Churn: a Key Effect on Real-World P2P Software

Cheng-Yun Ho
Department of Computer Science
National Chiao Tung University
Hsinchu, Taiwan
cyho@cs.nctu.edu.tw

Ming-Chen Chung
Department of Computer Science
National Chiao Tung University
Hsinchu, Taiwan
mcchung@cs.nctu.edu.tw

Li-Hsing Yen
Department of Computer Science and Information Engineering
National University of Kaohsiung
Kaohsiung, Taiwan
lhyen@nuk.edu.tw

Chien-Chao Tseng
Department of Computer Science
National Chiao Tung University
Hsinchu, Taiwan
cctseng@cs.nctu.edu.tw

***Abstract*—Churn refers to a large number of arriving and departing participants within a short time in peer-to-peer (P2P) networks. This paper studies the impact of churn on real-world unstructured P2P networks. To this end, we need collecting snapshots of P2P network topology and associated performance metrics. Because P2P topology changes dynamically, the time to take a snapshot must be sufficiently short for the snapshots to be accurate. We propose Third-party-to-servent Crawling with Servent-to-servent Sampling (TCSS) system. TCSS uses a third-party crawling technique to collect network topology information without disturbing the original P2P network under investigation. Furthermore, TCSS adopts distributed and parallel techniques to speed up the crawling process. TCSS also employs a servent-to-servent sampling technique to gather the corresponding performance metrics of the P2P network simultaneously. Empirical results show that TCSS takes around 7 minutes to take a topology snapshot of the P2P network. Besides, we found that churn is indeed a combined effect of peer arrivals/departures and neighbor replacements. As the number of peers increases, the number of very long-lived peers remains nearly constant and the P2P network possesses a small-world property. Moreover, as churn aggravates, the average booting time of peers increases and the variation is proportional to the degree of churn. The response time of the top-rank keyword searches is not affected by the degree of churn.**

Keywords-Churn; Real-world; Unstructured; Peer-to-peer; Performance; TCSS

I. INTRODUCTION

Over the past decade, peer-to-peer (P2P) applications have increased greatly in popularity and grown aggressively in Internet traffic. Unlike traditional client-server model, where the whole processing load is placed on centralized servers, peers in a P2P network share contents with each other in a distributed manner. For this reason, peers in P2P networks are also known as servents (servers and clients), possessing properties of both a server and a client. Typical P2P applications include file-sharing and multimedia streaming.

Many approaches have been proposed to enhance P2P network performance. These approaches include particular network structures, efficient search algorithms, and special mechanisms based on different overlay structures. However, many mechanisms do not perform well under real-world conditions. It is believed that the performance problem is caused by a phenomenon named churn, which refers to a large number of independent arriving and departing servents within a short time. This phenomenon results from unpredictable servent behaviors possibly due to network conditions, user behaviors, or servent overloading. Churn exists in a real-world environment but it is hard to precisely analyze churn in a simple analytical model.

Overlay is a logical networking structure built on top of P2P networks that provides connectivity and efficient route for P2P messages. Churn may affect the main functionality of the overlay and significantly degrade the performance of P2P applications. Researchers have analyzed servent behaviors [2] to study the cause of churn, but the performance impact of churn on overlay networks has not yet been considered. From [20], we know that structured file-sharing P2P has been well-studied. For further understanding of the correlation between churn and P2P overlay performance, we focus on analyzing how churn affects real-world unstructured file-sharing P2P software.

Each servent maintains a servent neighbor list to keep track of its neighboring servents. To keep the neighbor list updated, each servent removes neighbors that are no longer reachable from the list, and attempts to contact potential new neighbors. This process changes the overlay topology over time. The goal of this study is to figure out how such changes affect the performance of the overlay under a real-world environment. To this end, we need two datasets: a sequence of topology snapshots and the performance metrics of a specific topology. We therefore developed Third-party-to-servent Crawling system with Servent-to-servent Sampling (TCSS). TCSS integrates the crawler presented in [4] with accurate snapshot capture techniques to record overlay topologies as graphs. TCSS also deploys some customized P2P peers called modified servents which cooperatively run a sampling technique [3] to gather particular performance metrics. The combination of crawler and modified servent has not ever been implemented before. Consequently, TCSS can efficiently and accurately collect topology snapshots and performance metrics (such as initial boot time and search response time) for P2P overlay. We have used the data collected by TCSS to study the correlation between fluctuating topologies and corresponding performance metrics.

In summary, our goals are to accomplish the following tasks under a real-world environment: (i) datasets collection and (ii) analysis of overlay performance under various overlay topologies. The rest of this paper is organized as follows. Section 2 introduces our target P2P

overlay, Gnutella [8]. We present relevant works on dataset collection in Section 3. Section 4 describes TCSS in details and Section 5 shows our analysis results. The last section concludes this paper and discusses our future work.

II. BACKGROUND

There are several criteria to select a real-world P2P overlay for our study. First, the overlay type must be unstructured and file-sharable. Second, the overlay should be of large size, as a popular overlay will provide us rich data and thus statistically significant results. Third, the overlay protocol must be well-designed and well-defined with clear descriptions and open documentation. Thus, we decided to conduct our empirical study on the Gnutella overlay based on a number of its unique features.

A. Unstructured File-Sharing P2P and High-Popularity

Gnutella is widely regarded as the first fully distributed file-sharing P2P and is one of the most popular P2P applications [21]. From its inception in 2000, Gnutella is one of the most studied P2P networks in the literature. There are many measurement studies [3, 5, 7, 13-15] and tools [9-12, 16-17, 22] focusing on the Gnutella overlay, and these works have helped us gain an insight about the characteristics of Gnutella.

B. Well-Designed P2P Protocol

The Gnutella protocol is defined in RFCs [8] and its mechanisms are well-documented [18]. For scalability, current Gnutella (version 0.6) adopts a two-tier overlay architecture. As shown in Figure 1, there are two types of peers: top-level peers and leaf peers. Top-level peers include ultrapeers and legacy peers. High-performance ultrapeers handle connections and query requests from other peers. Leaf peers constitute the majority of the members in the Gnutella overlay; they connect to the overlay through a few ultrapeers. High-bandwidth and high-capacity leaf peers become ultrapeers if needed in order to maintain a proper ultrapeer-to-leaf ratio. This two-tier architecture of Gnutella greatly reduces the traffic caused by the flooding mechanism in traditional unstructured P2P architectures. Moreover, Gnutella provides a protocol hook that allows shared file lists and neighbor lists [19] to be easily extracted from a peer through a method called crawler protocol. These features enable users to learn of information about other peers. From this information, we can construct necessary datasets of the Gnutella overlay.

