Asynchronous Rumour Spreading in Social and Signed Networks

Christos Patsonakis  
University of Athens

Mema Roussopoulos  
University of Athens

Abstract

In this paper, we present an experimental analysis of the asynchronous push & pull rumour spreading protocol. This protocol is, to date, the best-performing rumour spreading protocol for simple, scalable, and robust information dissemination in distributed systems. We analyse the effect that multiple parameters have on the protocol’s performance, such as varying the rate at which nodes propagate rumours, using memory to avoid contacting the same neighbor twice in a row, varying the stopping criteria used by nodes to decide when to stop spreading the rumour, and others.

Prior work has focused on either providing theoretical upper bounds regarding the number of rounds needed to spread the rumour to all nodes, or, proposes improvements by adjusting isolated parameters. To our knowledge, our work is the first to study how multiple parameters affect system behaviour both in isolation and combination and under a wide range of values.

Our analysis is based on experimental simulations using real-world social network datasets, thus complementing prior theoretical work to shed light on how the protocol behaves in practical, real-world systems. We also study the behaviour of the protocol on a special type of social graph, called signed networks (e.g., Slashdot and Epinions), whose links indicate stronger trust relationships. Finally, through our detailed analysis, we demonstrate how a few simple additions to the protocol can improve the total time required to inform 100% of the nodes by up to 92.49%.

1 Introduction

Efficient information dissemination is a fundamental problem in distributed systems. In particular, rumour spreading is a class of randomized dissemination protocols that have been proposed for a variety of distributed applications such as maintaining consistency in replicated database settings [25], multicast [16], distributed ranking [24], and others.

Rumour spreading protocols are well-known for their robustness, simplicity, and scalability properties [30]. In distributed systems that use rumour spreading, information may be spread in one of three ways: 1) by having the informed nodes (those with the information to be disseminated) actively push the information to the rest of the network, 2) by having the uninformed nodes request, or pull the information, or 3) by combining both push & pull approaches. Nodes can engage in the protocol in complete synchrony by pushing and/or pulling at the same time (in rounds), or asynchronously, with each node running its own clock. The goal is to have the rumour (the information of interest) propagate quickly and efficiently throughout the system.

While there is a wealth of prior work on rumour spreading protocols, the existing literature has gaps in two important areas. First, the spectrum of possible improvements to the plain vanilla push/pull versions of these protocols is large and to date, remains mostly unexplored. There are several factors that affect the performance of these protocols, typically gauged as how quickly information is propagated throughout the system. Prior work is largely theoretical in nature and/or proposes improving performance by adjusting isolated parameters of the protocol such as choice of nodes to neighbor with to impose a more efficient topology of inter-connections between nodes, choice of neighbor to which to send the rumour, use of memory to avoid sending the rumour twice to the same neighbor and stopping criteria to avoid propagating the rumour more than needed. To date, no study has focused on studying how multiple parameters affect the protocol’s behaviour both in isolation and combination and under a range of values.

Second, while prior work examines the behaviour of rumour spreading protocols in a variety of topologies (e.g., meshes and tori [35], butterfly networks [41]), very few publications focus on social topologies. Several
and very diverse distributed applications link their nodes based on out-of-band, or, trust relationships, thus forming social graphs. Examples include LOCKSS [39], a digital preservation system, Tribler [40] and Maze [19], two social-based peer-to-peer file-sharing systems, and Diaspora [3] and PeerSoN [18], two distributed online social networking services (OSNs).

Until now, nearly all of the published results regarding rumour spreading protocols on social graphs are theoretical analysis. While they provide upper bounds on the number of rounds that these protocols require to achieve propagation, they do not provide any insight with regards to network-related metrics. Such information includes measured time delays and the network load incurred by the exchange of protocol messages. Moreover, a substantial portion of the published results focuses only on synchronous rumour spreading.

In this paper, we provide an in-depth experimental analysis of the asynchronous push & pull rumour spreading protocol. We focus on this particular type of protocol since it is known to combine the benefits of both push and pull [25, 36]. Furthermore, it has recently been proven that the asynchronous variant performs substantially better than the synchronous in both preferential attachment graphs (PA - synthetic graphs that resemble social graphs) and real-world social networks [28, 27].

Our methodology for extracting results is via simulations using a variety of real-world social network datasets (ten in total). We evaluate the impact of individual parameters on protocol performance over a wide range of values. We also present combined experiments, where we leverage the knowledge acquired from our single-parameter experiments to illustrate how an enhanced protocol with a few simple additions can significantly outperform the plain vanilla push and pull protocol. Moreover, we explore how information flows on a special type of social network, called signed networks, whose links indicate stronger trust values than those of common social networks. We are particularly interested in examining whether signed networks exhibit different behaviours with regards to information dissemination.

We believe that the aforementioned, as well as other social-network-based applications, can benefit from our findings. Tribler already uses a rumour spreading protocol for its "Buddycast" algorithm. LOCKSS nodes could gossip about the behaviour of other nodes in polls which, coupled with a sybilproof reputation mechanism [20], would lead to a reliable reputation system for handling peer introductions [34]. In Maze, peers can gossip about friends’ status updates, finding other collectors with whom they share similar interests, etc. PeerSoN and Diaspora peers can use a gossip protocol to quickly communicate system state information, an approach that has been successfully used for the Amazon Simple Storage Service (Amazon S3) [1].

In summary, we make the following contributions:

- We measure the effect that individual, as well as combinations of multiple parameters have on the plain vanilla asynchronous push & pull rumour spreading protocol over a wide range of values.
- We demonstrate that full rumour propagation is highly inefficient in social networks. However, the protocol’s efficiency is exhibited when the purpose is to inform large subsets of the node population, e.g., 90% to 97%, which is often sufficient for voting and quorum-based systems.
- In contrast to prior work, we take a pragmatic, empirical approach. Our study is based on a set of ten diverse real-world, publicly available social network datasets and our network model accounts for link latencies and bandwidths as well as the concurrent processing of network events at nodes.
- We present an enhanced protocol that is based on an intelligent selection of parameter values and illustrate that it improves the total time required to inform 100% of the nodes by up to 92.49%.

