Agent-Based Implementation of a P2P Publish/Subscribe System

Mihai Paraschiv, Alexandru V. Ștefănescu, Adela-Diana Almasi
University POLITEHNICA of Bucharest
Faculty of Automatic Control and Computers, Computer Science Department
Emails: {mihai.paraschiv, alexandru.stefanescu1708, adela.almasi}@cti.pub.ro

Abstract

The publish/subscribe paradigm offers many advantages for communication in unstructured decentralized peer-to-peer networks. One of the models that fit into this paradigm is topic-based event dissemination. This paper presents an agent-based system for publish/subscribe inspired by TERA and implemented on top of JADE. We describe the system's design and implementation and show several performance results obtained in a real-life setting.

Keywords: topic-based publish-subscribe, p2p networks, agents, JADE

1. Introduction

Publish/subscribe is a messaging paradigm of growing popularity for information dissemination in distributed systems. Due to the decoupled interaction model between participants, this approach accomplishes many of the interoperability needs of today's large-scale, dynamic, peer-to-peer applications. Communication participants can act both as producers (*publishers*) and consumers (*subscribers*) of information. Publishers inject information into the system in the form of *events*, while subscribers declare their interest in receiving some of the published events by issuing *subscriptions*. Subscriptions express conditions on the content of events (*content-based* model) or just on the category they should belong to (*topic-based* model).

Once an event is published, for each subscription whose conditions are satisfied by the event, the corresponding subscriber must be notified. The basic building block of systems implementing the publish/subscribe paradigm is a distributed *event diffusion* mechanism able to bring any published event from the publisher to the set of matched subscribers, while completely decoupling their interaction [1].

In unmanaged, inter-administrative systems (e.g., peer-to-peer), the event diffusion mechanism is usually implemented on top of an overlay network connecting all user nodes (either publishers or subscribers). Overlay networks are specifically designed to support information diffusion characterized by a high-level of reliability in large scale and unreliable environments.

One of the critical aspects in designing publish/subscribe systems is the implementation of the event diffusion mechanism. This can be accomplished trivially by flooding the overlay for each generated event. The downside is that nodes receive a large amount of events that do not match their subscription requirements. Ideally, event diffusion should be confined to the set of matching subscribers without affecting the whole network. This means that all subscribers with the same interests should be grouped in the same cluster. In this way, once the event reaches one member of the cluster, its dissemination can be limited to this group. Moreover, the number of messages needed to bring the event from the publisher to a node belonging to the target cluster should be as small as possible. In addition to this, event routing should be done without compromising reliability.

The goal of this paper is to present an implementation of a topic-based publish/subscribe architecture showing the above characteristics and implemented as an agent-based system.

The chosen approach is similar to TERA (Topic-based Event Routing for p2p Architectures) [13], which is one of the best performing pub/sub systems for large-scale, unstructured, completely decentralized peer-to-peer networks. The following section presents several approaches for developing architectures for peer-to-peer publish/subscribe systems. The usage of agents for implementing peer-to-peer networks is explored in Section 3, with exemplifications on the JADE framework [15]. The next two sections describe the implementation of the system using the agent-based approach, the experimental setup and discuss the results.

2. Overview of Pub/Sub Systems for P2P Networks

While the publish/subscribe pattern for managed systems has been widely studied and various solutions exist in the literature [2], [3], [4], publish/subscribe for unmanaged systems is today an active field of research [5], [6], [7].

SCRIBE [5] and Bayeux [9] are two pub/sub systems built on top of two Distributed Hash Table (DHT) overlays (namely Pastry [10] and Tapestry [11]), which enable them to achieve scalability, efficiency and self-organization. Systems like SCRIBE use the decoupled key/node mapping provided by the DHT to efficiently designate a rendezvous node for each topic which collects each event published for that topic and diffuses it toward subscribed nodes. However, the single node responsible for the management of each topic can quickly become a hot spot for very popular topics. Additionally, the usage of standard DHT routing protocols for event propagation involves many nodes that are not interested in the event.

An interesting variant of this technique was proposed in [12]: members of the system subscribed to the same topic form a separate overlay where events belonging to the corresponding topic are simply flooded. This mechanism for traffic confinement is similar to TERA's one. However, in [12] a single access point exists for each topic overlay.