III. RELATED WORKS

A. Dataset Collection Techniques

The authors in [2] classified prior collection methods into two groups based on how the peers interact with these methods. These groups are *passive monitoring* and *active probing*.

1) Passive Monitoring

Passive data collection techniques gather datasets by tracking logs in static devices, such as routers in large ISP networks or trackers in BitTorrent. In [6], the authors extracted logs of specific P2P protocols according to port number information in flow-level logs recorded at multiple border routers across the ISP's network. This approach misses some data since P2P applications may randomly generate port numbers. In addition, this technique finds only partial overlay topology since not all routers are self-controlled. Logs from the routers and the trackers may also be fragmental because some peers may only generate intermittent traffic throughout the observation period. These limitations degrade the accuracy of the measurements made by passive monitoring.

2) Active Probing

Active probing uses a crawling technique similar to a web spider that retrieves a list of links to other web pages. This crawling technique enables us to connect to overlay peers actively and then send requests to retrieve information of interest. Unlike passive monitoring, active probing has the ability to contact each peer on the overlay as long as it has the peer's address information. Active probing can gather more comprehensive topological information when compared with passive monitoring.

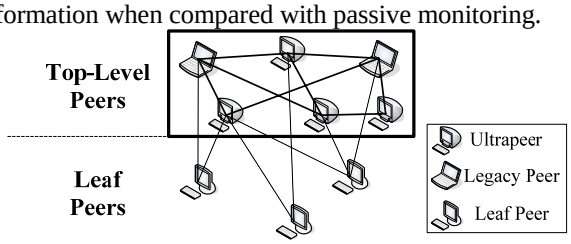

Figure 1. Two-Tier Topology of Modern Gnutella

B. Active Probing Approaches

As mentioned, we use active probing to collect dataset from the Gnutella overlay. There are two different crawling techniques for active probing, which are discussed below.

1) Third Party Crawler

This type of crawler collects datasets via some crawler protocol supported by Gnutella. For example, the crawler in [4] takes a snapshot of the overlay topology by progressively exploring the Gnutella overlay, querying peers for their neighbor lists, and adding fresh peers to the crawler's ready queue for the next round of crawling. Two kinds of retrievable data are defined in the crawler protocols: the neighbor list and the file-sharing list. The crawler is not a P2P peer. Therefore, it can be implemented with less overhead and can be customized to achieve high efficiency.

2) Modified Servent

The modified servent utilizes inter-peer messages to gather topological data. Different from the third party crawler, modified servent is a P2P peer and it can collect not only topological overviews but also performance related data through interactions with other overlay peers. However, this approach can incur high execution latency due to time-consuming peer interactions [22].

C. Speed-up Techniques for Accurate Datasets

The most common approach to measurement-based analysis of P2P system is to capture the overlay snapshot via the active probing technique. In addition, our goal is to analyze the impact of churn, a dynamically changing phenomenon, on overlay performance. To identify the change of churn in a given sequence of overlay snapshots, the intervals between consecutive snapshots must be of

similar size for a convincible result. Moreover, because of the large size of the overlay and the dynamic nature of P2P peers, snapshots taken with a long capturing period will be distorted. Thus, we need techniques that can shorten the time period of snapshot capturing.

1) Gnutella Overlay Protocol

a) Support of Crawler Protocol

Gnutella peers implement a special handshake feature designed to facilitate crawling, allowing a quick query to be sent to a peer for connectivity information. This allows the crawler to discover the addresses of the peer's neighbors, and for an ultrapeer, the addresses of its leaves.

b) Two-Tier Architecture

Since each leaf connects to an ultrapeer and there is no direct connection between leaves, we can capture all the nodes and links of the overlay by contacting only top-level peers. Furthermore, the high degree of peer connectivity within the top-level peers substantially increases the probability of detecting new ultrapeers.

2) Distributed System Architecture

The crawler employs a client-server architecture to achieve high-degree concurrency and to effectively utilize available resources on multiple workstations. A server process coordinates multiple client processes that act as virtual independent crawlers. Each crawler explores a particular portion of the network and all crawlers work in parallel. The clients are also responsible for detecting duplicated peer addresses in the reported neighbor lists. All newfound peers are reported back to the server, and the server dispatches these new peers to the clients for the next crawling.

3) Asynchronous I/O

Each client crawls hundreds of peers in parallel using asynchronous I/O. Servers also implement an adaptive load management mechanism to ensure that the clients' processes do not become overwhelmed. We adopt this speed-up technique in our FreeBSD and Linux workstations.

4) Sampling Technique

Contacting every peer on the overlay is very time consuming. A sampling technique finds out representative peers within a large scale P2P overlay. It greatly reduces the crawling time by contacting only a group of representative peers. A study in [3] presents a sampling technique named Metropolized Random Walk with Backtracking (MRWB) that shows a nearly unbiased selection of representative peers. MRWB can correct the bias towards high degree peers and cope with departed peers within a highly dynamic P2P overlay.

IV. THIRD-PARTY-TO-SERVENT CRAWLING SYSTEM WITH SERVENT-TO-SERVENT SAMPLING

A. Datasets vs. Active Probing Approaches

Since churn dynamically changes overlay topology, we want to track such changes in order to figure out the correlation between churn and the overlay performance. A sequence of topology snapshots provides us the scenario how the Gnutella overlay topology changes over time. Our task involves two datasets: a sequence of topology snapshots and the performance metrics of a specific topology. We used the active probing approach to gather these datasets. Section 3 already presents two approaches to active probing (the third party crawler and the modified servent). Each approach has its own features for dataset collection. In the following paragraphs, we analyze both the advantages and shortcomings of these two approaches and point out the design concept of our system.

