

Efficient SIP-Specific Event Notification

Bo Zhao
Network Solution Group
Bell Labs
Beijing, China 100102
bzhao@lucent.com

Chao Liu
Department of Computer Science
University of Illinois-UC
Urbana, IL, U.S.A. 61801
chaoliu@cs.uiuc.edu

Abstract

By Session Initiation Protocol (SIP), the entities in the network can subscribe to the resource or the call state for various resources or calls in the network. Those entities (or entities acting on their behalf) can send notifications when those states change. We propose a new approach to subscribe the information of part resources of the user's resource list. Unlike existing approaches that either use redundant SIP messages to subscribe to these resources one by one, or use minimal SIP messages to subscribe to the entire resource lists to get unnecessary information, our approach uses minimal SIP messages to get necessary information of these resources. We implement it in Enhanced Resource List Server (ERLS) project and compare it with previous approaches by the performance evaluation.

We systematically evaluated our approach under the same setting as previous ones. The system result demonstrated the power of our approach in reducing the CPU usage and the data size transferred: our approach only used one third as much CPU usage and data size transferred as the best previous approach at certain cases.

1 Introduction

The Session Initiation Protocol (SIP) [9] is an application-layer protocol used for establishing and tearing down multimedia sessions, both unicast and multicast. It has been standardized within the Internet Engineering Task Force for the invitation to multicast conferences and Internet telephone calls [10]. Next-generation Internet telephony networks are using SIP, including those proposed by PacketCable for cable modems and the Third-Generation Partnership Project (3GPP)[1] for next-generation wireless.

Developers have also extended the SIP to generate event notifications [6] and instant messages [8]. Users subscribe to an event using the SUBSCRIBE [9] method and receive notifications via NOTIFY messages [9]. Event notification

is typically used for presence notification and event signaling during telephone calls. The entities in the network can subscribe to the resource or the call state for various resources or calls in the network, and those entities (or entities acting on their behalf) can send notifications when those states change. Figure 1 shows the typical flow of messages.

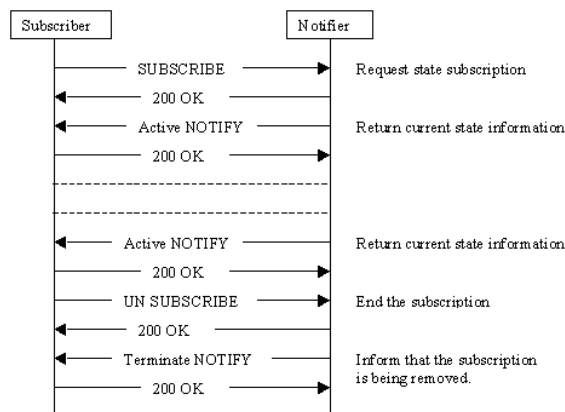


Figure 1. Call Flow of Notification Mechanism

The original approach [6] repeats the scenario of Figure 1 to send SUBSCRIBE messages and receive NOTIFY messages for each resource individually. Since the cost time of parsing SIP message is distinct [3], this approach increases the workload of the SIP server for subscribing multiple resources. In order to increase the efficiency of subscribing to multiple resources, another approach [7] is created. It extends the Session Initiation Protocol (SIP)-Specific Event Notification mechanism for subscribing to a homogeneous list of resources. Instead of the subscriber sending a SUBSCRIBE message for each resource individually, the subscriber can subscribe to an entire list, and then receive notifications when the state of any of the resources in the list changes. The subscriber only executes the scenario of Figure 1 once to get the information of multiple resources that are in the same list.

Although the second approach has succeeded in subscribing multiple resources, it is still not efficient and not flexible enough to subscribe to a part of the resource list(s). It has to subscribe to the entire resource list(s) and receive much unnecessary information of all the resources of the list(s) for only part of them.

Because of the above issue we are motivated to develop a new approach that integrates the advantages of the two above approaches in order to use minimal SIP messages to get necessary information of these resources. It is achieved by including URIs of the resources in the body of the SUBSCRIBE request. The subscriber subscribes to them, and gets notifications about changes in the resources' states via a single SUBSCRIBE dialog.

