Mobility support through caching in content-based publish/subscribe networks

Vasilis Sourlas, Georgios S. Paschos, Paris Flegkas and Leandros Tassiulas
Department of Computer & Communication Engineering
University of Thessaly, Greece
vsourlas, gpasxos, pflegkas, leandros @uth.gr

Abstract

In a publish/subscribe (pub/sub) network, message delivery is guaranteed for all connected subscribers at publish time. However, in a dynamic mobile scenario where users join and leave the network, it is important that content published at the time they are disconnected is still delivered when they reconnect from a different point. In this paper, we enhance the caching mechanisms in pub/sub networks to enable client mobility. We build our mobility support with minor changes in the caching scheme while preserving the main principles of loose coupled and asynchronous communication of the pub/sub communication model. We also present a new proactive mechanism to reduce the overhead of duplicate responses. The evaluation of our proposed scheme is performed via simulations and testbed measurements.

1. Introduction

The publish/subscribe paradigm has become an important architectural style for designing distributed systems. There are several research efforts concerned with the development of a pub/sub system [1] - [5]. Most of them address scalability and ease of implementation by realizing the broker tree as an overlay network. In those systems, any message is guaranteed to reach all interested destinations. This holds for all clients that their subscriptions are known to the network at publish time. However, there are cases where clients join the network after the publication of an interesting message, or roam around the network during their lifetime. In traditional pub/sub schemes it is not possible for a new subscriber to retrieve previously published messages that match his/her subscription. The same holds for a mobile client who was on the go during the publication. Therefore, caching for retrieval of past information is an important asset for mobile networks utilizing the pub/sub paradigm.

Caching as a mechanism for storing data in pub/sub systems has received little attention in the literature. In [6] a caching mechanism where brokers opportunistically cache information to make it available to future subscribers is introduced. A different aspect of caching is studied; the focus is on preserving the information over time instead of making information available in nearer space as the traditional caching schemes. Authors in [7] propose a caching mechanism, for wireless ad-hoc networks based on buffers, that offers a way to integrate data repositories distributed in the network. Their approach concentrates on the class of applications that commence normal operation after having seen a sequence of events. Finally, in [8] and [9] authors propose a historic data retrieval pub/sub system where databases (predefined caching points) are connected to various brokers, each associated with a filter to store particular information. In this paper, we will use the caching mechanism described in [6].

The majority of the overlay pub/sub systems are designed not to tolerate any form of topological reconfiguration, therefore they cannot be exploited in those application scenarios where decoupling would be most beneficial. The first pub/sub system that supported mobile clients was JEDI, where a client used two functions (*move-out* and *move-in*) to explicitly detach from the network and reconnect to it, possibly through a different broker. In [10] authors implement a mobility support service that is independent of the underlying pub/sub overlay and transparently manages active subscriptions and incoming messages when a client detaches from one broker until it reattaches at another. They use mobile service proxies which are independent, stationary components that run at the edges (where clients exist) of the pub/sub network. In other words they use a second overlay network to take care the mobility of the clients. That second overlay is responsible to gather the published events, that match the interests of the mobile client, and deliver them when the client reconnects to the network. The proxies of that second overlay should be aware of the topology of the mobile service network, since they should directly contact each other when a client moves among them. Finally in [11] authors present COMAN (COntent-based routing for Mobile Ad-hoc Networks), a protocol to organize the nodes of a MANET in a tree-shaped network able to self repair to tolerate the frequent topological reconfigurations. COMAN was designed to minimize the number of brokers whose routing information are affected by topological changes, but it is not support the retrieval of lost messages after the reconfiguration of the network.

Here we are interested in supporting the mobility of clients, where a client is disconnecting from the network and reconnects from a different point later in time. Particularly, we will use the caching mechanism of [6] and propose a modification which enables mobility for clients without further burdening the pub/sub mechanism. This way, we achieve the desirable result while avoiding adding the complexity introduced in [10]. Finally, we examine the resulting trade-offs between caching efficiency, system overhead and message delivery guarantees to the mobile clients.

