Observe Internet Peer-to-Peer Activities at Home*

Chun-Ying Huang[1], Shang-Pin Ma[1], Ching-Hong Wu[1], Chi-Ming Chen[1], and Chih-Hung Lin[2]
[1]Department of Computer Science and Engineering, National Taiwan Ocean University
[2]Networks and Multimedia Institute, Institute for Information Industry
Email: {chuang,albert}@ntou.edu.tw, {chwu,cmchen}@snsl.cs.ntou.edu.tw, chlin@iii.org.tw

***Abstract*—Peer-to-peer networking is a popular topic for network researchers. A fundamental step before digging into understanding peer-to-peer network behavior is to collect a sufficient amount of peer-to-peer traffic traces. However, it is not really easy to obtain traces. Due to privacy concerns, researchers usually have to either sign non-disclosure agreements or choose to use anonymous/obfuscated traces, which may hide important details of the traces. In this paper, we explore the possibility of observing peer-to-peer networks at home. A virtual machine based trace collection system is developed. We then show that with a limited resource, i.e., equipments and network nodes, it is possible to sample a number of peer-to-peer traffic traces and then make further analysis. Based on the traces we collected, a preliminary analysis is also made to show what can be observed in the traces. We believe it is helpful for researchers whom are interested in observing peer-to-peer activities and Internet dynamics.**

***Index Terms*—Internet measurement, peer-to-peer network, trace collection**

I. Introduction

It is generally recognized that Napster[2] is the first peer-to-peer system. The first generation peer-to-peer system is basically a centralized service. A big directory is stored in a central server to maintain the relationships between files and hosts. Hence, peer-to-peer end users are able to lookup files via the central server and then download files directly from involved hosts. However, due to performance and scalability issues, peer-to-peer techniques have evolved from centralized services to distributed services[1]. That is, there is not a central server maintaining the file-to-host relationships. Instead, all peer-to-peer nodes form a structured overlay network in a distributed manner. Searching and downloading activities are done through message exchanges over the overlay network. Modern peer-to-peer applications often implement both models, i.e., centralized and distributed models, to maximize the availability of the peer-to-peer network.

To peer-to-peer network researchers, a fundamental requirement for network traffic analysis researches is to collect a number of network traces. However, it is not really easy to obtain network traces. The difficulty is usually because of privacy issues. Most users would not like their network connections being watched, stored, or even analyzed. Therefore, to legally obtain large scale peer-to-peer network traces, a researcher often has to sign non-disclosure agreements (NDAs) with network operators or even end users. It is worse that even an NDA is signed, the obtained traces may be anonymized or obfuscated, which may hide important details of the traces. In addition to the difficulty of obtaining traces, another important issue for confidential network traces is that they can not be shared. It makes researchers difficult to compare their researches with others because data used by one group can not be shared with other groups. Therefore, it is not able to create a universal standard to evaluate various peer-to-peer network researches.

In this paper, we explore the possibility of observing peer-to-peer networks at home. Thanks to the distributed design of peer-to-peer networks and dynamic IP address assignment policies. The two techniques make this work possible. In a distributed peer-to-peer network, a node usually has to maintain a list of contacted neighbor nodes. The list is often called *the host cache* [1]. With the cache, a peer-to-peer node is able to expand its view of networks and hence accelerate peer-to-peer network operations including joining the network, searching for files, and even downloading files. However, due to the dynamic IP address assignment policy in most Internet service providers (ISPs), an IP address stored in a host cache may become invalid at any instant. This is because the address can be reclaimed due to expired lease time or disconnected hosts. The proposed system collect peer-to-peer network traces based on the above two observations. It tries to obtain IP addresses that is previously assigned to a peer-to-peer node. Then, it just passively waits requests from other peers and collect the trace. We show that with a limited number of nodes and resources, it is possible to collect a large number of traces from world-wide peer-to-peer nodes.

The rest of this paper is organized as follows. In Section II, we introduce the assumptions and the implementation of the proposed system. In Section III, we make some preliminary observations to the collected traces and show what can be observed. Finally, a concluding remark is given in Section IV.

