh, ad hoc or local area networks without wired Internet connection may connect to Internet via UMTS. These kinds of network may also work as a means of indoor coverage for UMTS operators. In these scenarios, the cellular link is shared by all the users connected to the mesh network and may provide low bandwidth (e.g., GPRS). Thus, optimizing the access to some Internet services may decrease the response time perceived by end users. The collaboration between a user-side gateway and a network-side proxy server may allow optimizing the traffic sent and received by users in the mesh network. Consequently, the PP functionality for

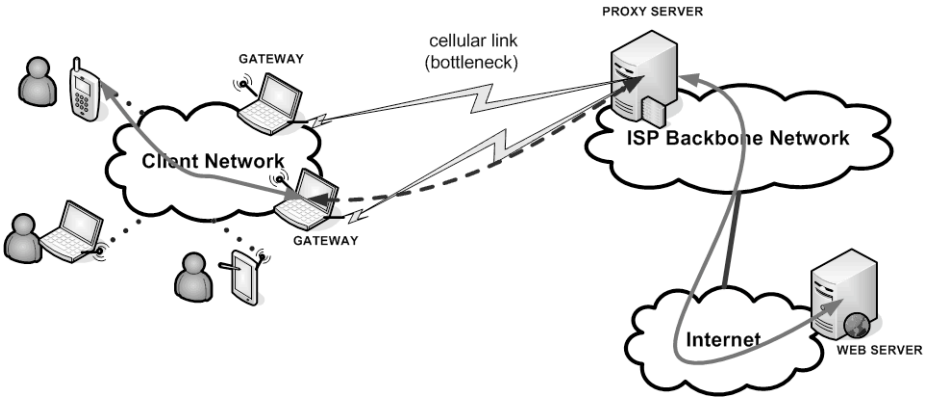


Figure 9.8: Scenario where some PP functionality is placed into a proxy server for HTTP optimization

HTTP traffic optimization described previously was integrated into the scenario in Figure 9.8. The CPM and SPM functionality for HTTP header reduction was placed into the client-side gateway and the network-side proxy server in Figure 9.8, respectively. The SPM functionality for HTTP content compression was placed into the network-side proxy server in Figure 9.8. DNS and HTTP caching was performed at the client-side network for reducing the traffic sent on the UMTS link. On the contrary, PP performs caching at the network-side, which reduces traffic on the network core rather than the access link. Although PP is mainly designed to be at the network-side, it may be possible to have PP working for users in a mesh network locally. In this case, multiple personal proxies would interact locally and act as user-side gateways for a UMTS network. Only in this particular scenario, the experimental results about HTTP and DNS caching reported below are applicable to the PP caching.

We tested the download time of various web pages when the PP techniques for optimizing HTTP traffic were performed in the above-mentioned scenario. To this end, a user was connected to a gateway for accessing to a UMTS network. For the tests performed, the user requested web pages through the gateway, which in turn forwarded the user's requests to the network-side proxy. In order to avoid the effect of public server congestion, we deployed a virtual web hosting system at the laboratory network that mirrored popular web pages. This system had the same hardware and software characteristics as the network-side proxy server. Thus, the user still accessed these web pages through the Internet but the servers' work load was kept under control. Tests

were executed with the equipment listed below:

- Gateway: Linux-based laptop (Debian 4.0 distribution with 2.6.18 kernel version) with Intel ipw3945 802.11a/b/g chips.
- Cellular link: HSDPA link, approximately 1 Mbps downlink and 384 Kbps uplink.
- Access Network: 802.11.b.
- End device: Windows-XP-based UltraMobilePC using Mozilla 1.7.12 for web browsing.
- Network-side proxy server: PC Intel P4 3GHz dual core processor and 2 GB of Random-Access Memory (RAM) with Linux Operating System (OS) (Debian 4.0 distribution with 2.6.18 kernel version). Proxy functionality is given by a Java-based HTTP server.

Each web page was requested 10 times, with optimizations disabled both on the gateway and the network-side proxy. Afterwards, the same experiment was repeated applying each optimization technique. A *Gzip* compressor was used to reduce the size of the textual content of the HTTP responses sent to the user. Textual content compression achieved an average decrease of 8% download time. We used a tool called *pdnsd*, which is a simple, lightweight DNS cache for linux. This cache decreased the download time by 7% on average. We used the HTTP cache *Squid* [211]. The improvement achieved by an HTTP cache obviously depends on the user's web browsing behavior. When the entire web page is saved in the cache the download time is reduced by about 90%, since the proxy responds locally. The methodology for testing HTTP header reduction is summarized below:

1. The user-side gateway and network-side proxy used a detection mechanism [208] that allows the latter to know the HTTP headers that are dispensable on the link between these two nodes and therefore will be removed by the former. The proxy stored these HTTP headers in a database.
2. Whenever the user sent a GET request, the gateway removed the dispensable HTTP headers of this request and forwarded the request to the network-side proxy.

