Designing an Asynchronous Group Communication Middleware for Wireless Users

Xuwen Yu
University of Notre Dame
xyu2@nd.edu

Surendar Chandra
Unaffiliated
surendar@acm.org

ABSTRACT

We evaluate an asynchronous gossiping middleware for wireless users that propagates messages from any group member to all the other group members. This propagation can either be implemented through distributed mechanisms or can be mediated through servers. Our analysis of asynchronous mechanisms using wireless user availability traces from an university, corporation and a hot spot federation shows that the fundamental impediment to the system performance is the wireless user availability patterns. We then investigate the relative performance for several distributed as well as server mediated approaches. We show that pull mechanisms effectively randomizes the times when messages are propagated and thus achieves better performance than push based mechanisms. We then develop an adaptive approach that customizes the propagation frequency using the last session duration and show that this mechanism exhibits good performance when the required propagation intervals are large. We also show that for a given number of gossips, it is preferable to propagate messages to all available nodes rather than increasing the frequency while correspondingly reducing the number of nodes to propagate messages. Our results allow middleware developers to choose the appropriate propagation model to satisfy their application constraints.

Categories and Subject Descriptors

H.5.3 [**Group and Organization Interfaces**]: Asynchronous interaction

General Terms

Performance

1. INTRODUCTION

Our primary goal is to design a middleware that propagates messages from any group member to the others. Such a middleware can be used for update propagation by group collaboration systems; results from this paper drives the propagation policies for the flockfs [4] collaboration system.

The performance of our system primarily depends on the user availability patterns. Contemporary users are wireless and operate from a variety of locations. Hence, we evaluate the system using user availability traces from an university, corporation and hotspot federation. Our earlier analysis of these traces showed a trend towards smaller user session lengths as well as longer durations between sessions with observable node churn. These availability durations cannot sustain synchronous propagation. Hence, we focus our attention on asynchronous mechanisms.

Messages in asynchronous mechanisms can either be propagated via a server or in a distributed fashion. We investigate the performance of both these approaches. Messages in each of these schemes can either be pushed from a member to another member or pulled from another member. The frequency of these push and pull operations affect the system performance. Frequent operations can help the system propagate messages quickly at the expense of more network traffic. We investigate these parameters using availability traces of contemporary wireless LAN users in an university, corporation and a city-wide hotspot federation.

We report poor system performance regardless of any specific propagation policy. For small groups, server mediated approaches achieve good performance although distributed approaches were competitive for large groups. In general, a pull based approach leverages the randomness of group availability to achieve better performance than push based schemes. We showed that the preceding session duration of an user was sufficient to adapt the propagation frequency. However, systems that required frequent message propagations did not benefit from history based prediction as much as systems that propagated messages less frequently.

Next, Section 2 describes the system architecture. Section 3 describes the results of our analysis. Section 4 describes related work with concluding remarks in Section 5.

2. SYSTEM ARCHITECTURE

We describe the propagation mechanisms, performance metrics and the wireless availability traces used for our study.

2.1 Message propagation mechanisms

We describe two approaches, distributed and server mediated for asynchronously propagating messages to the group.

2.1.1 Distributed approach

A distributed approach periodically propagates the mes-

MSWiM'09, October 26–29, 2009, Tenerife, Canary Islands, Spain.

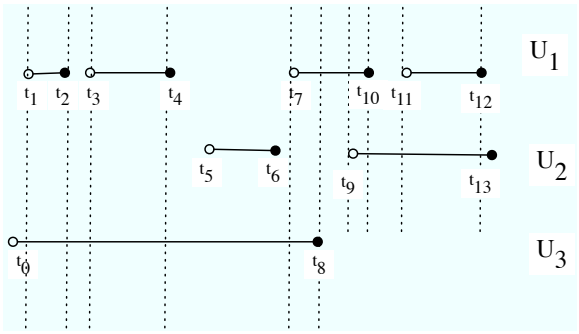


Figure 1: Availability behavior of three users

sages to the subset of simultaneously online nodes using a gossip operation. The time to perform each gossip operation depends on the size of the message as well as on the available network bandwidth. Given the bandwidth availability on wireless LANs, we do not model this duration ([5, 3]). Each message is subsequently propagated to other nodes through successive gossips in order to eventually reach all the group members. Some of these gossip sessions are unnecessary because the corresponding pair of nodes already have all the messages available with each other. The message can either be pushed to other nodes or pulled from other nodes. We refer to these policies as *P2P-push* and *P2P-pull*, respectively. When a node comes online, it always initiates a push or pull operation with other simultaneously available nodes. However, we assume that nodes go offline spontaneously without explicitly pushing its contents to other nodes. Frequent propagation improves system performance while incurring many unnecessary gossips.

