Responder Anonymity and Anonymous Peer-to-Peer File Sharing*

Vincent Scarlata† scarlata@cc.gatech.edu
Brian Neil Levine† brian@cs.umass.edu
Clay Shields* clay@cs.georgetown.edu

† Dept. of Computer Science, University of Massachusetts, Amherst, MA 01003
* Dept. of Computer Science, Georgetown University, Washington, DC, 20007

Abstract

Data transfer over TCP/IP provides no privacy for network users. Previous research in anonymity has focused on the provision of initiator anonymity. We explore methods of adapting existing initiator-anonymous protocols to provide responder anonymity and mutual anonymity. We present Anonymous Peer-to-peer File Sharing (APFS) protocols, which provide mutual anonymity for peer-to-peer file sharing. APFS addresses the problem of long-lived Internet services that may outlive the degradation present in current anonymous protocols. One variant of APFS makes use of unicast communication, but requires a central coordinator to bootstrap the protocol. A second variant takes advantage of multicast routing to remove the need for any central coordination point. We compare the TCP performance of APFS protocol to existing overt file sharing systems such as Napster. In providing anonymity, APFS can double transfer times and requires that additional traffic be carried by peers, but this overhead is constant with the size of the session.

1 Introduction

The Internet has become a pivotal medium for information dissemination and content sharing across the globe. While the the Internet Protocol suite (TCP/IP) takes into account numerous network performance issues, data transfer on the Internet inherently provides no privacy. All packets used for TCP data transfer can be easily traced back to the host specified in the IP source address field.

Protocols for maintaining anonymous Internet connections that have been proposed previously [12, 5, 11, 13] have specifically addressed maintaining the anonymity of the *initiator* of a connection. However, providing anonymity to a user awaiting connections — *responder anonymity* — is also crucial, as responder anonymity would allow privacy for network interactions such as anonymous web servers, anonymous e-mail, and private peer-to-peer file sharing.

In this paper, we offer a number of contributions with regard to comprehensive anonymity. First, we provide two techniques for adapting current initiator protocols to provide responder anonymity — one based solely on unicast communication and a second that takes advantage of multicast communication. Second, we show how protocols for initiator and responder anonymity can be composed to provide *mutual anonymity* for two parties forming a connection over the Internet. Our specific design of a mutually anonymous protocol is called the Anonymous Peer-To-Peer File Sharing (APFS) protocol. APFS takes advantage of the peer-to-peer environment to solve problems specific to maintaining responder anonymity.

We offer two versions of APFS based on our solutions to responder anonymity: one based on unicast transmissions, and a second which makes use of multicast routing protocols. We show that the use of multicast routing can eliminate the need for any centralized coordinator for the system, yielding, for APFS, a completely distributed file sharing service. Our unicast solution requires a bootstrapping point. We evaluate the network performance of APFS in comparison to the overt peer-to-peer file transfer protocol used in Napster and Gnutella. As APFS directly applies previous anonymous routing protocols, we do not offer a security analysis of those protocols. However, we do argue that we have not introduced any security flaws.

In the next section, we introduce recent work in peer-to-

*This study was supported in part by grants 0087639 and 0087482 from the National Science Foundation, and 2000-DT-CX-K001 from the U.S. Department of Justice, Office of Justice Programs. Its contents are solely the responsibility of the authors and do not necessarily represent the official views of the Department of Justice.

peer file sharing and anonymity. In Section 3, we propose two techniques for responder anonymity, and we detail the operation of APFS. In Section 4, we evaluate by simulation the additional network overhead caused by APFS as compared to overt file sharing techniques. Lastly, Section 5 summarizes our contributions.

2 Background

While there has been little specific research in the area of anonymous file-sharing, extensive research has been done in file sharing and anonymity separately, which we review in this section.

Although we make use of multicast routing in some of our proposals in this paper, we do not review IP multicast because of space limitations; extensive overviews of its design and operation can be found elsewhere [4, 6]. Multicast is not yet widely available, but, UUnet/Worldcom and Sprint have begun deploying PIM-Source Specific Multicast (PPM-SSM) [9, 8] multicast implementations.

2.1 Peer-to-Peer File Sharing

Distributed files systems can be classified into two broad categories, *centralized* and *distributed*. A centralized system has a main server through which all communication is coordinated; a distributed system lacks a centralized server. In general, a centralized file sharing system has a simpler search design and can return search queries faster than distributed systems. This is due to the fact that the server knows the location of all files, and as the user base increases, additional hardware can be added to the server to compensate for the heightened load. However, the server can be a bottleneck on the performance of the system, and if the server is unavailable the entire system halts. In the case of the Napster system of *peer-to-peer* file sharing, the central server is responsible for storing lists of available files and servicing queries for users searching for files; once peers learn the location of a desired file, they then directly contact the remote peer for to request the file.

