Characterizing Churn in Peer-to-Peer Networks

Technical Report CIS-TR-2005-03
University of Oregon

June 3, 2005

Daniel Stutzbach, Reza Rejaie
{*agthorr,reza*}*@cs.uoregon.edu*

Abstract

The user-driven dynamics of peer participation, or *churn*, are an inherent property of Peer-to-Peer (P2P) systems that should be taken into account in both the design and evaluation of any P2P application. While prior studies have shown that peer participation is highly dynamic, they have not provided detailed models, needed for simulation and analysis. More importantly, it is unclear whether the dynamics of peer participation exhibit a similar behavior across different classes of P2P applications.

In this paper, we present a detailed study of churn in widely-deployed applications from three different classes of P2P systems: an unstructured file-sharing system (Gnutella), a content-distribution system (BitTorrent), and a distributed hash table system (Kad). Our analysis reveals several interesting properties of churn including the followings: (i) the overall dynamics of peer participation is surprisingly similar across these systems, (ii) peer session times and downtimes follow power-law distributions, (iii) peer inter-arrival times follow a Poisson distribution, and (iv) peer session times across consecutive appearances are correlated in file sharing applications. These findings imply that a large portion of participating peers at any point of time are highly stable while the remaining peers turnover very quickly. We show how our findings can be used to simulate churn, and discuss the basic implications of our results on the design of P2P applications.

1 Introduction

During recent years, the Internet has witnessed a significant increase in the popularity of Peer-to-Peer applications ranging from file-sharing (*e.g.*, Gnutella and FastTrack) to conferencing (*e.g.*, End System Multicast [6]) and content distribution (*e.g.*, BitTorrent). In P2P applications, each participating peer voluntarily joins the system, uses available resources of other peers (*e.g.*, CPU, storage, bandwidth) while offering up its own resources, and leaves the system at some arbitrary later point in time. This implies that *(i)* peer participation in P2P systems is inherently dynamic, and *(ii)* these dynamics are primarily *user-driven* (at least in those P2P applications that do not run as a hidden "always-on" system daemons). The user-driven dynamics of peer participation, or *churn*, must be taken into account in both the design and evaluation of any large scale P2P application. To achieve this goal, application designers require a reliable, representative, and relatively simple characterization of churn.

Despite their importance, the characteristics of churn in P2P applications are not well understood. Several measurement-based studies have shed some light on the overall dynamics of peer participation and show that it is indeed highly dynamic (*e.g.*, [5, 16, 17]). These studies only present high level characteristics of churn (*i.e.*, the CDF of session time [15] or median session time [5]) as a part of comprehensive studies of specific P2P applications. Therefore, the provided characteristics are not sufficient to model or simulate churn. More importantly, previous studies on churn have not verified (or at least did not present any evidence) that their measurements have captured a *representative* population of sessions. Additionally, their findings are sometimes significantly different (*e.g.*, the reported median session time varies significantly in different measurements of the same P2P application [14]). Section 6 discusses the related work in more detail. In the absence of any reliable model, researchers have made simplifying assumptions about churn behavior (*e.g.*, a Poisson distribution for session time [14, 11]) in their design and evaluation. In a nutshell, the following important questions about churn still beg for an answer:

- What, if any, are the fundamental properties which consistently characterize peer churn?
- How similar (or different) are churn characteristics across different classes of P2P applications?
- What is the right model for churn characteristics for

use in evaluating P2P applications through simulation?

- What algorithms can P2P applications employ to handle churn intelligently?

This paper presents a detailed study on churn in widely-deployed applications from three different classes of P2P systems: Gnutella, an unstructured file-sharing system; Kad, a Distributed Hash Table (DHT); and BitTorrent, a content-distribution system. One contribution of this work is our measurement methodology, presented in Section 2, to capture an unbiased characterization of peer dynamics. This is particularly challenging in Gnutella and Kad since no built-in centralized monitoring exists. To address this issue, we capture all participating peers in Gnutella and leverage existing structure in the Kad network to acquire a uniformly random sample of sessions.

The main contributions of this paper are detailed characterizations and modeling of peer dynamics across different classes of P2P systems. Toward this end, in Section 3 we characterize churn at two levels: *(i) Group-level characterizations*, which present the behavior of the all participating peers collectively, and *(ii) Peer-Level characterizations*, which capture the behavior of specific clients (in Kad) or specific IP addresses (in Gnutella and BitTorrent) across multiple appearances in the system over time. We also discuss similarities and differences in churn behavior across the different classes of P2P applications. Our main results can be summarized as follows:

- Various characteristics of churn exhibit similar behavior across all three applications.
- Session time follows a power-law distribution. However, most of the peers in the system at any point of time are long-lived peers. In other words, most peers in the system are relatively stable, while the remaining small portion of peers turnover very quickly.
- Peer inter-arrival time follows a Poisson distribution.
- Session times across consecutive appearances of a client or an IP address exhibit a strong correlation in both Gnutella and Kad. Thus, past session time is a good predictor of the next session time.
- The availability of individual clients or IP addresses per day exhibit a strong correlation across consecutive days.

Additionally, Section 4 shows how our results can be leveraged to accurately simulate churn for a desired active population size, and Section 5 elaborates on key implications of our results on the design of P2P applications.

2 Measurement Methodology

Our goal is to accurately capture the dynamics of peer participation across a *representative* group of peers in three candidate P2P systems: Gnutella, BitTorrent and Kad. One attractive feature of these candidate systems is that they have open specifications and open source implementations. This allows us to conduct our measurement with more confidence since it minimizes any potential incompatibility error. To accurately capture churn in any P2P system, we need to address two issues: *(i)* how to select a representative group of peers, and *(ii)* how to accurately measure the arrival and departure times of each selected peer. Our ability to address these issues depends on features provided by each system as well as their population size.

While Gnutella and Kad applications build one large overlay for the purpose of locating files, BitTorrent uses separate overlays, called *swarms* for each file being transferred. Each BitTorrent overlay network is identified by a hash of the file to be transferred and uses a special rendezvous point, called a *tracker*, for coordination. The tracker's job is to keep track of all peers within the swarm. Towards this end, each participating peer contacts the tracker when it arrives and notifies the tracker when it departs. Additionally, peers contact the tracker periodically and the tracker removes peers which it has not heard from in some time (*i.e.*, soft state). The tracker logs each of these interactions, significantly facilitating the capture of the arrival and departure time of all peers in each BitTorrent swarm.

However, the Gnutella and Kad networks are extremely large in size (*i.e.*, more than a million concurrent peers) and do not have a central monitoring point. In our prior work [18], we developed a peer-to-peer crawler called *Cruiser* which can capture a snapshot of the peers in the Gnutella network over the course of several minutes (Δ). We have modified Cruiser by separating out the Gnutella-specific components and adopting a plug-in architecture which allows Cruiser to speak different protocols to crawl different P2P networks. The generic components of Cruiser, used to achieve high performance, include coordination, data recoding, and parallelization. The plug-in architecture allows Cruiser to speak different P2P protocols, such as Gnutella and Kad, with the additional of a relatively small, protocol-specific module.

Our basic measurement methodology for Gnutella and Kad is to use Cruiser to collect hundreds of back-to-back snapshots, and note the arrival and departure time of peers by comparing snapshots. A new peer p that was captured in snapshot i but was not present in snapshot $(i-1)$ must have arrived during the interval 2Δ between the start of crawl $(i-1)$ and the end of crawl

i[1]. Alternatively, a peer q that was part of snapshot $(i-1)$ but was not present in snapshot i must have left during the interval 2Δ from the start of one crawl and the end of the next. Therefore, we can measure with a granularity of 2Δ the departure and arrival times of every peer. We note that as the number of active peers grows, the duration of each crawl (Δ) increases, and thus the granularity of our measurements becomes coarser, *i.e.*, there is a tradeoff between the size of a snapshot and the accuracy of the measured arrival and departure times. Therefore, it is essential to minimize the duration of crawls (Δ). Once we determine the departure and arrival times of a peer p within a sequence of back-to-back snapshots, we can easily determine the duration of one appearance which is called its *session time* as follows: $SessionTime = DepartureTime - ArrivalTime$. We also use the term *uptime* for active peer p to denote the duration of time since its arrival.

To ensure that measured session times are not biased, we use the "create-based method" employed by Saroiu *et al.* [15]: Given a sequence of back-to-back snapshots during a window of τ minutes, we split the measurement window into two halves. Then, we only keep the session time for those peers that *(i)* arrive during the first half, *(ii)* leave during either the first or second half of the measurement window, and *(iii)* their session time is not longer than $\frac{\tau}{2}$. This guarantees unbiased results for sessions shorter than $\frac{\tau}{2}$, but tells us nothing about the distribution of longer sessions. To avoid time-of-day bias in our results we chose $\tau = 2$ days. Our initial measurements, as well as previous studies [3], show fluctuations in network size correlated with the time of day.