2 Background

Rumour spreading protocols are a series of randomized protocols which were initially proposed for distributing updates and ensuring eventual consistency amongst the sites of a replicated database [25]. Their simplicity, robustness, and scalability properties has made them attractive for use in a number of other applications including multicast [16], distributed ranking [24], and others.

In the simplest case, where there is only one piece of information to propagate, rumour spreading protocols resemble the random phone call model introduced in [36]. Each of these protocols assumes a start-up phase, where a piece of information (called update, rumour, or gossip) is injected at an arbitrary node, known as the originator. These algorithms then proceed in a series of synchronous communication rounds, based on the period of a globally accessible clock. In each round, nodes can be in one of the following states: Informed: a node that knows the rumour and will spread it; Uninformed: a node that has not yet received the rumour and will ask for it; Removed: an informed node who will refrain from spreading the rumour since it no longer considers it “hot”, i.e. it is old news. The purpose of the last state is to limit the amount of redundant communication while still trying to achieve rumour dissemination to all nodes.

There are three basic versions of rumour spreading protocols. In the push version, only the informed nodes
choose uniformly at random a neighbor to which they will transmit the rumour. This requires the sending of only one message and its receipt from the destination node marks the end of the current round. In the pull version, every uninformed node contacts a randomly selected neighbor and asks it for the rumour. The recipient of the pull message replies by sending back the rumour, iff it is informed (if it is uninformed, the reply will be an empty message). Note that in this version, in order for a round to be complete, it involves exchanging two messages.

One can combine the two aforementioned strategies and obtain the push & pull version, where, in each round, every node chooses uniformly at random one of its neighbors and, depending on whether it is informed or not, either pushes or pulls the rumour. Therefore, an informed node can, in a single round, push the rumour to one of its neighbors and also inform one additional node if it also receives a pull message from the latter. Thus, in this setting, a round can involve the exchange of up to a total of three messages. Nearly all prior publications (see Section 3 on rumour spreading protocols provide upper bounds regarding the number of rounds that are required for a rumour to spread across all nodes.

To avoid the constraints of synchrony, researchers have proposed asynchronous variants of these protocols in which every node is equipped with its own independent clock. These clocks comply with the asynchronous time model introduced by Boyd et al. [17]. Namely, node clocks are modeled as rate 1 Poisson processes, i.e., the time between two consecutive clock ticks is independent and exponentially distributed with \( \lambda = 1 \).

3 Related Work

The literature concerning rumour spreading protocols is extensive. A number of studies focus on how the topology (i.e., graph structure) of connections between nodes affects the number of rounds required for a rumour to spread throughout the system for a variety of graph topologies including meshes and tori [35], butterfly networks [41], sensor networks [17], random, regular, Erdős-Rényi and social graphs [22, 23, 21, 26, 27, 31, 32]. Other studies propose constructing or imposing a particular structure on the topology of connections between nodes in the system to enable more efficient dissemination [37].

Some studies focus only on the synchronous versions of randomized broadcasting [22, 23, 21, 26] whilst others [28, 27, 31, 32] consider both. Regarding the former set, it is well-known that achieving perfect synchrony in a distributed system amongst thousands of nodes distributed over a network with heterogeneous link latencies and bandwidths is difficult, at best. Moreover, recent results show that asynchronous rumour spreading protocols perform better in PA graphs than their synchronous counterparts [28, 27]. For these reasons, we focus on asynchronous rumour spreading in this work.

The majority of studies that focus on graph structure are theoretical analysis and aim to determine the number of rounds that are required to inform all nodes. The authors start with a series of assumptions, typically regarding the graph structure, and via a series of theorems and lemmas, reach a conclusion that is along the lines of: "In graphs with these properties, under protocol \( X \) and with high probability (w.h.p.), \( O(x) \) rounds suffice to broadcast a single piece of information to all nodes". For instance, in [23], the authors prove that the synchronous push & pull protocol can, w.h.p., broadcast a message within \( O(\frac{\log n}{\delta^c}) \) rounds to all nodes. In a later publication [22], this bound is improved to \( O(\frac{\log \phi - 1}{\delta^c} \log n) \) for PA graphs. These and other findings [31, 25] are important because they provide the necessary proof that rumours can "spread fast in social networks" (quoting [22]), which is our focus of interest here. However, they do not provide information regarding network-related metrics. Such information includes measured time delays and network load incurred by the exchange of protocol messages. Moreover, these studies suggest that nodes use derived formulas to determine when to stop propagating a rumour to avoid sending redundant traffic over the network. Unfortunately, these formulas are not feasible to implement in practice because they use values that are difficult to compute and/or require global graph knowledge such as the total number of nodes or graph conductance. In social networks, where there may be tens of millions of nodes, it is infeasible to have nodes compute, store, and update this type of global graph information on a regular basis. In our work, we take an empirical approach. Our aim is to study how asynchronous rumour spreading behaves in real social network settings and to find ways to design such protocols without the need for global graph knowledge.

To our surprise, despite the large number of prior works analyzing rumour spreading protocols, we find there is very little published work on optimizing or improving their performance. Instead, the main focus of the majority of works is the study of the plain vanilla versions of the push, pull, and push & pull protocols. There are three exceptions to this. First, Georgiou et al. (28) examine fault tolerance in asynchronous gossiping and propose algorithms that increase the robustness of the protocol on randomized graphs in the face of an adaptive and oblivious adversary hampering rumour dissemination. Second, Karp et al. (36) attempt to minimize the amount of traffic generated by the asynchronous push & pull protocol on random graphs by in-
introducing the median-counter algorithm. This is a stopping criterion nodes use to decide when to stop considering a rumour “hot” and thus stop propagation. We examine the performance of this algorithm and others in Section 5. Third, Doerr et al. [28] study the impact of equipping nodes with some “neighbor memory” that enables them to avoid contacting the same neighbor twice in a row. When the neighbor memory has space to hold the identity of one neighbor, Doerr et al. [28] show that the performance of the protocol improves by a factor of $\Theta(\log \log n)$. We note that this last study is the only work of which we are aware that contains experimental results based on PA and real-world social datasets. We validate the results of this work and extend them to include a study of how the protocol performs under varying sizes of neighbor memory.

