Using Tractable and Realistic Churn Models to Analyze Quiescence Behavior of Distributed Protocols *

Steven Y. Ko, Imranul Hoque, and Indranil Gupta
Dept. of Computer Science, University of Illinois at Urbana-Champaign
{sko, ihoque2, indy}@cs.uiuc.edu

Abstract

Large-scale distributed systems are subject to churn, i.e., continuous arrival, departure and failure of processes. Analysis of protocols under churn requires one to use churn models that are tractable (easy to apply), realistic (apply to deployment settings), and general (apply to many protocols and properties). In this paper, we propose two new churn models - called train and crowd - that together achieve these goals, for a broad class of stability properties called quiescent properties, and for arbitrary distributed protocols. We show (i) how analysis of protocol quiescence in the train model can be extended to the crowd model, (ii) how to apply the train and crowd model to several distributed membership protocols, (iii) how, even under real churn traces, the train and crowd models are reasonably good at predicting system-wide stability metrics for membership protocols.

1 Introduction

Today's distributed protocols (i.e., algorithms) have to withstand various patterns, degrees and rates of churn, when they are run in large-scale distributed systems such as peer-to-peer (p2p) systems, geographically distributed clusters (e.g., PlanetLab), the Grid, etc. [2, 8, 19]. Churn is defined as the continuous arrival, departure, and failure of processes from the distributed system. While one traditionally analyzes the fault-tolerance properties of a distributed protocol using a variety of failure models such as crash-recovery, crash-stop, etc. [15], several researchers have pointed out the need to explicitly define *churn models* [5, 10, 11, 13, 16] which assume neither a bounded number of failures, nor that the failures occur prior to protocol execution.

Churn models have three desirable characteristics: *tractability, realism, and generality*. Tractability means that the model is easy to handle and apply in analysis of a distributed protocol. Realism implies that the results from such an analysis can be generalized to arbitrary churn behavior in real deployed distributed systems. Generality implies that the model enables analysis of a broad class of self-stabilization properties for arbitrary distributed protocols.

We can classify existing work on churn models into two categories – node-based models and system-based models. Node-based models such as [7, 10, 11, 13] specify the behavior pattern for each process, typically via probability distributions for uptime and downtime for processes in the system. On the other hand, system-based models such as [5, 14, 16] specify a pattern for a *schedule* of processes joining, failing and leaving the system. The schedule typically specifies at what times and how many processes are churned, and optionally which processes are churned.

We focus only on system-based models in this paper. System-based churn models view churn in the distributed system as a holistic phenomenon, without modeling individual process behavior. Yet, they can be used to analyze emergent system-wide behavior arising out of individual process actions. In essence, any system-based model is equivalent to some node-based model, and vice-versa, i.e., each can be derived from the other. However, node-based models often have either low tractability, realism, or generality. For instance, exact uptime/downtime distributions derived from crawling real deployed distributed systems [2, 8, 19] are realistic, but cumbersome to apply in a mathematical analysis. Other node-based models use Poisson arrival and departure [10, 11, 13], making them less realistic. On the other hand, existing system-based models such as [5, 7, 14, 16] are typically not general, since they have been used to analyze only specific protocols, e.g., a DHT (p2p distributed hash table) in [14], replication [7], leader election in [5, 16], etc.

Our goals of tractability, realism, and generality motivate the four concrete contributions in this paper: (1) We present two new system-based churn models called *Train* and *Crowd* that are respectively tractable and realistic, where the crowd model is a generalization of the train model. (2) We prove how, for an important class of stability properties called *quiescent properties*, analysis using the tractable train model can be generalized to the more realistic model, for an arbitrary protocol under analysis. (3) We derive quiescence bounds for several membership protocols, including generic full membership protocols, generic DHTs, and gossip-based member-

*This work was supported in part by NSF CAREER grant CNS-0448246 and in part by NSF ITR grant CMS-0427089.

ship. (4) We use real churn traces from two deployed p2p systems: Overnet and Azureus – to verify that the train and crowd model are good at predicting quiescence behavior for gossip-based and DHT-based membership.

Intuitively, a quiescent property (defined in Section 4) is a global property that, at any given point of time, has a binary value for the entire distributed system, i.e., it is either true or false. Once true, the property continues to hold until churn happens, when it becomes false. In the absence of further churn, the quiescent property will begin to hold true at some of time again due to the actions of the protocol being analyzed. Thus, analysis of quiescent properties is invaluable in analyzing the correctness of the protocol's behavior under realistic churn settings. For instance, a distributed membership protocol would want to have all nodes know all the latest membership changes. Piergiovanni et al [17] discuss how to achieve connectivity during quiescent periods. In leader election, one desires a unique alive leader elected and known at all alive processes. These are all examples of quiescent properties that these protocols desire to achieve. Similarly, quiescent properties can be derived for protocols that seek to solve aggregation or consensus among alive processes.

The rest of the paper is organized as follows. Section 2 presents the system model. Section 3 presents the two new churn models. Section 4 defines quiescent properties and proves the relations between quiescence in the two different churn models. Section 5 uses the train model to analyze membership protocols. Section 6 shows simulation results with real churn traces and two real distributed membership systems. We elaborate on related work in Section 7, and conclude in Section 8.

2 System Model for Analysis

We analyze protocols that are intended to run in asynchronous message-passing distributed systems. However, we are interested in analyzing timing properties related to quiescence. Thus, it is inevitable for our analysis to assume synchronized clocks and bounded message delays. It is important to note that the analyzed protocol need not be synchronous; only the model for our analysis is synchronous. This means that our analysis also extends to asynchronous systems loosely synchronized via NTP.

Processes in our system have unique identifiers. We assume the crash-stop model of failure, which means that when a process fails or leaves, it can rejoin (if at all) only as a new process with a unique and new identifier. In addition, our mathematical analysis makes the following assumption:

Definition 2.1 - *System Size Invariant Condition:* For our analysis, we assume that the system size N, i.e., the total number of non-faulty processes, stays constant and stable at all times. Thus, in our churn models, for each process that leaves or fails at time instant t, a unique new process joins (or rejoins) the system at the same time instant t.

This condition is a part of our model and not assumed in the underlying system. The condition is needed to make our analysis tractable and is motivated by real trace data, as we explain below. Our experiments in Section 6 also show that our models work even under real traces where online population size varies over time. The above invariant condition and the successful experimental results, arise out of the fact that in p2p file sharing and streaming systems, the online population size does not vary much (in spite of churn) even over several weeks. For instance, the online population size varies by a factor of 2 per week and 3 per month for Overnet [2], 2 per day and per month in Gnutella [3, 20], and a factor of 9 per day and per week for PPLive [9]. Intuitively, our analysis with a value N holds for systems whose size varies around N.

