A P2PSIP event notification architecture

Georgios Panagiotou
Appear Networks AB,
Kista Science Tower,
164 51 Kista, Sweden
Email: georgios.panagiotou@appearnetworks.com

Alisa Devlic
Appear Networks AB,
Kista Science Tower,
164 51 Kista, Sweden
Email: alisa.devlic@appearnetworks.com

***Abstract*—This paper presents a P2PSIP notification architecture, where a traditional SIP notification server is replaced by a P2P overlay network for enhanced scalability, redundancy, as well as failure and recovery features. At the same time non-overlay SIP user agents continue to transparently access the overlay as they would in a centralized SIP notification architecture. The proposed mechanism is implemented with a structured P2P overlay network, an application level multicast protocol, and an accompanying modified SIP stack that allows the invocation of the P2P functionality. Not all the overlay peers need to have a SIP stack; only the ones acting as gateways to out of overlay SIP requests. Internally the application level multicast protocol distributes and decentralizes the entire notification process in terms of publications, subscriptions, and notifications. Finally, our proof of concept implementation is presented.**

I. Introduction

The Session Initiation Protocol (SIP) [1] support for event notifications (RFC3265 [2]) and event state publications (RFC3903 [3]), follows a topic based system approach. Thus, a topic identifier is used to uniquely identify different notification groups. Briefly, a publication destined to a particular notification group reaches the subscribed members as a notification request. In case of SIP, the Request URI header of a SIP message serves as the topic identifier. Notice that in this paper the term notification server refers to the SIP entity that processes publication and subscription requests and communicates the publications to the subscribers as SIP notifications.

Given a fixed publication rate the notification overhead experienced by a SIP server is proportional to:

$$\text{Notification overhead: } \propto \sum_{i=1}^{N} S_i$$

where N is the number of notification groups, and S_i is the number of subscribers of the i-th notification group. Such a centralized solution raises scalability concerns regarding high notification overheads and relevant benchmark studies prove this [4].

A state of the art notification architecture that meets the objectives for scalability, redundancy, as well as for failure and recovery is a challenging task. We argue that distributing and decentralizing the event notification process within a P2PSIP overlay network yields an expandable, yet scalable architecture able to accommodate an arbitrary large number of subscribers and notification groups. With a P2PSIP notification architecture we can distribute the notification groups and the actual notification load (overhead caused by subscription and notification signaling) within the overlay peers.

In order to distribute and decentralize the notification groups the hash and key based routing functionality of the overlay is used. Hashing the topic identifier of a notification group (i.e. SIP Request URI) results into a routable key address, called the group id (gid).

$$\text{gid} := \text{overlay\_hash(SIP Request URI)} \quad (1)$$

The peer that its routing table spans over gid is selected to serve this notification group. This technique is common among various application level multicast protocols ([5], [6], [7]), where the responsible peer is also known as rendezvous peer or multicast root. The hash function of the overlay guarantees that in the long run in an overlay of N peers and K notification groups each of the peers gets responsible for $\lceil \frac{K}{N} \rceil$ notification groups.

Additionally, the actual notification load can be distributed and decentralized with the aid of a tree-based application level multicast protocol. Such an improvement offloads the multicast root as the subscriptions can be handled lower the tree hierarchy. It also reduces the per link notification signaling as only one notification is needed to travel down the tree and received by the different subscribers. Regarding the notification delay, an upper bound can be considered given the fact that the length of a multicast tree is in the order of O(logN) hops, where N is the size of the overlay.

The rest of the paper is structured as follows: Section II presents the related work in the field and Section III elaborates on the proposed dissemination scheme. Section IV presents a proof-of-concept demonstrator, where Section V draws some conclusions and sets the guidelines for future work in the area.

II. Related Work

Various research proposals [8], [9] leverage SIP as the signaling protocol for a P2P overlay network. The SO/SIMPLE project [8] builds a structured P2P overlay network utilizing the Chord [10] algorithm, where ordinary SIP/SIMPLE (SIP for Instant Messaging and Presence Leveraging Excensions) [11] calls are used to encapsulate the overlay messages. The work proposed by Harjula et al. [9] extends SO/SIMPLE in the domain of mobile middleware. Similarly to SO/SIMPLE, Harjula [9] develops a middleware platform that organizes peers in a Chord ring and the SIP/SIMPLE is leveraged to

support the signaling requirements of the overlay. Additionally to the signaling demands, Harjula exploits SIP/SIMPLE for its event notification capabilities, since SIMPLE realizes the generic event notification (RFC3265) and event state publication (RFC3903) template packages. Thus, an arbitrary overlay peer can subscribe to any other peer serving event notifications. However, this approach raises resource management concerns, especially in the case where the peers are often mobile, resource constraint devices.