1) Topology of Gnutella Overlay

Both the third party crawler and the modified servent can capture the snapshot of the overlay topology. However, they differ in some ways. Here, we look at the two approaches in details.

a) Third Party Crawler

The third party crawler is implemented using the Gnutella crawler protocol. It works as follows: (1) The server gives bootstrapping peers to the crawler. (2) The crawler queries peers for neighbor lists via crawler protocol. (3) The crawler extracts fresh peers from the reported neighbor list. (4) The crawler repeats Steps (2) and (3) until no new peers can be discovered.

The third party crawler can be customized for topology collection, and it does not participate in the normal operations of the Gnutella network. This feature allows us to implement the third party crawler with low overhead for the lack of time-consuming peer interactions. Hence, the third party crawler can achieve higher efficiency than the modified servent.

b) Modified Servent

The modified servent operates using Gnutella inter-servent messages. It works as follows: (1) Modified servents connect to ultrapeers and request for services. (2) The ultrapeers broadcast requests via inter-servent messages over the Gnutella overlay. (3) Peers that receive request messages reply. (4) On receiving peer's replies, ultrapeers forward the replies back to the modified servent.

Current version of Gnutella uses several mechanisms to minimize the number of request messages and prevent the Internet from being flooded with broadcasting messages. For this reason, the topological information gathered by modified servent is fragmented and does not form a full-scale view. Furthermore, the modified servent is inefficient due to time-consuming peer interactions.

For the reasons stated above, we choose the third party crawler to capture the topology dataset. We implemented the third party crawler with several speed-up techniques that accelerate the process of data collection, minimizing the distortion of topology caused by long crawling period. We call the captured topology dataset crawler-to-servent dataset.

2) Performance Metrics

Since the third party crawler is not a servent in the Gnutella overlay, we can only gather metrics of overlay performance via the modified servent. We name the performance metrics dataset gathered by the modified servent servent-to-servent dataset. We enhanced the modified servent by adding the sampling technique to shorten the crawling period. However, there is no way to capture overlay topology via the sampling technique. We need complete topologies to identify the changing factors within each overlay snapshot.

In short, TCSS uses the third party crawler with speed-up techniques to capture overlay topology and the

modified servent with the sampling technique to gather performance metrics.

B. TCSS Overview

TCSS is designed to work on the Gnutella overlay. As shown in Figure 2, TCSS consists of a third-party crawler and modified servents. The third-party crawler contains three components: dispatching/sampling server, crawling clients, and central repository. The modified servents are Gnutella servents modified to gather performance metrics in our analysis. The datasets collected by TCSS are the crawler-to-servent dataset and the servent-to-servent dataset.

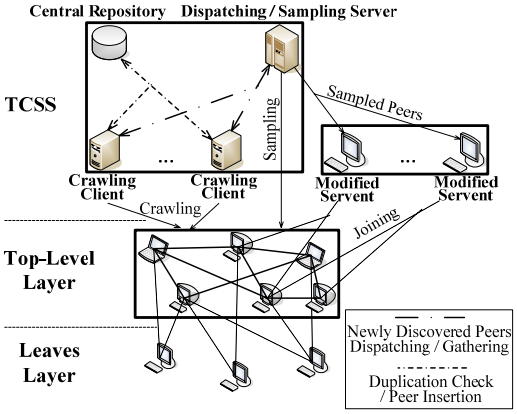

Figure 2. TCSS Architecture

The data-collecting work consumes considerable computing resources. As an improvement, we implement TCSS with divide and conquer concepts. The main part of TCSS is built upon distributed architecture. The dispatching/sampling server can master several independent crawling clients and modified servents, and assign them jobs depending on their functionalities. Both the crawling client and modified servents adopt asynchronous I/O for enhancing the parallelism when handling network communication.

C. Fast Fresh-Peer Refinement

Our experience in implementing the crawling function in TCSS revealed that numerous peers found in reported neighbor lists are duplicated. It therefore becomes a challenge to extract fresh peers in an efficient way. We devised two approaches that have been incorporated into crawling clients to deal with this problem.

1) Cache Mechanism

Since the number of neighbor lists to be reported is enormous, identifying and eliminating duplicated data can significantly reduce processing cost. To facilitate duplication checks and shorten the crawling period, it is necessary to have a cache in each client.

2) Multiprocess Programming

We augmented the clients with multiprocessing abilities to accelerate their crawling tasks. Each client has a main process that creates child processes called *workers*. Workers check whether peers in the reported neighbor lists are duplicated. Cache for duplication check is divided into two layers. The first-layer cache, maintained by the main process, stores all the peers that have ever been explored. The second-layer cache is maintained by each worker and synchronized with the first-layer cache periodically. The use of multiprocessing and divided caching enables a parallel duplication check and thus shortens the whole crawling process.

D. TCSS Design Details

In this section, we interpret the functions of the four components in TCSS.

1) Dispatching/Sampling Server

a) Fresh Peers Management

This server maintains fresh peers reported by the crawling clients. The server also manages the queues for each client and dispatches fresh peers to client's queues. It balances the load among crawling clients by controlling the sending rate of fresh peers to clients.

b) Representative Ultrapeers Sampling/Distributing

When clients crawl the overlay, the server extracts representative ultrapeers from the overlay simultaneously. A list of these sampled peers will be forwarded to the modified servents, which will then attempt to interact with these sampled peers to gather the servent-to-servent dataset.

2) Crawling Clients

a) Phase 1 – Neighbor List Query

Crawling clients know the presence of fresh peers from the server and contact these peers to capture the crawler-to-servent dataset. A two-tier overlay architecture eases the crawling task, since all leaf peers connect to the overlay through a small number of ultrapeers. This means that we can crawl only the top-level layer to capture the whole overlay topology. To avoid redundant crawling, the next phase will detect and remove peers found duplicated in the reported neighbor list.

b) Phase 2 – Duplication Check

Phase 2 is implemented using multiprocess programming and cache mechanism. Each process, called worker, parses out peer addresses from the reported neighbor lists and checks for duplicated peers. Crawling clients first check their own caches and, if no duplication is found, then the central repository. After the check, a list of fresh peers will then be forwarded to the server and inserted into the database of the central repository.

3) Central Repository

The Central Repository is a central database of crawled peers.

4) Modified Servents

We choose LimeWire [18] as our target servent. The modified servents are under the server's control. As the server gets sampled results, it will feed the results to modified servents. The modified servents then use the sampled peers to join and interact with the Gnutella overlay.

