Tracerouting Peer-to-Peer Networks

Wenli Liu and Raouf Boutaba
School of Computer Science
University of Waterloo
200 University Ave. W.
Waterloo, ON, Canada
{w7liu, rboutaba}@bbcr.uwaterloo.ca

Abstract

As Peer-to-Peer(P2P) networks are being developed and deployed on the Internet at a fast pace, the need for a traceroute like mechanism at the application layer surfaces. Such mechanism, once implemented and deployed, will significantly reduce the complexity involved in developing, researching and evolving P2P networks. It will also help users understand how their queries are handled in P2P networks and mitigate their frustration when their queries return nothing. In this paper, we propose and describe AppTraceroute, which is analogous to traditional traceroute but determines the application layer hops and hosts that a query has visited. Meanwhile, we studied the bandwidth consumptions involved when introducing AppTraceroute into pure P2P networks, hybrid P2P networks and structured P2P networks. We have implemented and experimented the concept of AppTraceroute on a structured P2P network, and the results are promising.

Keywords

AppTraceroute, Application Layer Traceroute, Pure Peer-to-Peer Networks, Hybrid Peer-to-Peer Networks, Structured Peer-to-Peer Networks

1. Introduction

With only a few years of development, Peer-to-Peer(P2P) networks have gained tremendous popularity and have reshaped the Internet's landscape ever since. From pure, fully decentralized P2P networks such as Gnutella, to hybrid P2P networks such as KaZaA, and to structured P2P networks such as Pastry[4], Tapestry[5], and CAN[14], P2P networks, once being deployed, can attract millions of users in the Internet. At the same time, P2P networks have been increasingly utilized in application layer multicast [11], Resilient Overlay Networks [6], Internet measurement [16], [15], Search for Extraterrestrial Intelligence [7] and P2P phone calls [13], albeit file sharing still remains as the prominent area. Measurements show that P2P networks have surpassed web applications in generating more than half of the total traffic in the Internet. It is estimated that such P2P applications will continue to be developed and deployed in the Internet in the foreseeable future.

As P2P networks can easily gain popularity and generate a large amount of traf-

fic, efforts have to be made to ensure their proper functioning from various aspects. One such aspect is whether queries are forwarded correctly across the P2P network to their destinations. In pure P2P networks such as Gnutella, peers should forward queries to all its neighbors and stop forwarding a query when the query has been forwarded for a certain times. In hybrid P2P networks such as KaZaA, a regular peer should always send its queries to the super node it connects to and the super node should forward the queries to other super nodes subsequently. The correct forwarding of queries is of particular importance for structured P2P networks, e.g., Pastry, wherein a query is supposed to be forwarded to its destination along one optimal path. Are the queries actually been forwarded along the optimal path? However, no known means are available to retrieve the paths taken by queries in a P2P network.

Nowadays P2P networks do not allow queries to reach all the peers in order to mitigate the effect of flooding. For example, queries in Gnutella will not be forwarded after their TTL expires [10]. This practice, however, lowers the chance for a query to find a file when the file actually exists in the P2P network. A repeat of the same query may visit some new peers, but most likely the majority of the peers will be visited again and again. When a query returns nothing, a user might resend the query again and again until the file is found or the user gets frustrated and gives up. It would be much better if P2P networks were equipped with some mechanism that would allow users to determine the peers visited by a query in P2P networks. To this end, a user would bootstrap from peers that the query has not visited before so that a resend of the query could reach more unvisited peers.

In this paper, we propose and describe a mechanism that is similar to traceroute but works at the application layer. The mechanism, i.e., AppTraceroute, is able to obtain the paths, if more than one path, that a query takes in a P2P network. Each of these paths is organized as an ordered list of application layer hops, starting from the source peer to the destination peers of the query. For each such application layer hop, AppTraceroute recovers the underlying IP layer hops in the same manner as traceroute does. Although AppTraceroute requires some minor modifications at each peer, it would allow users and researchers to easily recover the paths that queries take in P2P networks and to investigate the overlay topology and the network topology, thus offering opportunities in synchronizing the two topologies.

The rest of this paper is organized as follows. The related works are presented in Section 2 and Section 3 focuses on the detailed design and analysis of AppTraceroute. A prototype of a structured P2P network that adopts the concept of AppTraceroute has been implemented and Section 4 will describe the prototype. This paper will conclude with a summary of the contributions of this work and some of the planned future works in Section 5.