II. The Proposed Solution

A. Assumption

There are two assumptions used in the proposed system. First, the ISP used to collect traces must assign public IP addresses to its clients. Only public IP addresses are able to

*This work was supported in part by National Science Council under the grant number NSC 97-2218-E-019-004-MY2, NSC 99-2218-E-019-001-MY2, and by Taiwan Information Security Center at NTUST(TWISC@NTUST) under the grant number NSC 99-2219-E-011-004.

[1]It may be also because of copyright issues since an unlicensed central server providing references to copyrighted materials is usually considered as illegal.

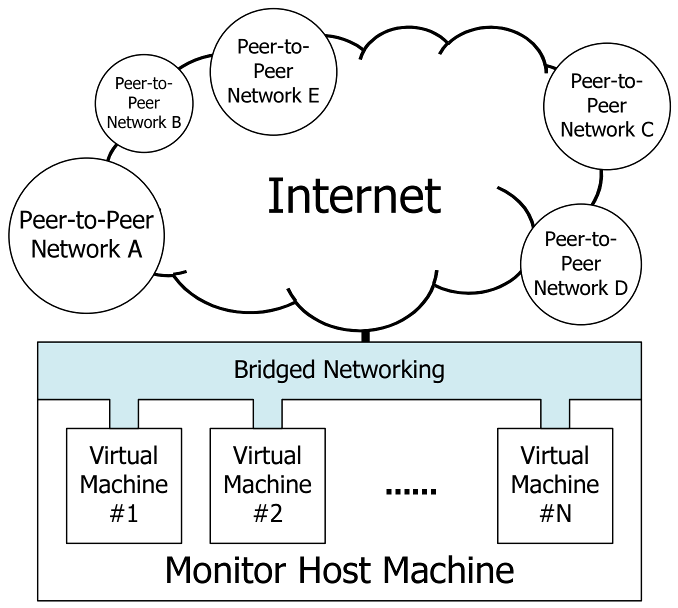

Fig. 1. The proposed system architecture.

passively receive incoming packets and connections. Second, the ISP should always assign dynamic IP addresses to its clients. If an on-line client disconnects itself from the Internet and then gets on-line again, the client should get an IP address different from that one used in the previous session. This assumption should be true in most ISPs because the number of IPv4 addresses are limited but the number of Internet users are tremendous. A just released IP addresses can be immediately assigned to another user and hence the IP address becomes unavailable.

B. Overview

The proposed system architecture is shown in Figure 1. Each traffic collector is implemented on a virtual machine, which is connected to the real network using a bridged interface. When a virtual machine boots, it first connects to the Internet and obtain an IP address by using a predefined user name and password. Then, a crafted program is executed to intercept all incoming data from the Internet. The program just passively listen for UDP packets and TCP connections, it does not response to anything. For UDP packets, the program simply stores everything it receives. For TCP connections, since data can be only sent after a connection is successfully established, the program accepts all incoming TCP connections and then waits for incoming data.

A TCP connection is said to be idled if there is no thing new received. If a established TCP connection is idled for more than a given threshold (T_c seconds), the connection is closed by the program. Similarly, an on-line virtual machine is said to be idled if it does not receive any packets from the network. If an on-line virtual machine has idled for a given threshold (T_m seconds), the IP address in-use is treated as an inactive IP address and the virtual machine reboots. In addition to reboot virtual machines on detection of inactive IP addresses, a virtual

```
iptables -t nat -A PREROUTING -p tcp \
         -d $IP -j DNAT \
         --to-destination $IP:$PORT
iptables -t nat -A PREROUTING -p udp \
         -d $IP -j DNAT \
         --to-destination $IP:$PORT
```

Fig. 2. Firewall rules used to redirect all TCP and UDP traffic to the sink program.

machine also reboots periodically. In the proposed system, a virtual machine is rebooted after T_r seconds since it is on-line. Hence, it is possible to collect as many traces as possible from different IP addresses.

C. Implementation

The proposed system is implemented on a Linux operating system. We install a Ubuntu Linux 9.04 server in each virtual machine and run our program over the Linux operating system. One difficult encountered during the implementation is that the program has to intercept all incoming connections and stored the data. Since network requests for all TCP and UDP ports have to be stored properly, we have tried to bind multiple network ports using either a single process or multiple processes. However, these methods do not work. If a single process binds all network ports, it may run out of available system descriptors. In addition, handling incoming connections from a large number of network ports is inefficient. It requires $O(n)$ time to iteratively check the state of each bound ports using either the `select(2)` or `poll(2)` system call. If multiple processes are used to bind all network ports, although the single process problem can be mitigated, the additional costs incurred by creating processes and context-switches degrade the performance of the system. The available memory space for each virtual machine is often limited. If a large number of processes is created, the overall system performance drops a lot.

