ChurnDetect: A Gossip-based Churn Estimator for Large-Scale Dynamic Networks

Andrei Pruteanu[1], Venkat Iyer[1], Stefan Dulman[1]

Embedded Software Group, Delft University of Technology, The Netherlands
{a.s.pruteanu,v.g.iyer,s.o.dulman}@tudelft.nl

Abstract. With the ever increasing scale of dynamic wireless networks (such as MANETs, WSNs, VANETs, etc.), there is a growing need for performing aggregate computations, such as online detection of network churn, via distributed, robust and scalable algorithms. In this paper we introduce the ChurnDetect algorithm, a novel solution to the distributed churn estimation problem. Our solution consists in a gossiping-based algorithm, which incorporates a periodic reset mechanism (introduced as DiffusionReset). The main difference with existing state-of-the-art is that ChurnDetect does not require nodes to advertise their departure from the network nor to detect neighbors leaving the network. In our solution, all the nodes are interacting with each other wirelessly, by using a gossip-alike approach, thus keeping the message complexity to a minimum. We only use easy accessible information (i.e., about new nodes joining the network) rather than presuming knowledge on nodes leaving the system since that is highly unfeasible for most distributed applications. We provide convergence proofs for ChurnDetect, and present a number of results based on simulations and implementation on our local testbed. We characterize the performance of the algorithm, showcasing its distributed light-weight characteristics. The analysis leads to the conclusion that ChurnDetect is an attractive alternative to existing work on online churn estimation for dynamic wireless networks.

1 Introduction

Recent technological advances have led to a tremendous increase in the number of embedded devices having processing and wireless communication capabilities. Large-scale networks of resource-limited devices are already in operation: wireless sensor networks (WSNs), mobile ad-hoc networks (MANETs) and vehicular networks (VANETs). Following this trend, current research projects show that significantly larger networks could be envisaged (e.g., programmable matter - claytronics [11], swarm robots [16], amorphous computing [1], etc.). Overall, devices tend to become smaller, networks increase in size and mobility becomes the basic assumption.

As such, there is a growing need for performing aggregate computations via distributed, robust and scalable algorithms. For example, the *online estimation* of network churn in dynamic scenarios is of crucial importance for a large number

of applications. While the churn rate can be computed offline from network traces, the online estimation of this quantity is a problem of increasing interest (e.g. for MANETs and WSNs). As we show in Section 1.1, the current state of the art presents unfortunately a quite limited set of alternatives. The particular aspects raising the difficulty of the problem are dynamic multihop architectures involving mobile nodes and failures at both node and communication levels.

A direct use-case for such an algorithm is the detection of traffic congestion on a highway. The potential imbalance between the inflow and the outflow of cars passing through a section of the road caused by an accident for instance, translates directly into a change of the churn level. Since our approach is fully distributed and convergences fast to a good estimate, we are able to prevent the drivers of a potentially hazardous situation faster than a centralized technique.

We approach the problem of estimating the network churn size by means of *diffusion algorithms* (also known as *gossiping*) [13]. This class of algorithms allows easy dissemination of information in a network, and has been already used to compute network aggregates such as averages, sums and aggregates, perform random sampling, compute quartiles, etc. (see [17]).

Inspired by real-world deployments of WSNs, where periodic resets of nodes are a known failure mode [3], we propose a new diffusion algorithm, by incorporating resets into a gossiping algorithm. We show that the new mechanism, called *DiffusionReset*, retains the properties of gossiping in terms of message complexity for achieving convergence exponentially fast. Although our algorithms are derived from gossiping algorithms sensitive to *mass conservation* [15, 17], our approach specifically exploits the property that total mass varies in a dynamic network. Our algorithm is able to track churn level evolution, even when it changes with time.

Based on *DiffusionReset*, we develop the *ChurnDetect* algorithm which we propose as a solution for the online estimation of the network churn rate (defined as the percentage of nodes that join/leave a network in a period of time). We are *not* assuming that the nodes advertise their departure from the network *nor* that nodes can detect when neighbors leave the network. In short, new nodes joining a network need to use a different reset value than "older" nodes. The results of the gossiping algorithm is an average aggregate value, available at all nodes, which can be used to compute the online churn estimate. To the best of our knowledge, this is the first work for the online estimation of churn that is addressing an arbitrary mobile multihop topology while still offering very good churn percentage estimates in a fully distributed manner.

We validate our work with both simulation and experiments on our wireless testbed. For the analysis of our algorithm we considered different mobility and network density scenarios that cannot be matched with corresponding traces from real deployments due to their scarcity, especially for mobile ad-hoc networks. The paper is structured as follows: Section 1.1 describes existing state-of-the-art. In Section 2 we introduce the underlying diffusion mechanism, while in Section 3 we present the network churn estimation algorithm. The algorithms are analyzed in Section 4 and we draw the conclusion in Section 5.