The rest of the paper is organized as follows. In section 2, a brief introduction to the pub/sub architecture is given, followed by the description the caching scheme to be used. In section 3 we present our approach to support mobility of clients while section 4 describes our proposed duplicate response dropping mechanisms. Moreover, sections 5 and 6 report on the performance evaluation of the proposed system through simulation and testbed measurements respectively. Finally, section 7 concludes our experience and discusses future work.

2 The pub/sub system with caches

2.1 The pub/sub architecture

We consider a pub/sub system that uses the subscription forwarding routing strategy [2]. In this setting, the routing paths for the published messages are set by the subscriptions, which are propagated throughout the network so as to form a tree that connects the subscribers to all the brokers in the network.

Particularly, when a client issues a subscription, a `Subscribe()` message containing the corresponding subscription filter is sent to the broker the client is attached to. There, the filter is inserted in a Subscription Table (ST), together with the identifier of the subscriber. Then, the subscription is propagated by the broker, which now behaves as a subscriber with respect to the rest of the network, to all of its neighboring brokers. In turn, the neighboring brokers record the subscription and re-propagate it. This scheme is usually optimized by avoiding subscription forwarding of the same event pattern in the same direction exploiting "coverage" relations among filters.

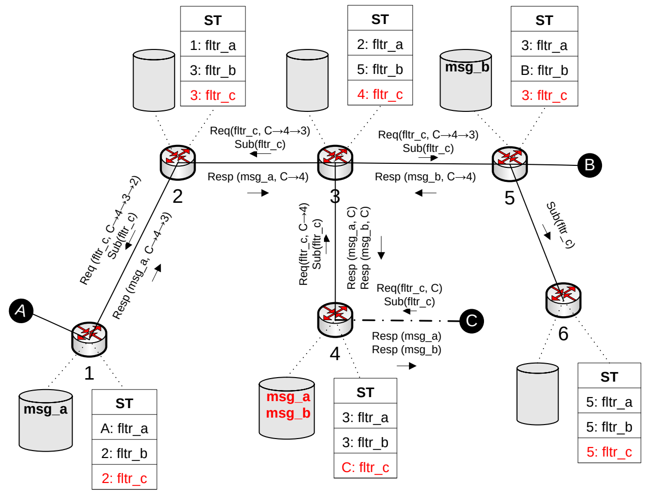

Figure 1. Caching and retrieving of old information (in red are the new entries of STs and created by the subscription of C).

2.2 Enabling caching in pub/sub systems

In this section, we give through the example of figure 1 a short description of the caching scheme firstly introduced in [6]. In our system, each broker is selected as a candidate caching point for a message as long as it has in its subscription table at least one client subscribed in this message, and depending on the caching policy (the broker) caches or not each published message matching the subscriptions of its clients. In this paper, we call that cached message as "old" message/information. In order to retrieve the old information, we added to the system two additional types of messages, `Request()` and `Response()`. A new client C interested in old content sends a `Request()` message with the interested filter $fltr\_c$. We used source routing for the forwarding of the `Request()` (the path is being built hop by hop and is included in the `Request()` header). Broker 4 upon receiving the `Request()` message checks in its Subscription Table (ST) for subscriptions matching the requested filter (we suppose that $fltr\_c$ matches both filters $fltr\_a$ and $fltr\_b$). The matching subscription can be either from another broker (broker 3 in this example) or from a client. The broker forwards the `Request()` message to every existing subscribed broker. Each broker–recipient of a request message–with at least one matching client subscription (brokers 1 and 5 in figure 1), searches in its cache for messages matching the initial filter (messages msg_a and msg_b accordingly) and for each match a `Response()` message is initiated.