3 A Family of Churn Models

This section presents the new system-based churn models Train and Crowd. As one goes from the train and to the crowd model, tractability decreases while realism increases.

A churn model specifies rules for generating a family of *schedules*. Each schedule consists of multiple *churn batches*. We define these terms below:

Definition 3.1 - *Churn Batch:* A churn batch is a group of processes joining, leaving and failing from the system simultaneously. A churn batch is associated with a *size* (say a). Such a churn batch consists of a unique process joins and a unique process failures or leaves, all occurring simultaneously. Notice that this satisfies the system size invariant condition (definition 2.1).

Definition 3.2 - *Schedule:* A schedule (or churn schedule) specifies a linear, possibly infinite, series of churn batches, giving the size and occurrence time for each batch.

A churn model thus has two components: (i) a timing component that decides when churn batches occur, and (ii) a choice component deciding which processes appear in churn batches. For the latter choice component, there are several possible options such as random, adversarial, benign, deterministic, etc. However, to maintain focus, we will assume the *random* choice component in the rest of this paper, unless otherwise mentioned[1]. Many of our results may apply to other choice components too, however a discussion of this is out of the scope of this paper.

The two models we define differ in their timing components as follows:

I. TRAIN MODEL: The train model allows churn batches to occur during protocol execution, but only at *periodic times* and in *fixed sizes*. Given a churn rate c, a train schedule is defined by a positive integer parameter denoted as K. Concretely: (1) processes are allowed to join and leave/fail at only time instances that are integer multiples of K time units, where $K \in R^+$, and (2) at each instance, $(c \times K)$ processes leave/fail from, and a new set of $(c \times K)$ processes join into,

[1] Our bounds analysis for generic full membership protocols in Section 5.1 uses the adversarial choice model.

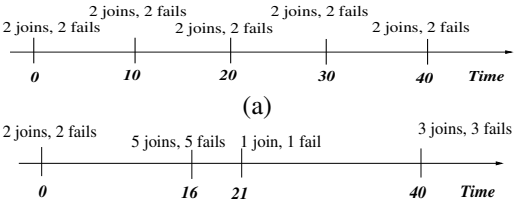

Fig. 1. Examples of (a) Train schedule and (b) Crowd schedule. Each of these two schedules has a churn rate of $\frac{8}{40} = 0.2$ processes per time unit.

the system, where $c \in R^+$. Notice that given c, one can only choose K so that $(c \times K) \in I^+$. We define the (long-term) *churn rate* of this train schedule as a $\frac{cK}{K} = c$ processes per time unit. Notice that this is half of the real churn rate, but we prefer the smaller value due to notational convenience. This model is illustrated with an example in Figure 1(a).

II. CROWD MODEL: The crowd model is a generalization of the train model. Here, churn batches are allowed to occur at *arbitrary times* and in *arbitrary sizes*. The churn rate of a crowd schedule is defined as $\frac{\sum_{\text{all batches}} a}{\sum_{\text{all batches}} t(a)}$, where $t(a)$ is the time between the arrival of the a batch, and its immediately consecutive batch. Figure 1(b) shows an example crowd schedule.

Notice that a train schedule is trivially a crowd schedule, but not the other way around. The crowd model is less tractable than train, but it creates a more realistic class of churn models. Examples of crowd models are the two α-churn models by Raynal et al [5, 16], and the half-life model [14].

Analyzing the behavior of a protocol in the train model is tractable since it involves considering only a single churn batch – the protocol's behavior between two consecutive batches can be generalized to the entire schedule. However, the arbitrariness of churn batch arrivals in the crowd model makes a similar analysis challenging. Motivated by this observation, the next section shows how to generalize conclusions from analysis using the train model, to the crowd model.

4 Analyzing Quiescent Properties under Train and Crowd Models

We consider an important class of stability properties called *quiescent* properties, defined for arbitrary distributed protocols (in Section 4.1). We then relate, for arbitrary protocols and quiescent properties, the behavior in the train and crowd models (in Sections 4.2 and 4.3). For ease of exposition and understanding, our discussion below uses a formal approach yet avoids formal temporal logic.

4.1 Quiescent Property, Reliable Quiescence, and Infinite Quiescence

Definition 4.1.1 - *Quiescent Property:* Let $prop$ be a global property for a distributed system. At any given point of time, $prop$ is either true or false in the system. $prop$ is usually defined w.r.t. a distributed protocol P, and typically expressed in terms of states of individual processes. Then, $prop$ is said to be a quiescent property if and only if (1) once $prop$ is true, it continues to hold until a churn batch occurs, and (2) if one considers a synchronous timing model, after occurrence of a churn batch that makes $prop$ false, the system takes a bounded amount of time for $prop$ to become true again. This time is a function of the churn batch size and protocol P.

Examples of quiescent properties include, in general, any liveness property that may be affected (i.e., made false) by churn in the system, but that may become true again after a while, due to actions by protocols running at individual processes. A useful and concrete example of such a property is *completeness in membership protocols*, i.e., "all non-faulty processes have been informed of all process joins and leaves in the system until now" (see Section 5). Quiescent properties can also be defined for classical distributed computing problems such as election, aggregation, consensus, system size estimation, etc., by considering the desirable liveness conditions for the problem at hand.

To analyze behavior of a given protocol P w.r.t. a given quiescent property, one needs to define a notion of *quiescence time* for the protocol and property. This is the only part of the analysis of P that uses the synchronous timing model. However, P itself is allowed to be an asynchronous protocol. The notion of quiescence time of a protocol (w.r.t. a property) is expressed as follows. Given a stable system size N, a protocol P can be associated with a function $f_P : I^+ \rightarrow R^+$, mapping the size a of a churn batch to the time taken thereafter for $prop$ to become true again. In other words, suppose a set of a processes joins and a processes leaves the system at time t, and the first time instant when $prop$ holds again is $(t + t')$; then $f_P(a) = t'$. We will assume that the function f_P is *smooth*, i.e., it is continuous and monotonically increasing. Notice that our notation for $f_P(.)$ omits $prop$ since we assume this to be implicit.

We are now ready to define *reliable quiescence* and *infinite quiescence*, two different metrics for analyzing the behavior of a protocol in a given churn schedule w.r.t. a given quiescent property.

Definition 4.1.2 - *Protocol Satisfying Reliable Quiescence (RQ) in a Churn Schedule:* [2] Given a quiescent property $prop$, a protocol P is said to "satisfy reliable quiescence (or ***RQ***) in a churn schedule C" if and only if for each churn batch in the schedule, there is a non-zero interval of time that starts sometime after the batch's arrival but before the immediately succeeding batch's arrival, during which $prop$ holds. Intuitively, a protocol satisfies RQ in a given churn schedule (w.r.t. a given property) if between every consecutive pair of churn batches, there is a non-zero time interval when quiescence

[2] Notice that this notion of quiescence is completely different from, and not intentionally related to, the notion of quiescence in failure detectors as defined by Chen et al in [1].