In the following subsections, we present a brief overview of our candidate applications, and discuss application-specific issues in capturing accurate and representative snapshots.

2.1 BitTorrent

BitTorrent is a popular P2P application that is often used for the distribution of very large files from a source to a large group of users (called a *swarm*). Peers form an overlay and exchange different blocks of the content until each peer has the entire file. Each swarm is coordinated by a rendezvous point, called a *tracker*, whose address is provided out of band. Each new peer contacts the tracker to join the swarm, periodically sends an update of its progress, and informs the tracker when it departs. Note that each peer may receive the entire file across multiple sessions, *i.e.*, it may obtain only a subset of blocks in one session and resume the download later. Since the tracker logs all its interactions with group members, the log provides detailed information about the arrival and departure times of each peer.

We have obtained tracker logs from two long BitTorrent swarms: distributions of Debian and Red Hat[2]. Close examination of these tracker logs reveals that roughly 50% of participating peers contact the tracker within every 5 minutes, and 99% of them contact the tracker within every 31 minutes. However, peers may depart in an ungraceful fashion and abruptly stop contacting the tracker. To identify these peers, we conservatively assume any peer that has not contacted the tracker within 35 minutes has ungracefully departed. These make up around one third of all sessions in our dataset and were eliminated since we can not measure their session time. We note that the session time for a BitTorrent client is a combination of time spent downloading the file (the *download time*) and additional time that the user leaves the client running after the download is complete (the *lingering time*). While the download time might be influenced by the size of the file or the number of other peers, the lingering time is directly determined by user behavior. Furthermore, the user can directly control the duration of each session by stopping the application during the download and returning at a later time to complete the file download. Since the tracker log presents the evolution of delivered content to each peer, it allows us to separate download time from lingering time in our analysis and examine them separately.

2.2 Gnutella

Gnutella is a popular P2P file-sharing applications with more than 1.3 million concurrent peers [19]. Each peer joins the network by connecting to a random group of participating peers. Since Gnutella is not run as a daemon, the arrival and departure times of each peer are triggered by user behavior, *i.e.*, session times are driven by when the user opens and closes the application. There is no central node in the Gnutella network that keeps track of all participating peers, therefore the only way to discover all peers is to crawl the overlay. Given a few participating peers in the session, a crawler progressively contacts peers to learn about their neighbors, until it discovers all the peers. The large size of the Gnutella network makes it a challenge to capture a complete crawl quickly. To address this, previous studies have selected a random subset of peers discovered by a partial crawl, and periodically probe those peers to measure their session time (*e.g.*, [15, 3]). The key question is "Does the session times of such a subset of peers represent the entire population *of sessions* in the Gnutella network?". With a heavy-tailed distribution of session time [16, 5], peers

[1]The interval is 2Δ rather than Δ because there is a possibility the peer arrives during crawl $i-1$ after the crawler has passed its neighborhood.

[2]We would like to thank Ernst Biersack from the Institut Eurecom who has kindly shared their Red Had tracker logs with us [7]. We obtained the Debian tracker logs directly from the Debian organization.

from a particular snapshot will have *significantly longer* sessions than the average session.

To ensure that captured session times are representative, we try to capture *all* peers in the overlay in each snapshot. We have recently developed a parallel crawler, called Cruiser [18], that can capture a complete snapshot of the Gnutella network with 1.3 million peers, in around 7 minutes (*i.e.*, $\Delta = 7$ minutes). Cruiser incorporates several features to achieve a achieve orders of magnitude higher crawling speed compare to all previously reported crawler as follows: *(i)* a master-slave architecture, *(ii)* parallel crawling of hundreds peers by each slave machine, and *(iii)* leveraging the two-tier architecture of modern Gnutella by crawling only top-level peers. In summary, Cruiser enables us to measure the departure and arrival times with around 14 minutes of accuracy for *every peer in the network*. Capturing the entire population eliminates any concern about sampling bias.

2.3 Kad

Kad is a Kademlia-based [12] P2P search network used by the eMule P2P file-sharing software [1]. To our knowledge, Kad is the largest deployed DHT, with more than 1 million simultaneous users according to our measurements. Similar to other DHTs, each node has a globally unique identifier of length b bits, and objects are stored on the node with the identifier closest to their hash. Each node stores a structured routing table to other nodes in the network such that the expected overlay distance between any two nodes is $O(\log n)$, where n is the population size.

Cruiser crawls the Kad network by querying nodes for their routing table information. This is possible due to the fact that in Kad peers do not actually route messages on behalf of other peers. Instead, each peer simply returns a message to the original peer referring it to the next hop in the routing table[3]. As a DHT, Kad uses a deterministic algorithm to specify the prefixes in each peer's routing table, based on that peer's Kad ID. Emulating this algorithm, Cruiser can send a query for each prefix in the routing table to extract the full routing table of a peer.

Since Kad uses a simple UDP query-response protocol, and extracting the entire routing table of a single peer requires many queries (around 40), Cruiser needs a mechanism to regulate the query rate to avoid causing congestion. Cruiser's plug-in Kad module implements a NewReno TCP-style congestion control algorithm [4, 8] to recover from dropped packets and throttle back the query rate when congestion is detected. The congestion control component includes the following mechanisms: round trip time variance estimation, slow start, congestion avoidance, exponential backoff, fast retransmit, and fast recovery. Our implementation differs from TCP in that it relies on specific, rather than cumulative, acknowledgements, so there is no risk of re-requesting packets already received.

[3]This is sometimes referred to as "iterative routing".

Dataset	Zone	Mean Crawl
Kad 1	0xab0/10	3.4 min
Kad 2	0x594/10	3.5 min
Kad 3	0xe14/10	3.3 min
Kad 4	0x734/12	1.1 min

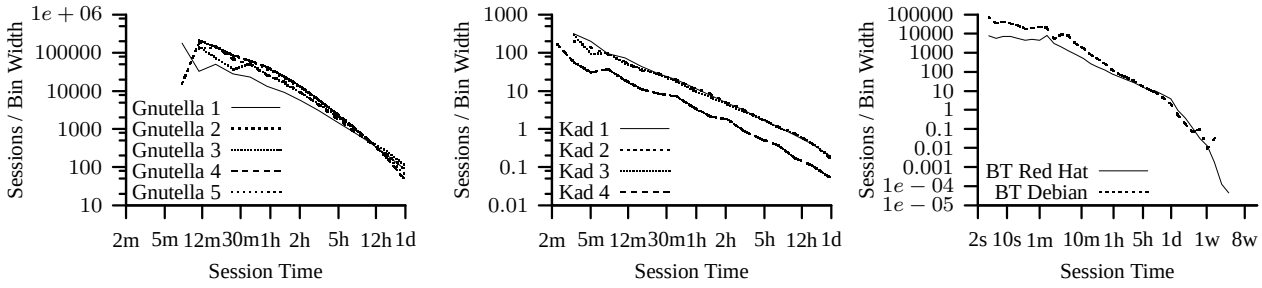
Table 1: Kad Zones

Our measurements reveal most Kad peers store over 700 pointers to other peers. However, many of these pointers are stale[4]. Due to the large number of stale entries in each routing table, Cruiser must contact each discovered peer to verify whether it's actually present during the crawl. To achieve a high crawling speed, Cruiser times out slow and unresponsive peers more aggressively than TCP. If Cruiser has not received any packets from a peer after 3 timeouts (2.5 seconds), it assumes the peer is not present. If Cruiser receives any packets from a peer, it notes the peer as present even if the full routing table is not extracted due to later timeouts. We empirically configured these timeout values by choosing the lowest timeout which did not generate false negatives in preliminary tests.

Due to the large network size as well as the large routing tables in each peer, crawling the entire Kad network takes a long time, even with our fast crawler. Normally this would unacceptably decrease the accuracy in measuring the peer arrival and departure time. To address this, we leverage the existing structure of the Kad ID space. We divide the identifier space into *zones*, where each zone consists of a continuous range of identifiers. If we visualize the identifier space as a ring, a zone corresponds with an arc of the ring. Since each peer selects its ID uniformly at random, each zone represents a uniformly random sample of the entire Kad network. Unlike sampling a set of addresses and probing them periodically, zones allow us to monitor the arrival of new peers. In other words, we are selecting a subset of all *potential sessions* rather than a subset of all *currently active peers*.

