Scalable Information Dissemination for Pervasive Systems: Implementation and Evaluation

Graham Williamson, Graeme Stevenson, Steve Neely, Lorcan Coyle, Paddy Nixon
Systems Research Group
School of Computer Science & Informatics
University College Dublin
Ireland
graham.williamson@ucd.ie

ABSTRACT

Pervasive computing systems require large amounts of information to be available to devices in order to support context-aware applications. Information must be routed from the sensors that provide it to the applications that consume it in a timely fashion. However, the potential size and *ad hoc* nature of these environments makes the management of communications a non-trivial task. One proposed solution to this problem uses *gossiping*, a class of probabilistic routing protocol, to disseminate context information throughout the environment. Gossiping algorithms require far less in the way of guarantees about network structure, reliability, and latency than alternative approaches, but are unproven in real world scenarios. We describe the on-going development of a framework for evaluating the performance of these algorithms within the context of pervasive environments.

Categories and Subject Descriptors

C.2.4 [**Computer-Communication Networks**]: Distributed Systems

General Terms

Algorithms, Performance

1. INTRODUCTION

Pervasive computing systems require a large amount of information to be made available to devices in order to support adaptive, context-aware, applications. Potentially consisting of large numbers of sensors and applications, the complexity of any inter-device communication must be managed correctly to allow systems to scale without deterioration in performance.

Building pervasive systems in a decentralised manner allows them to grow piecemeal while retaining some extent of co-ordinated behaviour through lateral relationships between components. This is preferable to the traditional approach where we rely on services and commands supplied by global control points. However, as the number of devices in any given environment increases, we face the problem of efficiently transporting information from the sensors that provide it to the applications that consume it and use it to adapt. Real-time interactions can only be supported with timely responses to user and application commands. The presence of devices with differing capabilities and connectivity, and applications with varying data requirements makes this an interesting challenge.

We may observe that data from sensors in these systems, in addition to having a limited lifetime, are often frequently repeated: e.g., a user's location may be updated every 5 seconds. The implication of this, is that we do *not* need 100% reliability of message delivery. Our pervasive systems architecture, *Construct* [2], makes use of this relaxed reliability constraint to use a non-deterministic mechanism called "gossiping" — an epidemic style algorithm — to underpin its communications. Gossiping requires far less in the way of guarantees about network structure, reliability, and latency than alternative approaches, whilst providing the desirable properties of decentralisation, and robustness to change, that are required for pervasive computing systems of this nature.

However, simple gossiping algorithms do not necessarily provide reliability and scalability guarantees. An algorithm that has been incorrectly applied to a given pervasive system can result in situations where an unacceptable number of peers fail to receive a message, or message delivery is not timely enough for a given application. Although gossiping has potential as an approach to data distribution in pervasive systems, its applicability remains unproven, especially in real world scenarios.

In order that we can characterise different gossiping algorithms, we need to be able to quantify the improvements and drawbacks that they introduce into a network. To that end, we describe the on-going development of a framework for evaluating the performance of these algorithms within the context of pervasive environments. We describe how the framework can be used to simulate variance in a set of key variables, and measure the properties of the system that may be affected as a result.

2. CONSTRUCT

We motivate our research into gossiping protocols by introducing *Construct*, our middleware infrastructure for the collection and dissemination of context information in smart

MPAC '06 November 27-December 1, 2006 Melbourne, Australia

environments. Construct forms a real-world empirical testbed for the gossiping communication substrate.

Construct is a software platform for integrating noisy data sources in a clean, dynamic, flexible and semantically well-founded manner. Construct provides applications with a uniform view of information regardless of how that information is derived, and supports extensive inferencing and sensor fusion within the platform [2, 5].

Construct has a fully decentralised architecture; devices in a smart-space each run an instance. Each instance manages the local data provided by sensors (physical or logical) connected to that device — a "local star" topology in which dumb sensors are connected to more computationally capable hubs which then exchange information between themselves. Should a device fail, any wireless sensors may re-connect to another visible device running the Construct platform. All data are modelled using the Resource Description Framework (RDF), which provides a standardised way in which to model contextual information and properties. Construct stores and manipulates this data using the Jena Framework [11].

Applications connect to the Query Service component of Construct, and use its services to obtain required information. Components with domain-specific knowledge may request and aggregate data from multiple sources in order to contribute new, or refined information. All data stored within the system are associated with metadata, which describe data lifespan and security restrictions.