TCSS is designed in a distributed manner. Crawling clients are distributed over several workstations and manipulated via Message Passing Interface (MPI). The time required for a complete snapshot is about 7.5 minutes.

E. TCSS Dataset Collection Flow

From a dataset's point of view, the complete processes for the two datasets are described below:

1) Crawler-to-servent Dataset

(1) [Server] Start a new round (snapshot). (2) [Server] Take bootstrapping peers as fresh peers. (3) [Server]

Dispatch fresh peers to clients. (4) [Clients] Probe fresh peers for neighbor lists. (5) [Clients] Log probing results for temporary storage of partial crawler-to-servent dataset. (6) [Clients] Perform duplication check with probing results. (7) [Server] Gather fresh peers reported by clients. If this round ends, go to Step (1). Otherwise, go to Step (3). A sketch of this process is depicted in Figure 3.

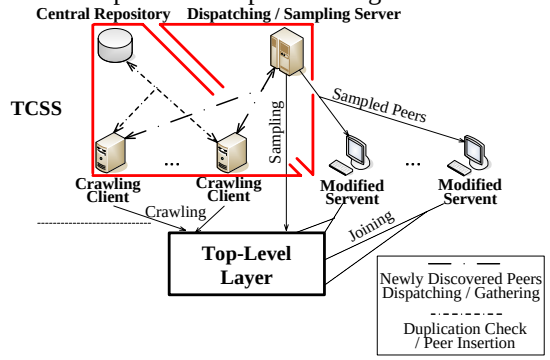

Figure 3. Process of Crawler-to-servent Dataset Collection

2) Servent-to-servent Dataset

a) Dispatching/Sampling Server

(1) [Server] Start a new round (snapshot). (2) [Server] Sample overlay for representative peers. (3) [Server] Distribute sampled peers to the modified servents. If this round ends, go to Step (1). Otherwise, go to Step (2).

b) Modified Servents

(1) [Modified Servents] Interact with sampled peers. (2) [Modified Servents] Log performance metrics for temporary storage of partial servent-to-servent dataset. A sketch of this process is depicted in Figure 4.

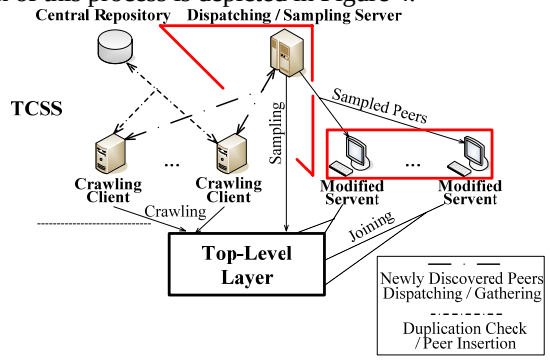

Figure 4. Process of Servent-to-servent Dataset Collection

F. Details in Collected Datasets

1) Crawler-to-servent Dataset

The main information in the crawler-to-servent dataset includes (i) peer address, (ii) peer type (ultrapeer or legacy), (iii) peer neighbors, and (iv) peer timestamp. With this information, we can identify changing factors in a sequence of topology snapshots. For example, we can observe the changes in participating peers between two successive snapshots. We can also derive session lengths of different peers from the timestamp information.

2) Servent-to-servent Dataset

The data in the servent-to-servent dataset include servent connection initialization time and response time for specific keyword search. To know which keywords should be used in later measurements, we modify the third party crawler to query for shared file lists instead of neighbor lists, and integrate this crawler with the sampling technique used in dispatching/sampling server. We call this modified crawler the third party file crawler. The third party file crawler ran for 24 hours and queried sampled peers for their shared file lists. The dataset that we received contain peer address, peer timestamp, and peer shared file list. After analyzing this dataset, we obtained some representative keywords for keyword search. The results are 'feat' (the most popular), 'thought' (a middle rank), and 'hills' (rank lower than middle).

V. Empirical Analysis

We ran TCSS for 20 hours from 19:00 21/06/11 to 15:00 22/06/11 and collected 163 snapshots totaling 17Gb. Table I presents a summary of our dataset, including (i) the average number of peers, (ii) the average number and percentage of top-level peers, and (iii) the average number and percentage of leaf peers.

TABLE I. Summary Information About Collected Dataset

Crawling Date	Avg. Total Nodes	Avg. Top-level Peers (%)	Avg. Leaves (%)
21~22/06/10	1,273,088	231,364 (18%)	1,041,723 (82%)

Overlay topology is inherently dynamic because the connections by peers are constantly changing. These dynamics may affect the main functionality of the overlay, which is to provide connectivity and efficiently route the P2P messages. We try to find the correlation between churn and the overlay performance. Referring to [1] and [2], we represent each snapshot in the crawler-to-servent dataset as a graph and analyze the change of several key factors. We pay special attention to the top-level overlay since it is the core component of the Gnutella overlay. We do not maintain peer's identities across snapshots because we do not identify peer's properties by their identifiers. We also consider a peer a newly joined one in the overlay if it is not present in the current snapshot but shows up in the next.

Table II shows all possible results of probing top-level peers. These results are kept in the crawler-to-servent dataset, where reachable peers are surely included. For accuracy, we preclude peers that do not join the overlay due to networking or resource problems. However, according to [1], most timeout peers are firewalled, and they constitute about 20 percent of the total peers. We need consider firewalled peers for accuracy of our analysis.

TABLE II. Classification of Peers in Crawler-to-servent Dataset

Connection Status	Reasons
Reachable	Peer replies its neighbor list
Bad Handshake	Incorrect protocol implement
Connection Dropped	TCP packages are dropped by router in path
Connection Reset	Peer crashes
No Route to Host	Routing setting error in path
Timeout	Peer shutdown or firewalled
Connection Refused	Peer's listen buffer overflows or port is not open

To identify peers that should be precluded from the analysis, we classify all peers into active peers and inactive

peers. If the status of a peer is either reachable or firewalled, it is an active peer; otherwise, it belongs to inactive peers. We only retain active peers in the dataset. To find out active peers in the crawler-to-servent dataset, we use a method called inactive peers elimination to rule out inactive peers. We think peers that had been reachable are not likely to change their firewall settings in a short time. If a peer's status has ever switched between timeout and reachable, we consider it an inactive peer because its peer status will change to timeout when a peer leaves the overlay. We also ignore all timeout peers that do not appear in at least two consecutive snapshots because we cannot distinguish them between being firewalled and being absent in the overlay. The flowchart for this checking is displayed in Figure 5.