2.1.2 Server mediated approach

This method uses an always available server for propagation. This mediation can either be initiated by the node or the server. In server initiated mechanisms, the server periodically pulls messages from nodes that are online and then pushes them to other online nodes. We refer to this policy as *Svr-ServInit*. On the other hand, in a node initiated policy, a node that creates a message periodically pushes them to the server while also retrieving messages from other nodes from the server. We refer to this policy as *Svr-NodeInit*.

2.2 Performance metrics

Next, we describe our metrics to measure the entropy as well as the network overhead.

2.2.1 Entropy using lagAmount

We measure the system entropy using *lagAmount*, a metric that quantifies the average amount of messages that are unavailable at a node. We assume that nodes create messages at a constant rate and quantify the amount of messages created by measuring the amount of time that a node was available. Consider a group of three nodes (U_1, U_2 and U_3) and their availability durations (Figure 1). Momentarily ignoring U_3, at t_5, U_2 does not have messages $(t_1 \dots t_2)$ and $(t_3 \dots t_4)$ from U_1. At t_5, the *lagAmount* at U_2 is $(t_2 - t_1) + (t_4 - t_3)$ with an average *lagAmount* of $\frac{(t_2-t_1)+(t_4-t_3)+(t_6-t_5)}{2}$. The *lagAmount* depends on the propagation policies (explored in §3.2). For example, U_3 could help in ferrying messages between U_1 and U_2.

Note that the number of node pairs is $O(n^2)$ on the group size, the *lagAmount* depends on the group size ($O(n)$). For distributed scenarios, the *lagAmount* also depends on the amount of time that pairs of nodes overlap.

2.2.2 Network overhead using number of gossips

Our system uses pair-wise gossips to propagate the messages; *numGossips* measures the number of anti-entropy operations. If a particular gossip only received messages from the corresponding node (without propagating messages from other peers), we consider the count of those as wasted gossips (*numWGossips*). Note that these metrics do not account for the amount of messages (say in kilobytes) that were actually propagated during a successful gossip operation; this assumption is reasonable [5, 3].

2.3 Wireless availability traces

We required user availability traces to analyze the message propagation. Our group members were wireless and mobile. Hence, we used availability traces from a variety of locations. We used traces collected at IBM Research [2] from 7/22/2002 through 8/17/2002, at the *îles Sans Fil* [10], a free city-wide hotspot federation from 8/28/2004 through 8/28/2007 as well as application level wireless user availability traces collected at Notre Dame (both on campus and the dormitories) from 12/3/2007 through 8/25/2008. We refer to these traces as *corporate*, *hotspot* and *university*, respectively. For our analysis, we focused on the two weeks of 8/1/2007 to 8/15/2007 in the *hotspot* trace (most recent), two weeks starting from 7/22/2002 in the *corporate* trace (typical) and in the two weeks starting at 12/04/2007 in the *university* trace (end of semester and hence busiest). During the trace duration, we observed 1,366, 2,724 and 2,729 unique users in *corporate*, *hotspot* and *university* traces, respectively. An in-depth analysis of these and other traces over longer durations are reported elsewhere. Our analysis showed that median user session durations in the *corporate*, *hotspot* and *university* were 2.8 hours, 35 minutes and 20 minutes, respectively. The median duration between sessions were 3.5 hours, 9.6 hours and 1.78 hours, respectively. We also observed node churn in all scenarios.