In summary, while there exist some studies that focus on isolated protocol parameters and their impact on performance, no prior study has focused on how multiple parameters affect the protocol’s behaviour both in isolation and combination and under a wide range of values. Such parameters include: choice of neighbor from which to push and/or pull a rumour, the use of neighbor memory to aid in faster dissemination and reduce network load and stopping criteria to end dissemination efficiently, and others. Our work is a first step in understanding how all of these factors together affect protocol performance.

4 Methodology

We use Narses [33], a discrete-event simulator that is written in Java. Since there are no publicly available network traces (that we know of) that indicate how social network nodes are connected, we chose to model the underlying network topology as a star. Nodes are linked to its center, which is assumed to have infinite packet switching capacity, i.e. we assume that the core of the network has no bottleneck links and that traffic is limited only by the end-link connections. Social-network-based applications are comprised by peers that lie at the edges of the network which tend, in most cases, to be the most limiting factor [15]. This approach, while simple, allows us to provide insight with regards to network-related metrics, without sacrificing accuracy [33], such as measured time delays and the network load incurred by the exchange of messages.

Values for the links’ bandwidths are chosen uniformly at random from the range of [3,50] Mbps. These are all cheap, commodity and widely available DSL line speeds. However, bandwidth speeds are not a significant factor in our experiments since the messages that the nodes exchange are only a few bytes long. Link latencies are also uniformly distributed from 10 to 100 ms ([9], [6]).

In our simulations, the “rumour” is simply a randomly selected long value. The simulator, at time $t = 0$, selects one node (typically this is a random selection, but more on this in Section 3) to play the role of the rumour’s originator from which this information will start to spread.

The overlays, i.e. the social relationships amongst the users of the system, are based on a large set of real-world, publicly available social graphs. Table 1 provides a listing, as well as some statistics, of the datasets that we used in our simulations. Nodes are initialized by receiving a list containing their social acquaintances. Each node then sets a timer that is programmed to fire after an exponentially distributed time period with parameter $\lambda$. When a node’s timer expires, its behaviour depends on its current state: Informed: The node selects one or more neighbors, at random, and sends a Push message; Uninformed: The node selects one or more neighbors, at random, and sends a Pull message; Removed: The node does not schedule any more timers since it considers the gossip that it is currently circulating in the network “old news”. Since we are interested in determining how quickly a piece of information can spread over social graphs, we assume that nodes do not perform any computation before they send or receive a message.

We use two metrics to evaluate protocol behaviour in our experiments. First, we measure the total time that is required to inform either all, or, specific percentages of each graph’s largest connected component (LCC). Second, because rumour mongering protocols are known to produce large amounts of network traffic ([37]), we also measure the load imposed on the network. Simply plotting the number of produced messages that are required to achieve some percentage of informed nodes over time does not suffice because the graphs of the social networks we study vary in size significantly. The protocol must produce more messages over a larger period of time to achieve the same target percentage for larger graphs compared to smaller ones. Thus, the network load metric must be independent of the number of nodes in the graph to allow us to evaluate protocol behaviour in a manner that is consistent across all graphs. We achieve this by plotting the network load as the ratio of the number of generated messages to the total time needed to achieve the desired percentage of informed nodes (on the y-axis) versus the desired percentage of informed nodes (on the x-axis).

5 Experiment Description and Results

This section provides a description and an analysis of the results for each set of experiments we ran to evaluate the effect of different parameters on the performance of the asynchronous push & pull protocol.

Performance bottleneck. The first thing that we observed when we began to simulate this protocol was the
Table 1: Structural characteristics of the real-world social network datasets used in our simulations. Type: connections between users can be one-sided (directed), two-sided (undirected) or even indicate a strong value of trust (signed); N: number of nodes; M: number of edges; \( \phi \): graph conductance; Avg.Deg.: the nodes average out-degree; Max.Deg.: the maximum node out-degree; LCC: all of the graphs have a few small isolated partitions. This value indicates the number of nodes that belong in the largest connected component, i.e. not in these small, cut-off communities.

| Network  | Type   | N      | M      | \( \phi \)  | Avg.Deg. | Max.Deg. | LCC    |
|----------|--------|--------|--------|-------------|----------|----------|--------|
| Slashdot1 | signed | 70491  | 396378 | 0.211025    | 5.6231   | 426      | 69863  |
| Slashdot2 | signed | 74899  | 422349 | 0.213837    | 5.63891  | 428      | 74200  |
| Slashdot3 | signed | 75144  | 425072 | 0.209647    | 5.65677  | 428      | 74444  |
| Epinions  | signed | 114222 | 717129 | 0.101079    | 6.74543  | 261      | 1788   |
| WikiSigned | signed | 126514 | 650444 | 0.0951449  | 5.14128  | 2524     | 121118 |
| Hamsterster | undirected | 1858  | 12533  | 0.110169    | 6.74543  | 261      | 1788   |
| Brightkite | undirected | 58228 | 214078 | 0.0868865  | 3.67655  | 1119     | 56739  |
| Facebook  | undirected | 63731 | 1545686 | 0.118187   | 24.2533  | 1098     | 63392  |
| TwitterLists | directed | 23370 | 33101  | 0.0167826  | 1.41639  | 238      | 22322  |
| Google+   | directed | 23628 | 39242  | 0.0418945  | 1.66083  | 2748     | 23613  |

Starting up from 90% and all the way up to (and including) 96% of informed nodes, time scales smoothly (as is the case in lower percentages that are not included in this plot). However, we observe a knee in the graph at 97% after which the total time increases exponentially. Through extensive analysis, we find that this extreme delay is caused by some communities which are connected to the giant component of the network via a handful of links. Since the number of available paths that can be used to reach the members of these communities is extremely small, their chances of being randomly selected for information dissemination decreases. Furthermore, in some other cases, we also observe that these communities are also connected, again via a handful of links, with other communities where, the latter, are almost never connected to the core of the network. Thus, small chains are formed, typically of length 2. For instance, assume that community A is connected via a couple of links to the giant component of the network and is also connected to another community, B, again via a handful of links. If community B is not directly linked to the core of the network, then its members will become eligible for receiving information only when the “hub nodes” of community A (the nodes connecting communities A & B) become informed. This creates a convoy phenomenon, which makes it harder for the protocol, and thus the increased delay, to inform these nodes.