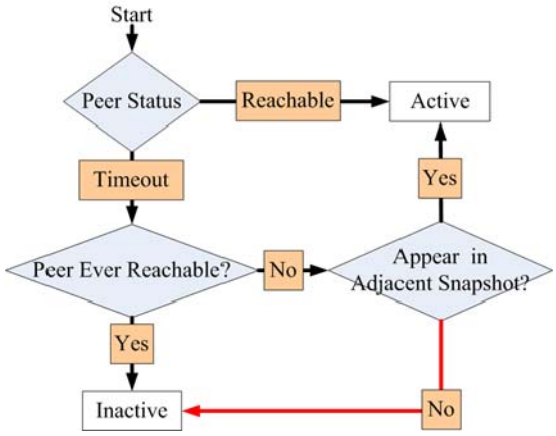

Figure 5. Methodology to Filter Out Non-firewalled Peers

In summary, the analysis of the crawler-to-servent dataset concerns the identification of active and inactive peers. If a peer is either reachable or firewalled, it is an active peer; otherwise, it is an inactive peer. Most importantly, this approach only applies to top-level peers. Based on this knowledge, we analyze the crawler-to-servent and servent-to-servent datasets from four different aspects: (i) member-based, (ii) neighbor-based, (iii) reachability-based, and (iv) servent-based analyses.

A. *Member-Based Analysis*

As mentioned above, active peers interact with other participating peers in the overlay. Inactive peers either have left the overlay or cannot be reached from any active peer in the overlay. Active peers provide steady connections to other peers. In contrast, the presence of inactive peers indicates that participating peers need extra efforts to repair failed connections to inactive peers or to overcome problems such as long search response time. Member-based analysis focuses on identifying the presence of a peer (by finding out its arrival and departure time) and the change of its properties in successive snapshots. We attempt to observe these membership related characteristics in the crawler-to-servent dataset and take them as basic metrics for the analysis in later sections.

1) *Arrival Rate*

The arbitrary arrival of peers in a P2P overlay is a major source of churn. Therefore, we should observe the arrival rate during crawling time by counting every newly arriving peer in the crawler-to-servent dataset. We consider a peer a newly arriving one if it appears in the *N*th snapshot but not in the (*N*-1)th snapshot. Then, we divide the total number of newly arriving peers by the length of crawling period to get the arrival rate.

Figure 6 shows that the arrival rate was high from 23:30 to 11:00. More than two thousand top-level peers joined the overlay every minute during this period, though the exact number continually fluctuated. Moreover, peak arrival rates occurred at 3:46, 8:57 and 10:11. We referred to [23] for the mapping between IP address and country and found that during the peak hour, most IPs came from Czech Republic, U.S., and Canada. According to the geographical analysis in [24], during 2004-2005, most users came from North America. In the sections below, we shall discuss whether other overlay analyses in our datasets exhibit a similar trend as the arrival rate.

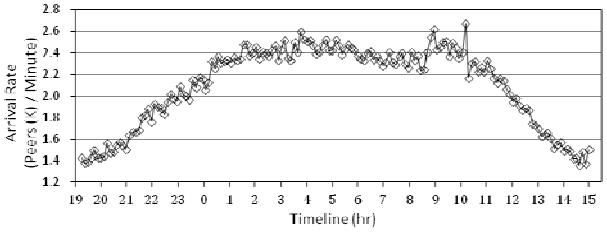

Figure 6. Arrival Rate

2) *Difference of Active Peers in Adjacent Snapshots*

Frequent arrival and departure of peers change the members of the overlay as well as the overlay topology. To observe the variation in active members, we use two adjacent snapshots (S_1, S_2) and define the difference of active peers (D_{ap}) as the percentage of peers that are active in S_1 (resp. S_2) but are inactive in S_2 (resp. S_1). The result shown in Figure 7 indicates that the difference smoothly changed during the crawling period. We suggest that as the size of the Gnutella network increases and decreases, the difference of active peers becomes slightly higher and lower, respectively.

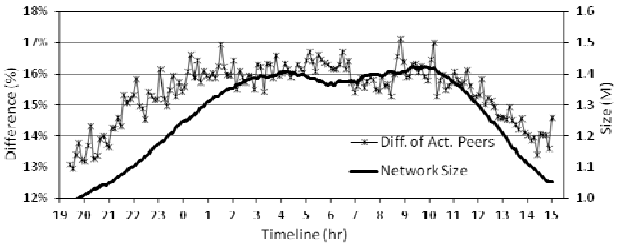

Figure 7. Difference of Active Peers

3) *Departure Rate*

Arbitrary departure is also a key source of churn. However, the probing technique implemented in TCSS does not report a correct number of leaving peers from the crawler-to-servent dataset, which is crucial to the calculation of the departure rate. In the result shown in Figure 8, the departure rate exhibits a trend similar to the arrival rate reported in Figure 6. A deep examination of the data reveals that the arrival rate sometimes exceeds the departure rate and sometimes not. The network size shown in Figure 7 also matches the growth and decline of the arrival and departure rates. Hence, the number of departures increases with a high population in the Gnutella network. To maintain a large network size, the number of arrivals must also rise.

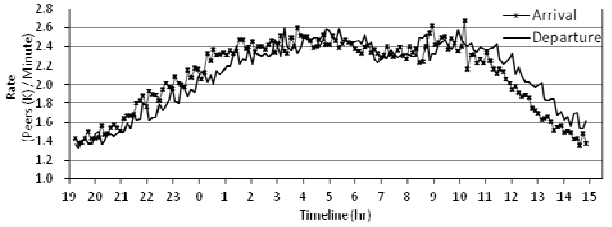

Figure 8. Departure Rate

4) Composition of Session Length in Snapshot

Prior studies reveal that session length is also important in observing churn. Generally speaking, the more long-session neighbors that a peer has, the better performance it achieves. Therefore, maintaining adequate neighbor connections can cope with dynamic changes in a P2P overlay. Such maintenance needs a large amount of control messages, causing non-negligible overhead. We examine the session length composition in each snapshot to determine whether session length is a key factor in analyzing churn.