To implement the system properly, there are some tricks. First, two traffic sink programs, `tcpsink` and `udpsink`, are written to handle only TCP and UDP traffic, respectively. The `tcpsink` program handles only TCP traffic received on a fixed port p. Similarly, the `udpsink` does the same thing for UDP traffic. The two programs do nothing but just read received data from the network and then drop it. The purpose of the two programs is to prevent the traffic sender from receiving ICMP host or port unreachable messages. In order to handle all TCP and UDP traffic from the Internet, two firewall commands, as shown in Figure 2, are used to redirect all TCP and UDP traffic to the sink programs. With the support of the Linux firewall system, the sink programs are able to act like accepting traffic destined for a single port but they actually accept traffic destined for all network ports. Finally, to collect traces for the obtained IP address, all involved network traffic is stored by a background `tcpdump` program [3] in a form of raw packets.

TABLE I
DOMINANT PEER-TO-PEER PROTOCOLS.

Protocol	Requests	Percentage
edonkey	115,385	4.7%
bittorrent	1,412,925	57.7%
gnutella	920,017	37.6%

III. TRACE COLLECTION AND OBSERVATION

A. *Environment*

Before introducing what has been observed, we have to introduce the real environment used to collect peer-to-peer traffic traces first. The proposed system installs in two different locations, but they are both using the same ISP. Each system runs four virtual machines because the ISP allows a maximum of four hosts connecting to the Internet simultaneously. The timers' threshold are given as follows:

- TCP connection idle timeout T_c = 30 seconds.
- On-line virtual machine idle time T_m = 1800 seconds (30 minutes).
- Virtual machine reboot timeout T_r = 1800 seconds (30 minutes).

To improve system efficiencies, we do not really reboot a virtual machine every T_r seconds. Instead, we just ask the virtual machine to disconnect itself from the Internet and then on-line again to obtain another new IP address. The entire trace collection experiment runs for more than 3 months (120 days).

In addition to peer-to-peer traffic, it is possible that an IP address also receives many non-peer-to-peer packets. These packets may be used to probe opened network ports or even launch attacks. Therefore, to filter out these noises, we use two strategies to recognize peer-to-peer traffic. First, we adopt patterns from the L7filter [6] project to identify unencrypted peer-to-peer traffic. The traffic identification program is based on one of our previous work [4]. Second, if a network connection can be recognized as a peer-to-peer connection, all packets send to the same destination are considered as the same type of peer-to-peer traffic. Here the destination is defined by the used transport layer protocol, the destination IP address, and the destination port number. Based on the two strategies, we find out three major peer-to-peer protocols in the collected traces, as show in Table I.

B. *Traffic Sinks*

During the entire experiment period, the proposed system connects to the Internet using 5,772 distinct IP addresses, which are used to collect peer-to-peer traces. Almost all these IP addresses, 97% of them receive peer-to-peer traffic. Figure 3 shows the overall statistics for the number of requests send to trace collection IP addresses. The x-axis indicates the number of requests that the trace collection IP addresses have received and the y-axis is the cumulative distribution function for the number of distinct IP addresses. The left plot in Figure 3 is the big picture and the right plot in the figure enlarges the part for the request numbers ranging from 1 to 200. In the figure,

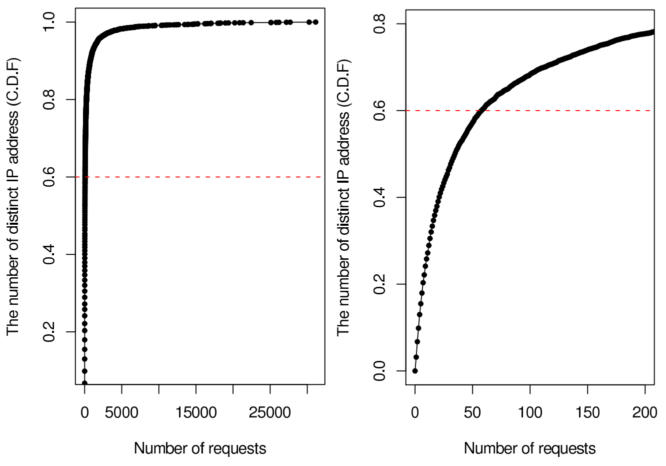

Fig. 3. Statistics for the number of requests send to trace collection IP addresses.