As an input parameter, Cruiser accepts a zone, specified as a Kad ID address and mask, analogous to an IP subnet address. The address specifies the location of the zone, and the mask specifies the zone's size. We use the "slash notation" to specify Kad ID zones. For example "0x594/10" specifies all Kad identifiers where the first 10 bits match the first 10 bits in the hexadecimal number 0x594. Furthermore, we present results for four different Kad zones, as shown in Table 1 to empirically verify that results are consistent across zones. The /10 zones have

[4]Kademlia includes a lot of redundancy in its routing tables with many potential paths for each next hop, using a LRU policy to push out stale entries.



(a) Gnutella (b) Kad (c) BitTorrent

Figure 1: Session Time Histograms

Datasets	Start Date	Duration
Gnutella 1	Oct. 14, 2004	2 days
Gnutella 2	Oct. 21, 2004	2 days
Gnutella 3	Nov. 25, 2004	2 days
Gnutella 4	Dec. 21, 2004	2 days
Gnutella 5	Dec. 27, 2004	2 days
Kad 1	Apr. 13, 2005	2 days
Kad 2	Apr. 16, 2005	2 days
Kad 3	Apr. 18, 2005	2 days
Kad 4	Apr. 21, 2005	2 days
BitTorrent Red Hat	Mar. 21, 2003	5 months
BitTorrent Debian	Feb. 22, 2005	2 months

Table 2: Measurement collections

approximately 1000 simultaneous peers at any time, and the /12 zone has approximately 250 simultaneous peers. We selected the /10 zone size because it is the largest zone which we can crawl within 5 minutes. We added the /12 zone to give us some insight into dynamics below 5 minutes[5]. We crawled each zone back-to-back for a 2-day period during which we observed many thousands of complete sessions in each zone.

2.4 DataSet

Table 2 provides summary information for the datasets we use throughout this paper. We use 5 two-day datasets from Gnutella and 4 two-day datasets from Kad, where each datasets consists of several hundred back-to-back snapshots. We also use 2 datasets from BitTorrent, each a few months long. In the remainder of this paper, we will simply refer to each dataset by the name given in Table 2.

3 Characterizations of Churn

We characterize churn at two different levels:
(i) Group-Level Characterization: We consider individual appearances by peers (*i.e.*, each measured session time) independently. More specifically, the focus is on the aggregate behavior of individual session times without any attention to the identity of peers.
(ii) Peer-Level Characterization: We examine any correlation across multiple appearances of a peer. Therefore, the identity of each peer must be determined in order to relate different appearances. We use both IP addresses and persistent, unique, application-assigned identifiers to identify individual peers. Note that changes of IP address across multiple appearances of a peer does *not* affect the group-level characterization, but must be addressed in the peer-level characterizations.

3.1 Group-Level Characterization

In this section, we explore properties of churn which do not rely on peer identity across sessions. We explore the relationship between the distribution of session times across all sessions and the distribution of session times across all peers coexisting at any particular moment. We will also examine the power of uptime as a predictor for remaining uptime, and examine the inter-arrival distribution for new peers. These results will provide sufficient information to model churn for simulation, which we will discuss in Section 4.

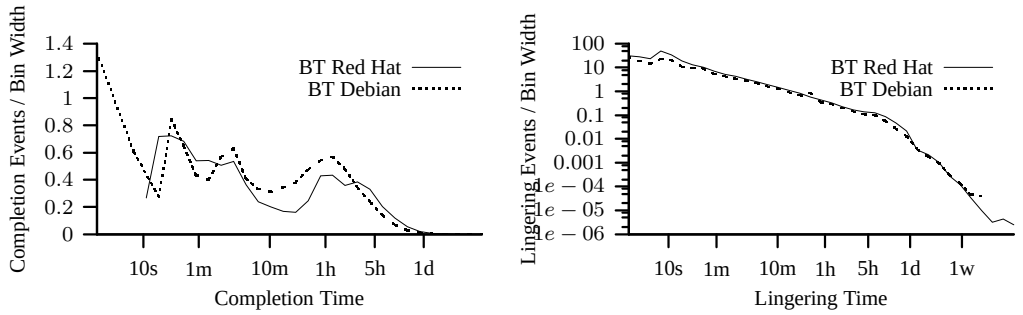
Distribution of Session Time: Figure 1 presents the distribution of session time across different datasets for each system[6]. These results demonstrate the following points: First, the distribution of session time across different datasets is very similar. This implies that the distribution of session time does not significantly change with time and thus our results are representative for each system. Second, and more importantly, in all three systems the distribution of session time, $s(t)$, within the range of a few minutes up to a day follows a power-law distribution:

$$s(t) \propto t^{-\alpha}$$

We also performed a least-squares fit on the log fo the data of the session time distributions shown in Figure 1.

[5]Reducing the zone size further does not significantly reduce our crawl time due to the time we must spend waiting for congested peers and timeouts.

[6]All log-log plots in this paper use log bins to smooth out the tail, where the number of data points in each bin would otherwise become small and noisy. To avoid weighting the fit towards the much larger number of data points in the tail, fits also use log bins.



(a) Completion Times

(b) Lingering after Download Completion

Figure 2: Interactions with Transfer Time

The reduced chi-squared (χ^2) value and the power-law parameter, α, are presented in Table 3. The parameter, α, varies across datasets, but is consistently in the range 1.1–1.7. Higher values of α correspond with a smaller fraction of long sessions. The reduced χ^2 values less than 3 indicate the distribution is a good fit.

This finding suggests that the power-law distribution of session times is independent of the application and the purpose of the application. Rather, it's determined by application-independent aspects of user behavior. To further examine this hypothesis, we divide the session time of BitTorrent clients into completion time (the time from appearance to download completion) and lingering time (the time that each peer stays in the network after completion of the download). These results are limited to peers which download the complete file in a single session. Figure 2 depicts the distribution of both completion time and lingering time for both BitTorrent datasets. Interestingly, these figures show that lingering time also follows a power-law distribution. However, completion time does not appear to follow any specific pattern, presumably since it is both group- and content-dependent. Since lingering does not serve the user in any way, this is yet another evidence, that user-controlled events, unrelated to the application's functionality result in a power-law.

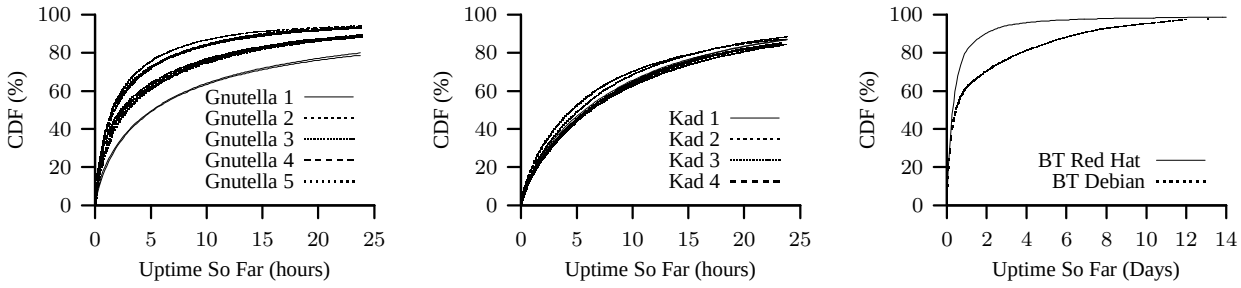
Furthermore, session times in BitTorrent (Figure 1(c)) exhibit a roughly uniform distribution for very short timescale ($<$ 1 minute), and power-law with a steeper slope for very long ($>$ 1 day) timescales. We expect that Gnutella and Kad behave similarly, but our current datasets are not of sufficiently long duration to characterize long session times and do not have adequately fine granularity to analyze the behavior of very short sessions. In general, for very short timescales the power-law distribution cannot hold since it would require an infinite number of infinitely short sessions. For very long timescales, we would expect a steeper slope as hardware issues begin to dominate over user behavior (*e.g.*, needing to reboot, power failure). Our findings present the behavior of session time within the range of a few minutes up to a day, which is adequate for most P2P modeling and simulation purposes.