2. Related Works

Traceroute has been a powerful and well-known debugging tool used to determine the hop by hop path from source to destination at the network layer by any one and

from anywhere in the Internet. During the early days of the Internet, traceroute was utilized extensively in identifying routing loops and black holes.

Van Jacobson implemented the first version of traceroute [3] based on the suggestions from Steve Deering. This version of traceroute works by sending out UDP or ICMP ECHO requests with an increasing value in the TTL field. The TTL field of the first packet is set to 1 and is incremented by 1 for each of the following packets*. This process continues until the TTL field has reached the value that is large enough for the packet to reach the destination, i.e., the ICMP PORT_UNREACHABLE response from the destination has been received, or until a maximum number of requests has been sent. When the TTL value is less than the required number of IP hops to reach destination, an ICMP TIME_EXCEEDED response will be generated by the IP router at which the TTL value reaches 0. By observing the received ICMP TIME_EXCEEDED or PORT_UNREACHABLE responses, the IP routers along the path from the source to the destination, as well as the destination itself, can be revealed.

Due to the widespread use of firewalls in the modern Internet, many of the UDP or ICMP ECHO request packets that traceroute sends out end up being filtered, making it impossible to completely trace the path to destinations. Layer four traceroute such as (LFT)[1] and tcptraceroute[2] were introduced. These variants operate almost the same way as traceroute, but send out TCP SYN packets instead of UDP or ICMP ECHO packets. As firewalls cannot differentiate one TCP SYN packet from another, ICMP TIME_EXCEEDED or PORT_UNREACHABLE responses will be echoed back, thus allowing LFT or tcptraceroute to penetrate more firewalls and tracing more destinations in the network.

Compared to traceroute series, i.e., traceroute, LFT and tcptraceroute, which all determine the hop by hop path that packets take from source to destination at the network layer, AppTraceroute works at the application layer, unveiling the application layer hops that queries take in the overlays. To have a vivid idea of how AppTraceroute differs from the former three, Figure 1 depicts a path consisting of two application layer hops, which are determined by AppTraceroute, and each of them in turn consists of a list of IP hops. While the application layer hops are determined by a single AppTraceroute process at the source, the IP hops in each application layer hop are determined by a separate traditional traceroute process running at the source of the application layer hop.

In this paper, the peer that initiates the query is called the source peer, the peers that forward the query to next hop peers are called forwarding peers, and the peers that the query ends up with are called destination peers. All the forwarding peers and the destination peers form the set which is termed the "reach of the query" in this paper. Accordingly, Peer 1 in Figure 1 is the source peer, Peer 2 a forwarding peer and Peer 3 a destination peer. The reach of the query is {Peer 2, Peer 3}.

*In its implementation, traceroute allows the sending of more than one UDP or ICMP ECHO requests per hop. As a matter of fact, the default value is 3.

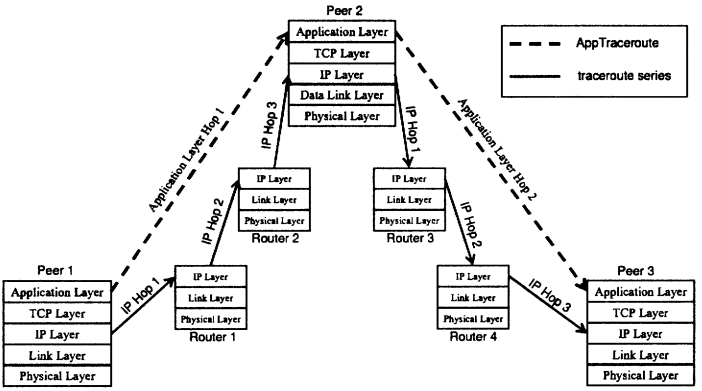

Figure 1: The Difference between AppTraceroute and Traceroute Series

3. Design

The goal of AppTraceroute is to determine the application layer paths that a query takes as well as the reach of the query after the query is initiated by a source peer. The characteristics of each peer in the reach of the query will be measured by AppTraceroute the same way as traceroute does for IP routers. The constituent IP hops for each application layer hop, along with their characteristics, will be revealed as well. During the operation, a source peer sends an AppTraceroute query. Upon the reception of an AppTraceroute query, each forwarding peer reacts accordingly and sends acquired results back to the source peer, which assembles all the received responses and interprets them at the end. The operation of AppTraceroute at the source peer as well as the actions performed by all the peers in a P2P network are illustrated in detail in the next two sections.