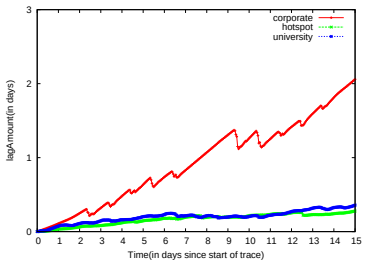
3. RESULTS

We use the traces described in § 2.3. We randomly created group sizes of three, fifteen and thirty and investigate the effects of the propagation frequency using the entropy and gossip metrics outlined in § 2.2. Three is the smallest meaningful group size for asynchronous propagation. We repeat the simulations for fifty different groups. Our experiments answered questions such as:

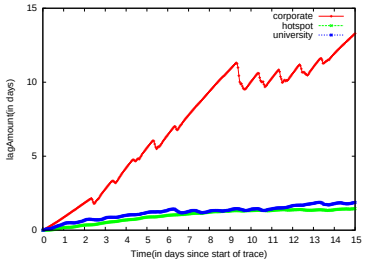
1. Gossips are considered to not be suited for quick dissemination [5, 3]. What is the best-case propagation behavior for wireless users?
2. Are push and pull mechanisms complementary?
3. What are the tradeoffs for more frequently propagating messages to fewer nodes?

3.1 Fundamental limitation for wireless users

First, we analyze the best performance achievable for wireless users. The best performance is achieved in a *Svr-ServInit* scenario where the propagation frequency is zero seconds.



(a) lagAmount, group size=3



(b) lagAmount, group size=15

Figure 2: Best performance achievable

Still, messages cannot be sent to nodes that are not online; the best performance is dependent on the user availability behavior. Also, this policy is not practical because it will incur tremendous network overhead for constantly pulling messages. Practical systems delay this operation and propagate the messages in less frequent intervals; we investigate these durations in Section 3.2.

We plot the *lagAmount* for groups of size three and fifteen in Figure 2. Groups of size fifty were similar in behavior to groups of size fifteen. We note that the *lagAmount* was high and depends on the group size and node session duration. The *lagAmount* also accumulates because of node churn; when a node leaves the system, we continue to compute its entropy for messages created by other nodes. We note that the *lagAmount* is high for the *corporate* trace; as high as two days worth of messages by the end of the trace. The session durations for the *hotspot* and *university* traces were smaller (§2.3), producing smaller *lagAmounts*. Unless efforts to improve the wireless user availability are successful, it is not possible to use asynchronous message propagation for applications that require speedier performance.

3.2 Practical propagation policies

Next, we investigate the system using realistic propagation intervals. Nodes propagate messages when they come online with subsequent propagations at regular intervals. Frequent propagation can reduce the system entropy while increasing the number of unnecessary gossips. The propagation frequency is primarily driven by the application that will use the middleware. The propagation frequency also depends on the user availability patterns. For example, if the node becomes unavailable before subsequent message propagation, then the messages created during this session will be delayed, increasing the system entropy.

Nodes are assumed to spontaneously go offline; it is impractical to require that nodes propagate their messages before going offline. Hence, we investigate whether we can predict the times when a node will go offline and correspondingly adapt the propagation frequency. We investigate history based approaches that use the past session durations to predict the next session duration. We used history depths of one, two and three. For each of these values, we choose base frequencies of five min., fifteen min., thirty min. or an hour. For each base frequency, if the predicted session duration was less than the predicted duration, we reduce the propagation frequency to the predicted value. We tabulate the cumulative percentage of times when a node goes offline before subsequent message propagation for our traces in Table 1. The table is read as follows: for the *university* trace, using a base frequency of an hour, 90.09% of the time, the node will go offline before subsequent propagation. Given the average session duration of 20 mins. (§2.3), most of these messages were delayed until the node came back online; average time between sessions was 1.78 hours for this trace (§2.3). The *lagAmount* values are reduced when the *times when no further messages were propagated* was low; using the last session duration can reduce this value to 48.86%.

From Table 1, we note that the percentage of time when nodes did not subsequently propagate messages was higher for the *university* than in other traces. The best performance was achieved when the base propagation frequency was high; this observation needs to be reconciled with the number of unnecessary gossips. An adaptive policy that used the recent session duration was effective. Hence, we use the past session duration to predict the current session. We investigate our propagation policies using the adaptive as well as application specified propagation durations.

We report the extra overhead imposed by a particular policy as compared to the best policy (evaluated in § 3.1). For example, if the *lagAmount* on a particular day for the *P2P-pull* policy was four while the corresponding values for the best policy was three, we report a relative *lagAmount* of one day. Analyzing the relative costs has the added benefit of eliminating the *lagAmounts* accumulated for not propagating messages to nodes that had already left the system. However, messages which were created on the node that will leave the system and that was not propagated to other nodes will still affect the relative costs. Ideally, we prefer a policy that minimizes the relative performance. In the interest of space, we only plot a few representative values.