1.1 Related Work

Several definitions exist for the term *network churn*, most of them coming from the peer-to-peer context (see [10] for an overview). In the following, we denote as *churn* the changes in the set of networked nodes due to joins, graceful leaves, and failures. The churn is thus the percentage of the nodes in the network that changes during a given time period.

The problem of estimating churn in large-scale networks has been mostly studied in the context of Internet peer-to-peer systems. For these applications, the importance of knowing how many nodes enter and exit the system at any given time is fundamental. Due to the highly distributed nature of these systems, gossip-based protocols [15] have emerged as one of the most used techniques for the estimation of churn ratios. The main assumption these algorithms make is that *detection of nodes leaving a network* is feasible, either by advertisement or by nodes periodically checking their neighborhood. Unfortunately, for the case of a MANET or WSN, these approaches are not feasible due to the difficulty of discovering and maintaining neighborhood information within a finite amount of time and with a reduced energy budget.

The gossip-based algorithm presented in [12], was designed for estimating churn in arbitrary topologies (that can be represented as undirected graphs) built on top of a peer-to-peer network. The algorithm cannot be used in the presented form on a multihop wireless network, suffering from the shortcomings described above. Even if assuming node departure detection is feasible, the convergence of the algorithm is dependent on the network diameter, rapidly degrading when the network size exceeds a certain threshold.

Aside from gossip-based methods, there are algorithms that estimate the level of churn based on the amount of time a peer spends while being connected to the system (online time) or while being disconnected from the system (offline time) [4, 9]. While for some distributed applications (i.e., peer-to-peer - where clients are *not* behind firewalls) it is feasible to presume that they can signal their departures, or the peers can ping each other at regular time intervals, for networks where most users are behind firewalls (cannot be checked by others for availability) or have other restrictions (i.e., in our case: energy consumption and unavailability of a cheap ping function on a multihop topology) this assumption cannot be made. Furthermore, although these papers claim to provide a churn estimate, they actually focus on a slightly different definition for churn, and showcase the estimation of the online time and not specifically on the amount of nodes that constantly enter or exit the system. We acknowledge the difference in terminology with the peer-to-peer community, and notice that the results presented there are not directly applicable in our case.

A large amount of work is actually targeted at reducing the effects of churn (in all the above mentioned communities: P2P, MANETs and WSNs). Most of them assume existence of network traces, and estimate churn offline [8,10,18]. We believe that to be able to enable algorithms to adapt at run time to a dynamic environment, online estimation of churn is an important building block.

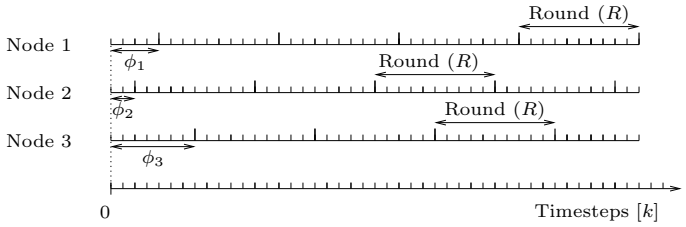

Fig. 1. Discrete time model for three nodes (ϕ_i - reset phase).

2 Diffusion Algorithms

We introduce the *DiffusionReset* algorithm under the same assumptions as used in [17] for the Push-Sum algorithm: i) communication operates in discrete time; ii) nodes do not need to have globally unique IDs (although at the lowest communication layer we need to be able *to distinguish* between neighbors); iii) the network does not become partitioned with time.

We make use of the discrete time assumption in order to simplify the description of the algorithm and ease the intuitive grasp of the concepts. The term *communication round* is being used in the sense described in [14] - it captures the fact that each node performs, in a given (large) time interval, an equal amount of actions. The beginning of rounds need not be synchronized (see Figure 1), in fact *DiffusionReset* is actually relying on this. Rounds are considered to be orders of magnitudes longer than the clock drift of the devices, thus, usual communication networks can be modeled as such. The last assumption (unpartitioned network or, equivalently, network as *a single* multihop cluster) should be interpreted from the perspective of large periods of time - it may be invalidated for the case of mobile scenarios for short moments (as in single nodes having no neighbors at a particular instance of time) but still holds for large time spans, so the assumption of the network not being disconnected does hold. Nevertheless, mobility actually helps by significantly accelerating the convergence of diffusion algorithms [19]. When introducing the *ChurnDetect* algorithm in Section 3, for the simulations we consider perfect radio communication between nodes. On the other hand, the usage of acknowledgments for messaging is *not* required. The effects of these two assumptions are addressed in Section 4.