A `Response()` message carries an old message as well as the sequence of nodes carried by the initiating `Request()` message (source routing). When a broker receives a `Response()` message, pops off its identifier from that sequence and forwards it to the first broker of the remaining sequence. In the end, client C will receive the message. With the above procedure, client C will receive every old message matching its filter and is still cached in a broker accessible by the request message.

In [6] a message is removed from a broker's cache when all the interested client subscribers have been unsubscribed, even if the cache is not full. This happens since future requests cannot reach that broker due to the lack of entries in the subscription tables of the rest of the brokers, pointing to that corresponding broker. Moreover, we used the first-in first-out (FIFO) with regenerations policy (similar to Least Recently Used) as a way to select the message to be dropped each time a cache is overflown. A Bloom-filter-based mechanism [12] could be used to solve scalability issues that might arise by the usage of source routing and the accumulation of broker ids at the header of the `Request()` and `Response()` messages, but such an analysis is out of the scope of this paper.

3. Mobility support

In this section, we describe a technique of using the already proposed caching scheme to provide support to mobile clients. Our approach is relative to [10], with the important difference that no extra functionality is required. Particularly, using a portion of each broker's cache, we allow brokers to manage subscriptions and publications on behalf of the mobile clients, both while they are disconnected and during the switch-over phase. This way no functions are needed other than the traditional pub/sub mechanism and the caching scheme explained above.

When the client is connected, publishes and receives messages directly to and from the pub/sub network. Before detaching, the client sends to the broker (broker 1 in figure 2), that he is attached to, a `Request()` message requesting to detach. That request message is similar to the message described in the above section but instead of the requesting filter it contains the "id" of the corresponding client. The broker has already in its Subscription Table "ST" the id of the client and its subscription filters so now whenever a message, matching those filters, arrives at the broker he directly caches it (apart from delivering it to the rest of the connected clients, if any, with a matching subscription). Until now the procedure is exactly the same with the procedure of the caching mechanism described above. The difference appears in the treatment of those cached messages.

In order to make the mobility support robust, we equip the caches with a preemption priority mechanism for those messages that are cached for a mobile client. Using such a mechanism, these messages are cached in a FIFO manner disregarding the rest of the messages which contend only

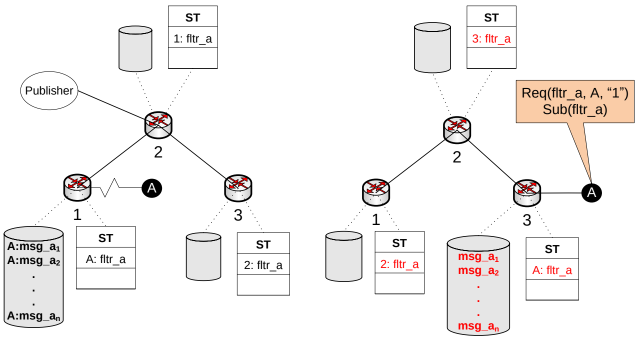

Figure 2. Mobility support mechanism. When client A is in movement messages msg_a_1 - msg_a_n appeared in the network.

for the remaining cache slots. Thus, a message cached for a mobile client is only dropped from the cache when the cache is full with such messages which arrived later than the given one.

When the mobile client reconnects to the network, from a different broker (broker 3 in the example), issues a `Request()` message with the subscription filter (or filters or part of them) that had subscribed to the pub/sub network before the movement and the "id" of the broker that was connected (broker 1 in the example). That request message will reach according to [6] broker 1. Broker 1, upon receiving that request, will i) respond (using `Response()` messages) with the cached messages (messages that arrived when the client was in movement, msg_a_1 - msg_a_n in the example), ii) unsubscribe the mobile client from its Subscription Table and iii) erase those messages that no other client has mataching subscriptions to them. This means that those messages are treated according to the scheme in [6] (like in figure 2).