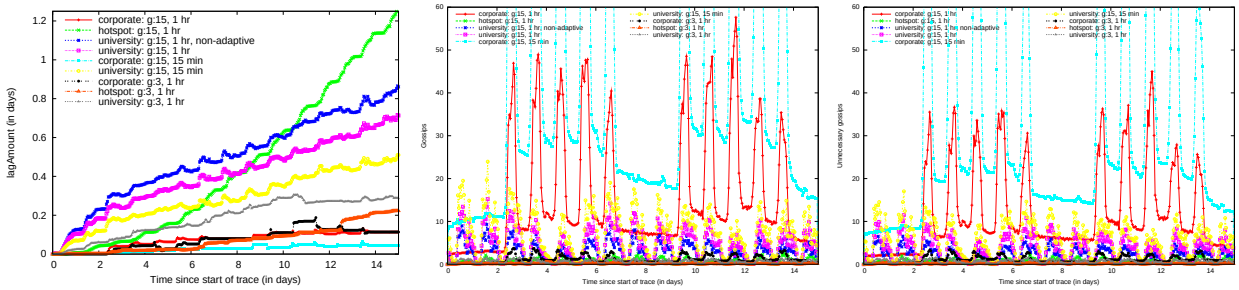
3.2.1 P2P-pull

We investigate the performance penalty introduced by the *P2P-pull* policy as compared to the best policy (§3.1) and plot the relative *lagAmount*, the number of gossips and unnecessary gossips in Figure 3. As nodes come online, they pull messages from other group members that are also online. Subsequently, they pull messages at frequent intervals that are either fixed or adaptive (§3.2). The times that nodes come online are random. Hence the pull operations to propagate messages created at any node to other nodes is also random. Note that the adaptive duration allows the local node to pull messages from other nodes before going offline; it does not affect when the local messages are sent to other nodes. Note that adaptive policies only made significant improvements in the *university* scenario.

First consider the relative *lagAmount* (Figure 3(a)); relative *lagAmounts* continue to increase because of the residual messages left in a node that were not pulled by other nodes before it left the system; the best policy (§3.1) would

trace	base freq = 5 min				base freq = 15 min				base freq = 30 min				base freq = 60 min			
	0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3
corporate	1.47	1.30	1.39	1.39	10.06	8.87	9.65	9.71	17.45	14.64	15.99	16.31	28.48	22.26	24.69	25.19
hotspot	2.52	2.36	2.49	2.53	20.06	16.68	17.41	17.24	38.84	28.73	30.18	30.10	60.63	39.39	41.36	41.06
university	18.88	15.68	17.33	17.84	39.98	30.09	33.86	35.46	63.38	41.58	47.08	49.21	90.09	48.86	52.44	53.74

Table 1: Cumulative percentage of times when node went offline before subsequent message propagation



(a) lagAmount (b) numGossips (c) numWGossips

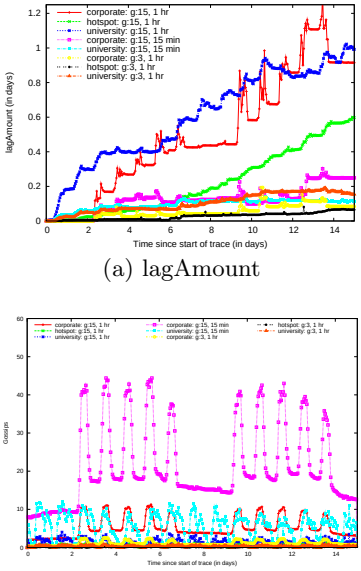
Figure 3: P2P-pull (adaptive duration by default)

have propagated these messages. This effect was pronounced in the *hotspot* scenarios. For a group of size fifteen in the *hotspot* scenario with an adaptive policy, base frequency of one hour and history depth of one, the relative *lagAmount* on the fifteenth day was over 1.2 days; i.e., each pair of nodes did not have about $\frac{1.2}{14} = 0.086$ days worth of messages. These values were slightly bigger for smaller groups which had fewer opportunities for message propagation (pairwise *lagAmount* of $\frac{0.2}{2} = 0.1$). The adaptive policy for the *university* scenario showed an residual relative *lagAmount* of 0.7 days vs 0.88 days for a non-adaptive policy.

Investigating the number of gossips and the number of unnecessary gossips in Figures 3(b) and 3(c), respectively, we note the high number of gossips as well as unnecessary gossips in the corporate setting. For other scenarios which exhibit poorer user availability, the number of gossips were low; about ten for groups of size 15. As we show in subsequent sections, these parameters are competitive with server mediated approaches.