The system proposed in [6] maintains, through the widespread use of probabilistic algorithms, a hierarchy of groups that directly maps a topic hierarchy. Without a general overlay network, one that would not be related to a specific topic, every publisher has to be part of the group corresponding to the topic it wants to publish. This implies that it would also have to receive events on topics that it might not have subscribed to. TERA solves the problem by not requiring publishers to join any topic; it is the general overlay that is responsible for diffusing events.

Content-based systems, such as Sub-2-Sub [7], assume that subscribers sharing the same interests are clustered with a self-organizing algorithm that continuously analyzes overlapping intervals of interest. In this system each peer is associated with one subscription. Sub-2-Sub relies on an epidemic algorithm to cluster similar subscriptions. Peers periodically exchange subscription information to get clustered to similar peers. Clustering involves the creation of a logical ring connecting all peers having the same subscription into a group. To publish an event, the system delivers it to any of the matching subscribers and from there the event traverses the logical ring connecting the subscribers. Information about subscriptions is periodically exchanged between nodes using a node sampling service employing Cyclon [8]. The main drawback of Sub-2-Sub is that the clustering mechanism leads to a high overhead imposed on nodes. This happens because the number of rings a node belongs to is not directly proportional to the number of subscriptions it manages, but it depends on the number of interest intersections that could be much larger than the number of subscriptions.

Most of the actual available systems are research prototypes, which concentrate on scalability and reliability rather than on durability in P2P environments. Durability refers to the property of the system to correctly send the events to *all* subscribers even if nodes or links in the underlying communication layer *fail*. Other problems that must be tackled are the load balancing of subscriber group membership data among the peers in the network and the separation of the communication layer from the subscription management layer.

The architecture for the peer-to-peer system presented in this paper is inspired by TERA [13], [14], which implements the publish/subscribe paradigm over unstructured, completely

decentralized networks. It uses probabilistic mechanisms and overlay management protocols in order to obtain event diffusion and high scalability.

The system embeds mechanisms that implement traffic confinement while supporting event diffusion reliability. In particular clustering is achieved by using dedicated overlay networks for each topic, where events can be diffused with high reliability. Routing of each event from the source node to the target overlay is realized through a probabilistic mechanism that shows a ratio of successes close to 1 involving a small and limited number of non-interested nodes. Moreover, the system is also shown to have a cost of diffusion per-event that scales with respect to the number of nodes constituting the system, the number of subscriptions/topics issued, and the event publication rate. Finally, TERA is shown to fairly distribute the system load according to the number of subscription currently issued by each participant.

TERA uses Cyclon [8] as overlay management protocol, which provides every node with a view representing a uniform random sample of the system through a simple epidemic algorithm. Each peer knows a small (around 20-100), continuously changing set of other peers, called its neighbors, and occasionally contacts a random one of them to exchange some of their neighbors (see Figure 1). This exchange (called a *shuffle*) is synchronous – a node chooses a number of neighbors, sends out to them information about its view of the network and waits to receive their answer. If they fail to respond in a timely manner, they are removed from the list of nodes. The view exchange technique lets the protocol build and maintain overlay topologies that closely resemble random graphs. Consequently, built overlays exhibit high connectivity and low diameter, thus being resilient to massive node failures, and are adequate topologies for implementing efficient broadcast primitives [14].

The main components of TERA are Event Management, Subscription Management, Access Point Lookup and Partition Merging and Broadcast.

3. Agent-Based Modeling with JADE

In recent years, agent systems have been a rapidly growing field of research in software. Agent-based technologies are nowadays considered the most promising means of deployment for enterprise level applications. An agent is a piece of software that acts on behalf of a user or other program in relationship with other agents [19]. There are some characteristics of agents which distinguish them from other pieces of software: reaction to the environment, autonomy, goal-orientation and persistence. The question of how to obtain interoperability across corporations and geographical distance in the context of heterogeneous systems is one for which agents offer a comprehensive solution [20].

JADE (Java Agent Development Framework) [15] is a software framework, implemented in the Java language, whose purpose is to aid the development of multi-agent systems [19]. It represents a middleware system fully compliant with FIPA [21] specifications, which allows users to deploy agents across different machines. Jade has the advantage of sharing the platform independence of the Java language. The agent platform is responsible for handling agent services such as messaging (transport, encoding and parsing), scheduling, agent lifecycle management and other common resources.