The rest of the paper is organized as follows. Section 2 provides a motivating example to illustrate the advantages of our approach over previous ones. We then develop the implement in Section 3. Systematic evaluations and comparisons are presented in Section 4, after which related work is discussed in Section 5. Section 6 concludes this study.

2 A Motivating Example

In this section, we present a motivating example showing the value of our approach compared to the existing ones.

For instance, a subscriber has 50 resources in its every list. It subscribes to the information of 10 resources that are divided in 2 lists. Normally, the size of the message header is 0.4 KB. The size of the information of every resource is 0.5 KB. The subscriber gets all the information by 2 active NOTIFY messages.

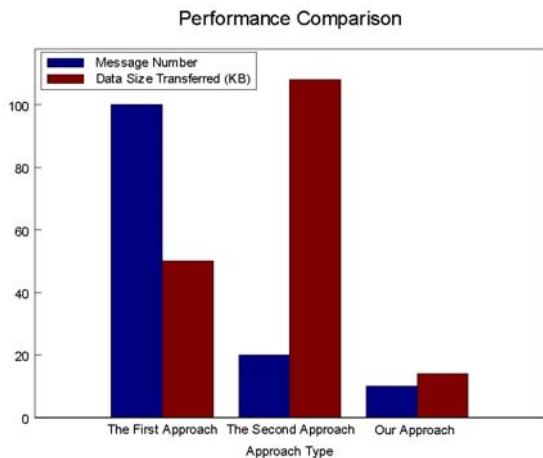


Figure 2. Performance Comparison

Figure 2 shows the number of messages and the data size transferred by three approaches. Y-axis stands for the number of both message number and data size transferred in KB.

For instance, while the first approach uses 100 messages, our approach only uses 10. While the first approach transfers 50 KB, our approach only transfers 14 KB. Clearly our approach uses much less messages and transfers much less data than others to achieve the same task.

Because the first approach has to repeat the scenario of Figure 1 for 10 times to subscribe to all the resources, it uses the most number of messages. Because the second approach has to transfer all the resource data on 2 lists, it transfers the most size of the data. Obviously, our approach executes the scenario of Figure 1 only once and transfers the data of only 10 necessary resources. Meanwhile we note that the number of messages and data size transferred by the first approach are linear with the number of resources subscribed. Furthermore, those of the second approach are linear with the number of lists where the resources subscribed are divided. We illustrate this by the analysis in section 3.3.

In sum, the above example illustrates a simple but representative case. It indicates that our approach is the best on both the number of messages and the data size transferred, compared to other two approaches. In the next section, we describe our implementation.

3 Implementation

In this section, we introduce the ERLS that implements our approach in section 3.1. Then we provide the service logic and call scenario that use our approach in section 3.2. Section 3.3 presents a detailed analysis that shows the value of our approach comparing to the existing ones. We leave the detailed implementation in section 3.4.

3.1 Introduction to ERLS

Enhanced Resource List Server (ERLS) is a resource management server that enhances the existing Resource List Server (RLS) [7] to support our approach. It supplements the handset capabilities through interactions with a network hosted server and through the seamless incorporation of dynamic presence information in the presentation of various contacts to the end-user using the handset. ERLS implements the Group List Management Server (GLMS)[5] [4] and RLS capabilities defined in the 3GPP and IETF standards respectively. It also interfaces seamlessly into the standards defined Presence Server entity. The RLS function is also sometimes referred to as the Presence List Server (PLS) functional element in the 3GPP architecture documents [10].

In this paper, we only focus on the function of dynamic presence information in the presentation of various contacts to the end-user via the handset. The client is the subscriber and ERLS is the notifier for the client.

Components and their functions:

1. Client: This logical element is on the handset. It publishes its presence information to PS, and performs three approaches of SUBSCRIBE messages and receives notifications of updates of contacts' presence information.
2. Presence Server (PS): ERLS relies heavily on the Presence Server to collect dynamic network presence information of an implicit nature based on devices' connectivity to the core network. In addition it obtains this information from various network elements, as well as the user specified presence information of an explicit nature gathered via the SIP PUBLISH messages. The PS also reacts to back-end SUBSCRIBE messages that ERLS may issue and respond with notifications for individual contacts. PS only supports the first approach of SUBSCRIBE.
3. ERLS: it implements the RLS function defined in IETF and enables clients to request three subscription approaches and receive notifications of updates of contacts' presence information. It also uses the first subscription approach to send back end SUBSCRIBE messages for the information of individual contact.