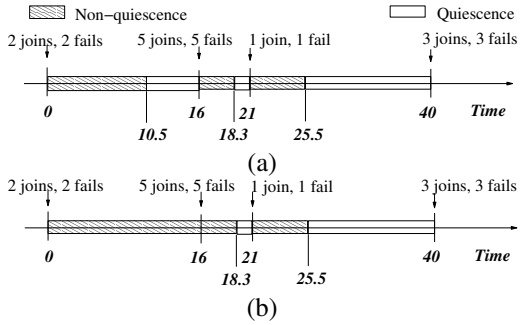

Fig. 2. Examples of (a) RQ and (b) IQ, both in the crowd schedule from Figure 1(b). We focus only on time $t = 0 - 40$. Notice that (a) also satisfies IQ, but (b) does not satisfy RQ.

holds, i.e., the quiescence property $prop$ is true. Another way of visualizing RQ is that every process that joins the system is guaranteed to see at least one non-zero interval of quiescence during its presence in the system. Figure 2(a) shows an example of RQ in a crowd schedule.

Definition 4.1.3 - *Protocol Satisfying Infinite Quiescence (IQ) in a Churn Schedule:* Given a quiescent property $prop$, a protocol P is said to "satisfy infinite quiescence (or *IQ*) in a churn schedule C" if and only if for each churn batch in the schedule, there is at least one non-zero interval of time that starts sometime after the batch's arrival, during which property $prop$ holds.

IQ is a weaker property than RQ since it does not guarantee that each process sees an interval of quiescence, only that each churn batch element is eventually incorporated into the system. Intuitively, a protocol satisfies IQ in a given churn schedule (w.r.t. a quiescent property) if all churn batches can be partitioned into finite-sized *sets of churn batches*, with each set containing a contiguous sequence of churn batches. For each set, there should be a non-zero time interval of quiescence after occurrence of the set's last batch but before occurrence of the following set's first batch. Figure 2(b) shows an example of IQ in a crowd schedule. Notice that there is no quiescence between the first two batches, but one right after both of them.

We now relate RQ and IQ for a given protocol and a given quiescent property.

Theorem 4.1.4: If a protocol P does not satisfy IQ in *any* train (or respectively crowd) schedule with churn rate $\geq c$, then it does not satisfy RQ in *any* train (or respectively crowd) model with churn rate $\geq c$.

Proof This is proved by observing the contrapositive - if P satisfies RQ in at least one train (or respectively crowd) schedule C with churn rate $\geq c$, then P satisfies IQ as well in C.

Next, we set up our discussion for the next few sections by classifying a given protocol P as either convex, concave, or linear, based on the nature of the quiescence time function $f_P(.)$:

Definition 4.1.5 - *Convex, Concave and Linear Protocols:* We call a protocol P *convex* (respectively *concave*, or *linear*) iff the function f_P is strictly convex (respectively strictly concave, or strictly linear).

In this paper, we are most interested in protocols that are either concave or linear. This is because quiescence time is minimized most effectively by such protocols. Consider a convex protocol P, i.e., $\forall a, b : f_P(a+b) > f_P(a) + f_P(b)$. Then the following modified protocol P' would quiesce much faster than P. For each batch of size $(a+b)$, P' prevents the batch b from being processed the system until the batch a has quiesced. The total time for P' to reach quiescence would be $f_P(a) + f_P(b)$, which is earlier than P's quiescence time of $f_P(a+b)$. Hence a protocol designer would prefer a non-convex protocol.

4.2 Relating Reliable Quiescence in the Train and Crowd Models

This section shows how conditions for a given protocol P to satisfy reliable quiescence (RQ) in the train model, can also be extended to the crowd model.

Theorem 4.2.1: Consider a distributed protocol P that is non-convex, i.e., function f_P is non-convex and smooth. If P does not satisfy RQ in *any* train schedule with given churn rate c, then P does not satisfy RQ in *any* crowd schedule with churn rate $\geq c$.

Proof The proof is by contradiction. Suppose P does not satisfy RQ in any train schedule with churn rate c, but satisfies RQ in a crowd schedule C' that has a churn rate $\geq c$.

No RQ in any train schedule C' implies that for any $K \in I^+$, for any churn batch of size $(c \cdot K)$, the time until the next batch, i.e., K time units, is insufficient for RQ to be achieved by P. That is, we have: $\forall K \in I^+ : K < f_P(c \cdot K)$. This is because all churn batches are similar to each other. Since f_P is smooth and non-convex, we write this as:

$$\forall a \in I^+ : \tfrac{a}{c} \leq f_P(a) \tag{1.1}$$

Now, consider that crowd schedule C' with churn rate $\geq c$, where P satisfies RQ. For each churn batch of size a in this schedule, suppose $t(a)$ is the time until the immediately next churn batch. Then, since RQ is satisfied:

$$t(a) > f_P(a) \tag{1.2}$$

But then, churn rate in the crowd schedule C' is:

$$= \frac{\sum_{\text{all batches}} a}{\sum_{\text{all batches}} t(a)} < \frac{\sum_{\text{all batches}} a}{\sum_{\text{all batches}} f_P(a)} \quad \text{(due to eq 1.2)}$$

$$\leq \frac{\sum_{\text{all batches}} a}{\sum_{\text{all batches}} \frac{a}{c}} = c \quad \text{(due to eq 1.1)}$$

However, this implies that the crowd schedule's churn rate $< c$, a contradiction. Hence proved.

The above theorem showed how to generalize *non-satisfiability* of RQ from the train model to the crowd model. The following observation relates the *satisfiability* across these two models:

Observation 4.2.2: Suppose there is *at least one* train schedule C (i.e., with a given K) with given churn rate c, where P satisfies RQ. Then (trivially) there is *some* crowd schedule with churn rate c that also satisfies RQ. (The train schedule C itself is a witness crowd schedule where P satisfies RQ.)

A stronger result than the above is difficult because of the following difficulty with the crowd model:

Observation 4.2.3: Given a crowd schedule C', with churn rate c, where P satisfies RQ, there is a different crowd schedule C'' with the same churn rate c, where P *does not* satisfy RQ.

The schedule C'' can be derived from C' by changing the arrival times of just one consecutive pair of batches in C' so that the first batch does not quiesce, while the second batch still does. In other words, consider any pair of consecutive batches of sizes a and b in the crowd schedule. Construct C'' from C' by changing the $t(.)$ values for only these two batches to $t_{new}(.)$ with:
$t_{new}(a) = f_P(a) - \epsilon$, and $t_{new}(b) = t(b) + t(a) - t_{new}(a)$
In C'', the a batch does not quiesce and thus RQ is not satisfied, yet the churn rate remains c.