Collectively, devices running the Construct software maintain a global model of the data within a smart-space. Applications then work with the local view of the data present on the device which they are connected to. Maintaining the global data model requires context information to be communicated between instances of Construct: this is accomplished by the randomised state-swapping algorithm, gossiping, described in detail in the following section.

3. GOSSIPING

This section provides an introduction to gossiping and describes a number of notable approaches using gossiping for information dissemination in distributed systems. It should be noted, however, that gossip-based technologies are not limited to information dissemination and can, in fact, be used for other purposes. Examples of such applications include data aggregation [9], load balancing [9], failure detection [15], and resource location [14].

The particular focus on information dissemination is motivated by our adoption of gossiping as the underlying communication mechanism for the Construct platform. Here, the fundamental problem gossiping addresses in the architecture of Construct is that of providing a reliable, scalable, and fault-tolerant group communication system.

The properties of reliability, robustness, and scalability are important design goals for Construct. In gossip-based algorithms, these properties stem from the purely local interactions between gossiping nodes, resulting in emergent behaviour which realizes these qualities: each node in the Construct network acts autonomously and a network built using gossiping can cope easily with faults, reconfiguring itself if necessary.

3.1 Epidemic Algorithms and Gossiping

Recently, research has been conducted into using randomised algorithms for communication. One class of such algorithms is based on the spread of epidemics.

Much of the current research in this area stems from the systems developed to maintain consistency in the Clearinghouse database servers of the Xerox Corporate Internet [4] and includes, for example, *pbcast* [1], *lpbcast* [7], and *pmcast* [6].

The term *epidemic* reflects how the information is disseminated through the network. Peers pass information to a small number of others, randomly selected from their neighbourhood, in much the same way that an infection spreads from person to person. During each cycle of the epidemic an infected host — one that has already received a particular message — may come into contact with, and infect a given number of, other hosts. These target hosts are chosen randomly and may, or may not, already be infected. The newly infected peers then, in turn, relay the 'infection' to further peers.

Epidemics have long been studied outside the field of computing and much of the mathematics predicting the behaviour of such processes is applicable [8].

The expression *gossiping* derives from another analogy to the process by which information spreads through the network. In a similar fashion to an epidemic, rumours may be spread through a community: people gossip with their neighbours and friends, who in turn gossip with people in their proximity. In this way, the rumour is propagated through the population.

3.2 Notable Gossip-based Techniques

Eugster et al. [8] identifies a number of challenges that developers of epidemic protocols face:

Network Awareness: How can the physical characteristics of the underlying network be taken into account in order to reduce load and increase performance?

Membership Management: How is information regarding peers discovered, updated and tracked?

Buffer Management: How can the storage, ageing, and removal of messages be managed effectively?

Message Filtering: How can it be ensured that nodes receive messages that they are interested in while reducing the probability that they must handle other messages?

Directional Gossip [10] begins to address the problem of incorporating network awareness into gossiping algorithms. They aim to reduce the overhead and increase the reliability of gossiping algorithms in wide-area networks. Directional Gossip uses an estimation of the minimum link cut set (the maximum number of link disjoint paths) between nodes to identify important paths through the network. Messages are always flooded across links to nodes identified with a low link cut set and gossiped probabilistically with other peers. The evaluation of this algorithm showed increased reliability of message delivery, however, in more realistic network settings the overhead performance was more akin to flooding than gossiping. It should also be noted that static 'gossip servers' are assigned to each LAN segment which constitute a single point of failure and require configuration information from network administrators to provide links between gossip servers.

Bimodal Multicast [1] aims to build a reliable, scalable, multicast protocol with the key properties of high and stable throughput. The algorithm they develop, termed *pbcast*, augments an initial unreliable multicast (such as IP multicast) with a gossiping stage to provide what may be viewed as a decentralised retransmission service. In this way, the initial multicast provides a quick, rough coverage of the group and the gossiping stage fills in the gaps where messages were lost. The main focus is on the optimisations made to the algorithm which are designed to improve performance and smooth throughput, and on the evaluation of the protocol. A cursory mention is made of network awareness and membership management, commenting that it is possible to view a wide-area network as a collection of interconnected local-area networks. Gossiping would then happen inside local-area networks as normal, with only certain nodes interconnecting these.