3.2.2 P2P-push

Next, we investigate the behavior of the *P2P-push* policy and plot the results of our analysis in Figure 4. The system performance is far more sensitive to the propagation rate of the local node. An adaptive policy in this scenario is likely to push its messages to another node before going offline. Hence, the adaptive policy has a more positive effect than when using *P2P-pull* mechanisms. On the other hand, any local message that was not pushed by the source to other nodes will not propagate to any other nodes; increasing the overall entropy. Note that in the case of *P2P-pull*, it is the responsibility of other nodes to pull local messages. For a group of size 15, there are 14 other nodes which are attempting to perform this propagation vs 1 for the *P2P-push* policy. Overall, the system performance is slightly worse than the performance of the *P2P-pull* policy. An adaptive policy for the *university* setting for a group of size 15 with base propagation frequency of one hour experiences a *lagAmount* of 0.75 days (as compared to 0.7 days for the *P2P-pull* policy). Hence, the *P2P-pull* is preferable to the *P2P-push* policy.



(a) lagAmount

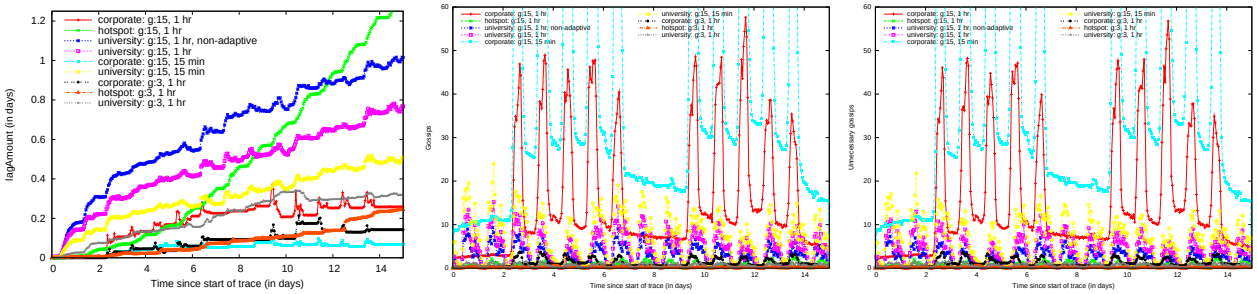
(b) numGossips

Figure 5: Svr-ServInit

3.2.3 Svr-ServInit

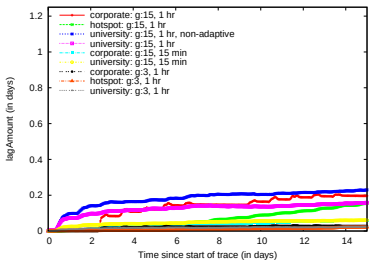
Next, we investigate the relative performance of server assisted approaches. We plot the relative *lagAmount* and the number of gossips in Figure 5. Note that the server cannot adapt to the session durations of individual nodes; if a particular node came online and went offline between the propagation duration, then its messages are not propagated to other nodes even though the server is continuously available. Also, all the gossips in this scenario are useful. Each gossip is comprised of two operations; a pull of messages from a particular node and a push of these to other nodes.

From Figure 5(a), we observe that the *lagAmount* progressively increases with time. Even if the server was continuously available, it is possible for a node to come online and

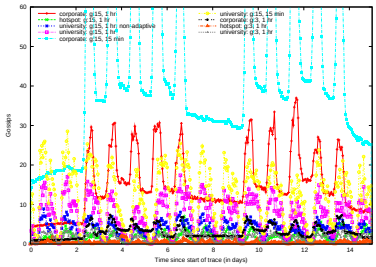


(a) lagAmount (b) numGossips (c) numWGossips

Figure 4: P2P-push (adaptive duration by default)



(a) lagAmount



(b) numGossips

Figure 6: Svr-NodeInit

go offline without propagating its messages to other nodes (especially if the session duration was below the propagation rate). For small group sizes, the *Svr-ServInit* policy exhibits better performance than distributed approaches. For example, for groups of size three in an *university* with base propagation every hour, the adaptive *P2P-pull* policy experienced a relative *lagAmount* of 0.3 days at the end of the 15^{th} day. However, the *Svr-ServInit* policy for similar settings experienced a *lagAmount* of 0.15 days. However, the distributed approaches are more competitive for larger groups. For the *Svr-ServInit* policy in the *university* scenario with a group size of 15 and a propagation frequency of an hour, the relative *lagAmount* is about a day. The corresponding values for an adaptive *P2P-pull* was only 0.7 days. The improvements were even more pronounced for groups of size 50. Since the distributed approaches allow any nodes to initiate the propagation operation, P2P schemes achieve good randomization of message propagation and better system performance.