Notations - $\mathcal{S}$ denotes the set of all n nodes in the network. The neighborhood of a node i, *including* the node itself is defined by $\mathcal{S}_i^+$ and has n_i nodes. We use i and j as node indexes, k to index time steps and r to index time rounds.

2.1 The DiffusionReset Algorithm

In this section we introduce the first contribution of this paper, the *DiffusionReset* algorithm (see Algorithm 1), which is the foundation of our solution to the churn estimation problem. We build upon a basic *diffusion algorithm* (lines 1–3 and 9–14 in Algorithm 1), adding the novel feature that each node periodically (albeit asynchronously), *resets its local variables to a default value* (i.e.,

Algorithm 1 *DiffusionReset*(μ, ϕ_i)

1: ▷ *state update step*
2: $m_i[k] \leftarrow \sum_{j \in \mathcal{S}_i^+[k-1]} \lambda_{j,i}[k-1] m_j[k-1]$
3: $\omega_i[k] \leftarrow \sum_{j \in \mathcal{S}_i^+[k-1]} \lambda_{j,i}[k-1] \omega_j[k-1]$
4: ▷ *reset step*
5: **if** rem $(k, R) == \phi_i$ **then**
6: $\{m_i[k]; \omega_i[k]\} \leftarrow \{\mu, 1\}$
7: Choose values $\lambda_{i,j}$
8: **end if**
9: ▷ *communication step*
10: **for all** neighbors j **do**
11: Send j: $\{\lambda_{i,j} m_i[k];\ \lambda_{i,j} \omega_i[k]\}$
12: **end for**
13: ▷ *return value*
14: $\{m_i[k];\ \omega_i[k]\}$

Algorithm 2 *ChurnDetect*(ϕ_i)

1: ▷ *initialization step*
2: **if** node i just entered the network **then**
3: Update phase: $\phi_i \leftarrow$ rem(k, R)
4: Reset mass value: $\mu_i \leftarrow 0$
5: **end if**
6: **if** node i inside network longer than R **then**
7: Reset mass value: $\mu_i \leftarrow 1$
8: **end if**
9: ▷ *diffusion step*
10: $\{m_i[k],\ \omega_i[k]\} \leftarrow$ *DiffusionReset* $(\mu_i,\ \phi_i)$
11: ▷ *return value*
12: $\frac{m_i[k]}{\omega_i[k]}$

the tuple $\{\mu; 1\}$ - lines 4–8 in Algorithm 1). The rationale for this mechanism is that we can model churn if a large number of nodes enter and exit the network constantly with time – actually, this is identical to having a number of nodes periodically reset (detailed in Section 3). The inspiration for this algorithm comes from a very common failure pattern met in real-world WSNs deployments – where nodes reset randomly [3] (see Figure 2 for the expected behavior).

Basic diffusion mechanism – *DiffusionReset* borrows parts of the *Push-Sum* and *Push-Vector*, introduced in [17] (lines 1–3 and 9–14 in Algorithm 1). In short, these work as follows: each node i holds a local state variable (given by the tuple of values $\{m_i[k]; \omega_i[k]\}$) at the beginning of the communication time step k (m_i is usually referred to as "mass"). During the time step, each node splits its local variable in several shares that get distributed to its neighbors. At the end of the time step, the node adds all the shares of received variables and updates to the new state value. The effect of this mechanism is that, with time, all local variables converge to the same value (the average of the original variable set) *regardless of the synchronization model* [17] (allowing us to relax the synchronous communication assumption).

Let i indicate the current node and j be the index of a neighbor $j \in \mathcal{S}_i^+[k]$. During each time step, node i defines a *share vector* $\mathbf{\Lambda}_i[k]$ of size $n_i[k]$, with elements corresponding to the share of local variables to be distributed to each neighbor. Let $\lambda_{i,j}[k]$ be the share assigned by node i to a neighbor j in time step k. The shares are chosen such that, at any time step k, $\sum_{j \in \mathcal{S}_i^+[k]} \lambda_{i,j}[k] = 1$ holds. During each time step k, each node i sends to all its neighbors a weighted vector: $\{\lambda_{i,j}[k] m_i[k]; \lambda_{i,j}[k] \omega_i[k]\}$ and receives the sets $\{\lambda_{j,i}[k] m_j[k]; \lambda_{j,i}[k] \omega_j[k]\}$ from its neighbors. At the time step $k+1$, the node updates its m_i (ω_i is updated similarly) value as follows: $m_i[k+1] = \sum_{j \in \mathcal{S}_i^+[k]} \lambda_{j,i}[k] m_j[k]$.