4.3 Relating Infinite Quiescence in the Train and Crowd Models

In this section, we study the reducibility between the train and crowd schedules w.r.t. the infinite quiescence (IQ) property defined in Section 4.1.

Unlike RQ, for IQ, we need to define a separate quiescence time function that takes into account the *sets* of churn batches that share a unique time interval of quiescence after their occurrence. Concretely, given a protocol P and a stable system size N, in the synchronous system model, one can define a function $g_P : (I^+)^m \rightarrow R^+$, where m is a finite, but variable, positive integer. g_P takes as input a (finite) set of churn batches (each of positive size), and outputs the *minimum* time that protocol P takes to reach quiescence after this set of batches is started to be introduced into the system. Notice that the minimum is taken across all possible interleavings and times of entry for each of the batches in the set.

For instance, for two batches of sizes a and b respectively, $g_P(a, b)$ represents the minimum time taken to reach quiescence, given that one is free to choose both (1) which of the a batch or the b batch is introduced first, and (2) the time when the second of these batches is introduced. Note that the time to quiescence will be counted from the time that the first batch of the two is introduced, and until the first moment of quiescence after both batches have been introduced.

In general, the function g_P is related to the function f_P for the same protocol. Specifically, for analysis, we assume there exists a constant $l_P \in (0, 1]$ such that, for any positive integer m and any numbers $a_1, a_2, \ldots a_m \in I^+$, we have:
$g_P(a_1, a_2, \ldots a_m) \geq l_P.(\sum_{i=1}^{i=m} f_P(a_i))$

Theorem 4.3.1: Consider a distributed protocol P for which f_P is non-convex and smooth, and g_P and l_P are defined as above. If P does not satisfy IQ in *any* train schedule with given churn rate c (for any given value of c), then P does not satisfy IQ in *any* crowd schedule with churn rate $\geq (\frac{c}{l_P})$.

Proof The proof works by contradiction. Suppose P does not satisfy IQ in any train schedule with churn rate c, but satisfies IQ in a crowd schedule C' that has churn rate $\geq \frac{c}{l_P}$.

From Theorem 4.1.4, absence of IQ satisfaction in any train schedule with churn rate c also implies non-satisfaction of RQ by P in any train schedule with churn rate c. Thus, similar to the proof of Theorem 4.2.1, we can write: $\forall K \in I^+ : K < f_P(c \cdot K)$. Thus: $\forall a \in I^+ : \frac{a}{c} \leq f_P(a)$ (2.1)

Now, in the crowd schedule C', consider a set of batches $A =< a_1, \ldots a_m >$ (m finite, a_i's are consecutive batches) that has no interval of quiescence between occurrences of a_1 and a_m, but does have a non-zero interval of quiescence between occurrences of a_m, and a_m's immediately succeeding batch. Let $t(A) = \sum_{i=1}^{m} t(a_i)$. Then, since A has a interval of quiescence at its end, we have:
$t(A) > g_P(A)$, which is $\geq l_P \cdot \sum_{i=1}^{j} f_P(a_i)$ (2.2)
This leads us to calculate the churn rate in the crowd schedule C', where P satisfies IQ, as:
$= \frac{\sum_{\text{all batches}} a}{\sum_{\text{all batches}} t(a)} < \frac{\sum_{\text{all batches}} a}{\sum_{\text{all batches}} l_P \cdot f_P(a)}$ (due to eq 2.2)
$\leq \frac{1}{l_P} \cdot \frac{\sum_{\text{all batches}} a}{\sum_{\text{all batches}} \frac{a}{c}} = \frac{c}{l_P}$ (due to eq 2.1)
This is a contradiction, since it implies the crowd schedule C' has a churn rate $< \frac{c}{l_P}$.

Finally, we make the following observation for relating satisfiability in the train and crowd models:

Observation 4.3.2: Suppose there is *at least one* train schedule C (i.e., with a given K) with given churn rate c, where P satisfies IQ. Then (trivially) there is *at least one* crowd schedule with churn rate c that also satisfies IQ. (The schedule C itself is a witness crowd schedule where P satisfies IQ.)

5 Bounds for Quiescence of Membership Protocols in Train Model

In this section, we demonstrate the applicability of the train model to real distributed protocols. Specifically, we study necessary conditions for quiescence in three distributed membership protocols: full membership, DHT-based, and gossip-based. A distributed membership protocol [4, 14, 21, 22] maintains, at each non-faulty process, a list (possibly partial) of other processes also currently in the system. The protocol actions update membership lists at non-faulty processes with information about processes that have joined, left or failed from the system. Membership protocols are used in p2p systems [14] and other large-scale distributed systems [4, 21, 22].

A *full membership protocol* is one that attempts to maintain, at each process, a list of *all* other processes currently in the system. This requires all membership changes to be communicated to *all* non-faulty processes. A membership protocol that is not full is said to be *partial*. Protocols [4, 21] are

full protocols, while DHTs [14] and [22] are partial.

Broadly, the quiescent property of interest for any such membership protocol is as follows: "all non-faulty processes have a correct membership list reflecting all churn batches that have occurred so far". This may be either a deterministic or probabilistic property, and is elaborated for each individual protocol below.

To standardize our analysis, we will assume an upper bandwidth bound of 1 message per time unit at each process, and restrict each message to carry at most one process identifier in its contents (besides sender and destination identifiers). The choice of homogeneous bandwidth in our model is for ease of analysis. Extension of our results to models with heterogeneous bandwidth bound is beyond the scope of this paper. In addition, we will also ignore the detection time for process failures; an extension of our analysis with failure detection times is straightforward.

5.1 Bounds for Arbitrary Full Membership Protocols under Adversarial Choice

In this section, we analyze, under the train model, arbitrary full membership protocols which aim to have all processes be informed of all membership changes. We assume that the choice of processes in each churn batch is *adversarial*. This is a worst-case scenario - per churn batch, we assume that a single unique non-faulty process (labeled as p) is responsible for initiating all membership updates for the churn batch. We first analyze RQ:

Theorem 5.1.1: Given adversarial choice in each churn batch, and a full membership protocol that is non-convex and smooth, it will not satisfy RQ in the train model if the churn rate is $\geq \frac{1}{2}$ processes per time unit.