Lightweight Probabilistic Broadcast [7] goes some way to address the concerns of scalable membership management and buffer management. The *lpbcast* protocol piggybacks membership information on gossip messages to provide a completely decentralised broadcast group architecture where each node maintains a fixed size partial view of nodes in the system. Interesting optimisations to the protocol are the age-based message buffer purging and frequency-based membership information removal heuristics.

Finally, the remaining challenge of the four identified earlier remains very much an open issue. The provision of a form of message filtering would move the algorithms from pure broadcast towards a scalable multicast. Probabilistic Multicast [6] defines the problem and moves towards a scalable multicast implementation which takes into account network awareness and the interests of each node in disseminating messages. Autonomous Gossiping [3] aims to solve a similar problem in the context of wireless mobile *ad hoc* networks. Selective dissemination is based on profiling of data items and host interests, with data items competing to survive, migrate, and replicate between hosts.

To illustrate the possible implementation of algorithms similar to those described, and to highlight some of the interesting details of gossiping protocols, we will proceed with a deeper examination of a simple gossiping protocol.

3.3 A Simple Gossiping Protocol

A skeleton gossip-based protocol is described with psuedocode in Figure 1. Each node has two main responsibilities: the first, to periodically wake up and initiate a gossip with a number of nodes; the second, to listen for incoming gossips and react to these. These responsibilities are illustrated in Figure 1a and Figure 1b respectively.

Each peer participating in the network maintains a local state which, at a minimum, consists of two tables: a list of peers that it considers to be its neighbours and a list of currently active messages.

Two important points may be implied from this description of local state. Firstly, we must have a way of discovering and managing membership information (i.e. what peers are our neighbours?); and secondly, we cannot infinitely buffer all messages and therefore messages eventually should be considered inactive and must be purged. Both of these areas — membership management and buffer management — are identified as challenges by Eugster et al. [8], and represent points where intelligence could be applied to the gossiping process.

The premise of gossiping is that, on each round of the algorithm, the nodes involved in a gossip reconcile any differences they have in their local states. Figure 1 represents this notion by having nodes gossip summaries of the recent messages that are present in their buffers (this could consist simply of sequence numbers of messages the node knows about). On the listening side, if the summary it receives differs from its own message history, the node can act to reconcile the differences in two ways (which are not mutually exclusive): either by requesting the messages it is missing from the sender of the gossip, or by forwarding the messages it has detected the sender is missing.

The remaining important points to note from the psuedocode description are that gossiping occurs in unsynchronised rounds — the duration of which may be tuned by adjusting the parameter T; the neighbours to gossip with are chosen randomly from the list of peers a node is aware of; the number of neighbours chosen on each round is termed the fanout; and the notion of ageing messages is included in order to measure how old an infection is and gauge when it may be appropriate to remove it from the buffer.

Adjusting T and fanout have a number of effects. For example, the load put on the network can be adjusted: increasing T reduces the frequency of gossips and hence reduces the bandwidth used (however the corollary is that latency will be increased). Also, the reliability of propagation of messages can be affected: increasing fanout increases the probability that nodes receive a message but also increases the network load as more messages are sent.

The advantage of gossip-based algorithms, such as that described heretofore, lies in the emergent behaviour: these simple local interactions lead to a scalable, resilient network which can react well to changes in the network structure.

4. GOSSIPING IN CONSTRUCT

The gossiping layer of Construct is a realisation of the simple gossiping protocol described in the previous section. The implementation aims to be compact and lightweight, allowing us to gain experience and make some early observations of how the algorithm functions. However, at the same time, we hope to make it flexible enough to allow the implementation of more complex gossiping algorithms. Important areas where flexibility is catered for in the core algorithm include: the choice of which node to gossip to on each round of the algorithm; the choice of which message to request when a received message summary differs from our own; and the choice of which message is removed from the buffer when it becomes full.