4 Handling multiple responses

While the proposed mobility support mechanism does not produce multiple duplicate responses, since only one broker responds to the mobile client's request, the caching and retrieving scheme has as side effect the possible production of multiple identical responses on a single request. To deal with this effect, we provide our system with two (reactive and proactive) duplicate preventing mechanisms.

In the reactive mechanism, every broker with at least one client subscriber, upon the arrival of each response message, checks whether the message already appears in its cache and if this is true, drops the response message. Otherwise, it forwards the message according to the technique described in section 2.2. The reason for searching the cache of every broker upon the arrival of each response, is because responses follow the reverse of the route that the requests follow. This means that the request for initiating the response has also been processed by the broker under question which may have responded to that request with the same message(s). Note also, that the requests cannot be dropped in a similar manner, because we consider a content-based network, and finding a matching message in a proximity broker does not guarantee that there are no other (different) messages in the network matching the same subscription.

In the proactive counterpart, every broker with a cached matching message, apart from responding to the `Request()` message, before forwarding it to its neighboring brokers appends to the `Request()` header the "*id*" of the responded message. The brokers–recipients of that request message–will only respond with messages matching the requested filter and their ids are not in the `Request()` message, since those messages have already been sent to the client issued that `Request()`.

In the example of figure 3, we suppose that brokers 1, 4, and 7 have in their cache the same "black" message while brokers 8 and 11 have in their cache the same "grey" message and broker 10 has in its cache a "blue" message. If now a client connects to broker 5 and requests with some filter matching "any color", the request will reach all the above mentioned brokers. Brokers 4 and 7 will reply with a response message but the response message sent by broker 7 will be dropped upon reaching broker 5 given that broker 5 has already cached the "black" message, after the reception of the response initiated on broker 4 (reactive mechanism). Broker 1 won't respond since the "black" message cached is the same with the one cached in broker 4 and the "message id" is carried by the `Request()` message (proactive mechanism). Similarly according to the proactive dropping mechanism only the response of broker 8 will be delivered to the client. Finally, the response initiated in broker 10 will also reach the client. With this simple example is obvious how important are those two duplicate dropping mechanisms and especially the proactive one. The disadvantage of the proactive mechanism is the usage and the accumulation in the request's header of the "message ids" cached in the network which is not scalable, but which can be beneficiary in a well defined environment where the total number of the published messages are finitely many.

5. Performance evaluation

In this section, we evaluate the proposed mechanism using a discrete event simulator. $N = 7$ brokers are organized in a balanced binary tree and clients are dynamically generated on each broker according to a birth and death process with birth rate λ_c and death rate μ_c. Those clients with rate λ_{mob} go mobile and reconnect to the network after a randomly selected period of time (with mean value $\frac{1}{\mu_{mob}}$). New publications occur to the network with rate λ_{msg}. Each

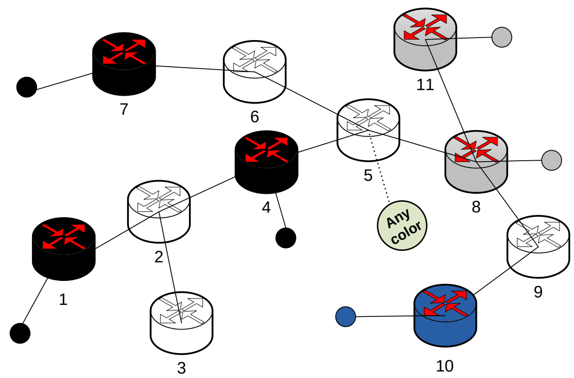

Figure 3. A dispatching network where the color of each broker represents the content of the cached messages and the their client subscription filters.

broker has a cache capable of storing k messages. We are looking at the following interesting metrics.