Proof Consider a train schedule with churn rate c and parameter K. Given a churn batch of size $(c \cdot K)$, the optimal time to disseminate the $(2 \cdot c \cdot K)$ entries (joins and leaves/failures) to all N processes is given by [6]: $f_P(c{\cdot}K) = (\lceil log_2(N) \rceil + 2{\cdot}c{\cdot}K - 1)$ time units. This is because in the adversarial model, the minimal time for disseminating a batch's updates to all other N processes (including to new processes, which initialize their membership lists from p) is via a spanning tree rooted at p, with the $(2 \cdot c \cdot K)$ updates pipelined via this tree. Thus, for RQ to be satisfied:

$(\lceil log_2(N) \rceil + 2 \cdot c \cdot K - 1) \leq K$

$\Rightarrow \frac{1}{2} \cdot (\lceil log_2(N) \rceil - 1) \leq K \cdot (\frac{1}{2} - c)$

Hence, if $c \geq \frac{1}{2}$, then for any given $N > 1$, there exists no K satisfying the above equation.

Notice that the above bound may not be tight, i.e., it is still possible that a given membership protocol does not satisfy RQ even in a train schedule with churn rate $c < \frac{1}{2}$.

Since full membership lists are updated via probabilistic protocols such as gossiping [4, 21], it is not enough to analyze only RQ, but also the *completeness* of individual membership entries. This leads us to define and analyze a new notion called *fractional inquiescence* for a protocol:

Definition 5.1.2 - *Fractional Inquiescence:* Consider a full membership protocol P that satisfies RQ in a given train schedule. Then the fractional inquiescence of P w.r.t. that schedule, is denoted as $FI(P)$, and is defined as the fraction of time that, a random non-faulty process p *does not know about* a random membership update u (i.e., join, leave or failure) in a churn batch B, considering only time instances between the occurrence of B and of its immediately succeeding batch.

Theorem 5.1.3: Given adversarial choice in each churn batch, in any train schedule with churn rate c and parameter K, the fractional inquiescence $FI(P)$ value of any non-convex, smooth and full membership protocol P is bounded from below by $(\frac{\lceil log_2(N) \rceil + 2cK - 1}{N \cdot K})$.

Proof We use the spanning tree from Theorem 5.1.1's proof for optimal pipelining of updates in a churn batch. With a bandwidth bound per process of 1 message per time unit, the i^{th} of these elements ($i = 1$ to $2cK$) is received by all processes by time [3]$(\lceil log_2(N) \rceil + i - 1)$. Since the i^{th} element starts spreading at time $(i-1)$ and the number of processes knowing it doubles every time instant thereafter, the expected time for a random process to receive it is: $(\frac{\lceil log_2(N) \rceil}{2} + i - 1)$ time units. Per churn batch, there are $(2 \cdot c \cdot K)$ updates intended for dissemination within K time units to N processes. Thus, denoting $X = \frac{(\lceil log_2(N) \rceil}{2}$, we have: $FI(P) \geq \frac{X + (X+1) + ... + (X+2cK-1)}{(2 \cdot c \cdot K) \cdot K \cdot N} = \frac{\lceil log_2(N) \rceil + 2cK - 1}{N \cdot K}$.

5.2 Bounds for Two Membership Protocols in Literature under Random Choice

In this section, we present our bounds analysis for RQ and FI in the train model for two systems - a generic DHT, e.g., [14], and a gossip-style membership protocol [21]. We consider only a random choice component in our churn models[4]. Although we have applied our analysis to two other protocols, Cyclon [22] and SWIM [4], we exclude these for brevity.

Bound for Distributed Hash Tables (DHTs): Consider a generic DHT where each process has d neighbors, and the routing length (in number of hops) between a random pair of processes is RL. The quiescent property (w.r.t. a churn batch) we are interested in is: "All neighbors of each non-faulty process are updated correctly, according to the DHT rules." Note that a DHT is a partial membership protocol.

In the train model, a churn batch of $(c \cdot K)$ will need to update $2 \times d \times (c \cdot K)$ other membership entries of non-faulty processes across the entire distributed system. In addition, each of the $(c \cdot K)$ incoming processes will need to hear of d neighbors. Each of the above updates consumes RL messages. However, due to our per-process bandwidth constraint

[3] All times are relative to the instant of occurrence of the churn batch.

[4] Unfortunately adversarial choice models appear to make the analysis intractable even for the simplest of partial membership protocols.

of 1 message per time unit, the maximum capacity of bandwidth available across the entire system between this churn batch and the next, is only $(N \times K)$ messages. Thus, the *necessary condition* for a DHT to satisfy RQ is that the number of messages is smaller than the capacity:
$(2 \times d \times (c \cdot K) + d \times (c \cdot K)) \times RL \leq (N \times K) \Rightarrow c \leq \frac{N}{3 \cdot RL \cdot d}$

This quantity $(\frac{N}{3 \cdot RL \cdot d})$ is thus the upper bound on the churn rate required for *any* DHT to satisfy RQ in *any* train schedule. Further, this bound also applies to *any crowd schedule* because DHTs are non-convex and smooth, and due to Theorem 4.2.1. DHTs are non-convex since for any churn batch of size $(a + b)$, the DHT can schedule the dissemination of the first a entries completely, and only then allow dissemination of the second batch of b entries. Thus, $f_{DHT}(a + b) \leq f_{DHT}(a) + f_{DHT}(b)$. Hence, $f_{DHT}(.)$ is non-convex.

Although our derivation did not assume a specific DHT, the above bound appears to tally with proven existing results for the Chord DHT [14]. For Chord, $d = O(log(N))$, and $RL = O(log(N))$, and Theorem 5.7 in [14] shows that the maximal rate of churn that Chord can tolerate, in order to remain "stable" and answer queries correctly w.h.p., is $O(\frac{N}{log^2(N)})$. This is asymptotically equivalent to $O(\frac{N}{3 \cdot RL \cdot d})$.

Bound for Gossip-style Membership Protocols: A gossip-style membership protocol [21] is a full membership protocol where each process attempts to maintain a complete list of processes currently in the system. In addition, each process gossips continuously about its membership list by sending out, at a rate of 1 entry per time unit, random elements of its membership list to target processes in the system. Since target selection mechanisms are typically probabilistic, the quiescent property of interest should also be probabilistic: (for any given churn batch) "a high probability fraction $(1 - \epsilon)$ of non-faulty processes in the system is informed about updates in the latest churn batch". The authors show in [21] that the time for a churn batch of size $(c \cdot K)$ to quiesce is: $f_{gossip-membership}(c \cdot K) = 2 \times c \cdot K \times N \cdot l \cdot log(N)$, where l is a constant that can be derived based on the value of ϵ, as well as the actual target selection mechanism used [21].