The gossiping subsystem itself consists of three components:

1. the core gossiping protocol
2. the message buffer responsible for storing the messages that we are currently gossiping about
3. the membership manager responsible for maintaining the list of contacts that we can gossip with

In the initial version, the core gossiping protocol is stateless. Gossiping is initiated periodically at each host by sending their message buffer summary to a single, randomly selected node from their contact list. When messages arrive,

```
repeat forever {
        sleep ( T );
        age_messages();
        digest ← message_history_summary();
        repeat fanout times {
                peer ← random_peer();
                send ( digest, peer );
        }
}
```

(a) Initiating gossip process

```
repeat forever {
        digest ← wait_for_incoming_gossip();
        my_digest ← message_history_summary();
        compare ( digest, my_digest ) {
                pull messages from sender
                        and/or
                push messages to sender
        }
}
```

(b) Listening gossip process

Figure 1: Skeleton Gossiping Psuedocode

they are dealt with based on the type of message and then discarded. The gossiping layer must handle three types of messages: data messages, message buffer summaries and requests for messages. Data messages are simply stored in the message buffer for future gossiping then passed up the protocol stack to the application. Message buffer summaries contain a list of the message IDs present in the remote nodes buffer. On comparison with our message buffer, a request for a message which we do not have is generated. These requests are fulfilled by sending a data message containing the appropriate information.

We do not address the discovery of instances of Construct to participate in gossiping at the moment. This function is currently accomplished with the zero configuration network protocol, Bonjour [12]. This provides a convenient solution, allowing us to work on the core gossiping protocol, but limits the scalability of the initial implementation, as we effectively assume global knowledge of Construct instances.

Currently, there is no evaluation or measurement taken of the performance of this implementation. To evaluate this effectively is a difficult problem and, in working towards a solution, we have developed a framework which describes the parameters we wish to tune and the measurements we will use to determine the effects of changing these parameters. The framework we will use is described in the following section.

5. AN EVALUATION FRAMEWORK FOR GOSSIPING PROTOCOLS

Gossiping algorithms involve trade-offs between a number of factors, for example: network overhead, reliability, or latency. As novel gossiping techniques are designed, there is a requirement to identify where improvements and drawbacks have been introduced. To better understand the impact of using new gossiping techniques we have designed a gossiping evaluation framework detailing parameters that may affect performance and the measurements that we will take to evaluate the impact.

For researchers it is a complex, even impractical, process to perform real-world tests on their gossiping implementations. Experimentation of this sort will require the use of large numbers of heterogeneous devices spread across the globe to gain insight into the behaviours of their systems. Initiatives such as Planet Lab [13] support global experimentation however there is a considerable overhead in preparing and launching such tests. To obtain the most out of a real-world test session we can first test by simulation.

By employing a framework for evaluating gossiping systems we aim to gain a comprehensive understanding of how algorithms will perform under varying conditions (arrangement of devices, link latency, node failure etc.). Through simulation we will have a basis to compare both existing algorithms and our new algorithms. Through real-world experimentation we can validate our results by observing actual deployments with emergent behaviour in the wild. By taking a systematic approach with the description of parameters and measurements, and evaluating at different points in the test space using both simulation and real-world experimentation, we will gain a comprehensive understanding of our algorithms.

5.1 Overview of the Framework

The gossiping evaluation framework takes a two-phase approach involving OMNet++ [16], a discrete event simulator that can model computer networks and communication protocols, and real-world experimentation on Planet Lab. The two parts of the experimentation method compliment and provide validation against each other. The results from simulation and real-world experimentation should have commonalities providing a consensus on the performance of the particular implementation.

We have identified the main set of variables that affect the performance of the gossiping algorithm under inspection. The impact of altering each is often seen to be combinatorial and multiplicative. To address this, we have also identified a set of measurements that we feel can be used to characterise and compare the performance of gossiping algorithms.

5.2 Evaluation parameters

There are a number of parameters that can be manipulated when deploying a network of gossiping nodes. Table 1 describes these.

Gossiping networks trade-off scalability with reliability properties. As the number of nodes in a gossiping network increases, the amount of data being transmitted will increase. This adds redundancy to the network but impacts on bandwidth utilisation. The optimal configuration of the network depends on tuning each parameter with consideration to the impact on other parameters plus the overall affect on the network.

Node tables store lists of peers that may be gossiped with. Smaller tables lead to a higher likelihood of node isolation or disconnected islands. A fully connected network, where node tables store the location of all other nodes, is infeasible on a global-scale. The memory footprint of a node will be the sum of node table size with the buffer size. The memory usage becomes highly critical in resource limited

Parameter	Remarks
Number of nodes	How many nodes are deployed in the network?
Node table size	How many other nodes does each node know about? This may vary between nodes.
Buffer size	How much data can each node hold?
Probability of nodes joining or leaving	How volatile is the network configuration?
Message size	How much data is gossiped in each cycle?
Message gossip frequency	How often do gossiping cycles take place?
The 'fanout'	How many nodes does a node gossip to per cycle?
Physical network	How do properties such as bandwidth, link latency, link failure, and topology impact performance?