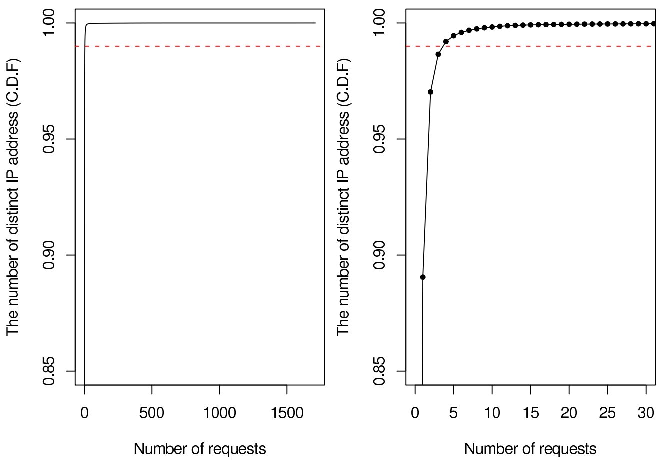

Fig. 4. Statistics for the number of requests sent from the source IP addresses.

we can see that the number of requests that a trace collection IP address receives is diverse. It is possible that an IP address receives less than ten requests but it is also possible that one receives more than 25,000 requests. From the figure, we can also see that more than 40% of IP addresses have received more than 60 requests during the period of observations. Since the T_r timer is set to 30 minutes, it means that 40% of IP addresses have a request incoming rate higher than 1 request per minute.

C. *Traffic Sources*

The proposed system monitored 1,998,049 source IP addresses that generate peer-to-peer requests. Similar to the sink addresses, Figure 4 shows the overall statistics for the number of requests sent from the source addresses. The left plot in the figure is the big picture and the right plot in the figure enlarges the part for the request numbers ranging from 1 to 30. In the figure, we can see that 99% of peers sent less than 5 requests during the entire observation periods. Since our trace collection system does not response to incoming requests, a remote peer should treat the sink IP address as a dead peer