Messages exchanged between JADE agents are assigned to specific protocols, allowing for easy filtering at reception. JADE offers a reliable messaging system both point-to-point between agents and topic-based.

We have chosen to use an agent-based implementation of a publish/subscribe system because these high-level abstractions offer valuable help in managing the complexity of the problem. Agents are best suited for use in applications that need distributed computations and an increased communication between entities, which is why they fit well for the project presented in this paper.

4. Agent-Based Pub/Sub Architecture

The architecture of the system that we present in this paper is illustrated by Figure 1. The main components of the architectural design correspond to those of the pub/sub architecture.

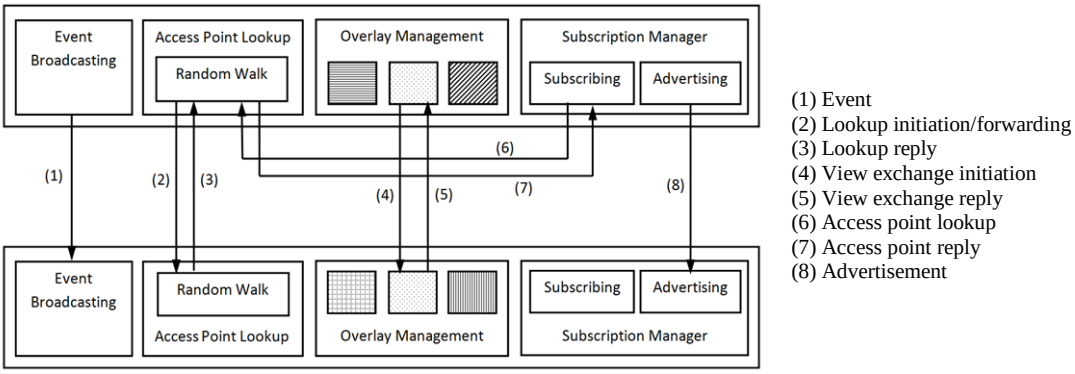


Figure 1 – Agent architecture

4.1. Design

In our system, each publish/subscribe agent has a set of components which are implemented as JADE behaviors. Each component contains multiple low level behaviors which are responsible for action planning, message sending or message receiving. In order to decouple components, some inter-behavior communication is performed through callbacks.

The following high level behaviors are found in the architecture: overlay management, subscription management, access point management, event diffusion and simulation. The latter is used to control the tests done on the system and is designed to interfere as little as possible with the other components.

4.2. Overlay management

At the base of our architecture there is an Overlay Manager, which is responsible for maintaining a set of agent networks. There are two types of overlays handled by the Overlay Manager. One of them consists in the underlying (base) network overlay, which contains all the agents. This is the level where processes like neighbor lookup and advertisement diffusion take place. The other overlay type is the one responsible for event propagation. Each topic has an associated overlay which is identified by the topic's name.

Both types of overlays are maintained using a view exchange protocol inspired by Cyclon. For all overlays of interest (base, subscribed topics), an agent holds a collection (fixed size) of known neighbors. At regular intervals, the agent picks a random receiver node from this set and initiates a view exchange. The first phase of this process consists in selecting a random subset of neighbors. A message is sent to the recipient, which is then removed from the agent's set of neighbors. The receiving agent responds with its own random selection of agents. Then, it randomly replaces nodes from its collection with those coming from the exchange initiator. The same action is performed by the initiating agent when receiving the reply message.

In order to account for disconnected nodes, a least recently used (LRU) cache is associated to each overlay. Nodes are added to this set when an initial view exchange message is sent to them. When a view is received from an agent present in this cache, the entry is removed and the sender of the view is added to the current agent's neighbor list. Additionally, nodes in the cache are removed from all incoming views.

Because we use JADE, the implementation of the protocol has to be asynchronous. This means that neighbors are not shuffled between nodes, as in Cyclon, because messages from

other agents can come before the reply for the current round is received. As a result, references to overlay peers could disappear. This may prove to be of concern when the neighbor collection size is very small or the network suddenly becomes partitioned.