We pre-scanned the crawler-to-servent dataset to find the distribution of session lengths for all reachable peers present in our dataset. Then, we re-scanned the crawler-to-servent datasets, snapshot by snapshot, to calculate the percentage of specific session intervals with references to each peer's session length computed in the first scan. We set up six specific session intervals: 10 minutes, 30 minutes, 2 hours, 5 hours, 20 hours, and 1 day. We placed peers with lifespan longer than or equal to 20 hours into the 1 day interval group and then classified all peers into two categories: short-lived part (10 minutes, 30 minutes, and 2 hours) and long-lived part (5 hours, 20 hours, and 1 day).

Figure 9 shows that neither the short-lived part nor the long-lived part varies significantly during our crawling period. The short-lived part grew slightly from 23:00 to 11:00, but the increasing rate was small. It does not well match the arrival rate shown in Figure 6. Nevertheless, there was a group of peers during the whole crawling period that we call very long-lived peers. The number of very long-lived peers almost remained constant in each snapshot. However, the percentage of very long-lived peers decreased from 23:00 to 12:00 in addition to a slightly increasing number of short-lived peers from 23:00 to 11:00. The ratio of the number of specific session intervals to the number of total peers in each snapshot varies smoothly, unlike the dynamically varying arrival rates shown in Figure 6. Session length is also used as a basic property for other analyses in the remaining datasets.

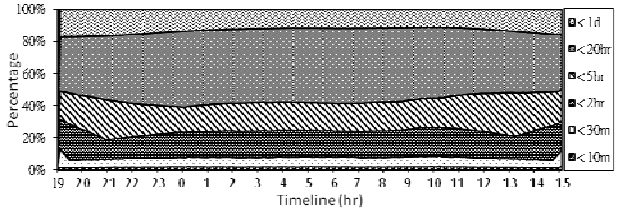

Figure 9. Session Length Composition

The above results show that the number of arrivals, departures and short-lived peers, as well as the variation of active peers in consecutive snapshots, stayed high from 23:00 to 12:00. We call this period high variation period.

B. Neighbor-based Analysis

The above results indicate that members in an overlay dynamically change and the degree of change is high during certain periods. It is sure that the changes of membership in an overlay will cause constant change of neighbor connections. Neighbor-based analysis focuses on the issue of churn from the neighbor connection's perspective. The reachable top-level peers in the crawler-to-servent dataset reply with their neighbor lists, where other top-level peers (or leaves if any) may be included. However, TCSS does not track neighborhood information for leaf peers. Since top-level peers are the core of the Gnutella overlay, we focus on top-level neighbors of reachable peers in this paragraph.

1) Difference of Peer Neighbors in Consecutive Snapshots

The dynamics of the overlay make connections change constantly. As a reaction to the dynamic changes in overlay members, participating peers attempt to repair the connections as their neighbors join or leave the overlay. Meanwhile, the dynamics also affect overlay functionalities as some neighbor connections become inadequate and unreliable.

This paragraph explores the variations in peer neighbor connections. For a specific peer that is present in two consecutive snapshots, we identify difference of its neighbor lists between snapshots. If the difference is significant, the peer may experience severe disturbance, which may be caused by arrivals and departures of peers or bad network conditions. Under such unstable condition, the peer may break several troublesome neighbor connections and/or establish new neighbor connections, but this behavior may cause further overhead and degrade the overlay performance.

The curve of our first analyzed result is not consistent with the trend shown in Figure 7. There exist several peaks and the curve in the rear part remains high. To explain this result, we conjecture that the changes in neighbor connections involve two factors: peer arrival-departure and neighbor replacement.

a) Peer arrival-departure

Peers may join or leave the overlay due to user behaviors, peer overloading or network conditions. Such changes either create or remove connections between peers. In other words, the peer's arrival-departure change is caused by peer's arrivals and departures.

b) Neighbor replacement

Active servents may seek peers with high available computing resources to replace inferior neighbors for a better overlay performance. In this scenario, the replaced neighbors would keep their connection status healthy (maintain a certain number of neighbors for normal operations in the overlay) by trying to connect with new neighbors. These neighbor replacement cases are difficult to study because different P2P applications may take different neighbor reselection policies. In addition, the factors that cause such change need to be considered so as to characterize the properties of neighbor replacement.

Knowing that neighbor changes can be caused by peer arrival-departure or neighbor replacement, we know which phase each neighbor in Figure 10 belongs to. Given two consecutive snapshots (S_1, S_2), neighbors that appear in

only one of these two snapshots (S_1 or S_2) are grouped under peer arrive-departure change. Other changed neighbors are grouped under neighbor replacement change.

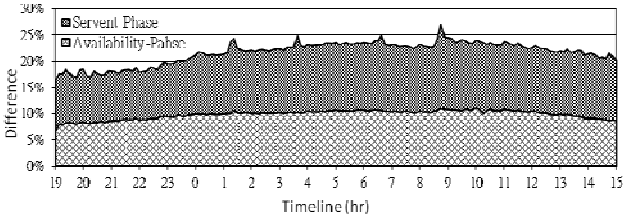

Figure 10. Difference of Reachable Peers' Active Neighbors

Figure 10 indicates that the difference (D_n) was high from around 0:00 to 12:30. The change in peer arrival-departure follows the trend of the arrival and departure rate depicted in Figure 8. This time period also nearly matches the time period having increasing short-lived peers in Figure 9. Peaks that occurred in our first analyzed result were obviously caused by the neighbor replacement change. The neighbor replacement change in the empirical environment at the time we collected the datasets can be explained by a high network jitter or a significant amount of busy active peers. The result in Figure 10 shows that churn is contributed by not only peers' arrival and departure but also different network conditions or the statuses of peers' capacities. In summary, two changes reshape the topology: peer's arrival-departure change, which depends on the rate of arrival and departure, and neighbor replacement change, which depends on the condition of overlay at the time.

C. Reachability-Based Analysis

The metric of reachability concerns the distance or hop count between peers. It decides how much effort a peer must make to reach a remote servent.