Table 2 illustrates the total amount of time required to inform 97% and 100% of the nodes in the LCC of each graph, as well as the percentage increase of the latter compared to the former. From Table 2, we see that the protocol exhibits significant difficulty informing these last few node percentages. To our knowledge, ours is the first study to document this phenomenon. This is important because many applications (e.g., voting or...
Table 2: Total time required to inform 97% and 100% of the nodes in the largest connected component of each graph and the percentage increase of the latter case compared to the former.

| Network          | 97% Inf. | 100% Inf. | % Incr. |
|------------------|----------|-----------|---------|
| Slashdot1        | 187.50   | 2566.66   | 1268.90%|
| Slashdot2        | 170.57   | 2342.61   | 1273.38%|
| Slashdot3        | 182.34   | 2573.80   | 1311.57%|
| Epinions         | 292.84   | 12520.14  | 4175.47%|
| WikiSigned       | 776.81   | 13602.26  | 1651.03%|
| Hamsterster      | 56.95    | 575.25    | 910.16% |
| Brightkite       | 146.61   | 6526.74   | 4351.63%|
| Facebook         | 16.74    | 1684.24   | 9961.63%|
| TwitterLists     | 378.62   | 1414.16   | 273.50% |
| Google+          | 2799.34  | 18779.10  | 570.84% |

Table 2: Total time required to inform 97% and 100% of the nodes in the largest connected component of each graph and the percentage increase of the latter case compared to the former.

quorum-based systems) may not need full propagation to all nodes to make progress. Since this phenomenon also affects the readability of graphs that plot time over node percentages, from hereon in, for most of the graphs, we plot data values whose maximum node percentages lies in the range of [90%, 97%].

Poisson rate $\lambda$. In this experiment, we examine how varying the parameter that determines the rate at which nodes engage in the protocol affects the total time required to inform the nodes of the graph. Nearly all prior publications model node clocks as Poisson processes with $\lambda = 1$ ([28], [27], [31]). Here, we assign $\lambda$ values that range from 200ms (the maximum propagation latency between source and destination in our network model) to a maximum of 10s with 50ms increments. Figure 2 shows the total time needed to inform 100% of the nodes versus the mean gossip period. For the biggest part of the plot, we observe that these two metrics share a linear relationship. As the value of $\lambda$ increases, however, there appears to be a deviation. We produced a linear fit for each curve (not shown here) and observe, over all topologies, an average maximum deviation of 27.59%. However, only a few data points deviate by that much; the average and standard deviations from the linear fit, again over all topologies, are 7% and 5.43% respectively. Thus, we conclude that these two parameters have an approximate linear relationship.

**Originator popularity.** We next study the effect that the connectivity, or, “popularity” of a node has on the dissemination of a rumour. We classify the nodes of the social graph into three different groups inspired by the scheme that is presented in [39]. Nodes reside in one of the following categories:

- Giant Component (Group 3): This group consists of highly connected individuals that are connected with a large fraction of the network. For instance, singers, actors, or other famous individuals are typical members of this category.

- Middle Region (Group 2): These are star-like shaped communities, with one or two nodes at the center, that mostly interact with other fellow group members and are sparsely connected to the giant component.

- Singletons (Group 1): These are the one-degree nodes.

For each of the aforementioned groups, we randomly sample 10% of its members to act as originators of rumours and measure the impact this has on the performance of the protocol. We repeat this procedure 100 times for each distinct source and average the results. In Figure 3 we plot the effect of originator popularity on the time to inform varying percentages of nodes for the Brightkite, TwitterLists, and Epinions topologies (note, y-axes are on different scales). These topologies are representative plots we have chosen due to lack of space.

The results illustrate an important difference in the total time required to inform 90% of the nodes in the graphs’ LCCs when the originator is in the first group. The data suggest an average slowdown of 25.98% and 30.14% compared to the case where the originator is in group 2 and 3, respectively. On the other hand, there is only a slight improvement when the information starts to spread from a node in group 3 than from a node in group 2. Indeed, the difference is only 4.16%. The (bottom) network load graphs, however, show nearly no improvement at all (2.27% and 2.43% on average).

**Neighbor Contact Criteria.** The fact that a node’s popularity can speed up the dissemination of a piece of data, inspired us to investigate if rumour dissemination
can be accelerated even more if nodes picked their communication partners based on a scheme that favors their most popular, or well-connected, neighbors. This biased selection deviates from the standard uniform random choice used by almost all prior works. We explore the effects of the following alternative neighbor contact criteria:

- Random Biased (RB): Selections under this policy are biased to select more often nodes that belong in group 3 than nodes that belong in groups 2 and 1. Also, it is more likely for a node to pick a neighbor that is part of group 2 than group 1.

- Quasirandom (Q): Nodes have a cyclic list of friends. They choose a random starting position and from then on, they contact neighbors in a round robin fashion. This model is presented in [29].

- Quasirandom Popular (QP): Nodes sort their lists according to popularity and cycle through them from the most popular to the least popular node.

- Quasirandom Unpopular (QU): The exact opposite of QP, i.e., nodes cycle from the least popular to the most popular node.