3.1 AppTraceroute Operation

The operation of AppTraceroute at the source peer can be roughly divided into three phases, as depicted in Figure 2. During the first phase and third phase, i.e., the AppTraceroute query formulation and handling phase and response interpretation phase, all actions happen solely on the source peer and no actions are required at the forwarding peers and the destination peers. In the response handling phase, however, AppTraceroute expects actions to be taken by forwarding and destination peers and responses to be sent back.

AppTraceroute Query Formulation and Handling

To obtain the paths that a regular query takes in a P2P network, an AppTraceroute query has to emulate the regular query so that it can be handled exactly the same as the corresponding regular query, and at the same time the AppTraceroute query has to trigger actions at forwarding peers and destination peers accordingly. Consequently, an AppTraceroute query has to be designed to search for the same item and to have the same format as the regular query. To accommodate this, the query format is

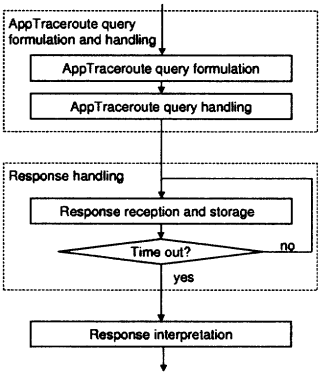

Figure 2: AppTraceroute Operation

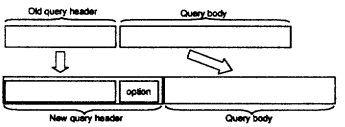

Figure 3: New Query Format

expanded with an option field in the header so that AppTraceroute queries can be differentiated from regular ones. Under this new query format, all regular queries will have the option field set to zero, whereas the option field of AppTraceroute queries are set to non-zero values. Figure 3 further illustrates the formation of the new query format.

Once an AppTraceroute query is formulated, the source peer handles the AppTraceroute query exactly the same way as a forwarding peer does to a received AppTraceroute query. It forwards the AppTraceroute query to all the next hop peers in the overlay as if the query was a regular query. Meanwhile, it starts tracerouting all the next hop peers to determine the IP hops along the path to each of them. The detailed handling of an AppTraceroute query at the source peer will be given in subsection 3.2 where AppTraceroute query handling at forwarding/destination peers is described. Once the AppTraceroute query has been sent and tracerouting processes have been handled, AppTraceroute enters its second phase of operation at the source peer.

Response Handling

When AppTraceroute enters its second phase of operation, the AppTraceroute query that was sent out in the first phase begins to reach its forwarding peers, to be forwarded to next hop peers, and ultimately to arrive at its destination peers. Each forwarding of the AppTraceroute query to a next hop will trigger the transmission of a response message to the source peer. A response message has three fields, i.e., $current$, $next$ and $result$. The $current$ field corresponds to the IP address of the current peer, i.e., the peer that initiates the response message, and the $next$ field is the IP address of the next hop peer, i.e., the peer to which the current peer forwards the query. The result field describes the constituent IP hops from the current peer to the next hop peer. In particular, the result field contains a list of four-tuples of the format $\langle ip, rtt_1, rtt_2, rtt_3 \rangle$. The i^{th} tuple in the result field contains the IP address of the i^{th} gateway router along the path from the current peer to the next hop peer and three measurements of the round trip delays from the current peer to the

<Current>	=	ipCurrent
<Next>	=	ipNext
<Result>	=	<ip1, rtt1, rtt2, rtt3> <ip2, rtt1, rtt2, rtt3> < ip3, rtt1, rtt2, rtt3 >

Figure 4: A Response Message

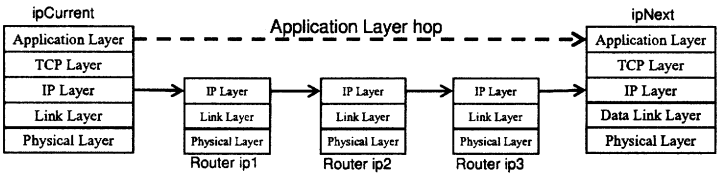

Figure 5: Graphical Representation of the Response Message

i^{th} gateway router. To illustrate this further, Figure 4 describes a response message with specific values for each of the three fields and the information conveyed by the response message is graphically represented in Figure 5.

In its second phase of operation, AppTraceroute at the source peer receives responses from forwarding peers, and records the received responses for further processing in the third phase. It continues to receive and save responses until all the expected responses have been received or a timeout value has been exceeded. Until then, AppTraceroute operation remains in its second phase at the source peer.