1) Shortest Path

Although the propagation delays vary in different network conditions, we want to minimize the delays. Furthermore, a long distance in the overlay means a high time-to-live (TTL) value in the Gnutella message to let the message reach a faraway target servent. Flooding a high-TTL message causes an extremely high cost. If the messages can arrive at the remotest targets with a low TTL value, the overhead brought to the network and the propagation delay can both be reduced. We therefore identify the shortest paths between all pairs of top-level peers and classify these paths into various groups depending on their lengths.

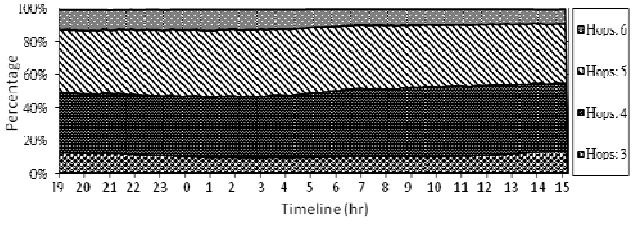

Figure 11. Shortest Path between All Pairs of Top-Level Peers

Figure 11 shows that most of the shortest paths between top-level peers were within 4 or 5 hops, and the hop count did not vary dramatically throughout the whole period. This result indicates that the shortest paths between any pairs of top-level peers remain fairly constant. According to [1], the shortest paths between leaves are 1 or 2 hops longer than those of top-level peers. Although peer's neighbor connections fluctuated during the high variation period in the crawler-to-servent dataset, we found that this reshaping topology still exhibited small-world network property. The maximum network diameter in our analysis was 9.

D. Servent-Based Analysis

We studied the servent-to-servent dataset to explore the correlation between churn and the performance of overlay. We examined two basic metrics to mimic the experience of regular users: the initialization time for neighbor connection when booting and the response time for searching specific keywords. Both metrics are important in P2P applications because high connection or response time may lower the users' will to use this service. Hence, we looked at the servent-to-servent dataset from a user's point of view and analyzed it with the results from the crawler-to-servent dataset.

1) Booting Connection Initialization Time

Figure 12 shows that the booting time fluctuated significantly from around 0:00 to 12:00. We found that the fluctuating period in Figure 12 matches the high variation period. We therefore make two inferences. First, the probability of getting low performance increases as churn becomes high. Second, the performance variation is proportional to the degree of churn. With high arrival and departure rate or bad overlay condition, churn becomes severer and the servents on the overlay are more likely to experience low performance.

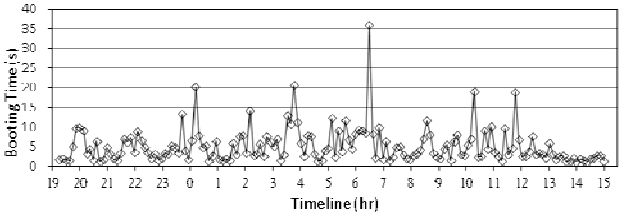

Figure 12. Booting Connection Initialization Time

2) Keyword Search Response Time

Figure 13 shows keyword search response time for the keyword 'hills'. The response time started to disperse and values higher than 40 seconds began to increase from 23:00 to 10:00, when the response times decrease gradually.

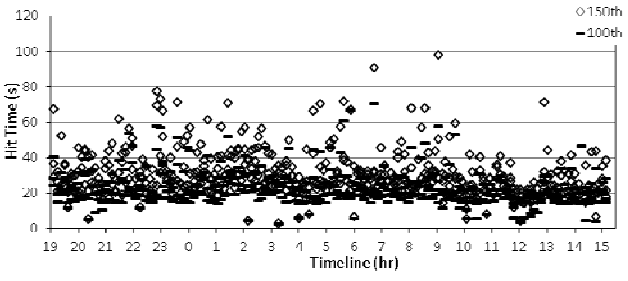

Figure 13. Keyword Search Response Time in Lower Rank: hills

Figure 14 shows the result for the keyword 'thought'. Response time rised earlier than that of 'hills' (around 22:30, the time at which the size of the overlay network started to grow linearly). Even the time for the 100th response increased until 11:00. Hence, we know that the

response time fluctuated significantly from 22:30 to 11:00 for the keyword ‘thought’.

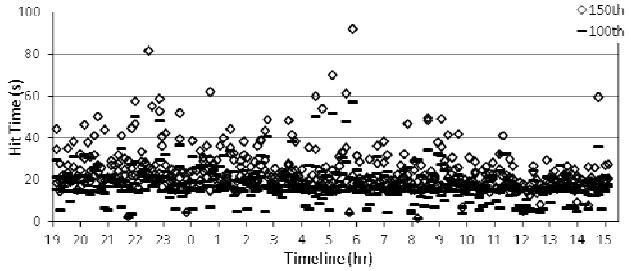

Figure 14. Keyword Search Response Time in Middle Rank: thought

Figure 15 shows the result for the most popular keyword ‘feat’. Here response time was usually low except for the time from 0:30 to 10:30. This result is still acceptable for most applications since almost all the searches respond quickly. In summary, the results of search response time support our two inferences. That is, the impact of churn depends on the network condition of the involved peers and more importantly, the conditions of their neighbors.

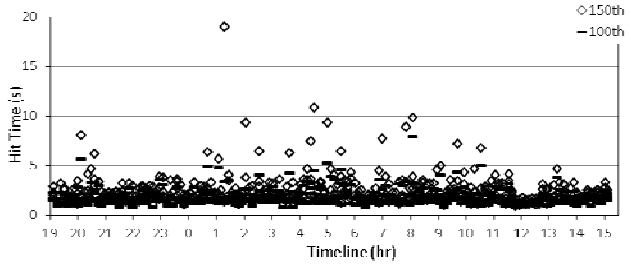

Figure 15. Keyword Search Response Time in Top Rank: feat

VI. Conclusion and Future Work

This paper presents TCSS, a system that can quickly capture an overlay snapshot containing both the overlay topology and overlay performance metrics. TCSS prevents collecting distorted snapshots by shortening the collection period and achieves our objective by analyzing the collected datasets.