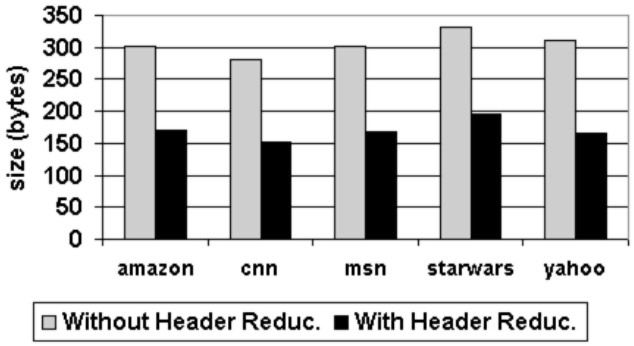


Figure 9.9: Average size of HTTP requests on uplink

3. Whenever the network-side proxy received a GET request, it added the removed HTTP headers back to the request and forwarded it to the web server.

4. The first time the user requested a page, the network-side proxy retrieved the headers from the database, but in subsequent requests the header values were cached and the database was not accessed.

5. The gateway calculated the size of the original and the reduced GET request for each web page request. Thus, we calculated the average number of bytes over the access link saved by applying HTTP header reduction.

Figure 9.9 shows the average size of GET requests for each tested web page, with and without HTTP header reduction. The different amounts of bytes saved by HTTP header reduction are due to the existence of cookies and similar dynamic headers. HTTP header reduction reduces the size of GET requests by removing only content negotiation headers, which are static. These headers obviously vary between different browsers and may even vary between two browsers of the same family and version because of different plug-ins installed.

Figure 9.10 shows the average improvement in response time for the tested web pages, which is about 10%. The differences in response time mainly depend on the number of objects rather than the type of these objects. The number of GET requests increases with the number of objects. Thus, the improvement achieved increases with the number of objects because more requests are optimized. In addition, it is demonstrated [212] that the downloading order of the objects of a web page affects the overall

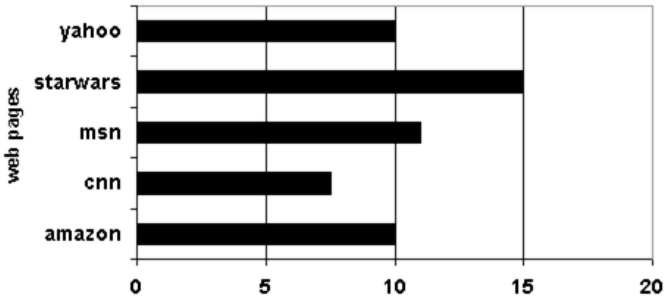


Figure 9.10: Average improvement in Response Time (%)

response time and that it is better to interleave small objects with large objects to keep the server transmitting rather than waiting. Differences in web document structure therefore affect the downloading order. Thus, the size of consecutive objects also affects the response time in conjunction with the number of objects that make up the web page.

9.5 Conclusions

We addressed the need to deploy intelligent, user-centric applications that require scalable and efficient platforms for collecting, processing and disseminating presence information in a user-personalized way. Thus, we proposed a decentralized proxy-based solution for handling fine-grained rules about presence information management, which we refer to as Personal Proxy (PP). This is a logical functionality that may be included in a home gateway or even a femtocell as a value-added service for managing the users' presence and communications in a distributed way. The nature of this architecture provides a great flexibility to converge multimedia protocols in a single point and tightly personalize user communications and privacy. PP also would ease the impact of the presence service on network operators since it does not require a centralized PS shared by numerous users. Although advanced users may have total control of their PPs, other use case is some network operator providing users with their PPs and therefore administrating it. We analytically estimated the performance of some techniques implemented by PP for reducing presence traffic. The reported results showed that these techniques help in reducing presence traffic. We performed some experiments for

testing the HTTP traffic optimizations implemented by PP. These experiments showed that PP decreases the response time perceived by users when browsing the Web.