Thus, the *necessary condition* for gossip-style membership to satisfy RQ is: $2 \cdot c \cdot K \cdot N \cdot l \cdot log(N) \geq K \Rightarrow c \geq \frac{1}{2 \cdot l \cdot N \cdot log(N)}$
This quantity $(\frac{1}{2 \cdot l \cdot N \cdot log(N)})$ is thus an upper bound on the churn rate required by the gossip-style membership protocol of [21] to satisfy RQ in *any* train schedule. This bound also applies to *any* crowd schedule, since $f_{gossip-membership}(.)$ is linear and smooth, and due to Theorem 4.2.1. This churn rate bound is asymptotically smaller than a DHT's churn rate bound of $(\frac{N}{3 \cdot RL \cdot d})$, since gossip-style is a full membership protocol, while DHTs are partial.

Finally, we calculate fractional inquiescence $FI(gossip - membership)$. If the expected time for a (single) gossip to reach a random process, among N given processes, is $\frac{log(N)}{M}$, where M is a constant > 1, derived from the actual target selection mechanism used [21], then we can write: $FI(gossip - membership) = \frac{l \cdot N \cdot \frac{log(N)}{M} \cdot 2cK}{(2 \cdot c \cdot K) \cdot K \cdot N} = \frac{l \cdot log(N)}{M \cdot K}$. Notice that the above $FI(gossip - membership)$ is independent of churn rate c. It turns out that the SWIM membership protocol [4] shares the same RQ bound and FI values as the gossip-style membership. We exclude these for brevity.

6 Experimental Results

Our experimental methodology has three characteristics. First, we relax the assumption of the system size invariant condition (definition 2.1), so that the total number of nodes in the system can vary over time. Second, we use real churn traces from two deployed p2p systems in order to inject churn workloads into our simulations. The two real churn traces are from the Overnet p2p system (collected by Bhagwan et al [2]), and from the Azureus p2p system (collected by Ledlie et al [12]). These churn traces have heterogeneous availability variations across nodes and across time, and also have varying number of online nodes over time.

Third, we evaluate quiescent properties for two practical membership protocols in the literature - the gossip-based membership protocol [21], and the Pastry DHT [18]. These are two different classes of membership protocols that have been widely used to develop many p2p applications, e.g., aggregation, publish-subscribe etc. We implemented a variant of the gossip-based membership protocol from [21], and used the FreePastry implementation by the authors of [18].

While the paper has so far looked at reliable quiescence (RQ) and fractional inquiescence (FI), our main goal in this section is to study key quiescence metrics of the above two systems, and observe how closely the train and crowd models predict them. Our studies find that several quiescence properties related to RQ and FI are predicted reasonably well by the train and crowd models.

We first describe the churn traces from Overnet and Azureus in Section 6.1, then study gossip-based membership in Section 6.2, and Pastry in Section 6.3.

6.1 Traces

Overnet Trace: We use the Overnet trace collected by Bhagwan et al [2]. This trace was collected over a period of 7 days by probing 2400 hosts every 20 minutes to measure the availability of hosts in the Overnet filesharing network. The churn rate of the Overnet trace is 3.7×10^{-5}/node/second. The average number of online nodes is 455.978.

Smoothened Overnet Trace: We modify the 20-minute Overnet trace so that it can be injected continuously. More concretely, let the *i*th Overnet trace be collected at time t_i. The hosts that join and leave in the *i*th trace must have joined and left the system over the period $(t_{i-1}, t_i]$. So, for each of the joins and leaves at t_i, we inject it at a time instant selected uniformly at random from the interval $(t_{i-1}, t_i]$.

Azureus Trace: We use the Azureus trace collected by Ledlie et al. where hosts were probed at a median rate of 248 seconds [12]. We use a subset (2400) of the 9849 host traces for our simulation. This is a higher probing frequency than the Overnet traces, and thus this trace is more realistic than the smoothened Overnet traces. The churn rate of the Azureus trace is 9.8×10^{-6}/node/second. The average number of online nodes is 301.11. Compared to the Overnet trace, the Azureus trace has a lower churn rate and a lower average number of online nodes.

Train and Crowd Models: For each churn trace above, in order to have a baseline for comparison, our train and crowd models are fixed to have the the same long term churn rates as the real trace, as well as the same number of initial nodes at the beginning of simulation (i.e., N in our models is set to the initial system size in the trace). For train, we set the inter-batch arrival time to 20 minutes. For crowd, the inter-batch arrival time follows a Poisson distribution with an average of 20 minutes; each batch size is scaled up proportional to the latest inter-batch arrival time. In models for the Overnet trace comparison, the average number of online nodes is 520.0 and 506.0 for train and crowd model, respectively. In traces for the Azureus trace comparison, the average number of online nodes is 292.0 and 293.0 for train and crowd model, respectively.

6.2 Gossip-based Membership Protocol

Methodology: Our implementation of the gossip style membership protocol from [21] works as follows. Each node maintains a membership list locally, containing a list of all nodes it believes are alive. Each entry in this list is associated with an integer-valued heartbeat counter. Each node runs the following steps asynchronously. Every T_{gossip} (= 2) seconds, a node: (i) increments its own heartbeat counter, (ii) chooses a target node at random from its own membership list, and (iii) sends a gossip message to the target node, containing both its heartbeat counter and its entire local membership list. A node receiving a gossip message uses the received list to both add new entries to its local list, and to update heartbeat counters for each local list entry (as a max of received and known heartbeat counters for the entry).

An entry is dropped (marked as failed) when its heartbeat counter has not changed in the past T_{fail} (= 22) seconds. However, the entry is retained for another T_{fail} seconds in order to avoid it from being added again to the membership list if it were to be received again via a gossip message. The detection time of this protocol turns out to be T_{fail}, since after a failure, it takes that much time for all entries pointing to the failed node to be deleted everywhere.

Now, membership lists may be incorrect in two ways: (1) an entry in the list may be *inaccurate*, i.e., it points to a node that has failed; (2) the list may be *incomplete*, i.e., it does not contain entries for at least one node that is currently alive. Thus, the quiescent property we study is *completeness and accuracy of membership lists*. Specifically, we study two metrics related to RQ and FI: (a) entry-level inquiescence metric: similar to FI, this is the fraction of entries that are either inaccurate or incomplete, across all membership lists in the system; (b) system-level quiescence metric: similar to RQ, this is the fraction of time that all nodes have correct membership lists. Our plots show the average value of these metrics, with each data point derived from 5 runs with different seeds.