Based on the Gnutella overlay, TCSS finds the correlation between churn and overlay performance in an unstructured file-sharing P2P network. Our results reveal that variations in the overlay topology stem from two causes, changes in peer arrival-departure and peer neighbor replacement. These changes may be caused by network condition, peer status, or peer arrival and departure, all of which are also additional sources of churn. Though the degree of churn varies dynamically, the Gnutella overlay still keeps a small world property. When churn is frequent, the probability of receiving a low overlay performance increases. In other words, when the degree of churn rises, the peers in the overlay have higher probabilities of getting low performance, depending on their neighbors’ statuses. We discover that the variation in overlay performance is proportional to the degree of churn.

This study provides an insight into the overlay topologies under churn and serves as a reference for possible improvement on P2P applications. Our results show that performance degradation caused by the change in neighbor replacement is hard to improve because it is hardly possible to consider many overlay conditions at that time. We may focus on the change in peer arrival-departure and attempt to lower the effect of peer arrivals and departures. Our results show that although a considerable number of top-level peers constantly join and leave the overlay, the top-level layer in the Gnutella overlay remains relatively stable. This leads us to think that when leaf peers have the capacities to become ultrapeers, the numbers of the degrees for top-level and leaf peers are not suitable. It is possible that these new ultrapeers do not stay in the overlay for as long as we expect. However, we can correct this problem if the ratio of new ultrapeers can increase over time. As a result, it may be possible to minimize the impact caused by the change in peer arrival-departure.

Acknowledgment

This work was supported in part by National Science Council, Taiwan, under Grants NSC 100-2221-E-009-072-MY3, NSC 101-2221-E-009-031-MY3 and NSC 101-2219-E-009-028, and D-Link Co., Taiwan.

References

[1] D. Stutzbach, R. Rejaie, and S. Sen, “Characterizing Unstructured Overlay Topologies in Modern P2P File-Sharing Systems,” IEEE/ACM Transactions on Networking, Vol. 16, Issue 2, pp. 267-280, April 2008.

[2] D. Stutzbach and R. Rejaie, “Understanding Churn in Peer-to-Peer Networks,” Proceedings of the 6th ACM SIGCOMM conference on Internet measurement, pp. 189-202, October 2006.

[3] D. Stutzbach, R. Rejaie, N. Duffield, S. Sen, and W. Willinger, “On Unbiased Sampling for Unstructured Peer-to-Peer Networks,” IEEE/ACM Transactions on Networking, Vol. 17, Issue 2, pp. 377-390, April 2009.

[4] D. Stutzbach and R. Rejaie, “Capturing Accurate Snapshots of the Gnutella Network,” INFOCOM 2005. Proceedings IEEE 24th Annual Joint Conference of the IEEE Computer and Communications Societies, Vol. 4, pp. 2825 - 2830, March 2005.

[5] S. Zhao, D. Stutzbach, and R. Rejaie, “Characterizing Files in the Modern Gnutella Network: A Measurement Study,” Proceedings of SPIE/ACM Multimedia Computing and Networking, Vol. 6071, 2006.

[6] S. Sen and J. Wang, “Analyzing Peer-to-Peer Traffic Across Large Networks,” IEEE/ACM Transactions on Networking, Vol. 12, Issue 2, pp. 219-232, April 2004.

[7] F. E. Bustamante and Y. Qiao, “Friendships that Last: Peer Lifespan and its Role in P2P Protocols,” Web Content Caching and Distribution, pp. 233-246, 2004.

[8] Gnutella Protocol Development, [Online]. Available: http://rfc-gnutella.sourceforge.net/

[9] S.K. Dhurandher, S. Misra, M.S. Obaidat, I. Singh, R. Agarwal, and B. Agarwal, “Simulating Peer-to-Peer Networks,” IEEE/ACS International Conference on Computer Systems and Applications, pp. 336 - 341, May 2009.

[10] S. Naicken, An. Basu, B. Livingston, and S. Rodhetbhai, “A Survey of Peer-to-Peer Network Simulators,” Proceedings of 7th Annual Postgraduate Symposium, 2006.

[11] R. Bhardwaj, V.S. Dixit, and A.Kr. Upadhyay, “An Overview on Tools for Peer-to-Peer Network Simulation,” International Journal of Computer Applications, Num. 19, Article 13, 2010.

[12] A. Brown and M. Kolberg, “Tools for Peer-to-Peer Network Simulation,” IRTF P2PRG, Internet Draft, July, 2006.

[13] Q. He, M. Ammar, G. Riley, H. Raj, and R. Fujimoto, “Mapping Peer Behavior to Packet-Level Details: A Framework for Packet-Level Simulation of Peer-to-Peer Systems,” MASCOTS 2003. 11th IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer Telecommunications Systems, pp. 71-78, Oct. 2003.

[14] J. Liang, R. Kumar, and K. W. Ross, “The KaZaA Overlay: A Measurement Study,” Computer Networks, Vol. 50, Issue 6, pp. 842-858, April 2006.

[15] G. F. Riley, R. M. Fujimoto, and M. H. Ammar, “A Generic Framework for Parallelization of Network Simulations,” Proceedings of 7th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems, pp. 128-135, 1999.

[16] PDNS – Parallel / Distributed NS, [Online]. Available: http://www.cc.gatech.edu/computing/compass/pdns/

[17] Packet-level P2P (Gnutella) Simulation, [Online]. Available: http://www.cc.gatech.edu/computing/compass/gnutella/

[18] LimeWire Wiki, [Online]. Available: http://limewire.negatis.com/

[19] Crawling Gnutella Network, [Online]. Available: http://www.ece.ubc.ca/~samera/TA/411/project/Gnutella-Protocol.html

[20] J. F. Buford, H. Yu, and E.K. Lua, “P2P Networking and Applications,” Elservier Inc, 2009.

[21] Slyck.com, [Online]. Available: http://www.slyck.com/

[22] Gnurella Crawler, Wikipedia, the free encyclopedia, [Online]. Available: http://en.wikipedia.org/wiki/Gnutella_crawler

[23] Max Mind GeoLite Country, [Online]. Available: http://www.maxmind.com/app/geoip_country

[24] A.H. Rasti, D. Stutzbach, and R. Rejaie, “On the Long-term Evolution of the Two-Tier Gnutella Overlay,” INFOCOM 2006. Proceedings 25th IEEE International Conference on Computer Communications, pp. 1-6, April 2006.