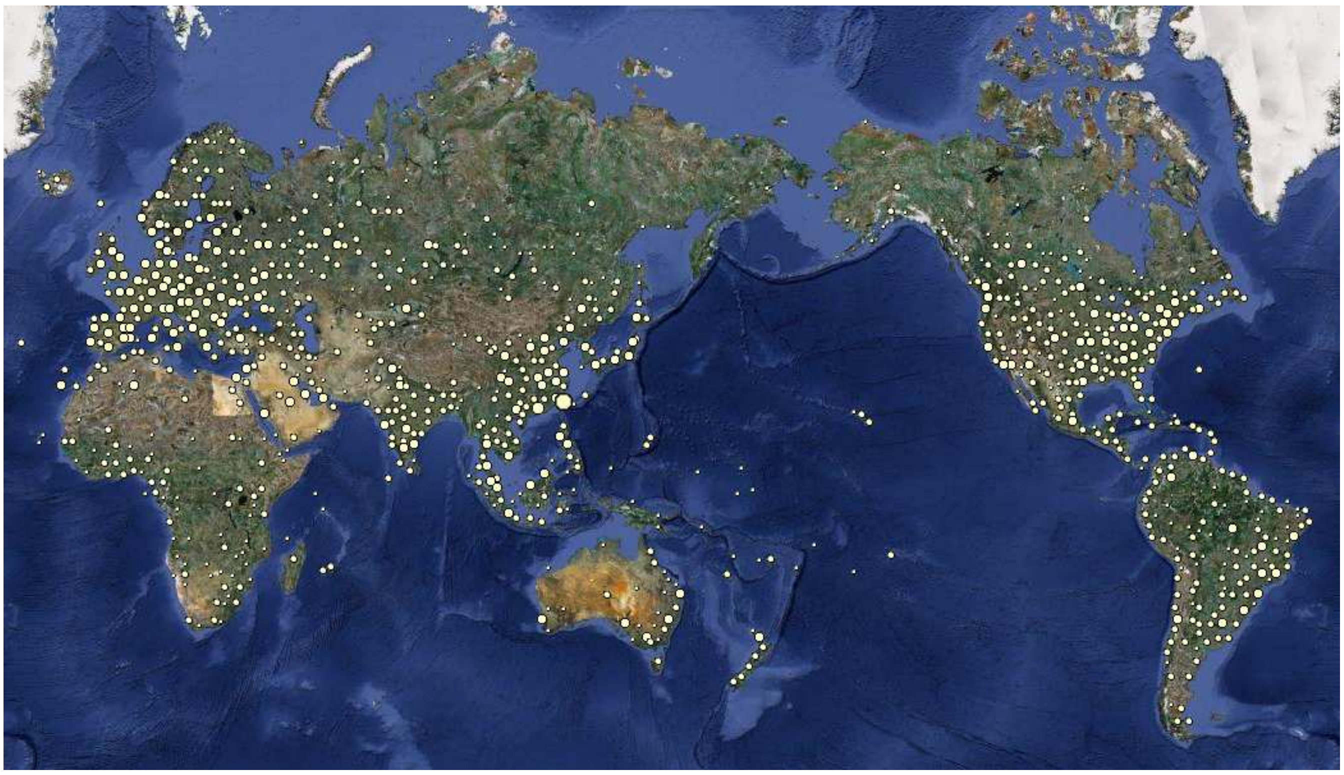

Fig. 6. Observed World-wide peer-to-peer users. The circles on the map indicate the location of observed peers. The larger the circle, the larger number of distinct IP addresses are observed.

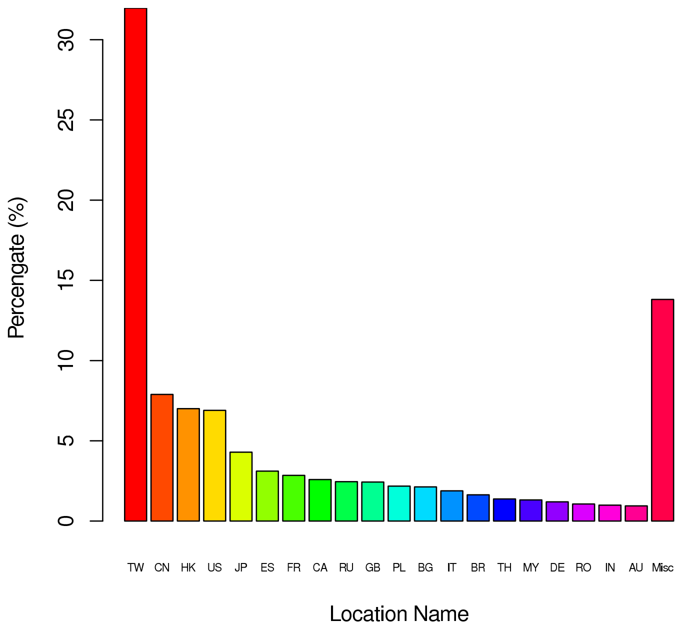

Fig. 5. Source countries of requests.

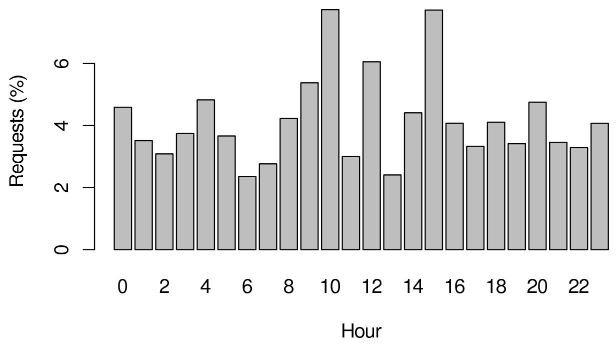

Fig. 7. Aggregated 1-day plot for requests.

and stop sending more requests to the IP address. However, we can still find that about 1% peers use a more aggressive strategy to communicate even if its peer is actually a dead peer.