In matrix form, ($\mathbf{M}$ and $\mathbf{\Omega}$ being column vectors with the m_i and respectively ω_i elements), we have $\mathbf{M}[k+1] = \mathbf{\Lambda}^T \mathbf{M}[k]$, $\mathbf{\Omega}[k+1] = \mathbf{\Lambda}^T \mathbf{\Omega}[k]$. As shown

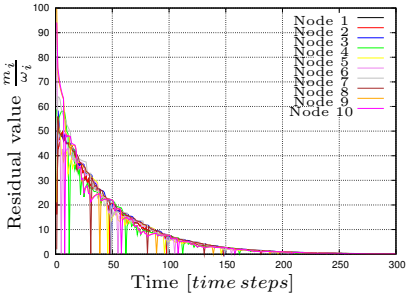

Fig. 2. *DiffusionReset* (200 nodes, tr. range 0.15 *units*, R(reset interval) = 50, $\mu = 0$).

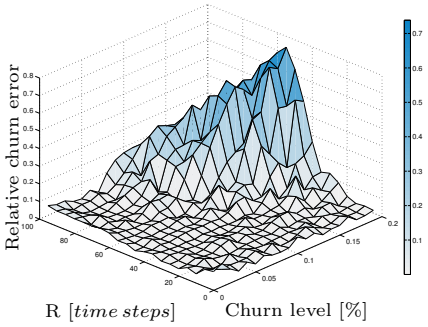

Fig. 3. Influence of reset period (500 nodes, tr. range 0.1 *units*).

in [17], if no errors occur and *the set of nodes remains the same*, the sums $\sum_{i\in\mathcal{S}} m_i[k]$ and $\sum_{i\in\mathcal{S}} \omega_i[k]$ remain constant over time. The usage of the share vector $\mathbf{\Lambda}_i[k]$ allows a great flexibility in the algorithm design: if all the elements in $\mathbf{\Lambda}_i[k]$ are zero, except for two entries (corresponding to i and a random neighbor j) equal to 0.5 each, the algorithm maps onto the classic definition for gossiping using unicasts. If all the entries in $\mathbf{\Lambda}_i[k]$ are taken to be $\frac{1}{n_i[k]}$ then we model a local broadcasting mechanism.

Reset mechanism – The reset mechanism (lines 4–8 in Algorithm 1) works as follows: every R time steps, a node resets its state value to $\{\mu; 1\}$. The reset phase of each node is ϕ_i (see Figure 1). Let $\delta[k]$ be the discrete Dirac function. The moment k when node i resets is signaled by $t_i[k] = 1$, where $t_i[k] = \delta\left[\text{rem}\left(k - \phi_i, R\right)\right]$ ($\text{rem}(a, b)$ gives the remainder of the division of a to b). Let $\mathbf{x}_i[k]$ be the local state variable on node i (i.e., the vector $[m_i[k], \omega_i[k]]$). The state transition can be written as

$$\mathbf{x}_i[k+1] = (1 - t_i[k]) \sum_{j\in\mathcal{S}_i^+} \lambda_{j,i}[k]\mathbf{x}_j[k] + t_i[k][\mu, 1]. \tag{1}$$

We define the vector $\mathbf{X} = [\mathbf{x}_1[k], \mathbf{x}_2[k], ..., \mathbf{x}_n[k]]^T$. Let $\mathbf{A}[k]$ be the adjacency matrix and $\mathbf{I}$ the identity matrix. We define the square matrix $\mathbf{\Delta}[k]$ with the terms $t_i[k]$ on its diagonal. Let $\mathbf{D}$ be a $n \times 2$ matrix with elements μ on the first column and 1 on the second column. The algorithm can be written in matrix form as $\mathbf{X}[k+1] = (\mathbf{I} - \mathbf{\Delta}[k])\,(\mathbf{I} + \mathbf{A}[k])\,\mathbf{\Lambda}^T[k]\mathbf{X}[k] + \mathbf{\Delta}[k]\mathbf{D}$, where the first term on the right side maps to the basic diffusion mechanism and the second term on the right side maps to the asynchronous resets.

2.2 Convergence of DiffusionReset

As shown by Dimakis et. al [19] mobility enables the construction of "short" routes between all pairs of agents, accelerating the diffusion process. If the entire network becomes mobile, the speed of information diffusion approaches the one of a fully connected network.

In our case, the influence of mobility and multihop topology is captured by $\mathbf{A}[k]$ and $\mathbf{\Lambda}[k]$ matrices - that change at each moment in time. Since a closed form solution for this type of equation is not available [5], we propose the following approach: we determined the convergence values and speed for the case of a fully connected network. Based on the results presented in [19], our convergence results will hold for a multihop mesh network *in which at least a small fraction of nodes is mobile*.