Dataset	Reduced χ^2	α
Gnutella 1	0.162202	1.30406
Gnutella 2	0.877162	1.33928
Gnutella 3	0.134783	1.44931
Gnutella 4	0.245367	1.67932
Gnutella 5	0.232318	1.59992
Kad 1	0.0133128	1.17703
Kad 2	0.0354993	1.11664
Kad 3	0.0342527	1.11935
Kad 4	0.0689354	1.15276
BitTorrent Debian	1.11564	1.27459
BitTorrent Red Hat	2.90033	1.27020

Table 3: Reduced χ^2 and fitted power-law parameter values for observed uptime distributions

In summary, these results show that *session times follow a power-law distribution across different classes of P2P systems within the timescale of a few minutes to one day*. Furthermore, this property is driven primarily by user action. Power-law is a heavily skewed distribution; compared to the Poisson distribution, it has some sessions with much longer duration, as well as a much larger fraction of sessions with very short durations. Therefore, the common modeling assumption of a Poisson session time distribution in recent studies (*e.g.*, [14, 11]) is inappropriate.

Distribution of Peer Uptime: In the previous subsection, we presented the distribution of session time across all sessions. However, it does not illustrate what combination of these peers might coexist in the system at any point of time, *i.e.*, Figure 1 does not present the distribution of session time across peers in a given snapshot. To address this issue, we turn our attention to the distribution of session time among participating peers in a snapshot, $s_u(t)$. We mathematically relate $s_u(t)$ to $s(t)$ as follows. If we imagine sampling the snapshots of the system, a session of length n has n opportunities to be present, while a session of unit length only has a single opportunity. Thus, $s_u(t)$ is also called the *time-weighted*



(a) Gnutella (b) Kad (c) BitTorrent

Figure 3: Uptime CDF

session time distribution, and is related to $s(t)$ as follows:

$$s_u(t) \propto \int s(t)\,dt \;\; \propto t \cdot t^{-\alpha}$$

There is also a relationship between $s(t)$ and the distribution of uptimes among each peer in the system, $u(t)$. We recall that the uptime of peer p at a given point of time is defined as the continuous time that the peer p has been in the system since its most recent arrival. In this case, an uptime of n will be represented once by every peer with a session length of n or longer. Thus, the relationship between $s(t)$ and $u(t)$ is described as follows:

$$u(t) \propto \int_t^{\infty} s(t)\,dt \;\; \propto t \cdot t^{-\alpha}$$

In other words, $s_u(t)$ and $u(t)$ follow identical distributions. In fact, the distribution of remaining uptimes[7] among all peers in a snapshot also follows the same distribution. This implies that both the distribution of session time and uptime within a snapshot follow power-law distributions, but the latter has a significantly gentler "slope", $\alpha' = \alpha - 1$. Lower α implies that *the distribution of peers within a snapshot is weighted more heavily towards long-lived peers*.

Figure 3 shows the CDF of $u(t)$ for co-existing peers within a snapshot[8]. To compute this distribution, we divide each measurement window into two halves, A and B. We annotate each snapshot with the arrival time of each peer. Then, for each snapshot in B, we sample the uptime for each peer. If it's uptime is greater than half of our measurement window (*i.e.*, one day for Gnutella and Kad), then we record it's uptime as unmeasurable. This allows us to determine what fraction of uptimes are greater than we can measure without bias[9]. For BitTorrent, where we have tracker logs instead of snapshots, we sample the state of the system once per minute.

Figure 3 shows several interesting points: First, the uptime distributions exhibit very similar behavior across different systems. Second, the gap in the tail of the distribution quantifies the percentage of peers whose uptime was longer than half our measurement period (*i.e.*, one day). More specifically, roughly 10%–20% of peers per snapshot in Gnutella and Kad have uptime longer than a day, and around 1–3% of BitTorrent peers have uptime longer than 14 days. Third, and most interestingly, these distributions are heavily weighted towards uptimes longer than a couple of hours, *i.e.*, they show that the majority of peers in each snapshot are long-lived peers. For example, if we randomly select a peer from these systems, the probability that the selected peer has an uptime more than five hours is roughly 40% in Gnutella, 55% in Kad and 60% in BitTorrent.

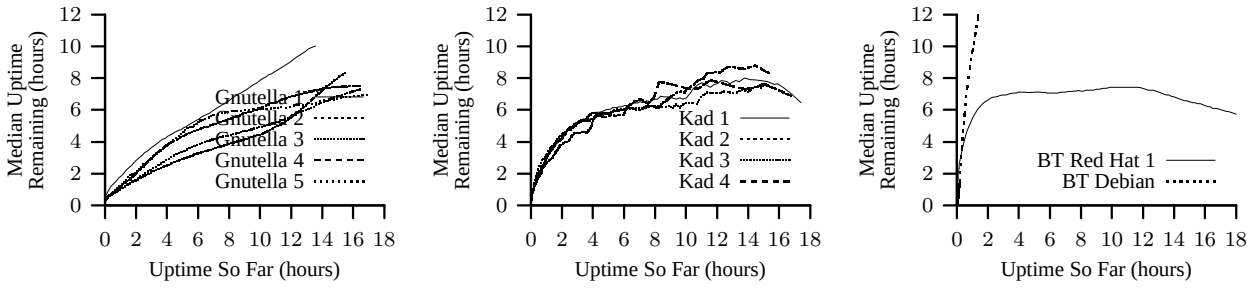
The combination of the uptime distribution (Figure 3) and the session time distribution (Figure 1) present an enlightening view of churn in P2P system as follows: *At any point of time, a majority of participating peers in the system are long-lived peers. However, the remaining small portion of short-lived peers join and leave the system at such a high rate that they constitute a relatively large portion of overall sessions.* Describing this from a different angle, the session time for a randomly selected session (Figure 1) is likely to be short whereas the session time of a randomly selected participating peer from a particular snapshot is likely to be long.

Uptime Predictability: In contrast to a Poisson distribution, the power-law distribution is not memoryless. This implies that at any point of time, peer uptime is a good predictor of its remaining uptime, *i.e.*, the longer a peer has been in the network, the more likely it will remain for a long period of time. To empirically show this property, we examine the correlation between uptime and remaining uptime at two levels. Figure 4 depicts the CDF of the median remaining uptime for peers with uptime x, across different datasets of all three candidate systems.

[7] By "remaining uptime" we mean the total session time minus the uptime so far.

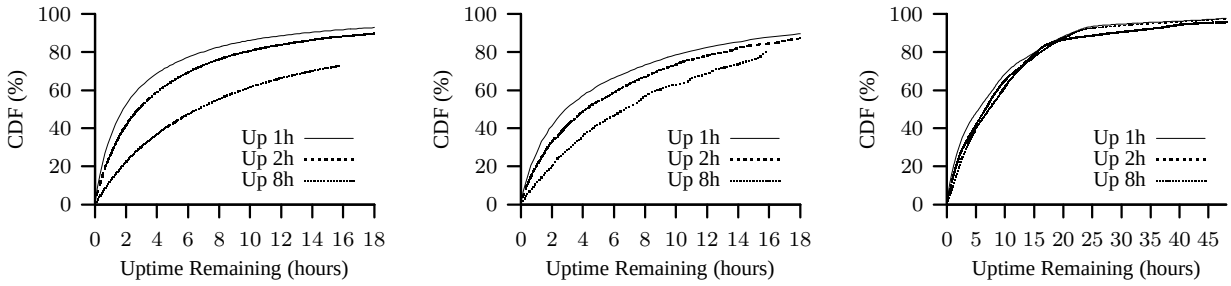
[8] We also looked at the histogram version of these figures to confirm the power-law shape, but omit these figures due to space constraints.

[9] This value is the difference between the highest plotted point in the CDF and 100%.



(a) Gnutella (b) Kad (c) BitTorrent

Figure 4: Median Remaining Uptime Function



(a) Gnutella 1 (b) Kad 1 (c) BitTorrent Red Hat

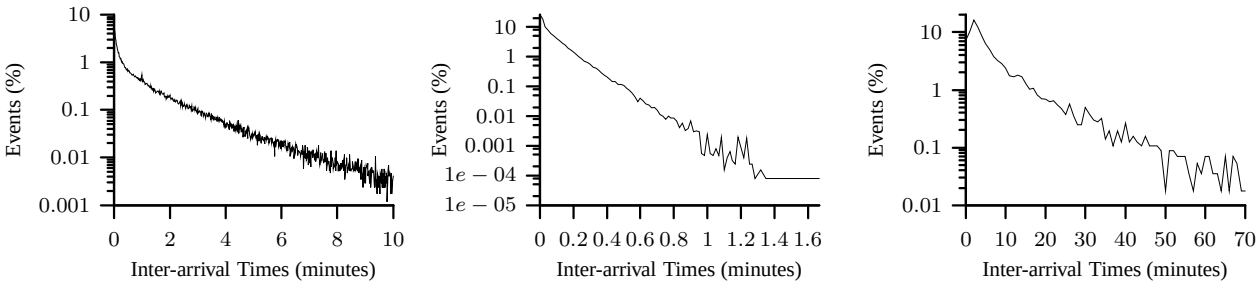
Figure 5: CDF of Remaining Uptime for peers already up 4 hours, 8 hours, and 12 hours

This figure shows that while uptime is in general a good predictor of remaining uptime, its strength is different among candidate systems and for different uptime values. More specifically, peer uptime in Gnutella is a good indicator of the remaining uptime regardless of uptime value; the median peer has a remaining uptime between 50% and 100% of its uptime so far. However, the uptime of Kad peers is a stronger predictor of remaining uptime up to around 4 hours. Beyond that, the median peer's remaining uptime increases only slowly. Nevertheless, at 16 hours of uptime, Kad peers have approximately the same median remaining uptime (around 7 hours) as their Gnutella counterparts. In the BitTorrent Debian dataset, we see a rapid increase in the median remaining uptime. Closer examination of this trace reveals that the observed remaining uptime distribution is significantly affected by the large number of seed peers run by the Debian organization.