The IETF P2PSIP [12] working group extends the aforementioned research efforts towards a fully distributed and decentralized SIP architecture. More specifically, the working group tries to leverage the use of SIP in a P2P environment where the traditional proxy-registrar and message routing functions are replaced by some type of DHT functionality. In a P2PSIP network the peers contribute to the overlay with their complementary SIP services (i.e. STUN service, SIP to PSTN gateway, SMS gateway, etc.) in order to provide a functional, yet distributed SIP network.

Finally, the recently submitted internet draft: "P2PSIP Event Notification Extension" [13] appears great similarities with our proposed notification architecture. However this approach only deals with the dissemination of the notification groups within the P2PSIP overlay, leaving the per group subscriptions issue open.

III. System Architecture

A scenario that describes the proposed architecture is shown in Figure 1, where a P2PSIP overlay network substitutes the functionalities of a centralized SIP notification server. The actors Bob, Dylan, and Alice are non overlay SIP user agents, with Bob and Dylan subscribing to Alice's publications. It is assumed that each of Bob, Dylan, and Alice maintain a connectivity degree with the overlay, through the overlay peers A, B, and E, respectively. The way a non overlay node accesses the overlay is beyond the scope of this paper. However, the IETF P2PSIP working group already proposes how this can be achieved (i.e. through an non overlay bootstrap mechanism, DNS SRV entries, etc.) [14].

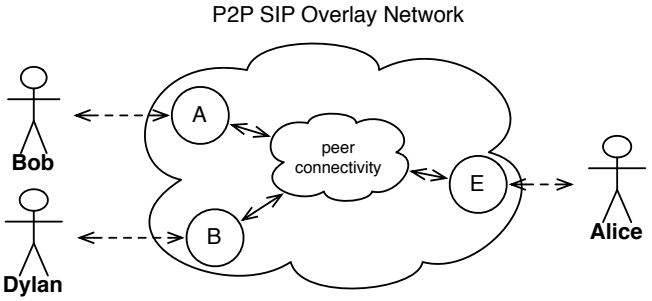

Fig. 1: SIP user agents Bob and Dylan subscribe to the overlay for Alice's publications. SIP user agent Alice publishes into the overlay and her publications reach Bob and Dylan as SIP notifications.

A. Summarizing the SIP notification server behavior

SIP event notifications can be seen as a two step process having a synchronous and an asynchronous behavior, as shown in Figure 2. On the reception of a valid SIP SUBSCRIBE request the SIP server has to immediately reply a SIP NOTIFY that encapsulates the latest known event state (*synchronous behavior*). In case the subscription is still active while a forthcoming publication reaches the SIP server; the asynchronous behavior is invoked and a subsequent SIP NOTIFY encapsulating the updated event state reaches the subscriber (*asynchronous behavior*). On the other hand, an application level multicast protocol implements **only** the asynchronous part of SIP notifications behavior. As soon as a publication is made to a group, this reaches the group subscribers.

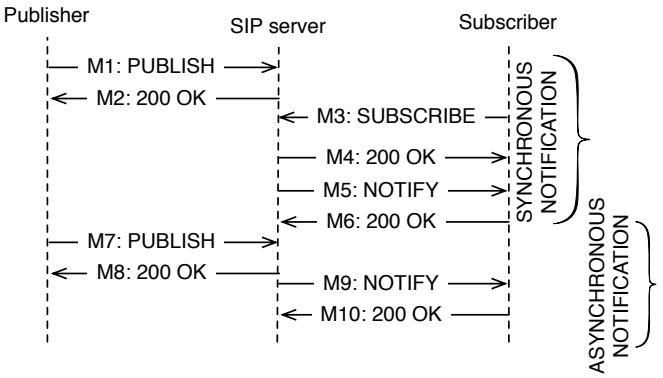

Fig. 2: The SIP server behavior on SIP PUBLISH and SUBSCRIBE calls

B. Architectural components

The proposed P2PSIP architecture utilizes a structured P2P overlay network, a custom application level multicast protocol, and the modifications required for an ordinary SIP stack to invoke the underlying P2P functionality. Primarily, the architecture should be able to receive SIP notification calls (PUBLISH, SUBSCRIBE) from non overlay SIP user agents and further distribute and decentralize the notification process among the overlay peers. A custom application level multicast protocol is required since the way SIP handles notifications (synchronous **and** asynchronous behavior) differs from the way application level multicast protocols do so (asynchronous behavior only).