We assume that nodes use broadcast communication ($\lambda_{j,i} = \frac{1}{n}$). Each node resets after R time steps and the reset phase for each node is random and follows an uniform distribution. This results in an approximately constant number of nodes resetting in a time step. The mass value to which all nodes reset is equal to μ. The expected values are $E\left\{\sum_{i=1}^{n} t_i[k]\right\} = \frac{n}{R}$, $E\left\{\sum_{i=1}^{n} t_i[k]^2\right\} = \frac{n}{R}$. where we used the fact that $t_i[k]$ can be either 0 or 1, leading to $t_i[k] = t_i[k]^2$. Let $f = \left(1 - \frac{1}{R}\right)$. The error on each node is defined as $|m_i[k] - \mu|$. We can prove the following two lemmas:

Lemma 1 (Convergence of Mass for *DiffusionReset*). *With time, the total mass of the system converges to:* $\lim_{k\to\infty} M[k] = n\mu$.

Proof. The total mass in the network at time $k+1$ is: $M[k+1] = \sum_{i=1}^{n} m_i[k+1]$. From Equation 1,

$$M[k+1] = \sum_{i=1}^{n}(1 - t_i[k]) \sum_{j=1}^{n} \frac{m_j[k]}{n} + \frac{n\mu}{R} =$$

$$= \sum_{j=1}^{n} m_j[k] \left(1 - \frac{1}{n}\sum_{i=1}^{n} t_i[k]\right) + \frac{n\mu}{R} = M[k]f + \frac{n\mu}{R}$$

$$M[k] = M[0]f^k + \frac{n\mu}{R}\left(f^{k-1} + ... + f^0\right) = M[0]f^k + n\mu\left(1 - f^k\right) \quad (2)$$

As $f < 1$, we obtain $\lim_{k\to\infty} M[k] = n\mu$. □

Lemma 2 (Convergence Speed of *DiffusionReset*). *The overall error* $v[k] = \sum_{i=1}^{n}(m_i[k] - \mu)^2$ *decreases exponentially fast in the squared norm form:* $v[k+1] = v[k]f^2$.

Proof. $v[k+1] = \frac{1}{n^2}\left(M[k] - n\mu\right)^2 \sum_{i=1}^{n}\left(1 - t_i[k]\right)^2 = \frac{1}{n}\left(M[k] - n\mu\right)^2 f$

Using Equation 2 to expand $M[k]$ leads to $v[k+1] = \frac{1}{n}\left(M[0] - n\mu\right)^2 f^{2k+1}$, and then to $v[k+1] = v[k]f^2$. □

For a static multihop network, standard gossiping is very expensive in terms of message complexity, requiring $O(n^2 \log e^{-1})$ messages [6] to compute the average within accuracy e. When even a small fraction of nodes are mobile, the communication complexity drops significantly to $O(n \log e^{-1})$ messages, the same order as a fully connected graph [19], this being the basis of our reasoning. Our algorithm has the same message complexity as standard gossiping and while node mobility improves the convergence speed, *ChurnDetect* does not require

the nodes to be mobile. As Lemma 1 shows, the average of the distributed variable $\overline{M}_i = \frac{m_i[k]}{\omega_i[k]}$, will converge to $\{\mu; 1\}$ with time, regardless of the initial values $\{m_i[0], \omega_i[0]\}_{i \in \mathcal{S}}$. Intuitively we can think of it as if the network "forgets" the initial value exponentially fast - see Figure 2. This property extends also to disturbances in the network: if a node local values become arbitrary, the system will converge back to $\{\mu; 1\}$ exponentially fast.

3 Churn Detection Algorithm

In this section we introduce the *ChurnDetect* algorithm, as a solution to the problem of online network churn estimation – i.e., the percentage of nodes that are entering/leaving the network in a given amount of time (see Algorithm 2). The main idea is that a network comprises two sets of nodes: ones that "are fresh" (entered less than R time steps) and the ones that already "belong" to the network (entered more than R time steps ago). The periodic reset mechanism in *DiffusionReset* is used with one change: the "fresh" nodes reset to $\{0, 1\}$ and the "old" nodes reset to $\{1, 1\}$. The value to which the algorithm converges (available readily at each node) is a function of the churn rate - thus each node can estimate it directly. The novelty in our approach is the fact that nodes need not advertise leaving the system. The algorithm automatically tracks their departure through the change of the global shared variable M. This is a fundamentally different when compared to classic approaches. The elegance of our approach comes from the fact that we drop the assumption of nodes advertising leaving the network.