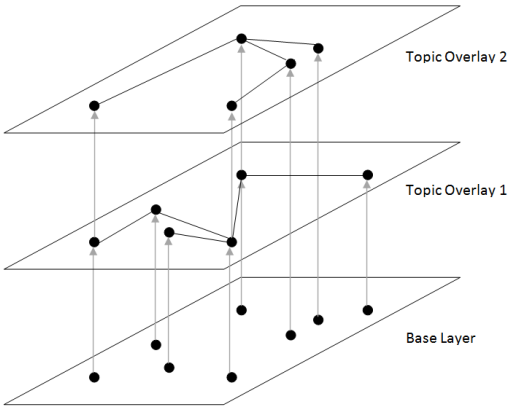


Figure 2 – Overlays: base and topics

Figure 2 shows how the base and overlay topics are represented. The base layer contains all the agents involved in the communication process. For each topic, an overlay containing all its subscribers is created (Topic Overlay 1 and 2 in the figure correspond to two topics). In every topic, agents have a set of neighbors, which are shown as connections in the figure. Agents can be included in multiple topic overlays depending on their subscriptions and in each overlay they may be both producers and subscribers.

4.3. Subscription management

The subscription management component is responsible with subscribing and unsubscribing the agent to/from a topic and facilitates the integration of an agent in a topic overlay.

The subscription operation consists of the following phases:

1. A topic overlay is created and registered with the Overlay Manager.
2. The Subscription Manager uses the Access Point Manager to look for nodes that are subscribed to the desired topic, which is accomplished using two methods. Firstly, the agent looks in its own access point cache. Secondly, the agent starts several random walks.
3. After the lookup returns, the identified peers are added to the topic overlay.
4. A view exchange is forced. This way, the peers know about the current agent before a new exchange round takes place.

Unsubscribing an agent from a topic is done by simply removing the topic registration from the Overlay Manager.

The Subscription Manager maintains a cache table of subscriptions, which are identified through advertisements made on the base overlay. The table contains one node per topic and has the purpose to speed up the discovery of the agents which can act as access points for a given topic.

4.4. Access point management

This component is responsible with the identification of nodes that can serve as access points to a topic overlay. The component is built on top of a random walk mechanism and provides two functions.

One of the functions is the node lookup initiation. Multiple random walks are started by a JADE parallel behavior, which waits for one message / all messages or a specified amount of time. After one of these conditions is fulfilled, the behavior makes a callback.

The other function of the Access Point Manager is to process incoming random walk requests. If a suitable peer can be identified in the agent's cache, a reply message is send to the initiator. Otherwise, the message is propagated to one of its own neighbors.

The rationale behind this search mechanism is that, given the uniform randomness of access point cache contents and of peer's neighbor set, it is possible to fix the lifetime of the walks and the cache size such that, given a topic, with a certain probability either (i) an access point for it will be found, or (ii) it will safely be considered as inactive [14].

If |T| is the total number of topics in the system, |APC| is the size of the access point cache table and *K* is the lifetime of the random walk, then the required cache size for which an access point for any topic is found with the probability *P* by visiting the *K* nodes of a random walk is [14]:

$$|APC| = |T|\left(1 - \sqrt[K]{1-P}\right).$$

4.5. Event diffusion

The first step in publishing an event is to find an access point to the topic overlay, which is accomplished by using multiple random walks through the Access Point Manager. After a subscribed agent is identified, the event message is sent to it. Next, the recipient processes the event internally and then broadcasts it. This last step is a forwarding operation, where the receivers are the topic overlay neighbors.

4.6. Differences from TERA

Our implementation diverges from TERA in several ways. One of these is the communication type. In TERA, messages are sent synchronously, which makes protocol properties easier to prove. JADE only supports asynchronous communication, thus resulting in a more complex design and a more difficult testing process.

The second difference is the use of a size estimation mechanism. Determining the size of an overlay is required in order to limit the amount of access points which are kept by each node. In our system, we do not use an estimation mechanism for the size of the network. This may prove to be a problem if there are many topics, but in scenarios like the ones we have tested, this does not seem to be a significant issue.