- Quasirandom Popular to Unpopular (QPU): Neighbor lists are again sorted based on neighbor popularity and nodes first choose the most popular node, then the least popular, then the second most popular, then the second least popular etc.

- Quasirandom Unpopular to Popular (QUP): The exact opposite of QPU.

In Table 3 we illustrate the performance improvements of these strategies compared to the standard uniform random choice. We note that these are all novel neighbor contact criteria that have not been evaluated in the past (apart from quasirandom).

To our surprise, the “Random Biased” strategy performed extremely poorly, i.e., in half of the cases it scored far worse than “Random”, up to 30%. For this reason we chose not to include it in Table 3. All of the variants of the quasirandom model provide tremendous improvements. However, it appears that the best option is to alternate between popular and unpopular neighbors, or vice versa. Indeed, the QPU and QUP contact criteria score the biggest percentages in nearly all of the topologies and have the lowest standard deviation amongst all alternatives. This is because, unlike “Random Biased”, they strike a nice balance between using popular nodes and their many links to disseminate the rumour, and unpopular nodes, who are the major cause of delay, as we illustrated in the beginning of this section. Their performance is almost tied, so we consider both of them as candidates for replacing the standard uniform random choice.

**Stopping Criteria.** Several theoretical papers ([31], [32]),...
stopping criteria: work. We evaluate the performance of the following of redundant information that is circulating over the net-

Table 3: Performance improvement in the total time re-

| Network | Q     | QP    | QU    | QPU   | QUP   |
|---------|-------|-------|-------|-------|-------|
| Slashdot1 | 84.50% | 84.47% | 84.50% | 85.05% | 84.76% |
| Slashdot2 | 83.88% | 84.33% | 84.51% | 84.48% | 84.85% |
| Slashdot3 | 83.05% | 82.62% | 83.02% | 83.64% | 83.54% |
| Epinions  | 84.94% | 84.96% | 85.05% | 84.88% | 84.90% |
| WikiSigned| 84.20% | 84.13% | 83.96% | 83.88% | 84.02% |
| Hamsterster| 56.43% | 47.85% | 45.54% | 75.05% | 73.16% |
| Brightkite | 81.47% | 80.68% | 81.14% | 80.74% | 80.84% |
| Facebook | 64.93% | 50.18% | 49.51% | 81.09% | 82.54% |
| TwitterLists | 80.60% | 80.88% | 80.51% | 80.13% | 79.65% |
| WikiSigned | 84.20% | 84.13% | 83.96% | 83.88% | 84.02% |
| Slashdot3 | 83.05% | 82.62% | 83.02% | 83.64% | 83.54% |
| WikiSigned | 84.20% | 84.13% | 83.96% | 83.88% | 84.02% |
| Facebook | 64.93% | 50.18% | 49.51% | 81.09% | 82.54% |
| TwitterLists | 80.60% | 80.88% | 80.51% | 80.13% | 79.65% |
| Epinions  | 84.94% | 84.96% | 85.05% | 84.88% | 84.90% |
| Google+  | 85.85% | 85.92% | 85.91% | 85.1%  | 85.91% |

Table 4 illustrates the average percentage of informed nodes that each strategy achieved after a large number of runs. According to the data, the most reliable stopping criteria are $O(n \log n)$ and Coin Toss, which manage to achieve an average percentage of 39% and 41.8% of informed nodes, respectively. However, we note that these percentages are drastically reduced due to the fact that they also consider the extremely low percentage of informed nodes that they scored in the TwitterLists and Google+ topologies. This is actually a phenomenon that affects all of the stopping criteria (except for this one case of the $O(n \log n)$ in the Google+ topology) in these directed graphs. We speculate that it could be related to the structural characteristics of these graphs, however, at the moment, we do not have sufficient data to support any claims and leave this issue as future work. Nevertheless, if we exclude these two cases, $O(n \log n)$ and Coin Toss achieve a 42.8% and 52.2% average percentage of informed nodes. Thus, Coin Toss appears to be the only viable candidate for protocols whose main focus is to spread a piece of information to the majority of the overlay.

Neighbor memory. The use of “neighbor memory”, i.e., the ability of nodes to remember which nodes they have communicated with and not to communicate with them in the future, has been shown to improve the performance of the synchronous push & pull protocol, even when it has space to hold only a single neighbor [28, 27]. For instance, Doerr et al. [28] illustrate that a 1-item memory yields an improvement of about 14%-21% compared to the case where there is no memory.

In general, no prior publication has examined how the asynchronous version of the protocol performs under varying neighbor memory sizes. While Doerr et al. [28] do examine how the asynchronous push & pull protocol equipped with a 1-item memory behaves in two large social network crawls compared to synthetic graphs, they do not compare with the case where there is no memory at all, meaning that they do not document the improvement that a 1-item memory offers. Moreover, in that same publication, there is no distinction between the time that a node’s clock ticks and the time that its round completes; they happen during the same asynchronous time step. This may seem to be a straightforward way to model asynchrony, but it fails to account for scenarios such as the following: Assume that the clock of node A, who is uninformed, ticks, and selects node B, who is also uninformed, from its list of neighbors. In a real network deployment, A would send B a pull message which would be delivered to node B after some time $t$. During this time, it is very likely, and we have observed it numerous times in our experiments, that node B receives a push message from a third node, C, who happens to be informed and has chosen node B as its communication...
partner. What this means is that by the time that B receives the message that A sent, it will be informed and will be able to reply to A with the gossip. The time model used in by Doerr et al. [28] is unable to account for scenarios like this. In contrast, our simulations do and thus, we believe they provide more accurate results.

We explore how the use of neighbor memory can enhance the performance of the “Random” and “Random Biased” neighbor selection strategies. In contrast to the quasirandom variants, these are the only strategies that do not use any memory by default. We illustrate the percentage improvements that a 1-item neighbor memory achieves, compared to the case where there is no memory available, with regards to the total time for the “Random” (Table 5) and “Random Biased” (Table 6) policies.