2. PS sends notification with the current information of Client B to ERLS.
3. Client A subscribes to the presence information of Client B to ERLS by one of the three approaches of SUBSCRIBE.
4. ERLS sends the notification with the current information of Client B to Client A.
5. Client B publishes its current presence information to PS.
6. PS sends notification with the last information of Client B to ERLS
7. ERLS sends notification with the last information of Client B to Client A.

This study only focuses on the service logic of scenarios 3, 4 and 7. When the user wants to subscribe to any presence information to ERLS, the client can automatically decide which approach of the SUBSCRIBE message is most efficient. Table 1 shows the selection logic.

3.2 Service Logic

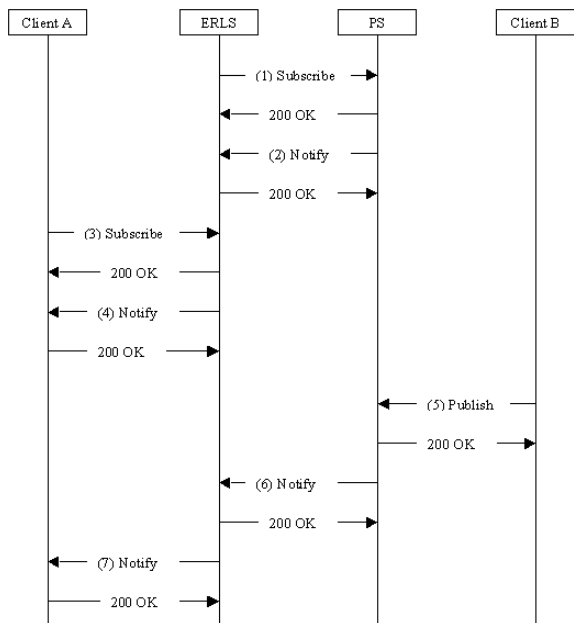


Figure 3. Call Flow of Service Logic

Figure 3 shows the whole scenario of the ERLS's service logic:

1. ERLS subscribes to the presence information of Client B to PS by the first approach of SUBSCRIBE message.

The Information Subscribed	The Approach Selected
A single resource	First Approach
All the resources of a list or lists	Second Approach
Part resources of resource list or of multiple resource lists	Our Approach

Table 1. Selection Logic

3.3 Analysis

In this section, we present a detailed analysis showing the value of our approach compared to the existing ones.

A subscriber has n resources in its every list ($0 < n$). It subscribes to the information of m ($0 < m < n$) resources that are divided in x ($1 \leq x \leq m$) lists. y is the size of the information of every resource in the body of NOTIFY message. z is the size of message header. l is the number of active NOTIFY messages sent from the notifier to the subscriber. Since l , y , z are impacted by the system environment and service logic, but not by the approach type, we consider that they are invariant.

Let

$$f_1(m) = (6 + 2l) \cdot m$$

be the number of messages used by the first approach to subscribe to the information of m resources. This approach has to subscribe to each resource individually.

Let

$$f_2(x) = (6 + 2l) \cdot x$$

be the number of messages used by the second approach to subscribe to the information of $x \cdot n$ resources from x lists for these m resources. This approach has to subscribe to each list individually.

Let

$$f_3() = 6 + 2l$$

be the message number used by our approach to subscribe to the information of m resources.

Let

$$B_1(m) = l \cdot y \cdot m$$

be the body size transferred by the first approach.

Let

$$B_2(x, n) = l \cdot y \cdot x \cdot n$$

be the body size transferred by the second approach.

Let

$$B_3(m) = B_1(m) = l \cdot y \cdot m$$

be the body size transferred by our approach.

The total size of message headers is $f_n \cdot z$ ($n = 1, 2, 3$)

(The size of the URIs of resources in the body of SUBSCRIBE request is very small and can be ignored.)