Response Interpretation

AppTraceroute constructs two directed graphs based on the received responses. The first directed graph $G_{app} = (V_{app}, E_{app})$ represents the application layer paths that the AppTraceroute query takes, wherein the vertex set V_{app} is the set of peers that the query has visited and a directed edge $(p, q) \in E_{app}$ indicates that peer p once forwarded the query to peer q. The second directed graph $G_{ip} = (V_{ip}, E_{ip})$, on the other hand, represents the IP layer routes that the AppTraceroute query takes, wherein the vertex set V_{ip} is the union of the set of the IP routers that the query has visited and the set of the hosts on which the visited peers run. Accordingly, a directed edge $(p, q) \in E_{ip}$ indicates that either router p forwarded the query to router q, host p forwarded the query to router q, or router p forwarded the query to host q. In addition, each edge in E_{ip} is labeled by the average one-way delay introduced by the corresponding IP hop. Figure 6 details the creation of the two graphs.

The visualization of the two graphs can be either a plain text printout or graphical representation, which is out of the scope of this paper and hence it will not be described here.

3.2 Actions Required at Peers in a P2P Network

Figure 7 sketches the actions a peer takes in a P2P network upon the arrival of a query. No matter which type the received query is, the peer determines all the next hop peers and forwards the query to each of the next hop peers. The peer then checks

$V_{app} = E_{app} = V_{ip} = E_{ip} = \emptyset$;
FOR each response $\langle ipCurrent, ipNext, (ip_i, rtt_i^1, rtt_i^2, rtt_i^3), i = 1...k \rangle$
  $V_{app} = V_{app} \cup \{ipCurrent, ipNext\}$;
  $E_{app} = E_{app} \cup \{(ipCurrent, ipNext)\}$;
  $V_{ip} = V_{ip} \cup \{ipCurrent, ipNext\}$;
  IF $ipCurrent \neq ip_1$
    $E_{ip} = E_{ip} \cup \{(ipCurrent, ip_1)\}$;
  ENDIF
  IF $ip_k \neq ipNext$
    $E_{ip} = E_{ip} \cup \{(ip_k, ipNext)\}$;
  ENDIF
  $V_{ip} = V_{ip} \cup \{ip_1\}$;
  FOR $i = 2...k$
    $V_{ip} = V_{ip} \cup \{ip_i\}$;
    $E_{ip} = E_{ip} \cup \{(ip_{i-1}, ip_i)\}$;
    labelEdge$\langle (ip_{i-1}, ip_i) \rangle = \frac{rtt_i^1 + rtt_i^2 + rtt_i^3}{6} - \frac{rtt_{i-1}^1 + rtt_{i-1}^2 + rtt_{i-1}^3}{6}$
  ENDFOR
ENDFOR

Figure 6: The Creation of the Two Graphs

the option field of the query to find out whether the query is an AppTraceroute query. If the query is an AppTraceroute query, i.e., the option field is not zero, the query is further processed as an AppTraceroute query. As mentioned before, the handling of an AppTraceroute query at the source peer and each of the forwarding peers is the same, whereas the handling at destination peers is slightly different. Figure 8 describes the handling of AppTraceroute queries in detail.

Given an AppTraceroute query, the current peer, either a source peer, a forwarding peer, or a destination peer, checks whether the query has arrived at its destination. In case that the current peer is actually the destination of the query, the current peer, i.e., the destination peer, discards the AppTraceroute query and takes no further actions. Otherwise, the current peer first retrieves its own IP address, *ipCurrent*. Then for each of the next hop peers that the AppTraceroute query has been forwarded to, the current peer retrieves the next hop peer's IP address *ipNext*, and starts a traditional traceroute process to discover the IP hops from the current peer to the next peer. At the end of the traceroute process, the current peer converts the traceroute result into an ordered list of four-tuples in accordance with the definition of the response message, and sends a response message back to the source peer. In the response message, the *current* field is set to *ipCurrent*, the *next* field is set to *ipNext*, and the *result* field is filled with the ordered list of four-tuples.