As the data in the tables suggest, this simple enhancement can provide radical improvements to the protocol’s performance. These are especially evident when the objective is to inform 100% of the nodes in the LCC. This serves as further proof that it is the isolated nodes who lie at the edge of the graph, along with the protocol’s inherent difficulty of randomly selecting them, that cause these extreme delays.

Further increases in the memory’s size do not yield any significant improvements compared to the case of a 1-item memory, i.e. a 2-item memory offers an average improvement of far less than 1%. From there on, we observe random miniscule performance fluctuations that we attribute to statistical error.

Communication fan-out. We have thus far assumed

| Network     | log_{2} n+ O(ln \ln n) | Median Counter | O(n \log n) | O(\log n) | O(\log^2 n) | Coin Toss |
|-------------|------------------------|----------------|------------|----------|-------------|-----------|
| Slashdot1   | 0.004995               | 0.00004        | 0.720768   | 0.002115 | 0.418157    | 0.288312  |
| Slashdot2   | 0.001746               | 0.001488       | 0.000027   | 0.002306 | 0.16935     | 0.773464  |
| Slashdot3   | 0.00484                | 0.001115       | 0.000027   | 0.001847 | 0.169795    | 0.676608  |
| Epinions    | 0.00002                | 0.002971       | 0.349494   | 0.005568 | 0.345994    | 0.279595  |
| WikiSigned  | 0.005057               | 0.001197       | 0.66976    | 0.000017 | 0.257827    | 0.721273  |
| Hamsterster | 0.022267               | 0.013423       | 0.175126   | 0.003356 | 0.315996    | 0.378531  |
| Brightkite  | 0.001185               | 0.000991       | 0.517028   | 0.000282 | 0.375403    | 0.061856  |
| Facebook    | 0.087496               | 0.024589       | 0.998359   | 0.032324 | 0.492806    | 0.998359  |
| TwitterLists| 0.000134               | 0.000246       | 0.013705   | 0.000112 | 0.00672     | 0.005824  |
| Google+     | 0.000408               | 0.000339       | 0.457111   | 0.000413 | 0.008126    | 0.00755   |

Table 4: Fraction of the nodes in the LCC that each strategy managed to inform before all of the nodes switched to a removed state. Coin Toss exhibits the best performance on average.

| Network     | 90% Inf. | 99% Inf. | 100% Inf. |
|-------------|----------|----------|-----------|
| Slashdot1   | 31.68%   | 55.24%   | 82.58%    |
| Slashdot2   | 27.77%   | 52.81%   | 80.82%    |
| Slashdot3   | 29.37%   | 54.46%   | 83.84%    |
| Epinions    | 26.46%   | 48.71%   | 84.78%    |
| WikiSigned  | 40.54%   | 59.31%   | 81.88%    |
| Hamsterster | 13.69%   | 41.50%   | 65.48%    |
| Brightkite  | 26.61%   | 41.00%   | 82.04%    |
| Facebook    | 7.80%    | 21.06%   | 52.68%    |
| TwitterLists| 39.45%   | 57.37%   | 78.54%    |
| Google+     | 30.02%   | 35.62%   | 52.85%    |

Table 5: Percentage improvement of the total time required by the “Random” neighbor selection policy to inform 90%, 99% and 100% of the nodes in the LCC of each network when each node is equipped with a 1-item memory compared to the case where there is no memory at all.

| Network     | 90% Inf. | 99% Inf. | 100% Inf. |
|-------------|----------|----------|-----------|
| Slashdot1   | 30.43%   | 39.22%   | 60.37%    |
| Slashdot2   | 26.03%   | 55.01%   | 82.37%    |
| Slashdot3   | 20.32%   | 51.94%   | 81.66%    |
| Epinions    | 30.73%   | 44.32%   | 85.59%    |
| WikiSigned  | 41.36%   | 56.35%   | 81.64%    |
| Hamsterster | 21.39%   | 45.64%   | 60.02%    |
| Brightkite  | 23.89%   | 47.96%   | 81.04%    |
| Facebook    | 7.80%    | 21.06%   | 52.68%    |
| TwitterLists| 39.45%   | 57.37%   | 78.54%    |
| Google+     | 30.02%   | 35.62%   | 52.85%    |

Table 6: Percentage improvement of the total time required by the “Random Biased” neighbor selection policy to inform 90%, 99% and 100% of the nodes in the LCC of each network when each node is equipped with a 1-item memory compared to the case where there is no memory at all.
that the communication fan-out parameter, i.e., the number of neighbors with which a node communicates on each round, has a value of 1. However, there are variants of rumor mongering algorithms (e.g., [37]) that use larger values to provide greater fault-tolerance in the presence of link-failures, or simply to disseminate information faster. We investigate the effects of this parameter on the protocol’s performance using two different approaches.

In the first approach, called absolute communication fan-out, we set a value \( f \) (for instance \( f = 3 \)) which specifies the number of nodes with which every node will attempt to communicate with at each clock tick. (It is possible that some nodes will communicate with less than \( f \) because they do not have that many neighbors). We vary the value of \( f \) starting from 2 to 50, in increments of one. Our results indicate that negligible improvements take place for values of \( f \) that are larger than 7 (for some topologies, it is even less). For this reason, we plot up to a maximum value of \( f = 7 \) in the figures concerning absolute fan-out below.

In the second approach, called relative communication fan-out, we fix a value in the range of \([0, 1]\) which indicates the fraction of neighbors with which nodes will communicate in every round. This means that nodes with larger neighbor lists will communicate with more nodes than the ones with smaller lists. Nodes whose neighbor lists are too small, i.e., multiplying their sizes by the fraction yields a value that is less than one, are set to communicate with one of their neighbors. We start by setting \( f = 1\% \) and proceed in increments of 3\%, up to a maximum value of 100\%, which results in flooding the network. As shown in Figure 4, we obtain the most drastic reductions in the total time required to inform all of the nodes in the LCC of each graph early on, i.e., in the first 10\%. Therefore, we plot up to a maximum value of \( f = 10\% \) in the figures concerning relative fan-out below.