From Table 2 and 3, we conclude that the numbers of messages and data size transferred by the first approach are linear with the number of resources subscribed (m). Those of the second approach are linear with the number of lists into which the resources subscribed are divided (x). The data size transferred by the second approach is also linear with the size of every list (n). The number of messages in our approach is invariant. The data size transferred by our approach is linear with the number of resources subscribed (m). The proportions of message number and data size transferred indicate that our approach uses the least number of messages and transfers the least data size, by comparing to the other two approaches. Our proposal is confirmed by the system result shown in section 4.

3.4 Implementation of Our Approach

The first approach [6] increases the workload of the SIP server for subscribing multiple resources. In order to increase the efficiency of subscribing to multiple resources, the second approach [7] is created. Instead of the subscriber sending a SUBSCRIBE message for each resource individually, this approach allows the subscriber to subscribe to an entire list, and then receives notifications when the state of any of the resources in the list changes. As such, a subscriber only executes the scenario of Figure 1 once to get the information of multiple resources in the same list.

Although the second approach has succeeded in subscribing to multiple resources, it is still not efficient to subscribe to a part of the resource list(s). It has to subscribe to the entire resource list(s) and get much unnecessary information of all the resources of the list(s) for only part of them.

The above issue has motivated us to develop our approach, which integrates the advantages of the two above approaches in order to use minimal SIP messages to get necessary information of these resources.

When the client wants to subscribe to part resources, it includes the URIs of these resources in the body of the SUBSCRIBE request message. After ERLS receives this request message, it parses its message body to get the URIs of resources. According to them, ERLS gets the presence information of these resources and follows the notification mechanism of RLS [7] to build the message body of NOTIFY request message. Finally, ERLS sends it to the client. Then client parses the message body of NOTIFY request [7] to get the presence information.

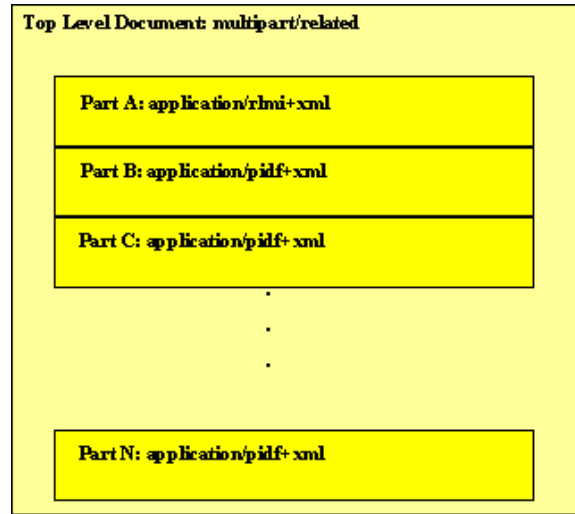


Figure 4. Body Structure of NOTIFY Request

Figure 4 shows the structure [7] of message body of NOTIFY request. It consists of a multipart/MIME document with one rlmf+xml part and up to N different pdf+xml parts, each of which corresponds to a subscription state resource addressed in the rlmf+xml element. The rlmf part of the document essentially indicates the subscription state for each resource. For detail of Figure 4, please refer to [7].

The following is an example of the message body of a SUBSCRIBE request, which carries URIs of part resources (A, B, C) of the list in its body and is sent by a subscriber to a notifier.

```
<?xml version="1.0" encoding="UTF-8"?>
<group>
```

	Message Number	Proportion of Message Number
The first approach	$f_1(m)$	m
The second approach	$f_2(x)$	x
Our approach	$f_3()$	1

Table 2. Proportion of Message Number

	Header	Body	Total	Proportion of Header	Proportion of Body
The first approach	$f_1(m) \cdot z$	$B_1(m)$	$f_1(m) \cdot z + B_1(m)$	m	1
The second approach	$f_2(x) \cdot z$	$B_2(x, n)$	$f_2(x) \cdot z + B_2(x, n)$	x	$x \cdot n/m$
Our approach	$f_3() \cdot z$	$B_3(m)$	$f_3() \cdot z + B_3(m)$	1	1