The intuitive explanation for why this works is the following: say that at each moment in time, a number n_{new} nodes enter the network (initialized with the value $\{0, 1\}$). At the same moment, a number of $n_{old} = n_{new}$ nodes leave the network taking with them the values $\{m_i, \omega_i\}$. This is equivalent to having *a network not changing its set of nodes*, but instead having *a subset of n_{old} nodes reset to $\{0, 1\}$ at each moment in time* (the subset needs not be the same at each moment in time). This equivalence maps churn directly onto *DiffusionReset* using two distinct reset values for the nodes. Please note that we are *not* assuming that the nodes advertise their departure from the network *nor* that nodes can detect when neighbors leave the network. The power of the algorithm is that only new nodes in the network are asked to behave slightly differently.

Lemma 3. *Convergence of* DiffusionReset *for various μ_i Assume that n_1 nodes reset to $\{\mu_1, 1\}$ and n_2 nodes reset to $\{\mu_2, 1\}$ ($n_1+n_2 = n$). Then $\lim_{k\to\infty} M[k] = n_1\mu_1 + n_2\mu_2$.*

Proof. We rewrite Equation 1 under the current assumptions. The formula is derived similarly to Lemma 1. □

Modeling-wise, the churn mechanism is equivalent to forcing each node to reset more than once per time round, to $\{0, 1\}$ instead of $\{1, 1\}$, the ratio being a function of the churn rate. Let ψ represent the churn rate, defined as the percentage of nodes entering/exiting the network at each *time step*. ψ can be

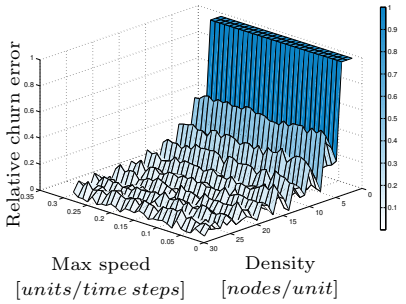

Fig. 4. Influence of density and speed. ChurnRatio = 0.1.

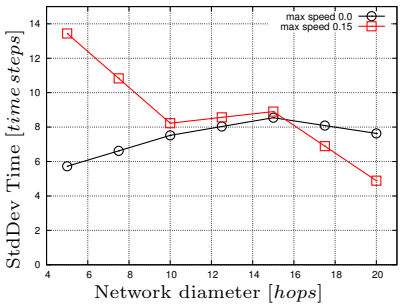

Fig. 5. Convergence speed. ChurnRatio = 0.1.

interpreted as well as the probability with which a node needs to reset to $\{0, 1\}$ at any given time step.

In *ChurnDetect* the nodes are forced to reset more often than once per period. The probability that a node *does not reset to* $\{0, 1\}$ for R consecutive time steps is: $P_{noreset} = (1 - \psi)^R$. In this case, $\overline{M}$ equals the ratio between the number of nodes resetting to 1, and the total number of nodes (resetting to either ($\{1, 1\}$ and $\{0, 1\}$ - see Lemma 3). From this definition, it follows actually that $\overline{M} = P_{noreset}$.This leads to each node being able to estimate ψ (making abstraction of the low-pass filter needed for a smooth estimate) as $\psi_e = 1 - \sqrt[R]{\overline{M}}$.

An important aspect of the algorithm is the selection of the reset period. Although synchronization is not needed, there is a dependency between the network dynamics (churn ratio) and the reset intervals.

As the shape of the graph in Figure 3 does not depend on the actual network size, nodes could check at run time if their reset period and the computed estimate are "in the safe zone", adapting otherwise. Due to size constraints this extension is not presented in this paper.

4 Analysis of ChurnDetect Algorithm

We base our evaluation by conducting simulations on Matlab. The mobile nodes are assumed to be deployed in a square space of 1 $units^2$. A circular disk communication model is assumed and the transmission range of the nodes is set to 0.1 *units*. The nodes move through space with a speed ranging from a minimum of 0.01 *units/time step* to a maximum of 0.2 *units/time step* (using the *Random Walk* [2] mobility model). Each experiment consisted of simulations running for 500 *time steps*. The maximum speed, the reset period and the number of nodes were varied across simulations to achieve different characteristics for mobility.

4.1 Experimental Evaluation via Simulations

Influence of Network Density and Mobility on Accuracy - As shown in Figure 4, the variation in network density from 5 to 25 nodes per squared

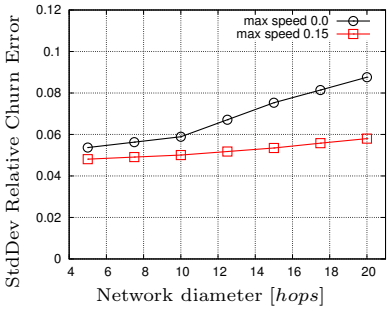

Fig. 6. Influence of network diameter. ChurnRatio = 0.1.