Another difference is that we have not employed overlay partition merging. In TERA, a topic can have more than one associated overlay. An agent that detects that its topic belongs to two overlays has to initiate a merging operation. A different approach is in our implementation, where we associate only one overlay to each topic. While this simplifies overlay management, it makes it difficult to estimate the size of a network. This happens because we have no easy way to determine which agent was the first in an overlay.

4.7. Implementation issues

Developing our system over JADE has been straightforward for most parts. However, we have encountered several issues which are summarized below.

One problem was identified when a large number (larger then 100) of agents were let to run in the same container. After a random interval of time, all agents would freeze. We determined that the problem was caused by the large number of timer tasks we were using. Each agent employs timers for mechanisms like topic advertisement sending or view exchange initiation. JADE uses one task per agent, but timers are handled by a singleton dispatcher and the mechanism through which this object is assigned is not directly available to the developer. In order to alleviate the freezing problems, we have replaced the dispatcher assigned by JADE with one coming from a pool designed by us. This is accomplished through simple introspection (the dispatcher field is private) at the setup phase.

Another stability issue was identified in early testing. Because the number of threads responsible with receiving the messages was very small compared to the number of threads sending the messages, the system would become unresponsive. The message buffer of the

recipient container would quickly fill and the system would freeze. This issue was solved by reconfiguring the JADE platform.

5. Tests

5.1. Test setup

For testing purposes, we have designed an additional agent – the facilitator (see Figure 3), which has two roles. The first one is to serve as the unique logging point, as the facilitator receives logging messages from all the other agents. The second role consists in initializing the agent network. The facilitator continuously waits for agents to register with it. After a predetermined number of agents have registered, the facilitator sends to them (and to agents registering afterwards) a random set of initial neighbors. After being added to the network and receiving the neighbor set, the initialization function of the facilitator is not used any longer. Later on, the facilitator will only be used as logging mechanism.

For our test scenarios, six machines have been used: one for the platform's main container and five for publish/subscribe agents. The machine on which the main container runs is an Intel Core2Duo at 1.73 GHz. The other five are Pentium 4 at 3 GHz. All machines are connected using a 100 Mbps LAN.

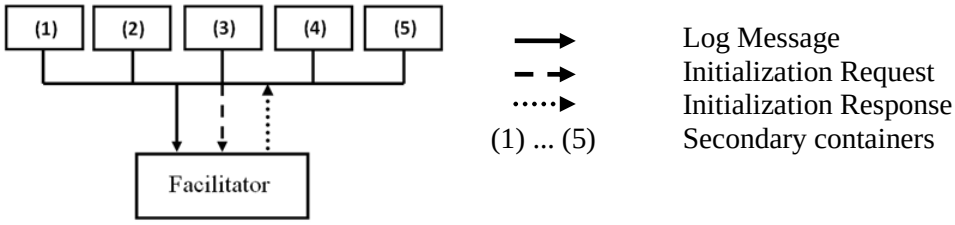


Figure 3 – Test Setup

The main container keeps the agents necessary for maintaining a JADE platform and a Facilitator agent. The secondary ones include only regular publish/subscribe agents and are launched as separate processes on each machine.

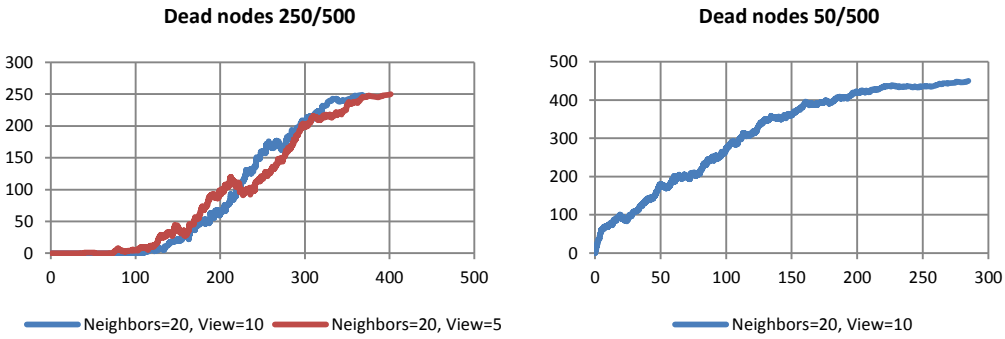
5.2. Experimental results

We have designed two test suites through which we have assessed the performance of two system components: overlay management and event diffusion.