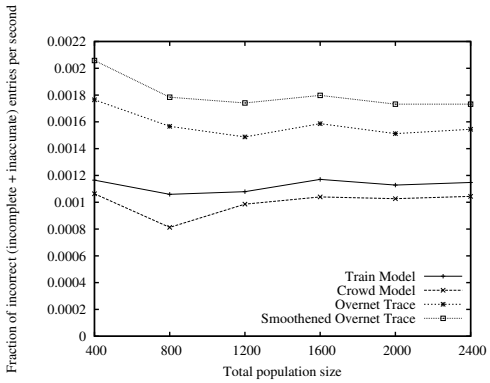

Fig. 3. Entry-level metric: Overnet trace

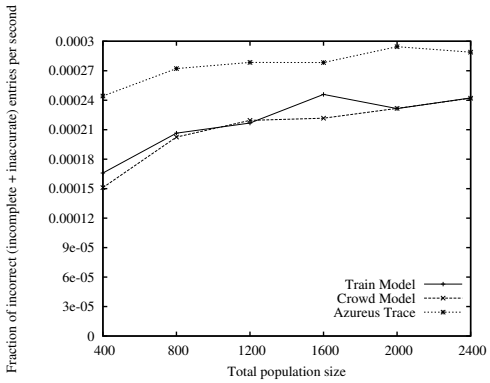

Fig. 4. Entry-level metric: Azureus trace

Results: Figures 3 and 4 show the entry-level inquiescence metric for the above protocol, for the Overnet and Azureus traces respectively. The plots show variation of the metric across different node population sizes. Each node population size x corresponds to the total number of nodes that are either offline or online; this set was selected by injecting traces for the first x nodes encountered in the churn traces.

We observe that the train and crowd model predict the average value of this metric within an error factor of 2 for the Overnet trace, and within an error factor of 1.5 for the Azureus trace. The higher error in the Overnet traces is because of the varying size of churn batches in the trace, as opposed to our train model which has fixed size batches. On the other hand, the error in the smoothened Overnet traces is because nodes are continuously joining and leaving (rather than in batches), thus giving the system little time to stabilize be-

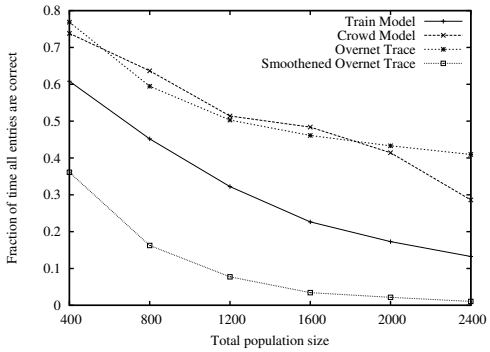

Fig. 5. System-level metric: Overnet trace

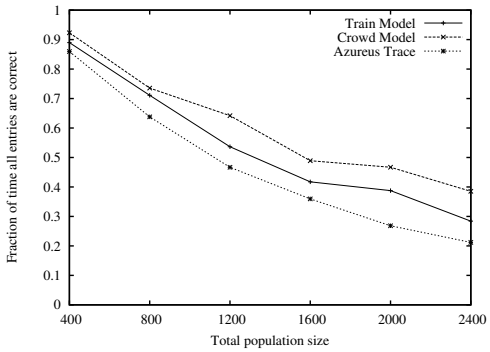

Fig. 6. System-level metric: Azureus trace

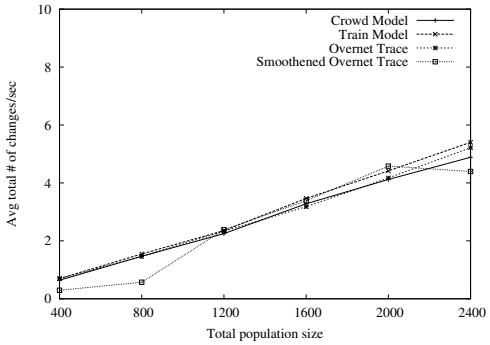

Fig. 7. Routing table changes: Overnet trace

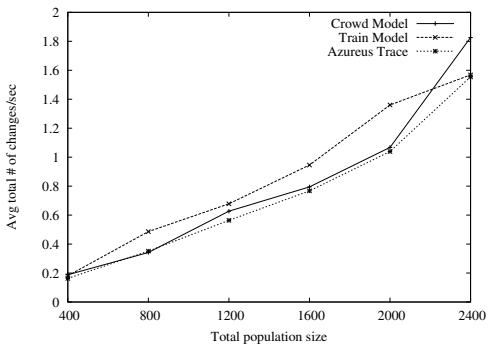

Fig. 8. Routing table changes: Azureus trace

fore the next churn event. For the Azureus trace, which accurately contains the time of each node join and leave, our models show a comparatively lower error factor (of 1.5). This is reasonable considering that the only things that are common between our models and the churn traces are the long term churn rate and the number of online nodes. Thus, we conclude that train and crowd models predict system-level quiescence metrics reasonably well for gossip-based membership.

Figures 5 and 6 show the variation of the system-level quiescence metric with a varying population size (we used a subset of nodes in the trace). The plots have a downward trend because with increasing population size, it takes longer for the status of a churned node to be propagated to all other nodes ([21] showed that the propagation time is $O(log(N))$ for N online nodes). We observe that the train and crowd model predict the average value of this metric within an error factor of 2 for the Overnet trace, and with a much smaller error factor for the Azureus trace. With Azureus trace, which is more realistic than Overnet and smoothened Overnet traces, the error rate is within 11.44% for train and 15.2% for crowd at 800 nodes, and within 34.5% for train and 17.5% for crowd at 2400 nodes. The causes for these error rates are similar to the explanation for the errors in the entry-level metric.

Thus, we conclude that for gossip-based membership, the train and crowd models are reasonably good at predicting both the entry- and the system-level quiescence metrics.

6.3 Pastry DHT

Methodology and Results: We rely on FreePastry, an open-source implementation of the Pastry DHT. We use the built-in simulator of FreePastry and drive our simulation with the traces described in Section 6.1. Although we do not measure latency-related metrics, in order to measure the performance in wide area settings, we use FreePastry's Euclidean topology. This is a random placement of nodes on a 2-dimensional plane with the inter-node latencies varying from 2 *ms* to 200 *ms* based on Euclidean distance.

Pastry has each node maintain a routing table pointing to a few other nodes (peers). These peers are selected according to certain structured rules (Pastry uses prefix matching). Churn in the system thus causes Pastry routing tables to be updated as joined nodes satisfy the prefix match, and departed nodes are deleted. We are interested in measuring the volatility of routing tables in the system. Thus, unlike the two metrics measured for gossip-based membership, for Pastry our main inquiescence metric of interest is the number of routing table changes per second, across the entire system. This is somewhat akin to the FI metric (see definition 5.1.2).

Our experimental setup uses the default parameters of FreePastry. Specifically, the parameters pertinent to our simulation are the leafset maintenance frequency and the routing table maintenance frequency, since those parameters contribute to routing table changes. The default values are 60

seconds for the leafset maintenance frequency and 900 seconds for the routing table maintenance frequency.