In summary, AppTraceroute can determine the routes that a query takes in a P2P network not only at the application layer, but at the IP layer as well. However, this is achieved at the cost of some minor modifications at each peer in the P2P net-

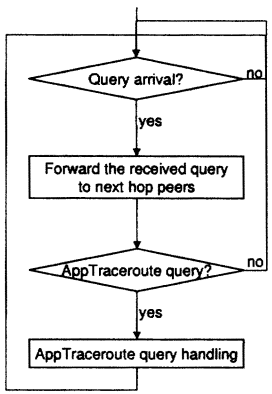

Figure 7: Modified Behaviors of Peers in a P2P Network

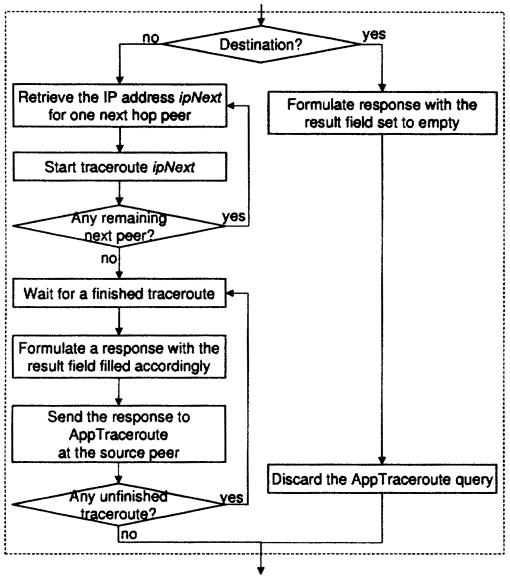

Figure 8: AppTraceroute Query Handling

work as described in this section. In addition, one traditional traceroute process is initiated and one response message is transmitted for each application layer hop that AppTraceroute queries undergo. This will inevitably introduce a certain amount of traffic into the network and the actual amount varies from one P2P network to another. In the subsequent sub sections, the amount of traffic introduced by a single AppTraceroute query is analyzed at the source peer and at a forwarding peer in three representative P2P networks, i.e., Gnutella for pure P2P networks, KaZaA for hybrid P2P networks and Pastry for structured P2P networks. The introduced traffic, fortunately, is acceptable.

3.3 AppTraceroute in Pure P2P Networks

In a pure P2P network like Gnutella [10], a source peer sends a query to all its neighbors, and its neighbors, upon the reception of the query for the first time, forward the query to all their neighbors except the one from whom they just received the query. To control the extent of flooding, a TTL mechanism is utilized, which is set to a default value at a source peer and decremented by 1 at each forwarding peer. Peers stop forwarding a query once the TTL value reaches zero.

To help quantify the amount of traffic introduced, Table 1 defines a list of parameters that will be used in the following analysis. For the ease of presentation, the length of messages, i.e., l and L, is assumed to have already taken into consideration the overhead introduced by lower layers in the protocol stack.

In Gnutella, a forwarding peer initiates $n - 1$ traditional traceroute processes for

Table 1 Parameter Definition

Name	Description
n	The number of neighbor peers on average
r	The default TTL value
h	The number of IP hops per application layer hop on average
d	The average delay in millisecond per application layer hop
m	The number of traceroute probes per IP hop
l	The length in bytes of the traceroute probes and responses
L	The length of AppTraceroute queries and their responses

each received AppTraceroute query and these processes are supposed to finish in about $2 \times d$ milliseconds. Meanwhile, it has to forward each AppTraceroute query $n-1$ times and send $n-1$ AppTraceroute responses back to the source peer. With all these taken into consideration, a forwarding peer is expected to send and receive $2(n-1)mhl + 2(n-1)L$ bytes in $2d$ milliseconds, an average of $\frac{(n-1)mhl+(n-1)L}{d}$ bytes per millisecond for a time duration of about $2d$ milliseconds.

The situation is slightly complicated for the source peer. During the time period $[0, 2d]$, the source peer sends the AppTraceroute query n times and handles n traceroute processes for all next hop peers. Afterwards, it is expected to receive r waves of AppTraceroute responses triggered by the AppTraceroute query, wherein the i^{th} wave has about $n(n-1)^{i-1}$ responses [10] that begin to arrive at the source peer $2id$ milliseconds after the source peer sends the AppTraceroute query. Assuming that all the responses within the i^{th} wave arrive at the source peer within the time period $[2id, 2(i+1)d]$, the bandwidth consumption at the source peer for the i^{th} wave is about $\frac{n(n-1)^{i-1}L}{2d}$ bytes per millisecond on average. Assembling all these together, equation (1) gives the average bandwidth consumption incurred by each AppTraceroute query at the source peer as a function of time t, which starts from time 0 when the AppTraceroute query is sent, to the time when all the responses have been received, i.e., $2(r+1)d$ milliseconds.

$$\begin{aligned} BandwidthConsumption(t) &= \frac{2nmhl + nL}{2d}, \ 0 \le t < 2d \\ &= \frac{n(n-1)^{i-1}L}{2d}, \ 2id \le t < 2(i+1)d \, \forall i = 1...r \end{aligned} \tag{1}$$

3.4 AppTraceroute in Structured P2P Networks

Instead of being flooded to all neighboring peers, a query in a structured P2P network like Pastry [4] is only forwarded to the best candidate peer that the forwarding peer is aware of. As such, queries are guaranteed to reach their destinations in at

most $\lceil \log_{2^b} N \rceil$ application layer hops, where N is the total number of peers in the structured P2P network and b is the number of bits per digit in a peer's identifier [4].