5.2.1. Overlay management

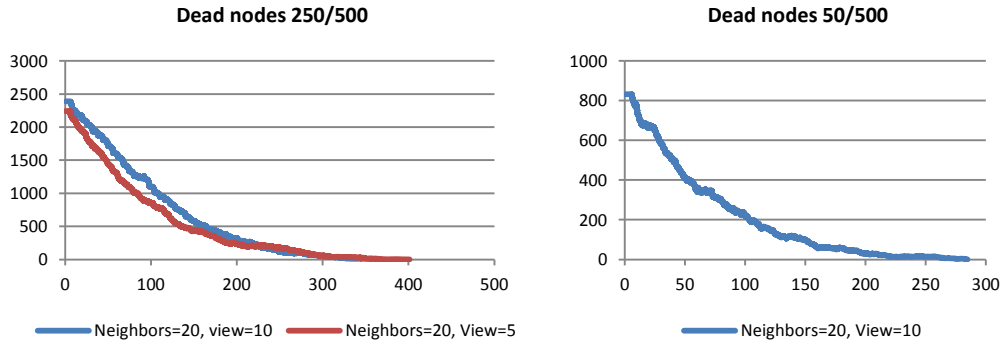
The first test suite evaluates the overlay management component. We tested three scenarios, in each case starting with a network with 500 agents (100 on each machine). We varied the number of removed nodes and the view size for the exchange phase. It is worth noting that nodes are started immediately one after another and that exchange rounds are scheduled every six seconds.

The graphs in Figures 4 and 5 show the number of agents that have cleared all references to nodes removed from the network. We can see that doubling the view size does not have a significant effect on the time it takes to flush all nodes.



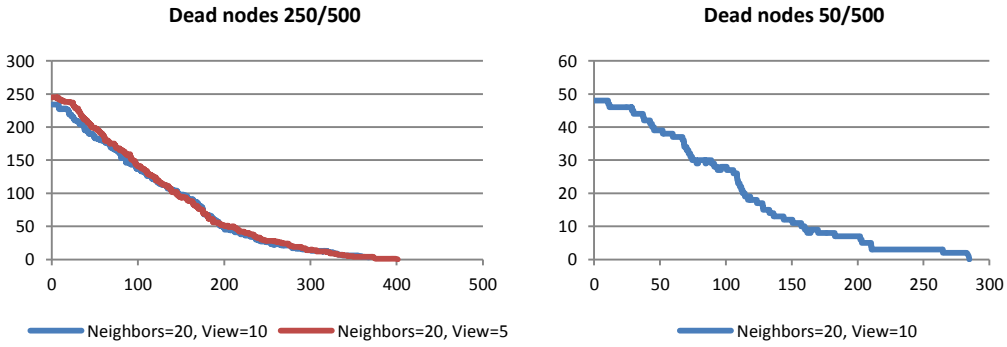
Figures 4 and 5 – Nodes which have no dead references remaining (in seconds)

Figures 6 and 7 show how the number of total remaining references changes. As we expect, the number of nodes decreases steadily over time. In several minutes, almost all references to dead nodes are removed. We observe that using a smaller view size can minimize this time.



Figures 6 and 7 – Remaining dead references (in seconds)