In a distributed system there is no server: the responsibility of coordinating communication is distributed across the users. In order to conduct a search, a query must be forwarded throughout the members of the system, as there is no centralized knowledge of where resources exist. This querying process limits the speed of searching and can be expensive in terms of network traffic. Gnutella is a distributed peer-to-peer system, and is reviewed in more detail below.

2.1.1 Napster

One of the most well known peer-to-peer file sharing systems is Napster. While Napster was designed specifically for the transmission of audio files, its theory can be employed for any peer-to-peer needs. Napster is a fully centralized system, and the Napster corporation maintains a cluster of servers for conducting searches. The Napster protocol is proprietary, but is simple to reverse engineer. *Open-Nap* [2] is the open-source reverse engineering of the protocol used by Napster. Because APFS is partially based on the Open-Nap protocol, we review its operation.

To join an on-going Napster session, a client sends the central server its *login id* and *file list*, which is a list of files that the client has available to share. To search the Napster community's files, a client sends a *query* to the server. The server replies with a list of matching files along with the name of the user sharing the file. The user may choose a remote file, and the client contacts the server again, requesting the IP address of that user. Finally, the client contacts the user directly and requests that the file be sent.

The Napster system lacks any significant security or privacy measures and has a single point of failure at the server cluster.

2.1.2 Gnutella

Gnutella is a popular, fully-distributed, file-sharing system. We review its operation here summarily, enough to point out the lack of anonymity in Gnutella[1].

Gnutella has no central point for querying or file transfer, though a coordination point allows new clients to learn of other clients running the protocol. In order to conduct searches, each client picks seven other clients to use for querying. If a query cannot be answered by these neighbors, the query is relayed to the neighbors of those neighbors. This process can recurse for up to 25 hops, and the cost in traffic for queries can be extreme. While the immediate originator of a query is not obvious except to its neighbors, querying is not entirely anonymous. Discovering who originated a query is as simple as answering a request falsely and waiting for the originator to directly request the desired file, as specified by the Gnutella design.

[1] Unfortunately, the popular press often claims Gnutella is anonymous; e.g., see J. Brown "The Gnutella paradox", Salon, Sept. 29, 2000.

2.2 Anonymity

Because IP addresses can uniquely identify users and their machines on a network, the goal of anonymous routing protocols is to disassociate a user's IP address from the traffic the user initiates on the network.

Previous research on anonymity has largely focused on initiator anonymity, with few exceptions [10, 7]. However, such protocols protect only half the connection. Our goal is to protect the anonymity of the initiator and the responder of a connection.

Most protocols that are designed for anonymous IP communication organize a group of members that forward messages for each other, trading bandwidth for anonymity. Examples of this type of protocol are Onion Routing [11], Hordes [13], and Crowds[12]. The general method of these protocols is that an initiator creates a path through the group, and the last member of the path, called the *tail node*, contacts the responder. The tail node then forwards the request to the responder and returns the reply back to the initiator. This method provides anonymity, because at each step on the path it is not possible to tell if the previous hop initiated the message, or was forwarding it for some other member.

This approach to anonymity requires other considerations be taken with respect to maintenance on the group [12]. Specifically, users are required to join and exit the group only at periodic intervals; all group members clear and reform their paths each interval. The reason for this *periodic join* is that a new member attempting to form a new path would be immediately recognizable as the initiator of that path if all other group members had already constructed, long-standing paths.

We choose to use Onion Routing as the base of our protocol; however, any path-based approach can be used. In the next section we discuss Onion Routing in detail.

2.2.1 Onion Routing

Onion Routing is a path-based anonymous protocol, which uses well-known public keys of participants to hide path information. A certificate authority can be used to store and distribute these keys [11, 14]. To send a message in an Onion Routing session, the initiator randomly chooses a path of proxies, then encrypts the message multiple times using the public keys of each proxy in the desired route creating a *onion*. For example, the onion for the path $P0$, $P1$, $P2$,$P3$, D is

$$\{P1, \{P2, \{P3, \{D, M\}_{K_{P3+}}\}_{K_{P2+}}\}_{K_{P1+}}\}_{K_{P0+}}$$

where $\{M\}_{K_{P+}}$ indicates a cipher created by encrypting a message, M, with P's public key.