- The *absorption time* of a message m is the time passed from the publication of m until it gets disappeared from the network. This metric is indicative of the capability of the network to maintain messages in its memory.
- The *responses per request* is measured for each successfully responded request and corresponds to the number of total responses that the system would have generated if no duplicate dropping mechanism was used. This metric is representative of the replication and the overhead in the network.
- The *loss ratio* is measured for each mobile client and is the ratio between the publications that matches his subscription and were published when he was disconnected and the messages that are finally delivered to the client after his reconnection to the network. This metric is indicative of the capability of the network to support mobile clients and is indicative of the contention in the caches when they support mobile clients.

The above metrics are random variables and we estimate their mean by simulating thousands of observations. We set two experiments, one varying the time interval that the client is mobile ($\frac{1}{\mu_{mob}}$) and one varying the publication rate λ_{msg}. Those two sets are similar to varying the dynamics of the mobile clients (rate of going mobile and rate of reconnected to the network) and varying the cache size respectively. We run those two sets with four different values of the client dynamics $\lambda_c/\mu_c = \rho_c = 0.1, 0.5, 1, 100$. Note that if we would vary the number of brokers, in terms of cache contention, it would scale the problem with respect to the values of ρ_c.

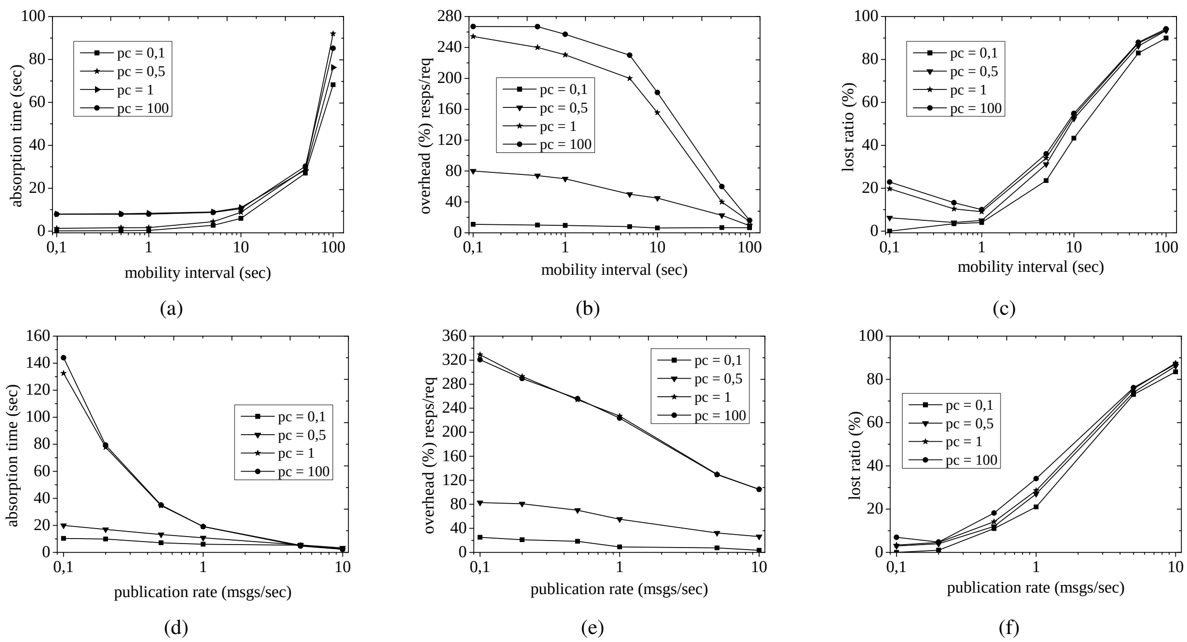

Figure 4. Performance of the mobility support mechanism for several values of the mobile time interval $\frac{1}{\mu_{mob}}$ (figures a-c) and the publication rate λ_{msg} (figures d-f).