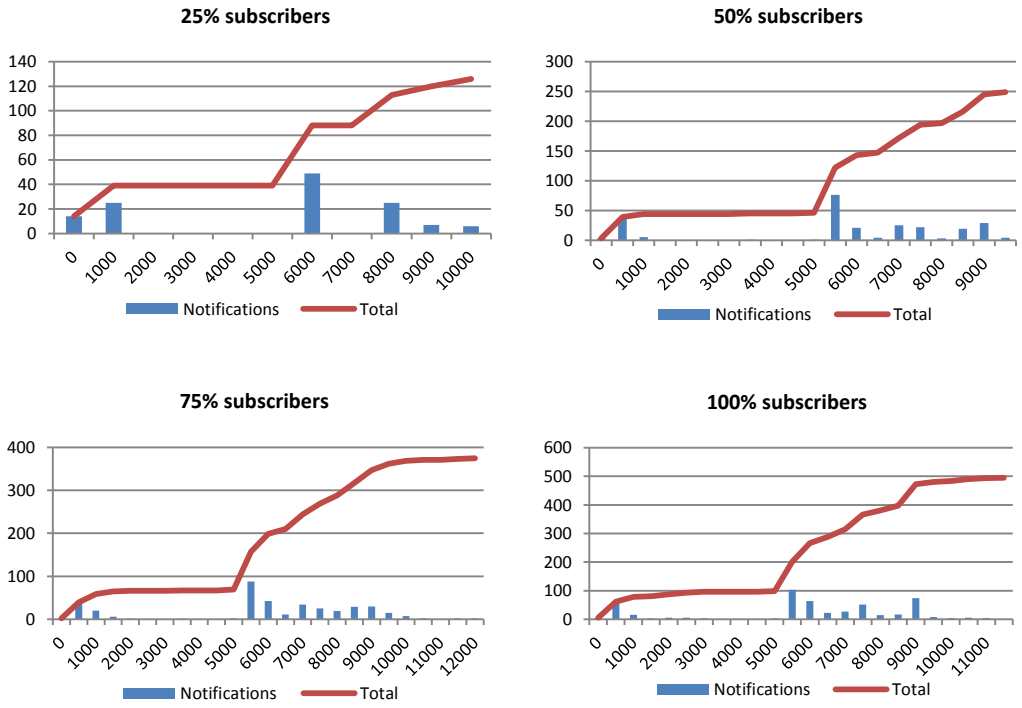
Figures 8 and 9 show the number of dead nodes remaining in the neighbor tables of live agents. Unlike references, nodes are removed from the network in an almost linear fashion.



Figures 8 and 9 – Remaining dead nodes (in seconds)

5.2.2. Event diffusion

The second test suite consists in having a single event published on a topic and varying the number of subscribers. We measure the time it takes to propagate the event in the entire overlay by log messages. A log message contains the recipient's timestamp and is sent to the facilitator immediately after the event is received.



Figures 10-14 – Event diffusion rate per time slice (in milliseconds) when the number of subscribers of the selected topic varies between 25% and 100%

All four plots show that at about 5 seconds into the test, the message starts to propagate very fast. In 9 to 12 seconds, almost all subscribers receive the message. Further testing is required in order to assess the real life behavior of our system.

6. Conclusions and Future Work

In this paper, we described the design and implementation of a publish/subscribe system over a peer-to-peer network, for which the main source of inspiration has been the TERA architecture. The characteristics and implementation of the system make it suitable for an enterprise environment, where the changing structure and flexibility requirements are capital to the applicability of message-based software. The system applies to topic based publishing using an overlay for each topic. Nodes that subscribe to a topic are added to their corresponding overlay and the system automatically manages traffic confinement, without unnecessarily flooding the network. Testing has shown that our system is capable of good event diffusion and can adapt to changes in network structure.

As future work, we consider the following tasks. First, we want the system to manage composite events. The system currently supports only simple events, and in order to extend its functionality, we would need to add event management capabilities to the existing overlays. Secondly, we consider implementing the partition merging functionality of TERA. This is needed in order to estimate the size of the network, and also to prevent problems such as the unwanted partitioning of the network. Finally, we feel it would be useful to further test the performance of the system in more varied circumstances than the ones already tested. The network on which we have tested comprised of only 6 computers, on which we have run the maximum number of agents that the platforms supported, which in our case was 500 agents. In order to confirm the system's scalability and fault-tolerance capabilities, it would be useful to observe how it performs in a massively distributed setup.

References

[1] P.T. Eugster, P.A. Felber, R. Guerraoui, and A.-M. Kermarrec, *The many faces of publish/subscribe*, ACM Computing Surveys 35, no. 2, 114–131, 2003.

[2] B. Oki, M. Pfluegel, A. Siegel, and D. Skeen, *The information bus - an architecture for extensive distributed systems*, Proceedings of the 14th ACM Symposium on Operating Systems Principles (SOSP), 1993, pp. 58–68

[3] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf, *Design and evaluation of a wide-area notification service*, ACM Transactions on Computer Systems 3, 2001, no. 19, 332–383.