Table 1: Tuneable parameters in a network of gossiping nodes

devices commonly found in *ad hoc* networks. As more data is stored in buffers, increasing redundancy, the reliability will increase.

Altering the probability of nodes joining or leaving the network impacts the rate at which data is propagated. Depending on the network configuration this may increase or decrease the rate at which data travels around the network or may result in islands of connectivity being introduced into the system.

With an increase in traffic levels, buffers will fill more rapidly. In some cases this may result in messages having to be dropped. Over communication may saturate the network links, leading to blockages. Message gossip frequency and fanout have a similar impact on the system. The computational cost of a gossiping algorithm places a restriction on the minimum value that the gossiping frequency can take. The advantage of more communication is an increase in redundancy as data is copied about the network.

In addition to these parameters, well understood properties resulting from the physical organisation of a network will also impact the performance of any given algorithm. Bandwidth, link latency and link failure are three examples of such properties. OMNet++ has support for modelling these network features, which we will use to test these algorithms under various network conditions.

5.3 Measurements

Manipulation of the parameters defined above impacts on the performance and behaviour of a network. To facilitate characterisation of different gossiping algorithms we need tangible figures for comparison. Table 2 describes these measurements to be taken as an output of experimentation.

Latency and saturation form the baseline measurements for our evaluation of gossiping systems. Latency, the average time taken for a unit of data to travel between randomly selected points A and B, defines the simplest basis for comparison. From this measurement we extend to a measurement of data saturation. In Eugster et al. [7] saturation is described as the average amount of time taken for messages within the system to reach 90% of the nodes in the network. However, the selection of an appropriate fraction depends on the requirements of an individual application, therefore evaluation at a number of saturation points is desirable.

Robustness can be measured with respect to latency and saturation measurements when subjected to network disruptions such as node or link failure, increased traffic levels, increased latency on communications links, and increased processing delay within nodes. Reliability, buffer and node table size may also be key properties affecting this.

In simple gossiping systems it is common for duplicate messages to arrive at a node. Any messages being transmitted to a node which already has the data contained within that message is wasted bandwidth. As in Eugster et al. [7], this can be measured by a change in the ratio of received information to already known information within a gossiping cycle over time.

Prevalence describes the proportion of nodes in a network to which an item of data is known at a specific time. It provides a useful measure of the commonality of data within a system. Incidence describes the number of new nodes that learn of an item of data in a given gossiping cycle. Prevalence will be greatly affected by the lifespan of data, the buffer size of nodes, and the rate that new data is generated. Incidence will be affected by both the selection of data to be gossiped in any given cycle and the selection of nodes to gossip with.

When investigating algorithms that are capable of using the semantics of data to inform the gossiping process we can provide additional measurements that relate to those semantics. Topic coverage and topic distribution are two such examples. Topic coverage measures the proportion of data associated with a given topic that is known to a node in relation to all data known about that topic. Topic distribution measures the prevalence of data in a network pertaining to a given topic.

Local data redundancy is a measurement of the amount of data stored by a node that can be considered redundant when taking into account the data stored by neighbouring nodes. A high redundancy figure implies a high tolerance in the network for failure of that node. This figure may be affected by many properties, including: node table size; the rate at which new data is contributed at that node; the rate at which data is gossiped; and the relative buffer size of neighbouring nodes.

Finally, the scalability of the system is a combination of changes in the other measurements as network size increases.

6. CONCLUSIONS

As pervasive computing systems grow and more sensors are added to the system, we gain access to increasing volumes of context data. A challenge arises in distributing this data around the system to the points at which it is required by applications in an efficient, scalable, and timely manner. Gossiping algorithms provide a promising means of addressing this challenge but remain unproven in the area. We have described the principles involved in goss-

Property	Description
Latency	A measure of how long it takes for datum X to go from A to B
Saturation	The time taken for a message to reach a given fraction of the nodes
Robustness	The percentage change in latency and saturation in the face of network disruptions (such as node failure, increased traffic...)
Redundancy	The ratio of information received by a node that is already known to it within a given gossiping cycle
Prevalence	The cumulative fraction of nodes that have received a message with respect to time
Incidence	The number of nodes receiving datum X per round
Topic coverage	The ratio of messages about a particular topic known to a node in relation to the total number of messages in the network about that topic
Topic distribution	As for prevalence, but for data pertaining to a specific topic
Local data redundancy	The ratio of the total data known to a node that is replicated in neighbouring nodes
Scalability	Defines the impact on the above measurements with increasing network size