The proposed application level multicast protocol builds upon Scribe [5] and extends it to support SIP like notifications behavior. Scribe is an application level multicast protocol that builds dissemination trees based on the reverse path forwarding technique, first implemented in IP multicast. Although similar realizations of tree based application level multicast protocols exist ([6], [7]), the advantage of Scribe is the ability to distribute and decentralize the entire notification process within the peer overlay; thus avoiding hot spots. Scribe also caters for peer failure, recovery and state replication for all the multicast peers including the multicast root. Comparative

studies [15] show that Scribe outperforms similar application level multicast implementations since the employed reverse path forwarding technique exploits the routing asymmetry of certain popular overlay algorithms (i.e Pastry, Chord) resulting in better a performance.

However, Scribe implements a multicast behavior: it builds multicast trees and asynchronously disseminates information down the tree on the reception of a multicast call. We would like to make Scribe SIP-aware, in order to implement decentralized and scalable notification server functionality. To achieve this we needed to change Scribe's behavior by following the SIP notification model. This results in a new protocol called SIP aware Scribe (sa-Scribe), that adds Scribe a new functionality to fetch the latest known notification state on a new subscription call (i.e. the synchronous behavior).

The need for synchronous response in Scribe increases the complexity of the protocol, and a simple acknowledgment mechanism similar to the one leveraged by SIP is needed to control the request and responses between sa-Scribe peers. Sa-Scribe needs to support three types of acknowledgements: the positive (ACK), the negative (nACK), and the provisional (pACK) acknowledgment. Provisional acknowledgements are needed because a sender often routes a message through multiple intermediate overlay peers without a-priori knowledge of the multicast root. A pACK maps to the SIP response codes 1xx, an ACK to the SIP response codes 2xx, where a nACK covers the SIP response range [3xx, 6xx]. Along with the acknowledgments an ordered transmission scheme is considered (leveraging sequence numbers).

The sa-Scribe also changes the way publications in the overlay are performed. The create and multicast calls of Scribe get merged into a new publish call, which creates a new notification group (if none exists) for a particular gid, and also carries the message to be multicasted to the participants of the group. In order to comply with the SIP PUBLISH behavior, sa-Scribe's publish is enabled entity tag (etag) support. The use of etags enables multiple publishers that publish information under the same gid, to know if their publication is the latest or if another publisher generated a more recent. This functionality is very desirable especially in situations where multiple publishers need to cooperate their publication actions without having to subscribe for their own publications. Analytically, the actions performed by sa-Scribe on the reception of publish and subscribe calls are given in the Subsection III-D and III-E.

C. The overlay network

The internal formation of the overlay network used in our message sequence charts is shown in Figure 3. It is assumed that the overlay utilizes 10-bit long addresses resulting in an identifier space of K = 2^{10} = 1024 unique addresses. The overlay consists of eight peers, each one holding equal portions of the original identifier space. Both the address space assigned to each of the peers as well as their respective routing tables are shown.

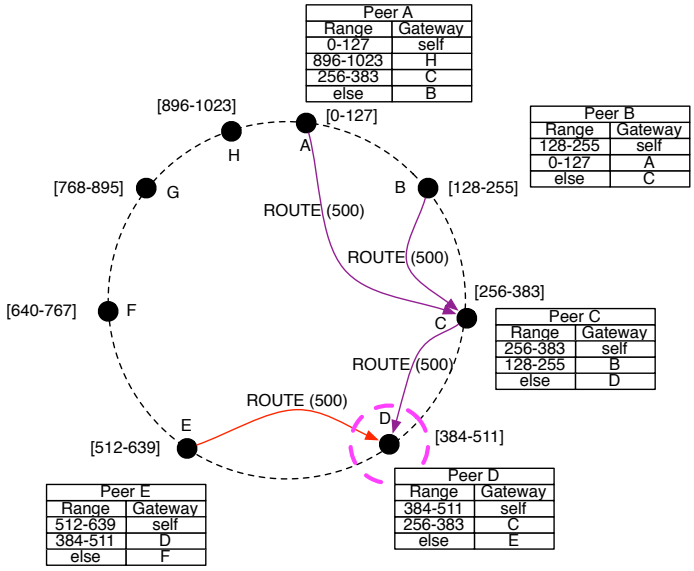

Fig. 3: Internal deployment and connectivity of the P2P SIP overlay utilized in the example

For example a route operation to the routing address 500 initiated from peers A and B would result at peer D, through peer C. On the contrary the routing table of E contains the information that for the key 500 responsible peer is D (since D's address space spans over the key range [384, 511]).