Table 3. Proportion of Data Size Transferred

```

<resource uri="sip:A@example.com" />
<resource uri="sip:B@example.org" />
<resource uri="sip:C@example.net" />
</group>

```

4 EXPERIMENTAL RESULT

Performance metrics are always a central issue in accurate and objective comparisons. We use the analysis and the means of each parameter that are already presented in section 3.3. In this section, we focus on measuring and comparing the cost of each approach on both data size transferred and CPU usage at different conditions. Our solution is to optimize the SIP-Specific Event Notification mechanism. Other protocols do not have it. Therefore, this experiment evaluation does not compare our approach with other existing protocols.

4.1 Testing Environment

Our tests run on an isolated Fast Ethernet network, using Sun Netra 1400. The machine is equipped with 4 Sparc 440 MHz CPUs with 4 GB physical memory. All the machines run Solaris* 9.

4.2 Description of Scenario

We developed a load generator. It simulates the subscriber and is responsible for controlling the overall test. It controls the test duration, generates the desired load, and guarantees a uniform distribution of new calls. It also starts and terminates calls, and collects statistics. ERLS is the notifier. Figure 1 shows the call flow of a single run in section 1.

The definition of a single call for each approach is that the scenario is produced by the mechanism of each ap-

proach once. The definition of the call and the call rate of each approach are shown below.

1. The first approach:

One CALL Definition: The scenario is produced at m times by the mechanism of this approach to transfer the information of m resources.

Call Rate Definition: Single call rate/ m (The subscriber need to subscribe to each resource individually).

2. The second approach:

One CALL Definition: The scenario is produced at x times by the mechanism of this approach to transfer the information of m resources.

Call Rate Definition: Single call rate/ x (The subscriber need to subscribe to each list individually).

3. Our approach:

One CALL Definition: The scenario is produced once by the mechanism of this approach to transfer the information of m resources.

Call Rate Definition: Single call rate.

4.3 Descriptions and Purposes of Each Test Case

For each approach, three independent tests were executed:

1. CPU usage with the change of Call Per Second (CPS): It collects data from the CPU usage of each approach with the change of CPS at the special condition. By analyzing the data, we can conclude which approach is the best at this condition. The other two tests analyze the impact of the change of the condition.

- Impact of the list size: It collects data of the CPU usage of each approach with the change of list size (n), one of the special conditions. By analyzing the data, we can conclude the impact of the change of this condition.
- Impact of the number of resources subscribed by the subscriber: It collects data of the CPU usage of each approach with the change of the number of resources (m), one of the special conditions. By analyzing the data, we can conclude the impact of the change of this specific condition.

After analyzing all the data collected by these tests, we can conclude which approach is most efficient on CPU usage and data size transferred.

4.4 Performance Comparison

4.4.1 CPU usage with change of CPS

Special condition: $n = 40, m = 5, x = 2, l = 3, z = 0.4$ KB, $y = 0.5$ KB

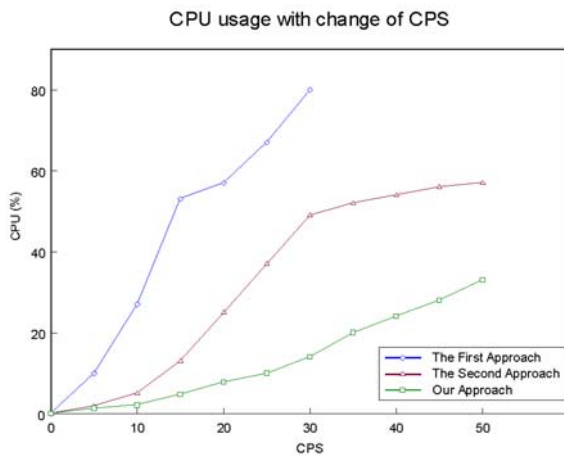


Figure 5. CPU Usage with Change of CPS

Figure 5 shows CPU usage with the change of CPS for three approaches. The CPU usage is the highest for the first approach, followed by the second approach. Our approach is the lowest in regards to CPU usage. When the CPU usage of the first approach is 82%, near the limit, those of the second approach and our approach are 49% and 14% respectively. The CPU usage per call increases linearly with the load for every approach [3]. As Figure 6 shows, the number of messages of approaches is linear with their CPU usage. Since the second approach has to transfer the data of entire resource lists, it transfers 129.6 KB data, much more than others' 31.5 KB and 12.3 KB. In Figure 6, the y-axis stands for the number of both message number and data size transferred in KB.