Table 2: Measurements to be taken on gossiping algorithms

iping protocols for information dissemination and the initial implementation of a basic gossip-style algorithm in our pervasive systems architecture, Construct. The complexity of evaluating the performance of this algorithm has led us to define a framework, which describes the parameters, measurements, and approach utilising simulation and real-world testing. We have presented an embryonic version of the evaluation framework; this will grow and evolve as we learn more about gossiping algorithms and applications that are likely to employ these techniques. Using this evaluation framework, we aim to confirm the suitability of gossiping for the dissemination of context information and allow us to gain an understanding of the issues and trade-offs involved when designing a gossiping protocol. Our next steps are to create a simulation of the current implementation in Construct and apply our evaluation framework to both the simulation and concrete implementation.

7. ACKNOWLEDGMENTS

This material is based on works supported by Science Foundation Ireland under Grant No. 04/RP1/I544.

8. REFERENCES

[1] K. P. Birman, M. Hayden, O. Ozkasap, Z. Xiao, M. Budiu, and Y. Minsky. Bimodal multicast. *ACM Trans. Comput. Syst.*, 17(2):41–88, May 1999.

[2] L. Coyle, S. Neely, G. Rey, G. Stevenson, M. Sullivan, S. Dobson, and P. Nixon. Sensor fusion-based middleware for assisted living. In *Proc. of 1st International Conference On Smart homes & heath Telematics (ICOST'2006) "Smart Homes and Beyond"*, pages 281–288. IOS Press, 2006.

[3] A. Datta, S. Quarteroni, and K. Aberer. Autonomous Gossiping: A self-organizing epidemic algorithm for selective information dissemination in mobile ad-hoc networks. In *IC-SNW'04, International Conference on Semantics of a Networked World*, LNCS, pages 126–143, 2004.

[4] A. Demers, D. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart, and D. Terry. Epidemic algorithms for replicated database maintenance. *SIGOPS Oper. Syst. Rev.*, 22(1):8–32, January 1988.

[5] S. Dobson, L. Coyle, and P. Nixon. Hybridising events and knowledge as a basis for building autonomic systems. In *Journal of Autonomic and Trusted Computing. September 2006*, 2006. To appear.

[6] P. T. Eugster and R. Guerraoui. Probabilistic multicast. In *Dependable Systems and Networks*, pages 313–322, 2002.

[7] P. T. Eugster, R. Guerraoui, S. B. Handurukande, P. Kouznetsov, and A. M. Kermarrec. Lightweight probabilistic broadcast. *ACM Trans. Comput. Syst.*, 21(4):341–374, November 2003.

[8] P. T. Eugster, R. Guerraoui, A. M. Kermarrec, and L. Massoulie. Epidemic information dissemination in distributed systems. *Computer*, 37(5):60–67, 2004.

[9] M. Jelasity, A. Montresor, and O. Babaoglu. Grassroots self-management: A modular approach. In *International Workshop on Self-* Properties in Complex Information Systems*, 2004.

[10] M. J. Lin and K. Marzullo. Directional gossip: Gossip in a wide area network. In *European Dependable Computing Conference*, pages 364–379, 1999.

[11] B. McBride. Jena: Implementing the RDF model and syntax specification. In *Proceedings of the 2nd International Workshop on the Semantic Web*, Hong Kong, may 2001.

[12] The Bonjour Developer Connection Homepage. http://developer.apple.com/networking/bonjour/.

[13] The PlanetLab Homepage. http://www.planet-lab.org/.

[14] R. van Renesse, K. P. Birman, and W. Vogels. Astrolabe: A robust and scalable technology for distributed system monitoring, management, and data mining. *ACM Trans. Comput. Syst.*, 21(2):164–206, May 2003.

[15] R. van Renesse, Y. Minsky, and M. Hayden. A gossip-style failure detection service. Technical Report TR98-1687, Cornell University, 1998.

[16] A. Varga. The OMNet++ discrete event simulation system. In *Proceedings of the European Simulation Multiconference (ESM'2001)*, Prague, Czech Republic, June 2001.