To assess how the fan-out parameter affects protocol performance, both in terms of total time and network load, as well as to determine which of the two aforementioned approaches is preferable, we present the following combined graphs: on the y-axis, we plot either the total time or the network load, on the x1-axis we plot the absolute communication fan-out values and on the x2-axis, the relative communication fan-out values. Figure 5 illustrates these combined plots for the TwitterLists, Slashdot2 and Facebook topologies. We observe that increasing the communication fan-out, regardless of whether it is absolute or relative, results in an exponential decrease in the total time required to inform the nodes in the LCC of each graph at the cost of an almost linear increase in the total network load. Moreover, it is evident that the relative approach yields much better results in terms of total time. With regards to total network load, however, it appears that the absolute case fairs better, although for small values, the difference is negligible. For instance, when comparing the case of \( f_{relative} = 1\% \) to \( f_{absolute} = 2\), the former achieves an additional percentage improvement of the total time by 4.68\%, 17.18\% and 32.07\%, with a percentage increase of the network load by -45.78\% (in this case we actually observed a decrease in the network load, hence the minus sign), 5.21\% and 29.98\% for the TwitterLists, Slashdot2 and Facebook topologies, respectively. Therefore, one can achieve substantial improvements in performance with a simple increase in the communication fan-out parameter which, if increased in moderation, does not significantly add load to the network.

**Enhanced Push & Pull Protocol.** We conclude this section by presenting the performance improvements that an enhanced push & pull protocol can provide. It combines all of the knowledge that we have acquired from our experiments and it has the following parameter values:

- A 1-item neighbor memory.
- A relative communication fan-out \( f_{relative} = 0.01 \).
- QPU as the neighbor selection policy.

We simulate two alternatives, one without any stopping criteria (Enhanced) and another one that uses the Coin Toss strategy (Enhanced Coin Toss). In Table 1, we illustrate the performance enhancements that both alternatives provide, compared to the vanilla version of the push & pull protocol. For the Enhanced Coin Toss protocol we also include the average percentage of informed nodes that it achieves.

The data in the table illustrate how an intelligent selection of protocol parameters can deliver tremendous performance enhancements.

![Figure 4: Effect of the percentage variance of the communication fan-out parameter f on the total time required to inform all of the nodes in the LCC of each graph.](image-url)
Table 7: Percentage improvement of the total time to inform all nodes that the Enhanced and Enhanced Coin Toss protocols achieve compared to the vanilla push & pull protocol. For the Enhanced Coin Toss, we also include the average percentage of informed nodes that it achieves.

| Network   | Enhanced | Enhanced Coin Toss | % Inf. | % Impr. |
|-----------|----------|--------------------|--------|---------|
| Slashdot1 | 93.72%   | 100%               | 92.81% |
| Slashdot2 | 93.61%   | 100%               | 92.86% |
| Slashdot3 | 93.16%   | 100%               | 92.69% |
| Epinions  | 98.71%   | 100%               | 98.65% |
| WikiSigned| 98.38%   | 100%               | 98.21% |
| Hamsterster| 76.33%  | 99.99%             | 66.50% |
| Brightkite| 96.98%   | 100%               | 93.34% |
| Facebook  | 90.56%   | 100%               | 89.64% |
| TwitterLists | 84.41%  | 99.99%             | 88.98% |
| Google+   | 99.08%   | 99.99%             | 98.67% |
| Average   | 92.49%   | 100%               | 91.23% |
| Std. Dev. | 7.23%    | 0%                 | 9.37%  |

Figure 5: Starting from left to right, the combined plots for the TwitterLists, Slashdot2 and Facebook topologies. The first row of figures illustrates how the absolute communication fan-out (x1-axis) and the relative communication fan-out (x2-axis) affect the protocol’s total time required to inform all of the nodes in the graphs’ LCCS (y-axis). The second row of figures illustrates how these parameters affect the protocol’s total network load (y-axis). The red and green curves correspond to the absolute and relative communication fan-out strategies.

6 Conclusions and Future Work

In this paper, we presented an in-depth experimental analysis of the asynchronous push & pull rumour spreading protocol. We studied the behaviour of the protocol over a large variety of real social network datasets (both common and signed). This is the first study that examines how multiple protocol parameters affect the protocol’s behaviour both in isolation and combination.

We illustrate how the convoy phenomenon, i.e., the inherent difficulty of the vanilla protocol to randomly select isolated nodes that lie at the edge of the network, can lead to extreme delays. However, when the goal is to inform large node percentages (e.g., 97%), the protocol manages to disseminate the rumour quickly. This can prove to be extremely helpful for applications such as voting and quorum-based systems.

Our measurements indicate that a popular rumour originator can provide an improvement as high as 30.14% in total dissemination time. However, biasing neighbor selection and favouring the more popular nodes does not necessarily yield improvements compared to the standard approach of choosing uniformly at random. The best alternative is our novel approach to cycle back and forth from popular to unpopular neighbors yielding an average improvement of 82.49%.

By leveraging the knowledge attained from our empirical study, we also proposed an enhanced protocol that combines a 1-item neighbor memory, our QPU as the neighbor selection policy and a relative communication fan-out \( f_{\text{Relative}} = 0.01 \). Our protocol delivers a 92.49%
average percentage improvement over the plain-vanilla version. We believe that our protocol provides strong evidence that rumours can indeed spread fast in real social networks.

Many and very diverse social-network-based applications can benefit from our findings. LOCKSS could build a reliable reputation system for peer introductions by having nodes gossip about the behaviour of other nodes in polls coupled with a sybilproof reputation mechanism. Tribler directly benefits since its peers already gossip as part of the “Buddycast” algorithm. Maze peers can gossip about friends’ status updates, finding other collectors with whom they share similar interests, etc. Lastly, Peer-SoN and Diaspora can use a gossip protocol to quickly communicate system information, an approach that has been successfully used for the Amazon S3.