D. Message sequence chart - Publication

As shown in Figure 4, peer A receives a SIP PUBLISH request and responds with a SIP 100 Trying provisional response. Additionally, peer A invokes a local publish call to the sa-Scribe protocol, providing the gid, an expiration value, and the publication data. In case the publish call from Alice carried etag information the etag value should be included in the local publication call. The publish call reaches the multicast root for that gid (peer D) and a new state for that publication is created (if there is not any). Subsequently, sa-Scribe running on D acknowledges the reception of the publish call with a positive acknowledgment that carries the gid, the final expiration value as decided by D, and the etag information. Upon the reception of this response peer E composes a SIP 200 OK response, which is propagated back to Alice.

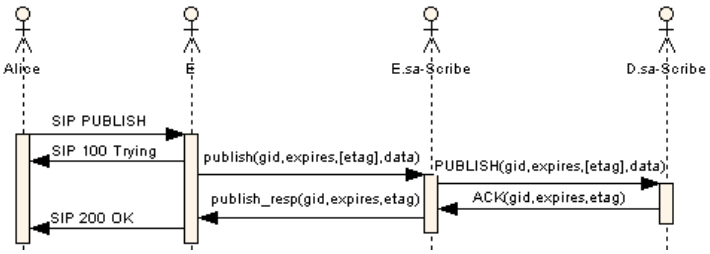

Fig. 4: Publishing information into the overlay

E. Message sequence chart - Subscription

The actions performed by sa-Scribe on the reception of a subscribe call are shown in Fig. 5. Following Fig. 5, the non overlay user agent Bob sends a SIP SUBSCRIBE request to the overlay with first point of contact peer A. The modified SIP stack of peer A issues a subscribe call to the sa-Scribe protocol running on the same node providing the gid and the proposed expiration value as arguments. Consecutively, the sa-Scribe issues a JOIN message and routes it through the overlay towards gid. The sa-Scribe protocol running on peer C receives the join and replies back to A with a provisional acknowledgment, while it further routes the JOIN request in the overlay towards D; which serves as the multicast root for this subscription.

In case D maintains event state for the gid of the JOIN request, an ACK and a subsequent NOTIFY carrying the latest event state are returned to C and eventually to A. On the contrary, if D maintains no state for that gid a NACK is returned first to C and then to A. On the reception of an up-call from sa-Scribe the SIP stack of A finds the associated SIP transaction state and communicates the final SIP response back to Bob. Any notification requests are communicated from D to Alice in the same fashion.