Given the same definitions and conditions for h, d, m, l and L as for pure P2P networks, a forwarding peer in a structured P2P network is expected to forward AppTraceroute query only once and at the same time to traceroute only one of its neighbors, thus sending an AppTraceroute response only once. This results in a total traffic of $2mhl + 2L$ bytes in $2d$ milliseconds, which equals a bandwidth consumption of $\frac{mhl+L}{d}$ bytes per millisecond for about $2d$ milliseconds. Similarly, a source peer in a structured P2P network is expected to initiate only one traceroute process and to receive at most $\lceil \log_{2^b} N \rceil - 1$ waves of AppTraceroute responses. However, one wave contains only one response in a structured P2P network. Accordingly, Equation (2) gives the average bandwidth consumption by the source peer as a function of time t, where $0 \leq t < 2d(\lceil \log_{2^b} N \rceil + 1)$.

$$\begin{aligned} BandwidthConsumption(t) &= \frac{2mhl + L}{2d}, \ 0 \leq t < 2d \\ &= \frac{L}{2d}, \ 2id \leq t < 2(i+1)d, \ \forall i = 1...\lceil \log_{2^b} N \rceil \end{aligned} \tag{2}$$

3.5 AppTraceroute in Hybrid P2P Networks

In a hybrid P2P network like KaZaA, regular nodes(RN) are dynamically elected as super nodes(SN) and all the SNs organize themselves into an overlay network. A regular node then attaches itself to one of the SNs and this SN becomes the parent node of the RN. In KaZaA, a RN always forwards its queries to its parent node, which in turn forwards the queries to other SNs. Queries initiated by a SN are forwarded to other SNs as well. Due to the use of encryptions in the signalling traffic, how a SN forwards queries to other SNs in KaZaA is unknown. [9] mentions that queries are forwarded to one or more SNs while [12] argues that queries are flooded among SNs. As a SN has about 40-50 SN neighbors [9], the extent of flooding, if it was the case, would be significant. On the other hand, the "one or more" behavior in [9] is relatively vague. For the purpose of this study, i.e, to understand the number of responses per wave after an AppTraceroute query is initiated, a SN is assumed to forward a query to next hop peers with a probability that decreases with the number of application layer hops that the query has travelled. In particular, the probability has the following probability density function: $f(x) = e^{-\lambda x}$ where x is the number of application layer hops a query has travelled so far and $\lambda \geq 0$ is a parameter that controls the extent of flooding. When $\lambda = 0$, queries are flooded and when λ gets positively larger, e.g. 3, the "one or more" behavior can be emulated.

In this work, λ is set to 3 and the average number of responses per wave is simulated for the first 5 waves when a SN is the source peer in a KaZaA overlay that is assumed to be well balanced with each SN having 50 neighbor SNs. At the same time, the number of SNs that a query is forwarded to is acquired for the KaZaA overlay as well.

Table 2 Branching Factor and Bandwidth Consumption at a Forwarding Peer

NetworksThe Branching Factor		Bandwidth Consumption (Bytes per second)
Pastry	1	23500
Gnutella	3	70500
KaZaA	2.59	60767

Gnutella and Pastry are studied as representatives for pure and structured P2P networks respectively. Both Gnutella and Pastry overlays are also assumed to be well balanced and to have as many peers as required for a query to travel 5 application layer hops. For Gnutella, the number of neighbors is set to its default value 4 [10]. An application hop in this work is assumed to have 10 IP hops and the traditional traceroute probes 3 times for each IP hop. In addition, each traceroute probe and its response have a length of 60 bytes while an AppTraceroute query and its responses have an average length of 80 bytes.

Table 2 lists the number of next hop peers a query is forwarded to, i.e., the branching factor, and the bandwidth consumed by a forwarding peer on average due to the initiation of traceroute processes and the transmission of responses back to the source peer. While Pastry consistently makes one forwarding only, Gnutella and KaZaA forward queries to 3 or 2.59 peers on average, respectively. For KaZaA, the branching factor is determined by solving $x+x^2+x^3+x^4+x^5 = Number\ of\ Visited\ Peers$, where the number of peers that a query can visit within 5 application layer hops is acquired during the simulation. As it shows in the table, the bandwidth consumption at a forwarding peer grows linearly with the branching factor, with Pastry consumes the least, Gnutella the most and KaZaA in the middle.