Future work includes exploring protocol behaviour in cases where there are multiple and distinct rumour originators that generate content dynamically. Note that this is substantially different from the well-known and well-studied gossiping problem.

References

[1] Amazon s3 availability event: July 20, 2008. http://status.aws.amazon.com/a3-20080720.html July 2013.
[2] Brightkite dataset. http://konect.uni-koblenz.de/networks/loc-brightkite_edges July 2013.
[3] Diaspora project. https://wiki.diasporafoundation.org/Main_Page July 2013.
[4] Epinions dataset. http://snap.stanford.edu/data/soc-sign-epinions.html July 2013.
[5] Facebook dataset. http://konect.uni-koblenz.de/networks/facebook-wosn-links July 2013.
[6] Global ip network latency. http://ipnetwork.bgtmo.ip.att.net/pws/network_delay.html July 2013.
[7] Google+ dataset. http://konect.uni-koblenz.de/networks/ego-gplus July 2013.
[8] Hamsterster dataset. http://konect.uni-koblenz.de/networks/petster-friendships-hamster July 2013.
[9] Ip latency statistics. http://www.verizonenterprise.com/about/network/latency/ July 2013.
[10] Slashdot dataset no.1. http://snap.stanford.edu/data/soc-sign-Slashdot081106.html July 2013.
[11] Slashdot dataset no.1. http://snap.stanford.edu/data/soc-sign-Slashdot090221.html July 2013.
[12] Slashdot dataset no.2. http://snap.stanford.edu/data/soc-sign-Slashdot090216.html July 2013.
[13] Twitterlists dataset. http://konect.uni-koblenz.de/networks/ego-twitter July 2013.
[14] Wikisigned dataset. http://konect.uni-koblenz.de/networks/wikisigned-k2 July 2013.
[15] Akella, A., Seshan, S., and Shaikh, A. An empirical evaluation of wide-area internet bottlenecks. In IMC (2003).
[16] Birmann, K., Hayden, M., Ozkasap, O., Xiao, Z., Budi, M., and Minsky, Y. Bimodal multicast. In ACM TCS (1999).
[17] Boyd, S., Ghosh, A., Prabhakar, B., and Shah, D. Randomized gossip algorithms. In IEEE/ACM TON (2006).
[18] Buchegger, S., Schöberg, D., Vu, L.-H., and Datta, A. Peerson: P2p social networking: Early experiences and insights. In SNS (2009).
[19] Chen, H., Li, X., and Han, J. Maze: a social peer-to-peer network. In CEC (2004).
[20] Cheng, A., and Friedman, E. Sybilproof reputation mechanisms. In P2PECON (2005).
[21] Cherichetti, F., and Panconesi, S. L. A. Rumor spreading in social networks. In ICALP (2009).
[22] Cherichetti, F., and Panconesi, S. L. A. Almost tight bounds for rumour spreading with conductance. In TOC (2010).
[23] Cherichetti, F., and Panconesi, S. L. A. Rumour spreading and graph conductance. In SODA (2010).
[24] Chiuso, A., Fagnani, F., Schenato, L., and Zampieri, S. Gossip algorithms for distributed ranking. In ACC (2011).
[25] Demers, A., Greene, D., Hauser, C., Irish, W., Larson, J., Shenker, S., Sturgis, H., Swinehart, D., and Terry, D. Epidemic algorithms for replicated database maintenance. In PODC (1987).
[26] Doerr, B., Fouz, M., and Friedrich, T. Social networks spread rumors in sublogarithmic time. In STOC (2011).
[27] Doerr, B., Fouz, M., and Friedrich, T. Asynchronous rumor spreading in preferential attachment graphs. In SWAT (2012).
[28] Doerr, B., Fouz, M., and Friedrich, T. Experimental analysis of rumor spreading in social networks. In DAA. 2012.
[29] Doerr, B., Friedrich, T., and Sauerwald, T. Quasirandom rumor spreading. In SODA (2008).
[30] Eugster, P. T., Guerraoui, R., Kermarrec, A.-M., and Massoulié, L. Epidemic information dissemination in distributed systems. In Computer (2004).
[31] Fountoulakis, N., Panagiotou, K., and Sauerwald, T. Ultra-fast rumor spreading in social networks. In SODA (2012).
[32] Georgiou, C., Gilbert, S., Guerraoui, R., and Kowalski, R. D. On the complexity of asynchronous gossip. In PODC (2008).
[33] Giuli, T. J., and Baker, M. Narses: A scalable flow-based network simulator. In CORR (2002).
[34] Giuli, T. J., Maniatis, P., Baker, M., Rosenthal, D. S. H., and Roussopoulos, M. Attraction defenses for a peer-to-peer digital preservation system. In USENIX ATC (2005).
[35] Juurlink, B., Sibeyn, J. F., and Rau, P. S. Gossiping on meshes and tori. In IEEE TPDS (1996).
[36] Karp, R., Schindelhauer, C., Shenker, S., and Vocking, B. Randomized rumor spreading. In FOCS (2000).
[37] Kermarrec, A.-M., Massoulié, L., and Ganesh, A. J. Probabilistic reliable dissemination in large-scale systems. In IEEE TPDS (2003).
[38] Kumar, R., Novak, J., and Tomkins, A. Structure and evolution of online social networks. In KDD (2006).
[39] Maniatis, P., Rosenthal, D. S. H., Roussopoulos, M., Baker, M., Giuli, T., and Muliadi, Y. Preserving peer replicas by rate-limited sampled voting. In SOSP (2003).
[40] Pouwelse, J. A., Garbacki, P., Wang, J., Bakker, A., Yang, J., Iosup, A., Epema, D. H. J., Reinders, M., van Steen, M. R., and Sips, H. J. Triller: A social-based peer-to-peer system. In P2P (2008).
[41] Sibeyn, J. F. Faster gossiping on butterfly networks. In TCS (2005).