Figures 7 and 8 plot for different population sizes, respectively for the Overnet trace and Azureus traces. We observe that the train and crowd models predict the values within an error factor of 5% for both traces, in spite of having only the long-term churn rate and average number of online nodes in common with the churn traces. Thus, our models are accurate at predicting quiescence of Pastry.

7 Related Work

Real Churn Measurements: Trace-based measurements have shown high rates of churn (up to 100% per hour) and short session times (O(minutes)) for p2p systems such as Overnet [2] and Gnutella [8]. Some past work has mathematically modeled session times in these systems, e.g., [19] for Gnutella, BitTorrent, and Kad, [8] for Kazaa and Gnutella.

Node-based Churn Models: Krishnamurthy et al [11] analyzed Chord's stability using a Poisson-based arrival/departure churn model and predicted the fraction of incorrect DHT membership entries and failed lookups, given arbitrary churn rates. Leonard et al [13] defined non-exponential process uptime/downtime distributions, and showed how to derive both residual and new process lifetimes. Kong et al [10] analyzed the probability of non-partitionability of seven different p2p systems under a one-time failure model. Godfrey et al [7] used arbitrary uptime/downtime distributions to analyze and compare different replica selection policies under churn.

System-based Churn Models: Fernandez et al [5] generalized traditional bounded-failure models to churn, by proposing a model with a known lower bound α on the number of processes that remain non-faulty across churn batches. They used this model to propose and analyze two eventual leader election protocols. Mostefaoui et al [16] used the same α-based churn model from above to adapt a leader election protocol originally proposed for a static system model to a churn-based distributed system. Finally, in [14] Liben-Nowell et al defined the notion of half-life for a p2p system as the time to either replace 50% of existing processes or add on 100% new processes. They then showed that, under steady-state system size, as long as the half-life of this churn model is $\Omega(log^2(N))$, the Chord DHT would continue to guarantee lookups completed successfully w.h.p.

8 Conclusions

In this paper, we have proposed two new churn models - a tractable and periodic train model, and a generic and practical crowd model. We then showed the relation between the train and crowd models w.r.t. the satisfaction of two types of protocol properties - reliable quiescence (RQ) and infinite quiescence (IQ). We used these models to study quiescence in several membership protocols, including full membership, gossip-based membership, and DHT membership. Our experiments with gossip-based and Pastry DHT showed that the train and crowd models predict the values of quiescence metrics within a factor of 1.5-2 (gossip-based membership), and within a factor of 5% (the Pastry DHT). Based on our results, we believe that simple churn models such as train and crowd can be used to provide tractable and practical analysis of quiescence properties in a variety of distributed protocols.

References

[1] M. K. Aguilera, W. Chen, S. Toueg, "On Quiescent Reliable Communication", *SIAM J. Comp.*, 29:6, pp. 2040-2073.

[2] R. Bhagwan, S. Savage, G. M. Voelker, "Understanding Availability", *Proc. IPTPS*, 2003.

[3] J. Chu, K. Labonte, B. Levine, "Availability and Locality Measurements of Peer-to-Peer Systems", *Proc. SPIE*, vol. 4868, 2002.

[4] A. Das, I. Gupta, A. Motivala, "SWIM: Scalable Weakly-Consistent Infection-Style Process Group Membership Protocol", *Proc. IEEE DSN*, 2002.

[5] A. Fernandez, E. Jimnez, M. Raynal, "Eventual Leader Election with Weak Assumptions on Initial Knowledge, Communication Reliability, and Synchrony", *Proc. IEEE DSN*, 2006.

[6] P. Ganesan and M. Sheshadri, "On Cooperative Content Distribution and the Price of Barter", *Proc. ICDCS*, 2005.

[7] P. B. Godfrey, S. Shenker, I. Stoica, "Minimizing Churn in Distributed Systems", *Proc. ACM SIGCOMM*, 2006.

[8] P. K. Gummadi *et al.*, "Measurement, Modeling, and Analysis of a Peer-to-Peer File-sharing Workload", *Proc. ACM SOSP*, 2003.

[9] X. Hei, C. Liang, J. Liang, Y. Liu, K. W. Ross, "A Measurement Study of a Large-Scale P2P IPTV System", *IEEE Transactions on Multimedia*, 9:8, Dec. 2007, pp. 1672-1687.

[10] J. S. Kong, J. S. A. Bridgewater, V. P. Roychowdhury, "A General Framework for Scalability and Performance Analysis of DHT Routing Systems", *Proc. IEEE DSN*, 2006.

[11] S. Krishnamurthy, S. El-Ansary, E. Aurell, S. Haridi, "A Statistical Theory of Chord under Churn", *Proc. IPTPS*, 2005.

[12] J. Ledlie, P. Gardner, M. Seltzer, "Network Coordinates in the Wild", *Proc. Usenix NSDI*, 2007.

[13] D. Leonard, Z. Yao, V. Rai, D. Loguinov, "On Lifetime-Based Node Failure and Stochastic Resilience of Decentralized Peer-to-Peer Networks", *IEEE TON*, 15:5, Oct. 2007, pp. 644-656.

[14] D. Liben-Nowell, H. Balakrishnan, D. R. Karger, "Analysis of the Evolution of Peer-to-Peer Systems", *Proc. ACM PODC*, 2002.

[15] N. A. Lynch, *Distributed Algorithms*, Elsevier, Mar. 1996, ISBN: 1558603484.

[16] A. Mostefaoui, M. Raynal, C. Travers, S. Patterson, D. Agrawal, A. E. Abbadi, "From Static Distributed Systems to Dynamic Systems", *Proc. IEEE SRDS*, 2005.

[17] S. T. Piergiovanni and R. Baldoni, "Connectivity in Eventually Quiescent Dynamic Distributed Systems", *Proc. LADC*, 2007.

[18] A. Rowstron and P. Druschel, "Pastry: Scalable, Decentralized Object Location and Routing for Large-Scale Peer-to-Peer Systems". *Proc. IFIP/ACM Middleware*, 2001.

[19] D. Stutzbach and R. Rejaie, "Understanding Churn in Peer-to-Peer Networks", *Proc. ACM IMC*, 2006.

[20] D. Stutzbach and R. Rejaie, "Characterizing Unstructured Overlay Topologies in Modern P2P File-Sharing Systems", *Proc. ACM IMC*, 2005.

[21] R. van Renesse, Y. Minsky, M. Hayden, "A Gossip-Style Failure Detection Service", *Proc. IFIP Middleware*, 1998.

[22] S. Voulgaris, D. Gavidia, M. van Steen, "CYCLON: Inexpensive Membership Management for Unstructured P2P Overlays", *JNSM*, 13:2, Jun. 2005, pp. 197-217.