Each proxy on the path decrypts the message to find the identity of the next proxy, then forwards the encrypted payload that only that next proxy can decrypt. This process repeats until the last proxy sends the message to its final destination. To enable the destination to respond, proxies pass a session ID along with the onion to their neighbors on the path. When the destination responds to the message it is routed back along the same path in the reverse direction.

2.3 Degradation of Anonymity

Our previous work [15] has shown the inability of protocols to maintain high degrees of anonymity with low overhead in the face of persistent attackers. This occurs because over time the path between initiator and responder must be reformed, and each path reformation provides attackers the opportunity to learn some information about the initiator. Over enough path reformations, the attackers can determine with increasing probability the initiator of a connection. With Onion Routing, the expected number of path reformations required for c attackers to determine the initiator out of n participants is $O\left(\left(\frac{n}{c}\right)^l\right)$ [15], where l is the length of the path between the initiator and responder. A persistent, coordinated attack applies to all protocols for untraceability as well.

In the next section, we show how responder anonymity is affected by this result.

3 Comprehensive Anonymity

Previous work has focused primarily on providing anonymity for the initiator of a connection. In this section we discuss protocols and techniques for hiding the identity of a responder, and for providing completely anonymous communication in which both the initiator and responder are anonymous.

There are two prominent difficulties with providing responder anonymity as compared to providing initiator anonymity.

1. Protocols for responder anonymity must be designed to route packets to a responder without the opportunity to set up a path between initiator and responder ahead of time, since we must assume the responder does not know the which initiator will desire communication.

2. Responders tend to be servers. Servers commonly need to stay up for a long period of time, which is in conflict with previous work on the degradation of anonymous protocols [15]. (See Section 2.3.)

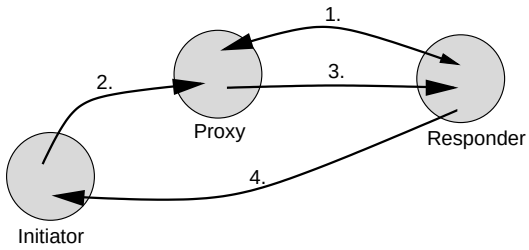

Figure 1: Steps for using proxies for responder anonymity.

In this section, we provide solutions to both problems.

For the first problem, presently we propose two different techniques to forward messages to an anonymous responder, which can then set up a connection to the initiator using a protocol for initiator anonymity (See Section 2). The first technique, which we call Proxy for Responder Anonymity (PRA), utilizes unicast routing and a proxy, which does not have to be trusted. The second technique, which we call Multicast Responder Anonymity (MRA), utilizes multicast routing and no proxies.

For the second problem, we take advantage of the properties of peer-to-peer networking services to allow the use of servers in such a way as to avoid the problem of long-term anonymous degradation.

3.1 Responder Anonymity

Any protocol for initiator anonymity can be used to provide responder anonymity. A publicly-advertised proxy can accept and relay datagrams to an anonymous responder. The technique requires only a few additional steps during which the responder contacts the proxy anonymously prior to receiving communications from any initiator. Figure 1 illustrates the technique.

Step 1: The responder, R, sets up a connection to a known proxy, P, using a protocol that provides initiator anonymity. The proxy maintains a public alias for R, without knowing the true identity of the responder.

Step 2: Packets from an initiator are sent to the proxy over the anonymous channel, encapsulating data and a header that specifies the alias of the responder.

Step 3: The proxy decapsulates received data and forwards them over the pre-established anonymous channel to the responder.

Step 4 (optional): To avoid the bottleneck of the proxy, the responder could set up a connection to the initiator using a protocol for initiator anonymity.

Anonymous responders may wish to have more than just their alias advertised. For example, the proxy may also identify the type of service the responder is providing. The responder may also wish to generate a public/private key pair and use it to sign the alias (along with a time stamp) so that past correspondents can identify responders from past sessions. The proxy need only be trusted to not engage in a denial-of-service attack by dropping requests, but does not have to be trusted to maintain the responder's anonymity.

3.1.1 Proxyless Techniques

A proxy can be avoided altogether if the responder listens on a multicast address for new requests. An equivalent idea has been put forward in the past for broadcast networks [10]; multicasting is a natural mechanism for applying this technique to IP networks. Instead of advertising the proxy's address and alias, the responder r advertises a *responder alias*: (m, id), where m is a multicast address, id is a random number. The id serves as a unique marker in case more than one anonymous responder chooses to listen on the same multicast address.

The choice of a multicast group may be one already in use by another user on the same network as the responder; hijacking an existing group allows the responder to be indistinguishable from all other receivers of the multicast group.