We also monitor the geographic location of peer-to-peer request sources. The IP-to-location database is obtained from `ipinfodb.com` website [5]. Figure 5 shows the originating countries and the percentage of the requests. The top 20 locations contributes 80% of requests. It is obviously that the monitored peer-to-peer traces have a high spatial locality. This is because traces are collected at a single place and the peers that sent requests to the trace collection IP addresses should have similar interests in common. However, we can still find that more than 60% of IP addresses come from many other different places. It also shows that the proposed system is able to reach peers all around the world. To be more intuitive, a world map plot can be done to give images about how these peers are distributed over the world, as shown in Figure 6.

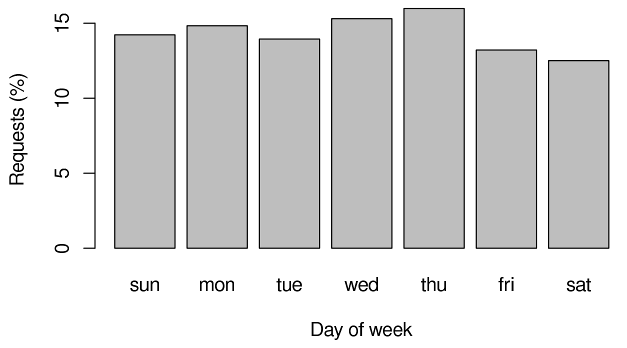

Fig. 8. Aggregated 1-week plot for requests.

D. Temporal Locality

We also observe the temporal locality of peer-to-peer behaviors. Figure 7 and Figure 8 shows the observed temporal locality from the collected traces. In the both figures, we interprets request timestamps by using the time-zone of request senders. We believe that interpreting timestamps using the senders' time-zone should be a more accurate choice to understand user behaviors. It is intuitive that peer-to-peer applications should be usually run in daytime. This intuition matches the result shown in Figure 7. However, if we see the one-week plot in Figure 8, we find that there are not much differences between weekdays and holidays.

IV. Conclusion and Future Works

In this paper, we propose and implement a peer-to-peer trace collection prototype to show that it is possible to collect representative peer-to-peer traces at home. With a limited number of network nodes and resources, a number of peer-to-peer traces can be collected and analyzed. Our preliminary observations also show that the quality of the traces, the spatial locality distributions, and the temporal locality behaviors matches common understandings to regular peer-to-peer behaviors. This work can be further improved in several ways. Since it is able to reach a large number of real peers, the system can be designed to interact with those peers. Then, it is possible to understand what are shared in the network, to make more advanced statistics for peer-to-peer protocols, or even compromise parts of the network. In addition to monitoring peer-to-peer network behaviors, the proposed system collects many non-peer-to-peer traces as well. These traces include network attacks, requests to dynamic proxies and servers, and many unknown traffic. It may be also a good material to understand the traffic and protocol dynamics on the modern Internet.

V. Acknowledgement

This work was conducted under the "Next Generation Security Technology Deployment and Enablement Project" of Institute for Information Industry which is subsidized by the Ministry of Economy Affairs of the Republic of China. We also thank the anonymous reviewers for their valuable comments and suggestions.

References

[1] S. A. Baset and H. G. Schulzrinne. An analysis of the skype peer-to-peer internet telephony protocol. In *Proceedings of the 25th INFOCOM Conference*, pages 1–11, Barcelona, Spain, 2006. IEEE.

[2] B. Carlsson and R. Gustavsson. The rise and fall of napster - an evolutionary approach. In *AMT '01: Proceedings of the 6th International Computer Science Conference on Active Media Technology*, pages 347–354, London, UK, 2001. Springer-Verlag.

[3] F. Fuentes and D. C. Kar. Ethereal vs. tcpdump: a comparative study on packet sniffing tools for educational purpose. *J. Comput. Small Coll.*, 20(4):169–176, 2005.

[4] C.-Y. Huang and C.-L. Lei. Bounding peer-to-peer upload traffic in client networks. In *DSN '07: Proceedings of the 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pages 759–769, Washington, DC, USA, 2007. IEEE Computer Society.

[5] ipinfodb.com. Free ip address geolocation tools. [online] http://ipinfodb.com/.

[6] M. Strait and E. Sommer. Application layer packet classifier for linux. [online] http://l7-filter.sourceforge.net/.