3.2.4 *Svr-NodeInit*

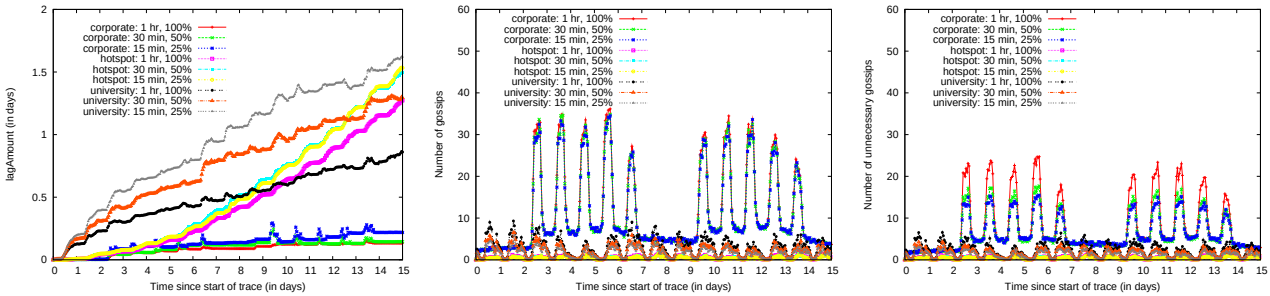
Finally, Figure 6 investigates the performance of the *Svr-NodeInit* policy. This policy is similar to the *P2P-push* policy; nodes periodically send its messages to the server while downloading new messages from other nodes. Thus, this policy leverages the availability of server nodes and propagates its messages quickly. Figure 6(a) shows that even after fifteen days, the *lagAmount* was a modest 0.2 days (can be as high as 1.2 days for the other propagation mechanisms).

3.3 Peer selection mechanisms

Increasing the propagation rate reduces *lagAmount* but requires more gossips. Next, we investigate a tradeoff that chooses higher propagation rates while reducing the number of nodes (among online nodes) to propagate messages.

We plot the relative performance for varying the number of nodes for propagation using the non-adaptive *P2P-pull* policy for groups of size 15 in Figure 7. We reduce the percentage of online users from 100% to 50% and 25% while increasing the propagation rates from 60 min. to 30 min. and 15 min. For the *corporate* trace, we observe that choosing fewer nodes has relatively minor effect on the *lagAmounts*. At the end of fifteen days, the *lagAmount* was about 0.1 days for propagating to 100% as well as 50% of the nodes (doubling the propagation rates). Propagating to 25% of the nodes (quadrupling the propagation rate) increased the *lagAmount* to 0.15 days. The corresponding reduction in unnecessary gossips was from 24 to 16 and 14, respectively. For the *hotspot* trace, reducing the frequency increased the *lagAmount* from 1.3 days to 1.5 days with small differences in the number of unnecessary gossips. However, for the *university* trace, reducing the percentage of online nodes to propagate messages from 100% to 50% and 25% (while exponentially increasing the propagation rates) drastically worsened the *lagAmount* from about 0.8 days to 1.3 days and 1.6 days, respectively. The number of unnecessary gossips showed a small improvement.

A similar analysis for groups of size fifty (not illustrated for lack of space) showed worsening performance for both the *hotspot* and the *university* trace. For the *hotspot* traces, the *lagAmounts* increased from 2 days to 2.6 and 3.2 days, respectively. For the *university* traces, the *lagAmounts* increased from 0.6 days to 1.2 and 1.55 days, respectively. The number of unnecessary gossips also worsened for the *corporate* trace; from 50 gossips to 225 and 125 gossips, respectively. For scenarios which exhibit poorer availability, reducing the percentage of nodes while increasing the message propagation rate is not a viable option; any middleware should propagate messages to all the nodes that are online.