[4] G. Banavar, T. Chandra, B. Mukherjee, J. Nagarajarao, R.E. Strom, and D.C. Sturman, *An Efficient Multicast Protocol for Content-based Publish-Subscribe Systems*, Proceedings of International Conference on Distributed Computing Systems (ICDCS '99), 1999.

[5] M. Castro, P. Druschel, A. Kermarrec, and A. Rowston, *Scribe: A large-scale and decentralized application-level multicast infrastructure*, IEEE Journal on Selected Areas in Communications 20, October 2002, no. 8

[6] S. Baehni, P. Th. Eugster, and R. Guerraoui, *Data-aware multicast.*, Proceedings of the International Conference on Dependable Systems and Networks (DSN), 2004, pp. 233–242.

[7] Spyros Voulgaris, Etienne Rivière, Anne-Marie Kermarrec, and Maarten van Steen, *Sub-2-sub: Self-organizing content-based publish and subscribe for dynamic and large scale collaborative networks*, Research Report RR5772, INRIA, Rennes, France, December 2005

[8] S. Voulgaris, D. Gavidia, and M. van Steen, *CYCLON: Inexpensive Membership Management for Unstructured P2P Overlays*, Journal of Network and Systems Management 13, 2005, no. 2.

[9] S. Q. Zhuang, B. Y. Zhao, A. D. Joseph, R. Katz, and J. Kubiatowicz, *Bayeux: An architecture for scalable and fault-tolerant wide-area data dissemination*, Proceedings of the 11th International Workshop on Network and Operating Systems Support for Digital Audio and Video, 25-26 June 2001, pp. 11–20.

[10] A. Rowstron and P. Druschel, *Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems*, Proceedings of IFIP/ACM International Conference on Distributed Systems Platforms (Middleware), 12-16 November 2001, pp. 329–350.

[11] B. Y. Zhao, L. Huang, J. Stribling, S. C. Rhea, A. D. Joseph, and J. Kubiatowicz, *Tapestry: A Resilient Global-scale Overlay for Service Deployment*, IEEE Journal on Selected Areas in Communications 22 (2003), no. 1, 41–53.

[12] Sylvia Ratnasamy, Mark Handley, Richard Karp, and Scott Shenker, *Application-level multicast using content-addressable networks*, Lecture Notes in Computer Science 2233 (2001), 14–34.

[13] R. Baldoni, R. Beraldi, L. Querzoni, V. Quema, and S. T. Piergiovanni. *A scalable P2P architecture for topic-based event dissemination*. MIDLAB Tech. Rep. 01-07, Dipartimento di Informatica e Sistemistica, Sapienza, University of Rome, Rome, Italy, Jan. 2007.

[14] R. Baldoni, R. Beraldi, V. Quema, L. Querzoni, and S. T. Piergiovanni, *TERA: Topic-based Event Routing for Peer-to-Peer Architectures*, 1st International Conference on Distributed Event-Based Systems (DEBS). ACM, 6 2007.

[15] Java Agent DEvelopment Framework (JADE). http://jade.tilab.com/. Available at June 27, 2010.

[16] PeerSim Peer-to-Peer Simulator. http://peersim.sourceforge.net/. Available at June 27, 2010.

[17] Mark Jelasity and Alberto Montresor, *Epidemic-style proactive aggregation in large overlay networks*, Proceedings of The 24th International Conference on Distributed Computing Systems (ICDCS), 2004, pp. 102–109.

[18] D. Kostoulas, D. Psaltoulis, I. Gupta, K. Birman, and A. Demers, *Decentralized schemes for size estimation in large and dynamic groups*, Proceedings of the 4th IEEE International Symposium Network Computing and Applications (NCA), 2005.

[19] Hyacinth S. Nwana, *Software Agents, an Introduction*, Knowledge Engineering Review, Vol. 11, No 3, pp.1-40, Sept 1996

[20] Fabio Bellifemine, Agostino Poggi, and Giovanni Rimassa, *Developing multi-agent systems with Jade*, Lecture Notes in Computer Science, vol 1986/2001 (2001), pp. 42-47

[21] Foundation for Intelligent Physical Agents (FIPA), http://www.fipa.org/. Available at June 27, 2010.