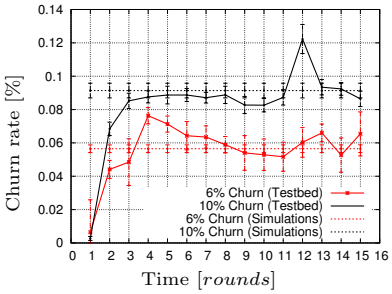

Fig. 7. Comparison between experiments and simulations.

unit affects the accuracy of the estimation as expected. This is in line with the generally agreed intuition that, for most distributed systems, increasing network density increases the information diffusion speed. On the other hand, as predicted in [19], even a small percentage of mobile nodes tremendously accelerates the information diffusion. In Figure 4 the difference between the cases with high and low mobility cannot be distinguished. One explanation for this graph is that the effects presented in these figures are dampened by the uniform spatial distribution of the churn nodes in the simulation. Nevertheless, Figure 4 confirms that an increased density decreases exponentially the relative churn error.

Influence of Network Diameter on Convergence Speed - One of the most interesting results is the influence of the network diameter and maximum node speed on the convergence time. As seen in Figure 5 where we showcase the standard deviation for the convergence time, the network diameter does show its role. The higher the diameter, the faster the algorithm converges to 90% estimation accuracy if the maximum nodes speed is non zero. Again, as predicted, having a percentage of node mobile, increases the convergence time even for the case of a high network diameter.

Influence of Network Diameter and Mobility on Accuracy - A trade-off exists between the network diameter and the speed of the nodes. While the former aspect is well understood [7], the latter is still a subject of active research [19]. Given a network diameter, if the mobility of the nodes is above a certain threshold, then the mobile multihop network becomes actually equivalent to a single hop network from the convergence speed perspective. Intuitively, a large network diameter increases the number of diffusion steps needed to spread the information, while node mobility helps the nodes "mix" faster, reducing the number of diffusion steps. The results confirm our hypothesis: as shown by the standard deviation in Figure 6, for the largest network diameter, the algorithm performs badly for the static case in comparison with the mobile one. The increased node speed accelerates the diffusion. For networks with low diameter, however, the increased speed of the nodes has relatively little effect. This confirms the results derived in Equation 1 - although derived for a fully connected graph, it holds for determining the churn ratio in a mobile case.

Communication Costs - A point of interest for a practical implementation of the *ChurnDetect* algorithm is the amount of communication that has to take place to ensure a good churn estimate. Although gossiping has a low message complexity, when the amount of nodes that constantly enter and exit the system increases, we have to reduce the reset intervals and thus linearly increase the amount of transmitted messages to fasten up the diffusion of information. Overall, the low complexity is maintained, making *ChurnDetect* an attractive distributed protocol.

4.2 Experimental Evaluation on the Testbed

In order to validate our results, we have implemented *ChurnDetect* on our wireless sensor network, consisting of 108 *GNode* nodes statically deployed across the floor of our department, using the TinyOS-2.x as the software platform. The *GNodes* are sensor nodes built around the MSP430-microcontroller combined with a Chipcon-CC1101 transceiver. Our experimental objective was two-fold; i. e., (1) To study the accuracy of the *ChurnDetect* algorithm on a real network, and (2) To recommend guidelines or implementation practices for protocols that operate on communication *rounds*. Figure 7 shows the comparison between the testbed runs (that had a network diameter of at most 3 communication hops) and simulation results. We tested *ChurnDetect* for varying values of churn, namely 6% and 10%. We implemented the network churn using a lookup table of nodeids, indexed by gossiping round number. Typically, for a gossiping round, every nodeid in the lookup table resets its value to 0. Each data point in the graph represents the average churn estimate over all nodes for a communication round. We note that apart from minor outliers, the testbed results are not only comparable with simulations, but also found to be within 10% of the actual churn value.

There was one notable issue we have had to face with real world experimentation, namely communication failures. Particularly, packet acknowledgment collisions early on within a gossiping round tends to increase the $\{m_i, \omega_i\}$ values on a node. This has an effect of an overestimated value of $\overline{M}$, which in consequence, underestimates the churn ratio. We alleviated this issue by allowing nodes to message randomly within a time interval (5 seconds in our experiments). We believe that in practice, every node should follow a probabilistic approach to messaging that adapts itself depending upon the number of channel contenders.