Figure 9 describes AppTraceroute bandwidth consumption at the source peer for Gnutella, Pastry and KaZaA. As illustrated in Figure 9, the source peer in all the studied networks consumes a relatively larger amount of bandwidth during time period $[0, 2d]$. This is due to the fact that a source peer has to send AppTraceroute queries and to handle the traditional traceroute processes for the first application hop. Afterwards, the source peer is only expected to receive waves of responses periodically. Since the source peer in Pastry consistently receives one response per wave, its bandwidth consumption levels off for the remaining waves. The source peer in KaZaA receives responses at an increasing pace at the beginning and then the number of responses decreases to almost nothing, thus yielding a high amount of bandwidth consumption at the beginning and almost nothing at the end. The bandwidth consumption by the source peer in Gnutella, however, keeps climbing due to the fact that more and more responses are sent back as queries travel further in Gnutella.

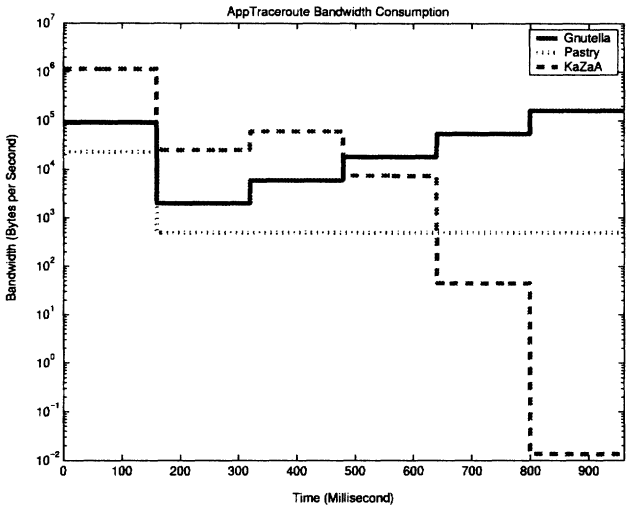

Figure 9: AppTraceroute Bandwidth Consumption at a Source Peer

4. Implementation

The concept of AppTraceroute has been implemented and experimented within a prototype of a structured P2P network based on Pastry[4]. More precisely, Pastry is only a distributed platform that implements scalable object location and routing. P2P applications, instead of building their own object location and routing facilities individually, can share the facilities provided by Pastry, which provides a common API [8] to facilitate the development of P2P applications.

In the prototype, each peer is given a 128*bit* identifier and manages a list of objects whose identifiers are numerically very close to the peer's identifier. To search an object, a query with the same identifier as the object is formulated and is then forwarded to a peer with an identifier that is numerically the closest to the object's identifier. The concept of AppTraceroute is completely incorporated into the prototype, thus for the first time offering users and researchers the capability to determine the paths that queries take in the P2P network. This can be used by researchers to study the dynamics of the P2P network and to establish more efficient and scalable P2P networks. In addition, the determined paths, both in terms of a series of application layer hops and a series of IP hops are visualized. Figure 10 is a screen shot that describes the operation of AppTraceroute in the prototype. At the top of the figure, users can specify an object to be sought and once the AppTraceroute button is pressed, an AppTraceroute query is formulated and the paths that the query takes in the P2P network are visualized at the bottom of figure. In the visualization, routers and hosts are represented by distinguishing icons and users can click on an item, e.g.,