Step 1: R chooses m, id, and a public key K_r and advertises these values on a public site (e.g., IRC channels, web sites, newsgroups, etc).

Step 2: The initiator I sends its message to the multicast group. This transmission reveals I's identity.

$$I \rightarrow m : id, \{\text{data}\}_{K_r} \quad (2)$$

Step 3: Once the responder receives the initiator's packet on the multicast address, it checks that the alias in the data portion of the packet is its own. The reply can be sent after a connection to the initiator is set up using a protocol for initiator anonymity, just as in PRA.

$$A_r \rightarrow I : \text{(initiator anonymity set up from R to I)} \quad (3)$$

The use of multicast can create more traffic than proxies, though our past work has shown that this is not always so [13]. However, proxies require the cooperation of a third party host, even though it need not be trusted to hide the identity of the responder. More details on using multicast for return paths to an initiator are in presented in our previous work on the Hordes protocol. The Hordes protocol is especially adaptable to multicast-based responder anonymity as initiators in Hordes already use multicast for their return path.

3.2 Mutual Anonymity - APFS

We have shown that responder anonymity is made possible by simple modifications to initiator-anonymous protocols. In this section, we address two other important problems. We show that providing *mutual anonymity* for both the initiator and responder simultaneously simply requires composition of both types of anonymous protocols. And, we address the conflict between long-lived servers and the degradation of anonymous protocols [15] by taking advantage of peer-to-peer networking. With peer-to-peer file services, such as Napster, file transfers are the responsibility of peers — the server performs only the duty of having a central location to store and query file indexes. APFS removes the single central server by recruiting peers to fill its position for a time short enough that they will not be revealed by anonymous degradation [15]. The peers therefore share the servers responsibilities serially, and shift those responsibilities between peers before anonymous degradation occurs.

In our explanation of the APFS protocol, we make the distinction between *overt clients*, where hosts act overtly to create anonymous routes, and *anonymous peers*, where hosts anonymously contact each other for the purposes of file sharing. APFS accordingly has two separate stages:

1. *Initialization*: clients join together to form anonymous connections between themselves (e.g., using Crowds, Hordes, or Onion Routing). All messages in this stage are overt, as the group membership is required to be known.

2. *Peer-to-peer services*: peers anonymously relay queries to the current server; the server anonymously answers queries, providing information allowing contact with peers who have the requested file. Periodically, new anonymous servers are chosen. All messages in this stage are anonymous.

APFS relies only on unicast communication, but requires an untrusted proxy called a *coordinator* to initialize the session. However, later in this section, we show how the coordinator can be replaced with multicast communication between clients, removing the single point-of-failure. For clarity, our subsequent discussion assumes that Onion Routing is the anonymous protocol used by all participants throughout the session, but any protocol will work.

3.2.1 APFS: Initialization

APFS begins with the start of a *coordinator*, which is a bootstrapping point for other clients. The coordinator's IP address and certifiable public key is well-advertised to potential clients (e.g., using web pages, e-mail, or IRC). Multiple sessions can be bootstrapped at the same coordinator when distinguished by unique identifiers. A client, I, begins the protocol by sending a join message to the coordination point, C, that contains its IP address and a session ID, S.

I	Initiator
p	Peer (anonymous client)
v	Server
C	Coordinator
M	Multicast address of responder
IP_n	IP address of client n
$A(n)$	Anonymous transmission from peer n
S	Session ID
T_n	Tail node of initiator n
N_v	Nonce chosen by server v
$time_i$	Time of reset i
f	Frequency of resets (in time)
ID_n	Anonymous alias ID of peer n
F_i	File i

Table 1: Table of Variables.

$$I \rightarrow C \quad : \quad IP_I, S \tag{1}$$

The client learns from the coordinator the set of other clients available for forming anonymous routes. New members must be allowed to join periodically. Therefore, as in many anonymous protocols, routes are periodically torn down and recreated, a process called *resetting*; otherwise, new routes in the group can be associated with new members [12].

The coordinator returns the time of the next reset, $time_1$, the frequency of resets, f, and a list of other clients, enabling the new client to begin construction of paths through the anonymous group.

$$C \rightarrow I \quad : \quad S,\ \text{time}_i, f, IP_1, \ldots, IP_n \tag{2}$$

Optionally, these messages may use authentication and authorization mechanisms, which we do not include here. Starting at $time_1$, paths are formed using Onion Routing (see [11] for a complete description of that process).

3.2.2 APFS: Starting Servers

After initialization, willing peers begin announcing themselves as servers to the coordinator and users begin issuing search queries to servers. All messages at this stage are sent anonymously.