To take a closer look at this correlation, we also examine the CDF of the remaining uptime for peers currently up for 1 hour, 2 hours, and 8 hours. Figure 5 depicts these CDFs for a single dataset in each candidate system. These figures show that the reliability of the predictions is highly variable, due to the high-skew of the underlying power-law distribution. For example, in Gnutella around 50% of peers up for 8 hours will be up for at least another 8 hours. However, the bottom 20% of the peers will be up for less than 2 more hours, while the top 30% will be up for more than 16 hours!

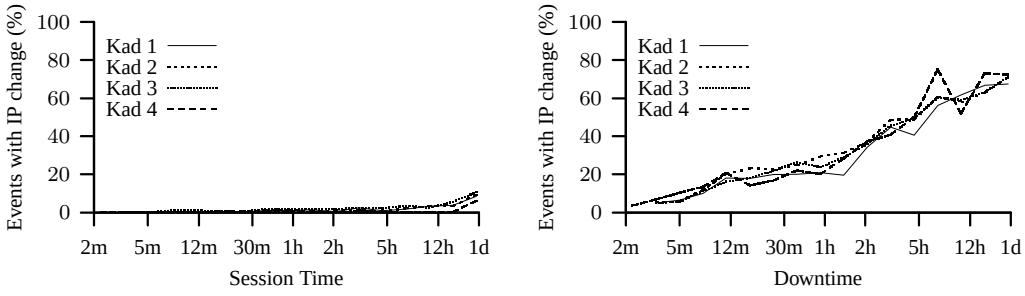
In summary, our results imply that while uptime is on average a good indicator of remaining uptime, it exhibits high variance. Therefore it should be used when a bad prediction does not have a major cost for individual peers but collectively making better predictions across all peers improves overall performance.

Distribution of Inter-Arrival Time: An inter-arrival time is the duration between the arrival of one peer and the next. The distribution of peer inter-arrival times is another dimension of group-level characteristics of churn. It's an important characteristic for simulating churn which has not be discussed in previous studies. To measure inter-arrival times, we must be able to observe individual arrival events. In Gnutella, this is not possible since tens of thousands of new peers arrive between two consecutive snapshots. In contrast, measuring inter-arrival time in BitTorrent is straightforward since tracker logs capture arrival times with one-second granularity. To examine inter-arrival time in Kad, we use our Kad 4 trace which monitors a smaller zone with 1 minute granularity. To further improve the granularity of the dataset, we divide the target zone into 16 smaller zones of equal size. This makes the inter-arrival times within each zone



(a) BitTorrent Red Hat (b) BitTorrent Debian (c) Kad

Figure 6: Inter-arrival Time Distribution Histograms



(a) IP changes during a session (b) IP Changes between sessions

Figure 7: IP Address Changes

sufficiently long to be measured accurately (*i.e.*, significantly longer than 1 minute) without reducing the total number of arrival events.

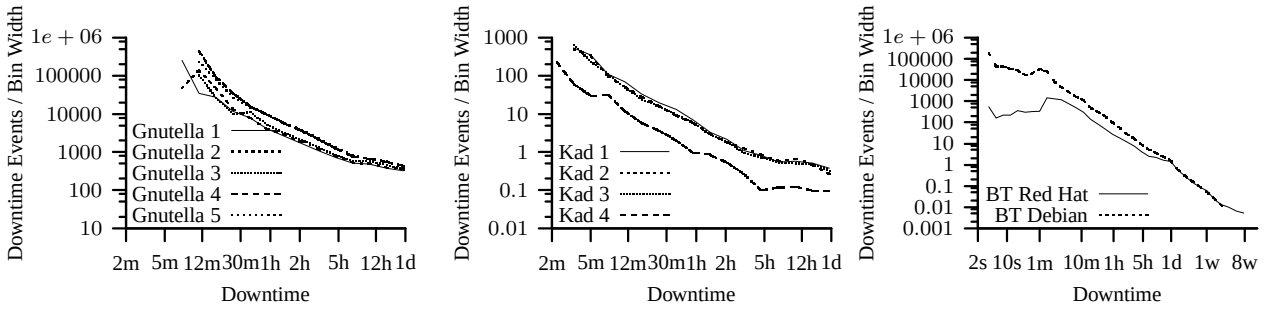
Figure 6 presents log-linear plots of the distribution of peer inter-arrival time for three datasets: BitTorrent Red Hat and Debian as well as Kad 4. These distributions appear to be straight lines on the log-linear plots, which is consistent with a Poisson distribution. This is reasonable because the Poisson distribution is memoryless which represents the fact that the arrival time of individual peers are independent events. It's worth noting that despite the similarity of these distributions, the median inter-arrival time is significantly different across these datasets, which reflects difference in the population size.

3.2 Peer-Level Characterization

In this section, we characterize the behavior of a peer across multiple appearances in the system. These characterizations are useful for both for design and evaluation of peer-to-peer systems. For example, they enable a peer to predict its session time early in a new appearance based on previous appearances. They are also a first step towards modeling appearances of a single peer within a simulation environment.

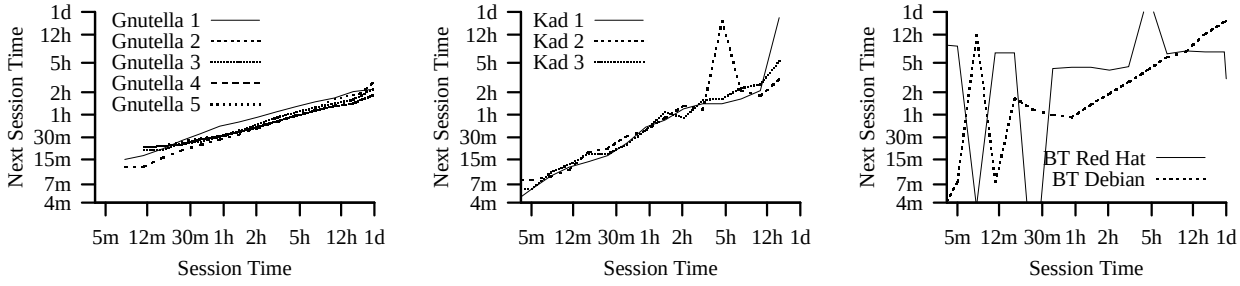
To achieve this goal, we need to determine the identity of each peer. We consider two types of identities: *(i)* Unique Client ID and *(ii)* IP Address. As previous studies have shown [2], each client might use a different IP address during each appearance in the system due to dynamic address assignment. Therefore, the characterization of peer behavior based only on unique, persistent client IDs represents the behavior of individual clients over time. Such characterizations are useful for those P2P applications that store persistent "state" information (*i.e.*, content or pointers) at specific clients, and thus are affected by their pattern of appearance in the system. However, IP-based characterization presents the behavior of specific IP address (that might be potentially used by different clients) across multiple appearances over time. IP-based characterizations are useful for bootstrapping, where it's important to rely on the availability of *any* peer at a particular IP address, regardless of its identity.

Kad is the only candidate system in which individual peers have a persistent unique ID. Thus, we use the Kad dataset for ID-based characterization, and both Gnutella and BitTorrent datasets for IP-based characterization across multiple appearances. We examine the following characteristics on both an IP- and an ID-basis: *(i)* the distribution of downtime, *(ii)* the correlation between consecutive session times, and *(iii)* availability. Figure 7 presents the variability of IP addresses in our Kad datasets both within sessions and between sessions.