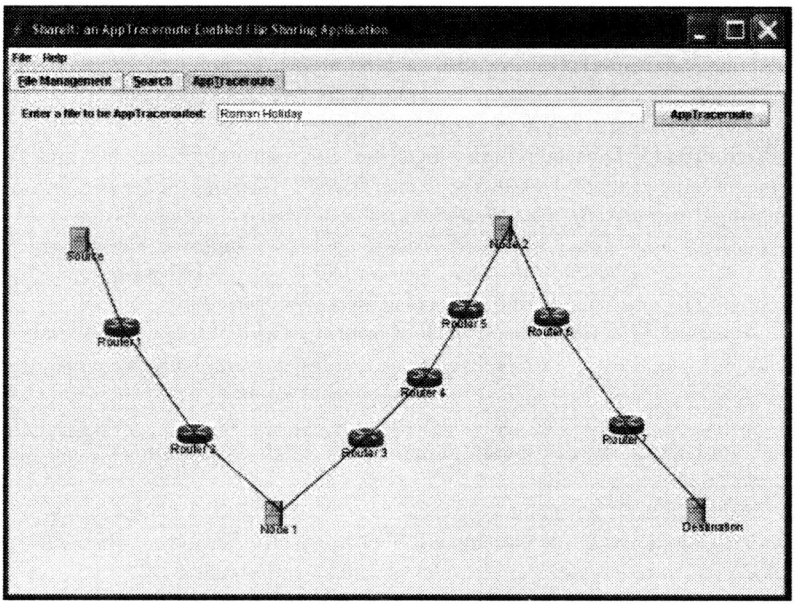

Figure 10: AppTraceroute Operation in the Prototype

a router, a host or a link, to retrieve a list of properties of that item, such as the IP address of the router or host, the delays of a link, etc.

5. Conclusion

In this paper, we proposed and described AppTraceroute, a traceroute like mechanism that operates at the application layer. We analyzed the bandwidth consumption at the source peer and at a forwarding peer when introducing AppTraceroute into pure P2P networks, hybrid P2P networks and structured P2P networks. In addition, we implemented and experimented with AppTraceroute in a structured P2P network that fully incorporates the concept of AppTraceroute. In the future, we will incorporate AppTraceroute into other P2P networks, such as Gnutella and KaZaA. Also, the bandwidth consumption of AppTraceroute at source peers and forwarding peers will be measured and compared with the analysis in this paper. As Gnutella and KaZaA have gained their popularity and boast large numbers of users in the Internet, the coexistence of AppTraceroute compliant and non-compliant peers will be inevitable when incorporating AppTraceroute into these networks. Hence, we will investigate the possible negative influence on AppTraceroute operation, and will study some adaptive algorithms to improve the correctness and precision of AppTraceroute in such networks.

References

[1] Layer Four Traceroute(LFT). Available at http://oppleman.com/lft/.

[2] tcptraceroute. Available at http://michael.toren.net/code/tcptraceroute/.

[3] traceroute. Available at http://www-nrg.ee.lbl.gov/.

[4] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, November 2001.

[5] Ben Y. Zhao, John Kubiatowicz and Anthony Joseph. Tapestry: An Infrastructure for Fault-tolerant Wide-area Location and Routing. Ucb tech. report ucb/csd-01-1141, University of California at Berkeley, April 2001.

[6] D. G. Andersen, H. Balakrishnan, M. F. Kaashoek, and R. Morris. Resilient Overlay Networks. In *18th ACM Symposium on Operating Systems Principles*, October 2001.

[7] E. Korpela, D. Werthimer, D. Anderson, J. Cobb and M. Lebofsky. SETI@home-Massively distributed computing for SETI. In *Computing in Science and Engineering*, January 2001.

[8] Frank Dabek, Ben Zhao, Peter Druschel, John Kubiatowicz, and Ion Stoica. Towards a Common API for Structured P2P Overlays. In *The Second International Workshop on Peer-to-Peer Systems(IPTPS)*, February 2003.

[9] J. Liang, R. Kumar, and K. Ross. The KaZaA Overlay: A Measurement Study. In *Technical Report*, September 2004.

[10] J. Ritter. Why Gnutella Can't Scale. No, Really. Available at http://www.darkridge.com/ jpr5/doc/gnutella.html., February 2001.

[11] M. Castro, P. Druschel, A-M. Kermarrec and A. Rowstron. SCRIBE: A large-scale and Decentralised Application-level Multicast Infrastructure. In *IEEE Journal on Selected Areas in Communications*, October 2002.

[12] N. Leibowitz, M. Ripeanu, and A. Wierzbicki. Deconstructing the KaZaA Network. In *3rd IEEE Workshop on Internet Applications(WIAPP)*, June 2003.

[13] S. Garfinkel. Peer-to-Peer Comes Clean. MIT Technolgoy Review Magazine, October 2004.

[14] S. Ratnasamy, P. Francis, M. Handley, R. Karp and S. Schenker. A Scalable Content-Addressable Network. In *The 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, 2001.

[15] S. Srinivasan and E. Zegura. Network Measurement as a Cooperative Enterprise. In *The First International Workshop on Peer-to-Peer Systems(IPTPS)*, March 2002.

[16] W. Liu, R. Boutaba and J. Hong. pMeasure: A Tool for Measuring the Internet. In *The Second Workshop on End-to-End Monitoring Techniques and Services*, October 2004.