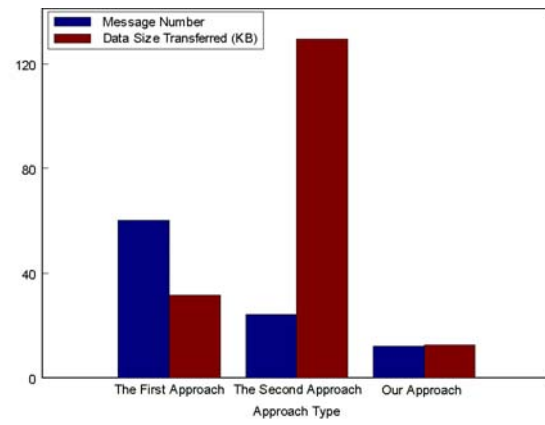


Figure 6. Performance Comparison

4.4.2 Impact of the list size

Special condition: $m = 5, x = 2, l = 3, z = 0.4$ KB, $y = 0.5$ KB

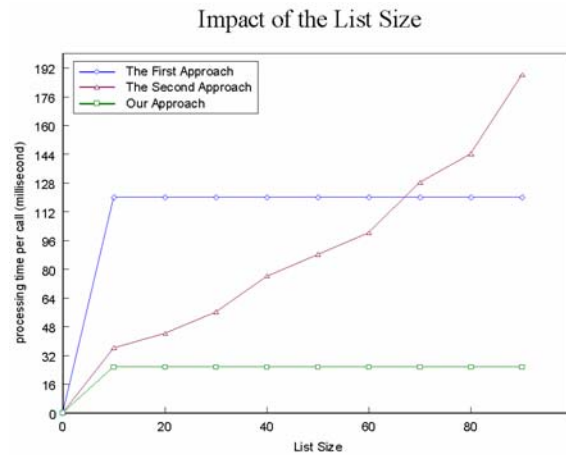


Figure 7. Impact of List Size

Figure 7 and 8 illustrate that the list size only impacts the second approach on both CPU usage and data size transferred. The CPU cost that is spent on building the body of NOTIFY message and the size of the body of the NOTIFY message, increases linearly with the list size for the second approach. Since the first approach uses more messages than the second approach, when the list size is small, the first approach costs more CPU time. However, because the CPU cost of the first approach is invariant, that of the second approach exceeds that of the first approach when the list size is more than 70. Our approach uses less CPU and transfers less data than others.