Figure 4 shows three pairs of figures. In the first pair (subfigures "a" and "d"), we can identify the exponential nature of absorption time. Particularly, increasing the time interval that a client is mobile increases the mean absorption time of messages since now more messages are cached in the mobility emergency cache and stay there for more time. But increasing the number of publications (publication rate) has as effect the contention of messages in the mobility emergency cache (more messages now have to be cached). The large values of absorbtion time in subfigure a are caused by the prioritized messages for mobile users. Since the mobility interval is large, these messages retain high priority for longer periods. This example shows that prioritization is capable of delivering high quality of service in this case.

In the second pair (subfigures "b" and "e"), we present the gain in overhead (percentage) as (duplicate) responses per request that we have in our system by the usage of the two duplicate dropping mechanisms compared to the case that we have no dropping duplicate mechanisms. Using those two dropping mechanisms, we have only one response per successful request. For low mobility time intervals, we have a gain of more than 250% something that also occurs in low publication rates (350% gain). In high mobility time intervals and publication rates that gain is less since now due to contention in caches we have less duplicate messages stored at each cache meaning less duplicate responses per successful request. In low values of the client dynamics, that gain is also low since now clients remain "alive" for less time in the system and messages are not stored based on the technique described in section 2.2. This overhead gain increases with the number of nodes in the network.

Finally, in the last pair (subfigures "c" and "f") we present the loss ratio described above. Evidently higher mobility time intervals lead to messages staying longer at high priorities and higher publication rates lead to more messages in the caches. Both ways the increasing cache contention leads to increased loss ratio.

6. System design and experimentation

We implementated the proposed system on top of REDS [5]. Apart from the modifications made and presented in [6] for the purposes of this paper, we altered the logic in caching to support the mechanisms described in 3 and 4. We used 5 laptops equipped with a 1,6 GHz Intel Celeron M CPU, 512 MB of RAM. The 3 computers were connected via Ethernet switch to set the pub/sub overlay network as shown in figure 2. Another computer played the role of publisher while the final laptop played the role of a mobile subscriber client. In our testbed experiments, the mobile client issues one subscription while a series of publications is sent (all publications match that subscription) at a constant rate of λ_p publications per second ($\lambda_p = 1, 2$), which lasts for 50 seconds (50 and 100 messages accordingly). The mobile client disconnects from one broker and connects to another only once while the mobility interval is fixed, $\Delta t = 15$sec.

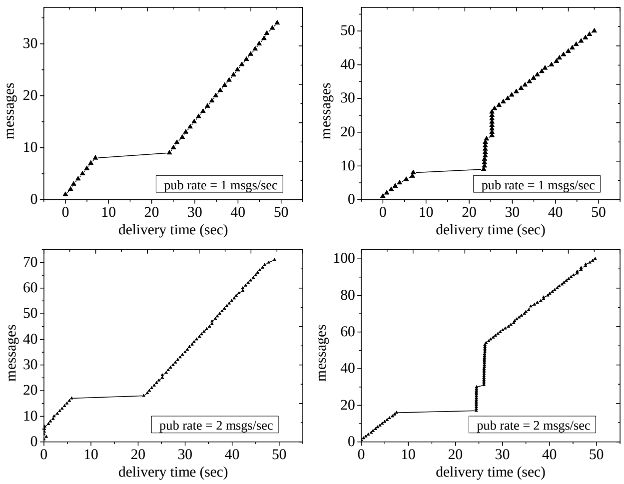

Figure 5. No mobility support (left-side), mobility support (right side).

Figure 5 shows two pairs of figures that we call *message/delivery time traces*. The y axis corresponds to the number of messages delivered to the mobile client, while the delivery time is when the message is received by him (set time to zero when the first message is delivered). To the left, the case where no mobility is supported is showcased, while to the right, the effect of our mobility support mechanism is demonstrated. Every point in the figures corresponds to a message received by the client either through the publish or the request process. The part of the figures where there is no message delivery represents the time interval that the client is disconnected from the overlay network, while the vertical part of delivered messages (right-side figures) after the reconnection of the client represents the responses delivered to the client to his request sent to the broker that was attached before the movement. It is obvious that all the published messages finally arrive to the client while there is no delay due to the processing associated with relocation.