Server Step 1. Peers willing to act as a query server send an anonymous message to the coordinator, encrypted with the coordinator's public key. The message

includes a unique server identifier, ID_v (a large random number should suffice), a nonce, N_v, the current tail node of the server, T_v, the *Server-Start* flag, and the associated session ID, S.

$$A(v) \rightarrow C : \{ID_v, N_v, T_v, \textit{Server-Start}, S\}_{K_{C+}} \quad (3)$$

Here we have denoted $A(I) \rightarrow C$ to mean that the node I uses its anonymous protocol to send the message. The use of the nonce and encryption prevents a denial-of-service attack described below. Optionally, the coordinator may acknowledge the message. The above message is resent to the coordinator whenever a server's tail node changes due to a path reset; a server's ID_v stays constant over path resets.

Server Step 2. The server begins waiting for peers to anonymously send lists of files to share. Upon receiving a file list from a user, the server records the newly available file list for future queries.

Server Step 3. Servers then process queries. For a query from an anonymous peer P yielding N results, the server sends the peer the information about each match. For each result n, this includes the filename, F_n, the anonymous ID of the sharing peer, ID_n, and the tail node for which the user should use to contact the sharing peer, T_n.

$$A(v) \rightarrow A(P) : (F_1, ID_1, T_1), \ldots, (F_N, ID_N, T_N) \quad (4)$$

Server Step 4. When a server wants to stop participating in the session, it must first remove itself from the server lists stored by the users. The server sends a message to the coordinator declaring itself a non-server encrypted with the coordinator's public key. This message is the same as message (3) with the start flag changed to *Server-Stop*.

$$A(v) \rightarrow C : \{ID_v, T_v, \textit{Server-Stop}, N_v + 1, S\}_{K_{C+}} \quad (5)$$

If not for the incremented nonce, it would be too easy for other nodes to forge this last message, resulting in a denial of service attack. If the former server continues to receive search requests, they may be ignored, or an error message may be anonymously sent to the requesting peer. If an error message is used, a nonce exchange, similar to that described above, could be used to limit the effectiveness of denial-of-service based on forged error messages.

3.2.3 APFS: Anonymous Peers

Peer Step 1. Peers start by anonymously querying the coordinator for available servers in the current session S. For a peer, P:

$$A(P) \rightarrow C : S, \textit{Server-List-Request} \quad (6)$$

Peer Step 2. The coordinator responds with the current list of servers. For each server, a server ID and the current tail node is provided.

$$C \rightarrow A(P) : (ID_1, T_1), \ldots, (ID_N, T_N) \quad (7)$$

Peer Step 3. The peer sends a message to some or all known servers announcing the user's shared content. This includes the peer's anonymous ID, ID_P, its current tail node, T_p, and a list of all the files the user is sharing, $F_1, \ldots, F_N$.

$$A(P) \rightarrow A(v) : ID_P, T_P, F_1, \ldots, F_N \quad (8)$$

Users send message (8) whenever they first learn of a new server. The user periodically sends updates of its file list, which also servers to inform the server that the user is still active. However, when sending message (8) to a previously known server, the file list in the message can be omitted if no changes have occurred.

Peer Step 4. The anonymous peer can begin anonymously sending queries. The user can spread queries across different servers to avoid poorly connected or heavily loaded servers. Multiple queries also might aid in finding files that have been reported to only a subset of the possible servers.

Peer Step 5. When a client is ready to leave the session it should simply send the server a message stating that it is no longer sharing any files. At the next join period the user does not re-announce itself as a member.

3.2.4 APFS Multicast

APFS Multicast has a few differences from the unicast version. The protocol has some added complexity, but has the key advantage of having no central coordinator.

Here, we describe the differences between the two protocols.

APFS Multicast begins with an initial participant who starts the session with the selection of a multicast address, M. Currently, IP multicast doesn't allow reservations of addresses; however, collision with other applications is easily detected, although cumbersome. The chosen address is posted in a well-advertised form; e.g., web pages, mailing list, or IRC channel. The advertisement includes a *base join time* and a *frequency of future join periods* given in GMT. As we see later, only loose clock synchronization is required. (The advertisement is optionally signed by the initial participant for integrity.)

Participants join the session by subscribing to the multicast group and then overtly sending messages (1) periodically to the multicast address. Clients learn of other clients by waiting long enough to hear a sufficient number of other participants.

To become a server, peers send a message (3) to the multicast group M instead of the coordinator; following our assumption of Onion Routing as the unicast protocol,

this means that the tail node multicasts the message to M.

Servers handle queries and return query results unicast.