(a) Gnutella (based on IP address) (b) Kad (based on node ID) (c) BitTorrent (based on IP address)

Figure 8: Downtime Distribution Histograms



(a) Gnutella (based on IP address) (b) Kad (based on node ID) (c) BitTorrent (based on IP address)

Figure 9: Correlation between Consecutive Session Time

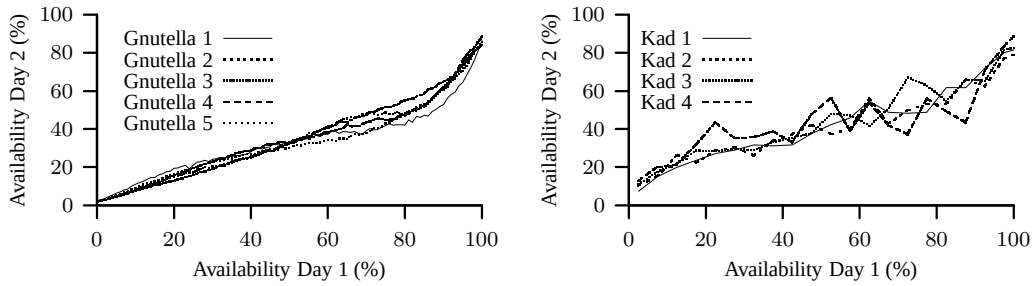
While a peer does not typically changes its IP address within a session, there is a high degree of IP address changes between sessions. Furthermore, Figure 7(b) shows there is a correlation between the downtime and the likelihood of a change in IP address. For example, almost 60% of the peers that are away from the network for 12 hours exhibit a change in address.

Distribution of Downtime: We define downtime as the interval between a departure time and the next arrival time for the same peer. To ensure an unbiased sampling of downtime, we again apply the create-based method (described in Section 2). Figure 8 presents the distribution of client downtime for Kad (based on ID), and IP downtime for Gnutella and BitTorrent (based on IP address). This figure shows that client downtime in Kad follows a power-law distribution within the timescale of few minutes up to 5 hours. Interestingly, the distribution of downtime based on IP address also follows a power-law distribution for both Gnutella (between a few minutes up to 5 hours) and BitTorrent (between a few minutes up to eight weeks!). In Gnutella and Kad, after around 5 hours the distribution begins to curve slightly upward. Further investigation is required to reliably explain the underlying cause for the shape of this distribution for downtimes longer than 5 hours.

One caveat with characterizing downtime is that we cannot quantify *(i)* the number of long-absent peers which have not logged into the system during our measurement period[10], and *(ii)* the actual downtime values of these peers. Information about these long-absent peers are necessary for simulating multiple appearances over long times scales (weeks). It's worth noting that there is also a population of peers who never return to the system and thus their downtime cannot be defined. We reserve studying these issues for future work.

Correlation in Session Time: Another interesting question is "How correlated are session times across different appearances of a single peer?" Characterizing such correlations illustrates whether past session times of a peer are good predictor of future session times. For example, in Gnutella, a peer can promote itself to an Ultrapeer earlier if it can reliably estimate that its remaining uptime is long. Given all pairs of consecutive session times for each peer in our measurement window, Figure 9 presents the distribution of the median of the second session times for all peers with first session time x. In other words, we have $(n-1)$ pairs of consecutive sessions for peer p if it appears n times during our measurement. These figures show that there is a strong correlation between consecu-

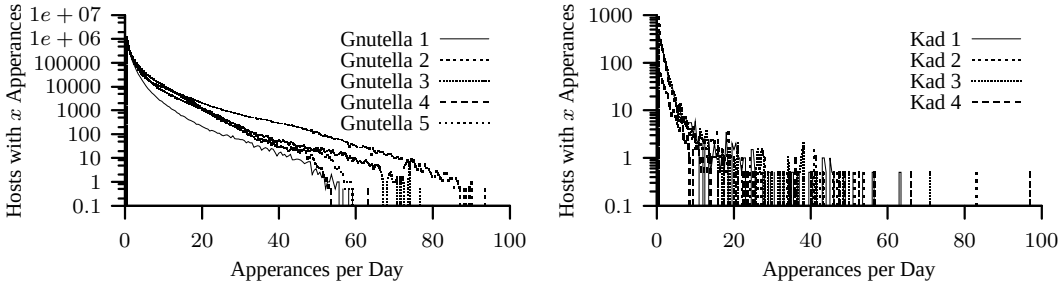
[10]Note that the analogous quantity for session time (*i.e.*, the fraction of very long-lived peers) is easy to measure since the peers are actually present in the system. This fraction is shown by the gap between the highest point and 100% in Figure 3.



(a) Gnutella (based on IP address)

(b) Kad (based on node ID)

Figure 10: Correlation in Availability



(a) Gnutella (based on IP address)

(b) Kad (based on node ID)

Figure 11: Appearances

tive session times (based on peer ID) in Kad, and (based on IP address) in Gnutella. However, session times in BitTorrent do not exhibit such a correlation. This result is not surprising because the pattern of client participation in BitTorrent is likely to be different from Kad or Gnutella. A BitTorrent client tries to download a desired file, possibly during multiple sessions, and leaves. While lingering time across all peers follows a power-law distribution as shown in Figure 2, once a peer has completed a download it has no motivation to return. In contrast, both Kad and Gnutella are file-sharing applications where clients casually join the system, possibly on a regular basis, to search for various contents, and may also download the content. Therefore, their typical session time is more likely to exhibit correlation across different appearances. *In summary, the main implication of this finding is that the past session time of individual clients or IP addresses is a good predictor of their next session time in both structured and unstructured file-sharing applications.*

Another important issue is the correlation between non-consecutive session times and any potential impact the gap between sessions might have on the correlation. Examination of this issue requires a much longer dataset (*e.g.*, across a few weeks) and remains as a future work item.

Correlation in Availability: Similar to the correlation in session time, we can examine the correlation in availability of participating peers across consecutive windows of time. The availability of a peer is the portion of a window that the peer has been available in the system. For example, a node with 50% availability during one day, might appear just once for 12 hours, or appear 4 times and stay 3 hours during each appearance. In essence, the availability of a peer during a window is a coarse measure that represents the peer's overall degree of participation during that window regardless of its pattern of appearance. In the previous subsection, we focused on the fine-grained relationship between consecutive sessions. In this subsection, we examine availability which captures a coarser view of the relationship between sessions, regardless of the number of sessions.

To study peer availability, we divide each two-day dataset into two windows of one-day length, and examine the correlation between the availability of peers in two consecutive days. Figure 10 depicts the median availability in the second day for all peers with the availability of $x\%$ in the first day. These results show that a strong correlation exists between availability of individual clients (in Kad) and individual IP addresses (in Gnutella) across two consecutive days. Bhagwan *et al.* [2] show the availability of different peers are mostly independent. It is worth noting that this is orthogonal to our result, since we examine the correlation in availability of a *single peer*

Listing 1 Discrete Event Simulator

```
1  now = 0
2  eventq = []
3
4  def arrival(now):
5      num_peers += 1
6      heappush(eventq,
7               (now + random_session(),
8                departure))
9      heappush(eventq,
10              (now + random_interarrival(),
11               arrival))
12
13 def departure(now):
14     num_peers -= 1
15
16 heappush(eventq,
17          (random_interarrival(), arrival))
18 while True:
19     now, func = heappop(eventq)
20     func(now)
```

across multiple measurement windows.

To further explore the underlying dynamics of the availability in Gnutella and Kad, we present the distribution of appearances per day across all peers in Figure 11. This figure shows that more than half the peers in both systems appear at most once per day while a very small number of clients may return to the system more often (up to 80 times per day). However, since the number of peers per Gnutella snapshot is significantly larger than in our Kad snapshots, this has increased the number of peers in each group. Furthermore, shared IP addresses among participating peers could be another factor that increases the number of appearances of an IP address in Gnutella.

In summary, the availability of individual clients or specific IP addresses, across two consecutive days, are strongly correlated. Furthermore, most observed peers appear only once per day.

4 Simulating Churn

In this section, we discuss how our findings can be used to accurately simulate churn in a P2P network. A typical goal in a simulation is to generate churn by generating arrival and session times[11] for individual peers such that the average group converges to a desired size, P, often specified based on the available resources of the simulator or the expected population size for a P2P system. There are two ways to model churn:

- *Retaining Peer Identity*, where each peer may leave and re-join the system multiple times during a simulation

[11]The departure time of a peer is simply determined by adding its arrival and session times.