7. Conclusion And Future Work

In summary, we have extended our proposed caching scheme to support mobility of clients and we have proposed a new duplicate dropping mechanism to reduce the system overhead. Evaluation via simulations and testbed measurements depict the performance of the system regarding information survivability, overhead and quality of the proposed mobility support scheme. This work can be extended in many ways, from deriving applications to extensions in mobile ad-hoc networks where both brokers and clients are allowed to move.

References

[1] Aguilera M. K., Strom R. E., Sturman D. C., Astley M. and Chandra T. D., "Matching events in a content-based subscription system," 18th ACM Symposium on Principles of Distributed Computing (PODC '99) Atlanta, GA, May 4-6, pp. 53–61, 1999.

[2] Carzaniga A., Rosenblum D. and Wolf A., "Design and evaluation of a wide-area event notification service," ACM Transaction On Computer Systems, vol. 19, pp. 332–383, 2001.

[3] Segall B. and Arnold D., "Elvin has left the building: A publish/subscribe notification service with quenching," Proceedings of AUUG97, Brisbane, Australia, Sept. 3-5, pp. 243–255, 1997.

[4] Cugola G., Nitto E.D. and Fugetta A., "The Jedi event-based infrastructure and its application to the development of the opss wfms," IEEE Trans. Softw. Eng. 27, 9 (Sept.), pp. 827–850, 2001.

[5] Cugola G. and Picco G., "REDS, A Reconfigurable Dispatching System," 6th International workshop on Software Engineering and Middleware, pp. 9–16, Oregon, 2006.

[6] Sourlas V., Paschos G. S., Flegkas P. and Tassiulas L., "Caching in content-based publish/subscribe systems," to appear in IEEE Globecom 2009 Next-Generation Networking and Internet Symposium.

[7] Cilia M., Fiege L., Haul C., Zeidler A., and Buchmann A. P., "Looking into the past: enhancing mobile publish/subscribe middleware," Proceedings of the 2nd international Workshop on Distributed Event-Based Systems (DEBS 2003), pp. 1–8, San Diego, California, 2003.

[8] Li G., Cheung A., Hou S., Hu S., Muthusamy V., Sherafat R., Wun A., Jacobsen H., and Manovski S., "Historic data access in publish/subscribe," Proceedings of the 2007 inaugural international Conference on Distributed Event-Based Systems (DEBS 2007), pp. 80–84, Toronto, Canada, 2007.

[9] Singh J., Eyers D. M., and Bacon J., "Controlling historical information dissemination in publish/subscribe," Proceedings of the 2008 Workshop on Middleware Security (MidSec 2008), pp. 34–39, Leuven, Belgium, 2008.

[10] M. Caporuscio, A. Carzaniga, A. L. Wolf, "Design and evaluation of a support service for mobile, wireless publish/subscribe applications," in IEEE Transactions on Software Engineering, Vol. 29, pp. 1059- 1071, 2003.

[11] L. Mottola, G. Cugola, G.P. Picco, "A Self-Repairing Tree Topology Enabling Content-Based Routing in Mobile Ad Hoc Networks,"in IEEE Transactions on Mobile Computing, Vol. 7, pp. 946-960, 2008.

[12] P. Jokela, A. Zahemszky, C. Esteve, S. Arianfar, and P. Nikander, "LIPSIN: Line Speed Publish/Subscribe Inter-Networking" in SIGCOMM'09, Barcelona, August 17-21, 2009.

[13] J. Wang, "A survey of web caching schemes for the Internet," ACM SIGCOMM Computer Communication Review, 29 (5), pp. 36-46, 1999.

[14] G. Pallis, and A. Vakali, "Insight and perspectives for content delivery networks," Communications of the ACM, vol. 49, no. 1, pp. 101-106, 2006.