(a) lagAmount (b) numGossips (c) numWGossips

Figure 7: Tradeoff between propagation frequency and choosing fewer online nodes (group size: 15, *P2P-pull*)

4. RELATED WORK

Birman et al. [3] surveyed recent developments on the strengths and limitations of gossip protocols. They showed that gossip based protocols are not designed to expeditiously propagate updates. We use empirical wireless availability data to study the impact of the propagation parameters.

Vahdat et al. [11] used epidemic routing to propagate updates in an ad hoc networking scenario. They simulated a random node mobility pattern and analyzed the propagation behavior by varying the radio range. Jain et al. [6] investigated the routing behavior in a DTN. They used simulations and progressively increased the amounts of network topology information available to the routing mechanism. We validate propagation rates using empirical node availability behavior rather than by using simulated behavior.

Bakhshi et al. [1] surveyed formal analysis techniques for gossiping protocols. Our evaluation metrics were influenced by a consistency count metric described by Kuenning et al. [8]. Jelasity et al. [7] addressed the problem of selecting peers for gossiping. Similarly, Kwiatkowska et al. [9] evaluated gossip protocols using probabilistic model checking. In contrast to simulation based studies, this technique provides both an exhaustive search of all possible behaviors of the system, including best and worst-case scenarios and exact quantitative results. Their work is concerned with identifying the set of gossip peers; each node maintains a relatively small local membership table providing a partial view of the network. Using wireless users availability patterns, we show that the system should gossip with all the available peers to achieve good performance.

5. DISCUSSION

We analyzed the behavior of asynchronous message delivery mechanisms using empirical access traces of wireless users from a variety of locales. We developed a metric to measure the entropy of our system. We investigated the performance bounds of a best policy that instantaneously propagated the messages created at any user to other users. We show that factors such as node availability and node churn play an important role in the system performance. We then compared the performance of several practical distributed as well as server mediated approaches with this policy. We showed that *Svr-NodeInit* policy achieved the best performance with the *P2P-pull* policy exhibiting competitive performance. Overall, distributed policies are competitive, especially for larger groups.

Acknowledgments

We thank the contributors, maintainers and supporters of CRAWDAD archive. Supported in part by the U.S. National Science Foundation (CNS-0447671).

6. REFERENCES

[1] R. Bakhshi, F. Bonnet, W. Fokkink, and B. Haverkort. Formal analysis techniques for gossiping protocols. *SIGOPS Oper. Syst. Rev.*, 41(5):28–36, 2007.

[2] M. Balazinska and P. Castro. CRAWDAD data set ibm/watson (v. 2003-02-19). `http://crawdad.cs.dartmouth.edu/ibm/watson`, Feb. 2003.

[3] K. Birman. The promise, and limitations, of gossip protocols. *SIGOPS Oper. Syst. Rev.*, 41(5):8–13, 2007.

[4] S. Chandra and N. Regola. flockfs, a moderated group authoring system for wireless workgroups. In *Mobiquitous '09*, Toronto, Canada, July 2009.

[5] 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. In *ACM PODC*, pages 1–12, Aug. 1987.

[6] S. Jain, K. Fall, and R. Patra. Routing in a delay tolerant network. In *Sigcomm '04*, pages 145–158, 2004.

[7] M. Jelasity, R. Guerraoui, A.-M. Kermarrec, and M. van Steen. The peer sampling service: experimental evaluation of unstructured gossip-based implementations. In *Middleware '04*, pages 79–98, New York, NY, USA, 2004. Springer-Verlag New York, Inc.

[8] G. H. Kuenning, R. Bagrodia, R. G. Guy, G. J. Popek, P. L. Reiher, and A.-I. Wang. Measuring the quality of service of optimistic replication. In *ECOOP '98: Workshop ion on Object-Oriented Technology*, pages 319–320, London, UK, 1998. Springer-Verlag.

[9] M. Kwiatkowska, G. Norman, and D. Parker. Analysis of a gossip protocol in prism. *SIGMETRICS Perform. Eval. Rev.*, 36(3):17–22, 2008.

[10] M. Lenczner, B. Gregoire, and F. Proulx. CRAWDAD trace ilesansfil/wifidog/session/04_07 (v. 2007-08-27). `http://crawdad.cs.dartmouth.edu/ilesansfil`, Aug. 2007.

[11] A. Vahdat and D. Becker. Epidemic routing for partially connected ad hoc networks. Technical Report CS-2000-06, Duke University, July 2000.