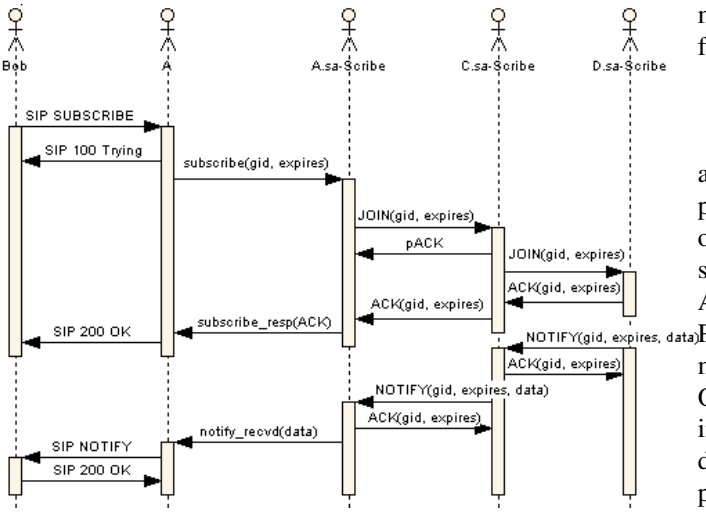

Fig. 5: Subscribing for event notifications through the P2PSIP overlay

Up to this point we have assumed that the gid results from (1). In a real implementation of the proposed architecture this calculation strategy is insufficient. This is because the event state notification and publication mechanisms of SIP are abstract (template) mechanisms that in practice are used by many event packages (i.e. presence, xcap-diff, etc.). Thus, two SIP SUBSCRIBE requests for the same Request URI but with different Event type headers are unique subscriptions that require different notification data. To overcome this challenge the SIP event header information may be used in the calculation of the gid as well. This results in a more fine grained distribution of notification groups among the overlay peers, as the multicast root now serves notifications coupled to a particular event package type.

F. Failure and recovery

Failure and recovery is effectively handled by Scribe. A parent in the multicast tree sends periodic heartbeat messages to its children. If a child fails to receive heartbeat messages from its parent, then it re-initiates the JOIN procedure towards gid. The JOIN message is routed through the overlay and the multicast tree is repaired [5].

The sa-Scribe extends this functionality and modifies the JOIN message to carry the latest known state maintained by a child. Thus, the new parent in the multicast tree (eventually the new multicast root) becomes aware of the latest known notification state of the notification group. Notice that the new multicast root uses the sequence numbers to identify the latest event state, if JOIN requests from different children with different event state arrive.

An upcoming issue is the replication of publication related information that is only kept at the multicast root (i.e. etag and expiration timers). Scribe proposes a solution to this problem by replicating the state associated with the multicast root to the roots K nearest peers. Thus, upon a multicast root failure the multicast root group identifiers will be handled by one or more of these replication points that will act on behalf of the failed multicast root as its replacement peers.

IV. Proof of concept

Our demonstrator utilizes the P2P simulator Oversim [16] and a modified version of the open source SIP proxy server project openSIPS (open SIP Server) [17]. The binding between openSIPS and Oversim is depicted in Fig. 6. At the openSIPS side we have implemented a new module invoking sa-Scribe API calls on a predefined Oversim peer. As soon as a SIP PUBLISH or SUBSCRIBE request reaches openSIPS, this module intercepts the call and invokes the sa-Scribe API. OpenSIPS and Oversim are connected through XML RPC interface, based on the open source code of an equivalent demonstrator [18] setup, that speeded up our prototype implementation.

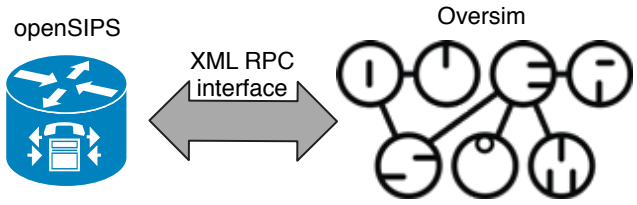

Fig. 6: The components of our proof of concept demonstrator

Once the sa-Scribe layer of the particular overlay peer receives the subscribe call, it follows the behavior shown in Subsection III-E. Similarly, when a publish call is received the behavior shown in Subsection III-D is implemented. Internally Oversim is configured to run the example network shown in Fig. 3 where the selected overlay algorithm is Pastry, and

each node runs an sa-Scribe instance. A current limitation of the demonstrator is that each opensips instance bounds to a particular peer in the Oversim overlay network. Thus, our demonstrator implements a $\frac{1}{N}$ gateways to overlay peers ratio, where N is the size of the overlay and N=8 in the current demonstrator setup.

V. Conclusions

This paper presents a P2PSIP event notification architecture that utilizes a P2P overlay algorithm, and an application level multicast scheme to distribute and decentralize the notification process. The proposed architecture adds scalability, fault tolerance, and recovery features which were lacking in a centralized SIP/SIMPLE infrastructure. Moreover, not all the overlay peers need to run a SIP stack and be aware of SIP operations; only the ones serving as gateways for out-of-overlay SIP user agents. Our proof of concept demonstrator combines a P2P simulation framework and a modified SIP server that invokes the proposed P2P functionality. This approach makes the rapid prototyping of the presented architecture feasible. Since this project is currently under implementation, we plan to perform performance evaluation of this system with regard to scalability and fault tolerance. More specifically, we intend to perform different simulation runs in order to evaluate our architecture under high notification overhead and focus on scalability expressed in terms of throughput and delivery ratio and delay.