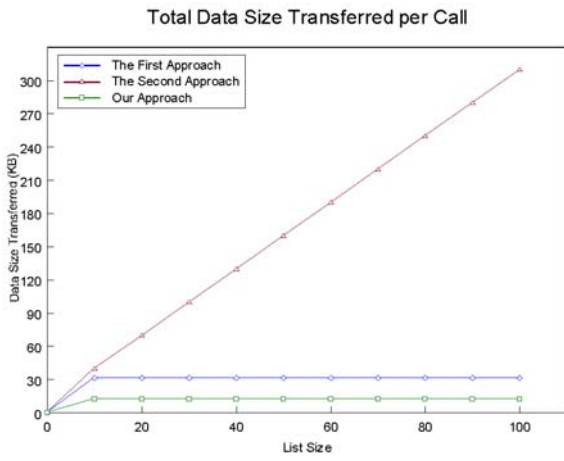


Figure 8. Total Data Size Transferred per Call

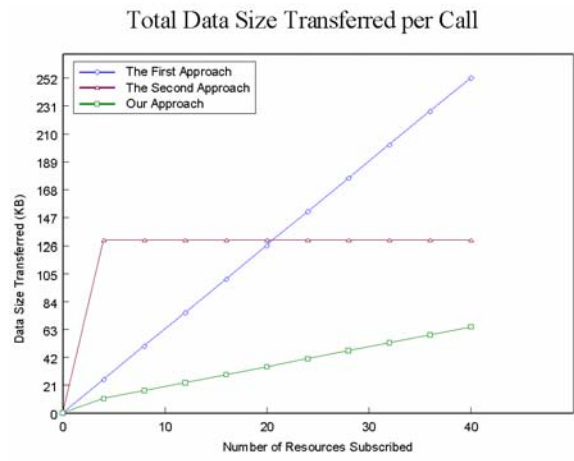


Figure 10. Total Data Size Transferred per Call

4.4.3 Impact of the number of resources subscribed by subscriber

Case 1: Specific condition: $n = 40$, $x = 2$, $l = 3$, $z = 0.4$ KB, $y = 0.5$ KB

Impact of the number of resources subscribed by subscriber

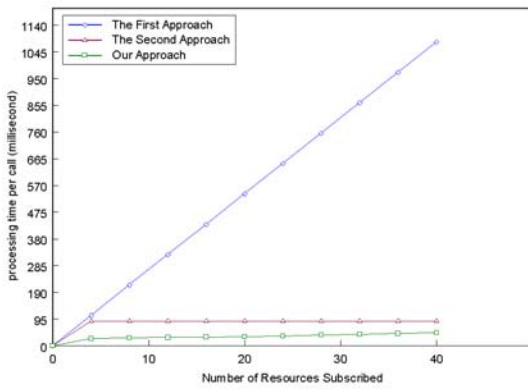


Figure 9. Impact of Resource Number

Case 2: Specific condition: $n = 40$, $x = 1$, $l = 3$, $z = 0.4$ KB, $y = 0.5$ KB

Figure 9 and 10 show that the CPU usages and the data size transferred by the first approach and our approach increase linearly with the number of resources subscribed. When more resources are subscribed, more messages are used by the first approach. Meanwhile the Notifier spends more time to parse the body of the SUBSCRIBE message and build the body of NOTIFY message for our approach. When the number of resources subscribed is small, since the body size of NOTIFY message of the second approach is more than that of the first approach, the second approach transfers more data than the first. Since the message sizes

Impact of the number of resources subscribed by subscriber

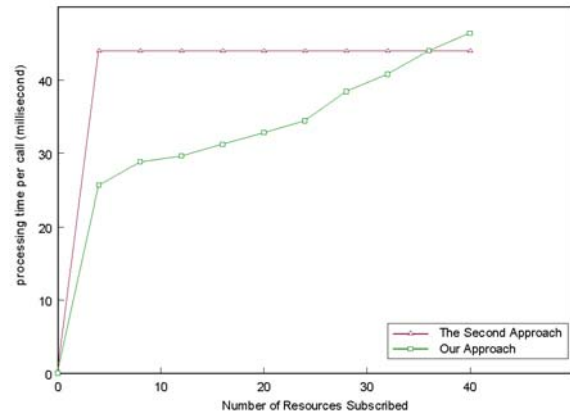


Figure 11. Impact of Resource Number

of the second approach are invariant, and the message headers of the first approach increase linearly with the number of resources subscribed, the data size transferred by the first approach is more than that of the second approach when the number of resources subscribed exceeds 21. The CPU usage and data size transferred by the second approach and our approach become close. However, they cannot be the same. Since x is equal to 2, the second approach uses twice as many messages as our approach. The Case 2 investigates the system result when they subscribe to the same number of resources with the same number of messages.