Symbol	Meaning
User Inputs	
P	Target Mean Simulation Population
Empirical inputs	
α	Power-Law Parameter
f	Fraction of Peers with Uptime $\leq t_{max}$
t_{min}	Minimum Modeled Session Time (2 min)
t_{max}	Maximum Modeled Session Time (1 day)
Functions and Computed Values	
x	A value generated uniformly at random
$s(t)$	Session Time Probability Density Function
$S(t)$	Session Time CDF
$S^{-1}(x)$	The inverse of the Session Time CDF
A	Proportionality Constant
$s_u(t)$	Time-Weighted Session Time Probability Density Function
m	Mean Remaining Uptime
r	Mean Arrival/Departure Rate

Table 4: Variable and Function Definitions

- *Ignoring Peer Identity*, where no state is retained about a peer once it departs (*i.e.*, returning peers are modeled as new peers)

Modeling the first scenario is challenging since it requires not just uptime and downtime distributions, but a comprehensive model of how peers arrive for the first time and how often peers leave never to return. In other words, it must also model the *lifetime* of peers, which is beyond the scope of this paper. We therefore focus on the second, simpler case which only requires the inter-arrival and session time distributions.

The basic simulation methodology is to construct a Discrete Event Simulator based on a priority queue (*i.e.*, a heap) as shown in Listing 1. Each element in the queue includes a future event, such as a peer arrival or departure, and a scheduled time for that event to occur. We prime the queue with a single arrival event (line 16). Processing each arrival event requires the following steps: *(i)* generating a new peer, *(ii)* determining the peer's session time in order to schedule its departure event (line 6), and *(iii)* scheduling the next arrival event (line 9). Two functions, `random_session()` and `random_interarrival()`, generate samples of the session and inter-arrival distributions. The following subsections explain the implementation of these functions to reproduce the models derived in Section 3 (*i.e.*, power-law and Poisson, respectively). Table 4 summarizes the list of variables used in this section.

Session Time: To generate proper session time values, *i.e.*, define `random_session()`, we need to address the following issue: the session time distribution has three distinct regions (as shown in Section 3.1). These regions are as follows: sessions less than two minutes,

sessions between two minutes and one day, and sessions longer than one day. For reasons of brevity and ease of exposition, we ignore the peers with session durations less than two minutes. Our model could be extended to include these peers without much difficulty. Our data is not comprehensive enough to model sessions longer than one day accurately. However, in many cases simulations will be much shorter than one day. Therefore, we treat these long-lived peers as a fixed (*i.e.*, static) portion of the population that will be present in the system for the entire simulation. The accuracy of this simplification decreases for simulations that last more than a few days. The observed percentage of peers (f) with session time of one day or less are given in the CDFs in Figure 3. The number of fixed peers is $P(1-f)$. For example, in the Gnutella 5 dataset, we have $f = 93\%$, and 7% of P is fixed. When developing an actual protocol, a range of different input values should be selected to ensure robustness against slight changes in the simulation model. When designing a protocol that leverages long-lived peers, it may be best to conservatively assume $f = 100\%$, and there are no fixed peers.

We now turn our attention to modeling churn for the Pf peers with session time between $t_{min} = 2$ minutes and $t_{max} = 1$ day. Toward this end, we can convert a uniformly random number between 0 and 1 to a session time in the range $[t_{min}, t_{max}]$ using the inverse of the power-law cumulative distribution for session time *i.e.*, the function that maps $[0,1] \rightarrow [t_{min}, t_{max}]$. To derive the inverse CDF, recall from Section 3.1 that the the session time density function is given by: $s(t) \propto t^{-\alpha}$ The session time cumulative distribution function is simply the integral of the density function:

$$S(t) \propto \int_{t_{min}}^{t} t^{-\alpha}\, dt \propto t^{-\alpha+1} - t_{min}^{-\alpha+1}$$

To make it a proper cumulative distribution, we must also compute the proportionality constant, $A = \frac{1}{S(t_{max})}$, to satisfy:

$$S(t_{max}) = A\left(t_{max}^{-\alpha+1} - t_{min}^{-\alpha+1}\right) = 1$$

Finally, we reverse the function through algebraic manipulation so that it accepts x (selected uniformly at random between 0 and 1) as an input, and generates a session time:

$$S^{-1}(x) = \exp\left(\frac{\log\frac{x + A t_{min}^{-\alpha+1}}{A}}{-\alpha+1}\right)$$

Inter-Arrival Time: Generating Poisson inter-arrival times for `random_interarrival()` is straightforward given the mean inter-arrival rate, r. The standard formula $t = \frac{-\log x}{r}$ translates the uniformly random variable x into the Poisson variable t. To ensure a stable group size in steady state, the arrival rate must be equal to the mean departure rate, $\frac{P}{m}$, where m denotes the mean session duration *of peers in the system*. The difficulty is in computing m.

Listing 2 Random Generators

```
A = 1.0 / (tmax**(-alpha+1)
           - tmin**(-alpha+1))

m = ((-alpha+1)/(-alpha+2)
     * (tmax**(-alpha+2)-tmin**(-alpha+2))
     / (tmax**(-alpha+1)-tmin**(-alpha+1)))

def random_interarrival():
    return -log(random()) / ((P*f)/m)

def random_session():
    p = random()
    t = exp(log((p+A*tmin**(-alpha+1))/A)
            / (-alpha + 1))
    return t
```

To compute the mean session duration (m) of peers in the system, we divide the sum of the time-weighted session times by the number of peers that generated those sessions. This can be thought of as dividing the total time each peer is up by the number of peers. Recall the time-weighted session time distribution is given by:

$$s_u(t) \propto \int s(t)\, dt \propto t \cdot t^{-\alpha}$$

Then, dividing the integral of $s_u(t)$ by the integral of $s(t)$ to compute m gives us:

$$\begin{aligned} m &= \frac{\int_{t_{min}}^{t_{max}} s_u(t)\, dt}{\int_{t_{min}}^{t_{max}} s(t)\, dt} = \frac{\int_{t_{min}}^{t_{max}} t \cdot t^{-\alpha}\, dt}{\int_{t_{min}}^{t_{max}} t^{-\alpha}\, dt} \\ &= \frac{-\alpha+1}{-\alpha+2} \cdot \frac{t_{max}^{-\alpha+2} - t_{min}^{-\alpha+2}}{t_{max}^{-\alpha+1} - t_{min}^{-\alpha+1}} \end{aligned}$$

Finally, we can compute the mean rate of departure, after subtracting out the fixed peers (which never depart): $r = \frac{Pf}{m}$.

To summarize, Listing 2 presents an implementation of our routines to randomly generate the inter-arrival and session time values as pseudo-code. We have also implemented this code in a small simulator to empirically validate its accuracy. Our results show that after a t_{max} warm-up period, the population stabilizes around the desired mean group size P.

5 Design Implications

In this section, we discuss a couple of key implications of our findings on the design of churn-aware P2P applications. To gracefully cope with churn, information, including both state (*e.g.*, routing) or content, should be

maintained at stable peers. Otherwise, the overhead of updating state in the system can become expensive, or the availability of stored content can be affected. The fundamental design question is "How to reliably identify stable peers?" Our results show that the naive approach of randomly selecting from an accumulated list of observed peers during a session is more likely to capture short-lived peers because the distribution of session time among all sessions follows a power-law similar to Figure 1. Our findings suggest two more reliable strategies for identifying long-lived peers are as follows: *(i)* randomly selecting participating peers within a short period of time is likely to capture some long-lived peers since the distribution of session time among peers *up at any moment* (Figure 3) is weighted more heavily towards long-lived peers, or *(ii)* weight observed peers by the number of times they are observed. The key point is that observations made *over time* becomes skewed towards the larger number of short-lived peers if we do not weight the observations back in favor of the long-lived peers. For example, if a peer has a few neighbors in a (structured or unstructured) P2P overlay that it periodically interacts with, the neighbor peer with highest uptime is more likely to have a longer remaining uptime. Therefore, it is more useful to keep information about these more stable neighbors, and store content at these neighbors.

Our results indicated that 10%–20% of peers at any moment have an uptime longer than one day. This motivates a scalable and low-cost bootstrapping mechanism for existing peers in the system (not for first comers) that does not require any well-known bootstrapping node as follows: Each peer can select and cache IP addresses of several long-lived peers during a session using the strategies we described earlier. To connect to the system at any later time, each peer can contact cached long-lived peers to locate a participating peer in the system. Maintaining a sufficiently large number of *long-lived peers* ensures that each peer with a high probability can find another peer to bootstrap into the system without any need to contact a centralized bootstrapping node, and without having to waste time trying to contact many short-lived addresses.