5 Conclusions

Online computation of network churn, in a distributed and reliable manner, is a common research interest for several communities such as MANET, WSN, VANET and peer-to-peer. In this paper, we introduced *ChurnDetect*, an algorithm for the online estimation of churn in dynamic networks. To the best of our knowledge, this is one of the first algorithms specifically targeted at multihop, mobile networks. The solution we proposed is derived from gossiping algorithms and incorporates the notion of periodic asynchronous resets for being able to

provide at each node an estimation of the churn percentage. We analyze the *ChurnDetect* algorithm and validate our contribution analytically, through simulations and experiments on a wireless sensor network. As future work, we plan to address some of the shortcomings that we encountered with respect to the implementation of the algorithm on the testbed platform.

References

1. Harold Abelson, Don Allen, Daniel Coore, Chris Hanson, George Homsy, Thomas F. Knight, Jr., Radhika Nagpal, Erik Rauch, Gerald Jay Sussman, and Ron Weiss. Amorphous computing. *Commun. ACM*, 43(5):74–82, 2000.
2. Christian Bettstetter. International workshop on modeling analysis and simulation of wireless and mobile systems. In *MSWIM 2001*, pages 19 – 27, 2001.
3. Jan Beutel, Kay Römer, Matthias Ringwald, and Matthias Woehrle. Deployment techniques for sensor networks. In Gianluigi Ferrari, editor, *Sensor Networks*, Signals and Communication Technology, pages 219–248. Springer, 2009.
4. A. Binzenhöfer and K. Leibnitz. Estimating churn in structured p2p networks. *Managing Traffic Performance in Converged Networks*, pages 630–641, 2007.
5. Stephen Boyd, Arpita Ghosh, Balaji Prabhakar, and Devavrat Shah. Randomized gossip algorithms. *IEEE/ACM Trans. Netw.*, 14(SI):2508–2530, 2006.
6. A.D.G. Dimakis, A.D. Sarwate, and M.J. Wainwright. Geographic gossip: Efficient averaging for sensor networks. *Signal Processing, IEEE Transactions on*, 56(3):1205–1216, 2008.
7. Alexandros G. Dimakis, Anand D. Sarwate, and Martin J. Wainwright. Geographic gossip: efficient aggregation for sensor networks. In *Proceedings of IPSN 2006*, pages 69–76, New York, NY, USA, 2006. ACM.
8. R. Friedman, D. Gavidia, L. Rodrigues, A.C. Viana, and S. Voulgaris. Gossiping on MANETs: the Beauty and the Beast. *ACM SIGOPS Operating Systems Review*, 41(5):67–74, 2007.
9. C. Giuffrida and S. Ortolani. A Gossip-based Churn Estimator for Large Dynamic Networks. In *Proceedings of ASCI 2010*, 2010.
10. P. Brighten Godfrey, Scott Shenker, and Ion Stoica. Minimizing churn in distributed systems. *SIGCOMM Comput. Commun. Rev.*, 36:147–158, August 2006.
11. Seth Copen Goldstein, Jason D. Campbell, and Todd C. Mowry. Programmable matter. *IEEE Computer*, 38(6):99–101, June 2005.
12. V. Gramoli, A.M. Kermarrec, and E. Le Merrer. Distributed churn measurement in arbitrary networks. In *Proceedings of the twenty-seventh ACM symposium on Principles of distributed computing*, page 431. ACM, 2008.
13. S. M. Hedetniemi, S. T. Hedetniemi, and Arthur L. Liestman. A survey of gossiping and broadcasting in communication networks. *Networks*, 18 (4):319–349, 1988.
14. K. Iwanicki and M. Van Steen. On hierarchical routing in wireless sensor networks. In *Proceedings of IPSN 2009*, pages 133–144. IEEE, 2009.
15. M. Jelasity, A. Montresor, and O. Babaoglu. Gossip-based aggregation in large dynamic networks. *ACM Trans. on Computer Systems*, 23(3):219–252, 2005.
16. M. Karpelson, Gu-Yeon Wei, and R.J. Wood. Milligram-scale high-voltage power electronics for piezoelectric microrobots. In *ICRA 2009*, 2009.
17. David Kempe, Alin Dobra, and Johannes Gehrke. Gossip-based computation of aggregate information. In *FOCS 2003*, 2003.
18. N.N. Qadri, M. Alhaisoni, and A. Liotta. Mesh based P2P streaming over MANETs. In *Proceedings of MOMM 2008*, pages 29–34. ACM, 2008.
19. A.D. Sarwate and A.G. Dimakis. The impact of mobility on gossip algorithms. In *INFOCOM 2009, IEEE*, pages 2088 –2096, 19-25 2009.