Servers quit answering queries by sending message (5) to the multicast group instead of the coordinator. If the former server continues to receive search requests, a message (5) may be resent, multicast or unicast, until search requests cease.

Peer searches do not have any operational differences with the unicast version of APFS. Peers learn of new servers from the advertisements on the multicast group.

3.2.5 Overlapping Periodic Reset

To perform periodic resets efficiently and without a centralized server, we propose that sessions overlap construction and deployment. Each session i has three points of interest called Build_i, Deploy_i, and Shutdown_i. These points are when route building begins, when the session routes begin use, and when the old routes must cease use, respectively. The overlap between sessions is such that

- $\text{Build}_i = \text{Deploy}_{i-1}$
- $\text{Deploy}_i = \text{Shutdown}_{i-1} = \text{Build}_{i+1}$
- $\text{Shutdown}_i = \text{Deploy}_{i+1}$

For a give session i, the creation and implementation of the session $i+1$ follows the following steps (see Figure 2).

Reset Step 1. During the time period [Build_i, Deploy_i) all nodes must announce themselves via message (1) shown previously.

Reset Step 2. During [Deploy_i, Shutdown_i) = [Build_{i+1}, Deploy_{i+1}) all routes must be determined. Also, servers need to multicast message (3) to ensure all the users know about the servers' new tail node. (Peers may request retransmissions after timeout.)

Reset Step 3. At time $\text{Shutdown}_i = \text{Deploy}_{i+1}$ all clients and server must stop using routes from session i and begin using the new session $i+1$ routes. Clients can now update the servers by sending message (8). The client should again omit the file list for servers it sent up to date file lists to in session i. At this point normal functioning may resume using session $i+1$, and preparations for session $i+2$ should be underway.

4 Performance Analysis

The primary cost of providing anonymity to a set of clients is the additional latency of data delivery to responders and the additional bandwidth consumed forwarding

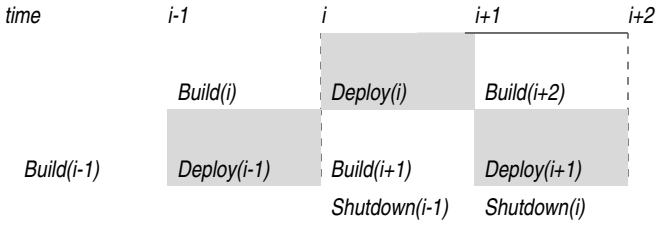

Figure 2: Overlap between session phases.

traffic for other members. There was also additional latency from querying. We chose to study the costs of data delivery rather than the expenses involved in maintaining a system for performing queries and the costs of those queries. The rationale for this decision was that data delivery was likely to be the more expensive operation and that there was no point in studying anonymous query mechanisms if data delivery was not feasible. As our results show, such delivery is possible, and our future work will include a study of the costs of querying.

We simulated our protocol over a hierarchical, Internet-like topology to determine how our protocol compares with equivalent overt protocols, such as Napster. We evaluated the TCP performance of APFS based on a generic path-based protocol similar to Crowds or Onion Routing. We also calculated the amount of work required to support other peers in the session.

Previous performance studies of anonymous protocols have not considered TCP performance. Reiter and Rubin have calculated the amount of work required of participants in the Crowds protocol [12]. Levine and Shields compared by simulation the round trip time and link utilization of Crowds and Onion Routing, Hordes, and overt connections [13] and computed the work required for Hordes participants. (Syverson et al report performance results of Onion Routing only in terms processing overhead [14].)

4.1 Methodology

We simulated APFS against an overt peer-to-peer file transfer using the ns2 [1] network simulator. Our simulations ran over internet topologies generated by the Georgia Tech Internetwork Topology Modeler (GT-ITM) [16, 3, 17]. Our performance analysis focuses on the overhead introduced directly by APFS; for this reason, we evaluated the performance of transfers between peers.

We simulated a variety of sizes of client groups: 10, 50, 100, 200, and 285. Each size client group was simulated over each of three different GT-ITM topologies of 1000 nodes each. Our client base was limited to 285 because

we choose clients only from the edges of the generated topologies. Each edge generated by GT-ITM had a random latency, from 10ms to 2300ms, randomly selected in a fashion that approximates an internet network. All links had a simulated bandwidth of 1.5 Mbs. Routing between clients was determined by ns2's internal shortest path unicast routing protocol.

At startup, we preselected a path from one peer to another peer. To simulate a generic overt peer-to-peer application, the path was simply the initiating peer and one other peer chosen uniformly at random. To simulate APFS, the path between peers consisted of 4 or 8 other peers, simulating a mutually anonymous connections of two paths of length two and two paths of length four respectively. Peers transfered a 2Mb file to the remote peer using TCP Reno between clients on the path.