Finally, the strong correlation in consecutive session times (or availability) of clients (in Kad) implies that a P2P application can properly estimate the duration of its session time (or availability per day) based on its last observed behavbior. The advantage of this approach is that it does not require the application to wait for a while to measure uptime as a sign to predict user behavior for the rest of the session.

Our finding that session times are independent of download time in BitTorrent has interesting consequences. The power-law session duration results in a significant population of peers in a swarm who are lingering after their download has completed. Since these peers are donating resources to the system without using further resources, they significantly increase the performance of other peers in the swarm. In other words, *power-law session times implicitly lead to good performance for peer-to-peer file distribution*.

6 Related Work

We are not aware of any study that has exclusively examined different aspects of churn in P2P networks. However, several studies have presented a passing investigation of session time as part of wider characterizations of P2P applications. We divide these studies into two groups based on their measurement technique: *(i)* passive monitoring of P2P traffic and *(ii)* active probing.

As part of a study on P2P flows in a large ISP network, Sen *et al.* [16] use passive measurement at several routers to monitor flows in FastTrack, Gnutella, and Direct-Connect. They present a CDF of the duration an IP address is active (the "ontime"), based on a threshold, $\delta = 30$ minutes, of inactivity. They show that the ontime is heavy-tailed, but does not follow a Zipf distribution when ranked. As part of a study on workload characterization in Kazaa, Gummadi *et al.* [5] present session durations based on passive monitoring of a router at the University of Washington. They found that session durations are heavy-tailed, with a median session length of 2.4 minutes while the 90th percentile is 28.25 minutes. In general, passive monitoring techniques to characterize churn will underestimates session times when peers are not generating traffic through the observation point. The difficulty in correctly identifying peer-to-peer flows [9] can also limit measurement accuracy. Furthermore, it is difficult to determine whether the subset of users monitored with passive measurement is representative of the entire P2P user population, especially if data is collected at a small number of measurement points.

Several studies use active probing or crawling to characterize P2P networks and present the behavior of session time across peers. While studying the locality of files in Napster and Gnutella, Chu *et al.* [3] present the session time distribution based on probing and fit it to a log-quadratic distribution[12]. In their insightful characterization of peers in Napster and Gnutella, Saroiu *et al.* [15] present a CDF of session durations based on probing, showing session durations are heavily skewed. They also present CDFs of peer availability in these systems. More recently, Liang *et al.* [10] similarly provide a CDF of session durations for Supernodes in the Kazaa network, based on active probing. Finally, Bhagwan *et al.* [2] examine availability in the Overnet DHT using probing. They show that IP address changes are fairly

[12]This distribution is defined by $p \propto \exp(\alpha_1 (\log x)^2 + \alpha_2 \log x)$. and can be thought of as a second-order power-law.

common in these systems and there is little correlation between the availability of different peers (*i.e.*, that each peer acts independently).

There have also been a few studies on BitTorrent [13, 7], using tracker logs which allow for high accuracy. Pouwelse *et al.* [13] show that session times follow a Zipf distribution when ranked, except for the longest-lived peers[13]. Izal *et al.* [7] study a 5-month tracker log from the Red Hat 9 ISO images and present a CCDF of session times and show that they are heavily skewed. We obtained a copy of their data and use it in our analysis.

Our study differs from these prior work in several ways. Previous studies often present only the distribution of session time which is not sufficient to simulate churn. Furthermore, they neither investigate other aspects of churn nor conduct any comparison across different classes of P2P applications, as we do in this paper. More importantly, none of the previous studies on P2P file-sharing applications have shown that they have captured a representative population of sessions in their target system. We believe that the lack of representative sampling could be the main contributing factor in the different results reported by previous studies (*e.g.*, dramatically different median session time ranging from 1 minute to an hour [14]). More specifically, these studies do not distinguish sampling peers from the total session population, $s(t)$, and sampling peers within a specific snapshot, $s_u(t)$.

7 Conclusions and Future Work

This paper presented a detailed characterization of churn in three different classes of P2P systems: Gnutella, Kad and BitTorrent. Our measurement methodology leverages application-specific features to capture representative sessions in each system. In particular, we have shown that it is critical to differentiate sampling from all *sessions* and sampling from all *peers*. We characterize both group-level as well as client- and IP-level characterizations of churn. Our main findings, listed in Section 1, present new insights into peer dynamics that can be used in the design and evaluation of churn-aware P2P applications. Finally, we show how our findings can be incorporated for simulating churn.

We are pursuing this work in several directions, primarily based on collecting longer datasets from Gnutella and Kad. First, we intend to more closely study the correlations between consecutive sessions over longer time scales. Second, we plan to conduct more detailed characterizations of downtime and changes in the user population (up or down) to develop simulation models for multiple appearances. Finally, we are exploring the implications of churn on existing P2P systems. We have already conducted a limited analysis of how churn affects the overlay topology structure in Gnutella [19].

[13]We note that the Zipf approach of plotting values based on their rank compresses the typically sparse data points of measurements of the tail of a power-law distribution.

References

[1] eMule. http://www.emule-project.net, 2005.

[2] R. Bhagwan, S. Savage, and G. Voelker. Understanding Availability. In *International Workshop on Peer-to-Peer Systems*, 2003.

[3] J. Chu, K. Labonte, and B. N. Levine. Availability and Locality Measurements of Peer-to-Peer File Systems. In *ITCom: Scalability and Traffic Control in IP Networks II Conferences*, July 2002.

[4] S. Floyd, T. Henderson, and A. Gurtov. The NewReno Modification to TCP's Fast Recovery Algorithm. RFC 3782, Apr. 2004.

[5] K. P. Gummadi, R. J. Dunn, S. Saroiu, S. D. Gribble, H. M. Levy, and J. Zahorjan. Measurement, Modeling, and Analysis of a Peer-to-Peer File-Sharing Workload. In *SOSP*, 2003.

[6] Y. hua Chu, S. G. Rao, S. Seshan, and H. Zhang. A Case for End System Multicast. *IEEE Journal on Selected Areas in Communication (JSAC), Special Issue on Networking Support for Multicast*, 20(8), 2002.

[7] M. Izal, G. Urvoy-Keller, E. W. Biersack, P. A. Felber, A. A. Hamra, and L. Garces-Erice. Dissecting BitTorrent: Five Months in a Torrent's Lifetime. In *PAM*, Apr. 2004.

[8] V. Jacobson. Congestion Avoidance and Control. In *SIGCOMM*, pages 314–329, Aug. 1988.

[9] T. Karagiannis, A. Broido, N. Brownlee, K. Claffy, and M. Faloutsos. Is P2P dying or just hiding? In *Globecom*, Nov. 2004.

[10] J. Liang, R. Kumar, and K. W. Ross. The KaZaA Overlay: A Measurement Study. *Computer Networks Journal (Elsevier)*, 2005.

[11] D. Liben-Nowell, H. Balakrishnan, and D. Karger. Analysis of the Evolution of Peer-to-Peer Systems. In *Principles of Distributed Computing*, July 2002.

[12] P. Maymounkov and D. Mazieres. Kademlia: A Peer-to-peer Information System Based on the XOR Metric. In *International Workshop on Peer-to-Peer Systems*, 2002.

[13] J. Pouwelse, P. Garbacki, D. Epema, and H. Sips. The Bittorrent P2P File-sharing System: Measurements and Analysis. In *International Workshop on Peer-to-Peer Systems (IPTPS)*, Feb. 2005.

[14] S. Rhea, D. Geels, and J. Kubiatowicz. Handling Churn in a DHT. In *USENIX*, pages 127–140, 2004.

[15] S. Saroiu, P. K. Gummadi, and S. D. Gribble. Measuring and Analyzing the Characteristics of Napster and Gnutella Hosts. *Multimedia Systems Journal*, 8(5), Nov. 2002.

[16] S. Sen and J. Wang. Analyzing Peer-To-Peer Traffic Across Large Networks. *IEEE/ACM Transactions on Networking*, 12(2):219–232, Apr. 2004.

[17] K. Sripanidkulchai, B. Maggs, and H. Zhang. An Analysis of Live Streaming Workloads on the Internet. In *Internet Measurement Conference*, Oct. 2004.

[18] D. Stutzbach and R. Rejaie. Capturing Accurate Snapshots of the Gnutella Network. In *Global Internet Symposium*, pages 127–132, Mar. 2005.

[19] D. Stutzbach, R. Rejaie, and S. Sen. Characterizing Two-Tier Overlay Topologies in Modern P2P File-Sharing Systems. Technical Report CIS-TR-2005-01, University of Oregon, Feb. 2005.