References

[1] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler, "SIP: Session Initiation Protocol," Internet Engineering Task Force, RFC 3261, Jun. 2002. [Online]. Available: http://www.rfc-editor.org/rfc/rfc3261.txt

[2] A. B. Roach, "Session Initiation Protocol (SIP)-Specific Event Notification," Internet Engineering Task Force, RFC 3265, Jun. 2002. [Online]. Available: http://www.rfc-editor.org/rfc/rfc3265.txt

[3] A. Niemi, "Session Initiation Protocol (SIP) Extension for Event State Publication," Internet Engineering Task Force, RFC 3903, Oct. 2004. [Online]. Available: http://www.rfc-editor.org/rfc/rfc3903.txt

[4] V. K. Singh and H. Schulzrinne, "Simplestone - benchmarking presence server performance," Columbia University, Tech. Rep., 2004.

[5] M. Castro, P. Druschel, A. marie Kermarrec, and A. Rowstron, "Scribe: A large-scale and decentralized application-level multicast infrastructure," Sep. 01 2002. [Online]. Available: http://www.ovmj.org/GNUnet/papers/scribe.pdf

[6] S. Zhuang, B. Y. Zhao, A. D. Joseph, R. H. Katz, and J. Kubiatowicz, "Bayeux: an architecture for scalable and fault-tolerant wide-area data dissemination," in *NOSSDAV*. ACM, 2001, pp. 11–20. [Online]. Available: http://doi.acm.org/10.1145/378344.378347

[7] D. Tam, R. Azimi, and H.-A. Jacobsen, "Building content-based publish/subscribe systems with distributed hash tables," 2003, pp. 138–152.

[8] D. A. Bryan, B. Lowekamp, and C. Jennings, "SOSIMPLE: A serverless, standards-based, P2P SIP communication system," in *AAA-IDEA*. IEEE Computer Society, 2005, pp. 42–49. [Online]. Available: http://doi.ieeecomputersociety.org/10.1109/AAA-IDEA.2005.15

[9] E. Harjula, J. Ala-Kurikka, D. Howie, and M. Ylianttila, "Analysis of peer-to-peer SIP in a distributed mobile middleware system," in *GLOBECOM*. IEEE, 2006. [Online]. Available: http://dx.doi.org/10.1109/GLOCOM.2006.190

[10] I. Stoica, R. Morris, D. R. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," in *SIGCOMM*, 2001, pp. 149–160. [Online]. Available: http://doi.acm.org/10.1145/383059.383071

[11] J. Rosenberg, "A Presence Event Package for the Session Initiation Protocol (SIP)," Internet Engineering Task Force, RFC 3856, Aug. 2004. [Online]. Available: http://www.rfc-editor.org/rfc/rfc3856.txt

[12] P2P-SIP Working Group. [Online]. Available: http://www.p2psip.org

[13] J. Wang, "P2PSIP Event Notification Extension," Internet Engineering Task Force, Internet-Draft draft-wang-p2psip-event-notification-extension-00, Mar. 2009, work in progress. [Online]. Available: http://tools.ietf.org/html/draft-wang-p2psip-event-notification-extension-00

[14] C. Jennings, B. Lowekamp, E. Rescorla, J. Rosenberg, S. Baset, and H. Schulzrinne, "REsource LOcation And Discovery (RELOAD)," Internet Engineering Task Force, Internet-Draft draft-bryan-p2psip-reload-03, 2008, work in progress. [Online]. Available: http://www.ietf.org/internet-drafts/draft-bryan-p2psip-reload-03.txt

[15] M. Castro, M. B. Jones, A. marie Kermarrec, A. Rowstron, M. Theimer, and H. Wang, "An evaluation of scalable application-level multicast built using peer-to-peer overlays," Dec. 27 2003. [Online]. Available: http://www.research.microsoft.com/ antr/Pastry/../PAST/infocom-compare.ps

[16] I. Baumgart, B. Heep, and S. Krause, in *Proceedings of 10th IEEE Global Internet Symposium (GI '07) in conjunction with IEEE INFOCOM 2007, Anchorage, AK, USA*, May, p. 79.

[17] openSIPS. [Online]. Available: http://www.opensips.org

[18] I. Baumgart, B. Heep, and S. Krause, in *Proceedings of 7th IEEE International Conference on Peer-to-Peer Computing (P2P2007), Galway, Ireland*, Sep., p. 243.