Figure 11 and 12 show that, when the second approach is used to subscribe to the resources of one list ($x = 1$), and our approach is used to subscribe to resources of equal number to the entire resources of one list, these two approaches nearly have the same efficiency both on CPU usage and data size transferred. However, since our approach has to spend

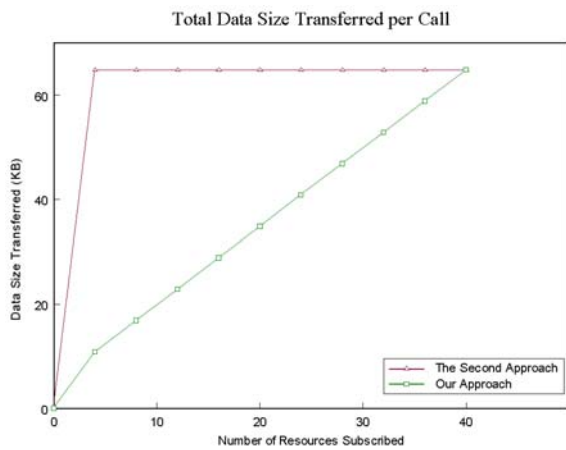


Figure 12. Total Data Size Transferred per Call

additional effort to parse the body of the SUBSCRIBE message for the URIs of resources, the second approach is a little better than ours at this special condition.

5 Related Work

“Session Initiation Protocol (SIP)-Specific Event Notification” [6] defines the general concept that entities in the network can subscribe to each resource or call state individually for various resources or calls in the network, and those entities (or entities acting on their behalf) can send notifications when those states change.

“A Session Initiation Protocol (SIP) Event Notification Extension for Resource Lists” [7] presents an extension to the Session Initiation Protocol (SIP)-Specific Event Notification mechanism for subscribing to a homogeneous list of resources. Instead of the subscriber sending a SUBSCRIBE for each resource individually, the subscriber can subscribe to an entire list, and then receive notifications when the state of any of the resources in the list changes.

“Subscriptions to Request-Contained Resource Lists in the Session Initiation Protocol (SIP)” [2] specifies a way to create the subscription to a list of resources in SIP. This is achieved by including the list of resources in the body of a SUBSCRIBE message. Instead of having a subscriber send a SUBSCRIBE message for each resource individually, the subscriber defines the resource list, subscribes to it, and gets notifications about changes in the resources’ state using a single SUBSCRIBE dialog.

Our approach takes the advantages of first two approaches together and refers to the structure of the third one.

6 Conclusions

In this paper, we propose an approach that integrates the advantage of the two previous approaches in order to use minimal SIP messages to get necessary information of part resources. It is achieved by including the resource entities in the body of the SUBSCRIBE request. The subscriber subscribes to them, and gets notifications about changes in the resources’ state using a single SUBSCRIBE dialog. Systematic evaluations by comparing to three approaches through multiple well-chosen cases, clearly demonstrates the advantages of our approach in terms of CPU usage and data size transferred.

References

- [1] 3GPP. Architecture principles for release 2000, (release 2000). <http://www.3gpp.org/ftp/Specs/html-info/23821.htm>.
- [2] G. Camarillo, A. Roach, and O. Levin. Subscriptions to request-contained resource lists in the session initiation protocol (SIP). In *draft-ietf-sipping-uri-list-subscribe-03*, 2005.
- [3] M. Cortes, R. Ensor, and J. Esteban. On SIP performance. *Bell Labs Technical Journal*.
- [4] OMA. Group management architecture, draft version 1.0. In http://member.openmobilealliance.org/ftp/public_documents/PAG/Permanent_documents/OMA-PAG-GM-AD-V1_0_0-20041118-D.zip, 2004.
- [5] OMA. Group management requirements, draft version 1.0. In http://member.openmobilealliance.org/ftp/public_documents/PAG/Permanent_documents/OMA-RD_GM-V1_0-20040926-D.zip, 2004.
- [6] A. Roach. Session initiation protocol (SIP)-specific event notification. In *RFC 3265, Internet Engineering Task Force*, 2002.
- [7] A. Roach, B. Campbell, and J. Rosenberg. A session initiation protocol (SIP) event notification extension for resource lists. In *draft-ietf-simple-event-list-07*, 2004.
- [8] J. Rosenberg. SIP extensions for instant messaging,. *IETF Internet Draft (work in progress)*.
- [9] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. R. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler. SIP: session initiation protocol. In *RFC 3261, Internet Engineering Task Force*, 2002.

- [10] H. Schulzrinne and J. Rosenberg. The session initiation protocol: Providing advanced telephony services across the internet. *Bell Labs Technical Journal*.