We ran a total of 900 simulations. For each of the five group sizes and three path lengths, we generated 30 runs of the simulation: 10 simulations with 10 different randomly chosen paths for each peer for each of the three topologies. Moreover, we simulated two separate loss rate scenarios on the links: geometrically-distributed loss processes of 1% and 5% each. These loss rates were aggressive as they appeared on each link in the network, and as such they accumulated end-to-end. Because the anonymous protocols traversed more links, they were at a disadvantage in the simulation environment.

For all graphs in this section, error bars report 95% confidence intervals.

4.2 Transfer Latency

First we present our analysis of the difference in latency between overt systems and APFS. Of all latencies in a distributed file sharing system, the most import is the transfer times for files, as this has a direct bearing on the interactive usability of the system. Figure 3 shows the average transfer time for 2MB files initiated simultaneously at every peer for various group sizes and path lengths with 1% loss in the links. Note that a path length of zero in this case is the equivalent to overt protocols like Napster. There transfer time increases by 65% by using APFS with a path length of 2. However we also note that to increase the path length to 4, we only see an additional increase of 13%. From this we gather that after the initial increase in transfer time, we can increase the level of anonymity relatively cheaply. This is due to the pipelining of TCP connections that is used to forward data down the path between clients. Figure 4 shows the same experiments with 5% constant loss on the links. The extreme loss greatly delays the TCP transfer time, affecting the

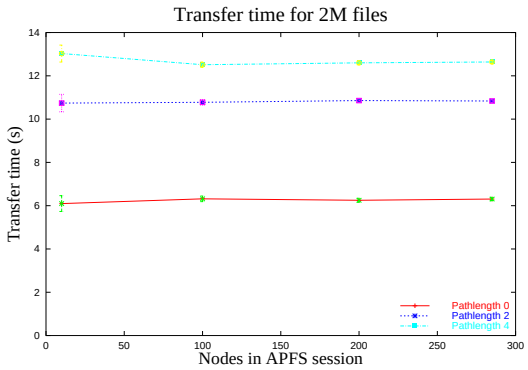

Figure 3: Transfer latency for 1% loss on all links.

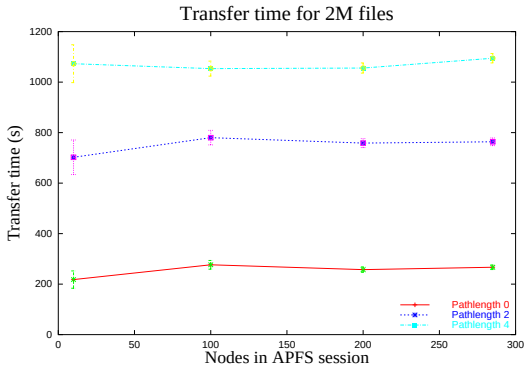

Figure 4: Transfer latency for 5% loss on all links.

anonymous protocols worse than overt protocols due to the additional network links traversed.

4.3 Resource Consumption

When a user decides to join an APFS session, the client also offers their own network resources to be used for proxying data. We analyzed what factors effect the amount of resources consumed by proxying data versus traffic generated by that users own file requests. Figure 5 shows the amount of data in bytes carried for other clients during the session where every peer transfered exactly 2MB. In comparison to the peer's own file transfer, the amount of traffic carried for others is high. Fortunately, the amount of traffic forwarded for other clients is directly related to the path length chosen by other clients and not the number of clients in the session; this is predicted by Reiter and Rubin's analysis [12]. While clients cannot limit the path length chosen by other nodes, they can deny requests from nodes to join a path so as to control the amount of work they perform. Fortunately clients in Onion Routing locally pick routes and can avoid overtaxed nodes that haven't denied their route requests.

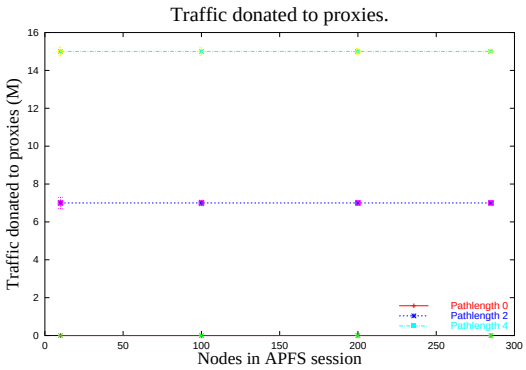

Figure 5: Percentage of data carried from other connections.

4.4 Scalability

The most prominent quality of the performance results presented in Figures 3–5 is the excellent scalability of the anonymous protocols in terms of resulting latency and carried traffic. As the number of clients (who are all each peers) grows in the simulation, the APFS protocol has perfect scalability with respect to transfer times and resource consumption. This is due to the fact that as more users join the session, not only do they bring more data to require proxying, but also they themselves add to the proxy count. The end effect of this is a more distributed path structure, but no change in transfer time or resource consumption.

As noted in Section 2, Gnutella is not an anonymous protocol, though it can be mistaken for one. Although we did not simulate the protocol, it is a safe conjecture to say that Gnutella's scaling properties are extremely poor as more clients join the session. This is due to the expensive querying mechanism being used. APFS offers true anonymity but, like Gnutella and unlike Napster, has no central serving point.

5 Conclusions

We have contributed new techniques for providing responder anonymity using existing protocols for initiator anonymity. We solve two important problems in providing responder anonymity: the problem of allowing initiators to locate and contact an anonymous responder, and the problem of providing anonymity for a long-lived server in the face of anonymous degradation. APFS can use either an untrusted central coordinator or multicast routing to eliminate the problems in providing responder anonymity. APFS also uses features of peer-to-peer networking to solve the problem of degradation of anonymity over time in servers.

Our simulations show that the latencies incurred in providing reliable anonymous peer-to-peer communication can increase the transfer times for data, and that APFS suffers more from high network loss rates than non-anonymous protocols. The trade-off in gaining anonymity is an increase in bandwidth consumption that does not scale with the size of the session but instead with the length of the anonymous paths constructed.

References

[1] Network simulator version 2. http://www.isi.edu/nsnam/ns.

[2] Opennap: Open source napster server. http://opennap.sourceforge.net.

[3] GT-ITM: Georgia Tech Internetwork Topology Models. http://www.cc.gatech.edu/fac/Ellen.Zegura/ graphs.html, 1996.

[4] K. Almeroth. The Evolution of Multicast: From the MBone to Inter-Domain Multicast to Internet2 Deployment. *IEEE Network*, January 2000.

[5] D. Chaum. The Dining Cryptographers Problem: Unconditional Sender and Receipient Untraceability. *Journal of Cryptography*, (1):65–75, 1988.

[6] C. Diot, B.N. Levine, B. Lyles, H. Kassan, and D. Balsiefien. Deployment Issues for the IP Multicast Service and Architecture. *IEEE Network*, January/February 2000.

[7] I. Goldberg and D. Wagner. Taz servers and the rewebber network: Enabling anonymous publishing on the world wide web. *First Monday*, 1998.

[8] H. Holbrook and B. Cain. Source-specific multicast for ip. IETF Internet-Draft, March 2001. ¡draft-holbrook-ssm-arch-*ast*.txt¿.

[9] H. Holbrook and D. Cheriton. Explicitly requested source-specific multicast: Express support for large-scale single-source applications. In *ACM SIGCOMM '99*, September 1999.

[10] A. Pfitzmann and M. Waidner. Networks without user observability — design options. In *Eurocrypt '85, LNCS 219*, pages 245–253, 1986.

[11] M. Reed, P. Syverson, and D. Goldschlag. Proxies for anonymous routing. In *12th Annual Computer Security Applications Conference*, pages 95–104. IEEE, December 1995.

[12] M. K. Reiter and A. D. Rubin. Crowds: Anonymity for Web Transactions. *ACM Transactions on Information and System Security*, 1(1):66–92, November 1998.

[13] C. Shields and B.N. Levine. A Protocol for Anonymous Communication Over the Internet. In *Proc. 7th ACM Conference on Computer and Communication Security (ACM CCS 2000)*, November 2000.

[14] Paul F. Syverson, D. M. Goldschlag, and M. G. Reed. Anonymous connections and onion routing. In *18th Annual Symposium on Security and Privacy*, pages 44–54. IEEE CS Press, May 1997.

[15] M. Wright, M. Adler, B. Levine, and C. Shields. An analysis of the degradation of anonymous protocols. Technical Report, April 2001. University of Massachusetts, Amherst.

[16] E. Zegura, K. Calvert, and S. Bhattacharjee. How to Model an Internetwork. In *Proceedings of IEEE Infocom '96*, San Francisco, CA, 1996.

[17] E. Zegura, K. Calvert, and M. Donahoo. A Quantitative Comparison of Graph-based Models for Internet Topology. *IEEE/ACM Transactions on Networking*, 5(6):770–783, December 1997.