Forensic Identification of Anonymous Sources in OneSwarm

George Bissias, Brian Neil Levine, Marc Liberatore, and Swagatika Prusty

Abstract—OneSwarm is a p2p system for anonymous file sharing. We quantify the system's vulnerability to three attacks that identify the sources of files. First, we detail and prove that a timing attack allows a single attacker to investigate all its neighbors for possession of specific files. We prove the attack is possible due to OneSwarm's design and is unthwarted by changes made to OneSwarm since we released our attack. Second, we show that OneSwarm is much more vulnerable to a collusion attack than previously reported, and we quantify the attack's success given a file's popularity, a factor not evaluated earlier. Third, we present a novel application of a known TCP-based attack. It allows a single attacker to identify whether a neighbor is the source of data or a proxy for it. Each of these attacks can be repeated as attackers quit and rejoin the network. We present these attacks in the context of forensics and the investigation of child pornography. We show that our attacks meet the higher standards required of law enforcement for criminal investigations.

Index Terms—privacy, digital forensics, legal aspects of security, child sexual exploitation, p2p networks

1 Introduction

OneSwarm [6] is p2p system for anonymous file sharing designed to resist traffic analysis attacks, while providing file transfer performance comparable to BitTorrent. OneSwarm thwarts attribution of both queries for content and the sources of downloadable files.

Our law enforcement partners were able to download images of child sexual exploitation from OneSwarm by sending out search queries with keywords typical of child pornography (CP). Identifying sources of CP is of great interest to law enforcement for several reasons. First, such investigations are the primary proactive method of discovering persons physically abusing children. Victims come forward to authorities uncommonly, and it is an impossible task for infants and toddlers. Past studies have found that 16% of persons sharing and possessing CP are *contact offenders* [24]. Second, possession and distribution of CP is illegal and worthy of investigative resources to protect the privacy of the victims. Finally, CP is used to "groom" new victims by normalizing the acts perpetrated against them.

Given OneSwarm's protections against traffic analysis, we ask, Can forensically valid evidence be acquired from peers that share CP on OneSwarm? We show that it can, while adhering to the higher standards required of criminal investigations. Civil investigations of copyright infringement, for example, have a much lower standard, and those who seek to invade the privacy of OneSwarm users might be satisfied with less. We present three attacks on OneSwarm that identify sources of files of interest. In each case, we attempt to generalize our assumptions beyond the specifics OneSwarm to ensure the broadest application of the results.

Peers find content in OneSwarm by sending to all neighbors search messages with query terms. If a peer receiving the query has the requested file, it replies after a purposeful delay. Otherwise, with a probability p it forwards the query to its neighbors, again after a delay. Replies are routed back to the querier along the reverse of the search message's path. Search messages do not have a hop-count field. Instead, *search cancel* messages avoid unnecessary flooding after sufficient replies are received. Peers can name neighbors as *trusted friends*. Queries from trusted friends are answered without delays.

Contributions. We quantify the efficacy of three attacks on OneSwarm, which can be carried out independently.

Our analysis of a first attack shows that OneSwarm's delays do not effectively hide the identity of content sources. An attacker can connect to a peer and verify whether the peer is a source (or a *trusted friend* of the source) of files of interest by merely comparing the response delay against a pre-measured upper bound. When attackers comprise 6% of OneSwarm peers, we expect over 90% of remaining peers will be attached to at least one attacker and therefore vulnerable. With a smaller fraction of attackers, the entire network can still be investigated: the attackers must quit and rejoin the network with a new identity, which is easy to do. Trusted relationships are purported to defeat this timing attack. In contrast, we show that the attack succeeds in that it finds peers that are trusted friends of CP sources. In that case, conspiracy statutes are generally additionally applicable; trusting peers is a significant criminal liability. In our analysis, we derive corrected parameter settings that defeat the attack but degrade performance.

As discussed by Isdal et al. [6], OneSwarm is vulnerable to a *collusion attack*. A set of k attackers that are neighbors

• *Authors are listed alphabetically. All are with the College of Information and Computer Sciences at the University of Massachusetts Amherst. E-mail: (gbiss,brian,liberato,prusty)@cs.umass.edu*

of the same peer can infer if it is a source of a query response by comparing the probabilistically forwarded responses. We show that the architecture is much more vulnerable than previously reported, and that it also depends on content popularity, a factor not considered earlier. For example, when attackers comprise just 10% of available peers, 50% of the network can be investigated with 95% precision to find sources of content that is available from 1 in 1,000 peers. As content popularity increases, the attack's success also increases. In comparison, should attackers comprise 25% of an Onion Routing network [14], about 9% of the circuits are vulnerable to an analogous attack.

Finally, we show that OneSwarm is also vulnerable to a novel application of a known TCP-based attack [18]. The attack attempts to determine if a neighbor is the source of a file transfer or merely proxying data from the source. By optimistically ACKing data, the send-rate is increased as much as possible. If the neighboring peer is the source, the send rate will increase with the ACKs; if not, the transfer will be rate-limited by the true source. This attack requires only one attacker and can successfully distinguish sources and proxies with TPR=89% and FPR=4%. Only users that turn off the default rate-limit setting are susceptible to this attack. Onion Routing, on the other hand, is not at all vulnerable to this attack.

While our attacks are based on OneSwarm's design, our results are applicable to broad design principles for anonymous file sharing systems. For example, our TCP-based attack works on anonymous systems that do not use onion-based encrypted layers when streaming data, and several such open-source systems exist, including MUTE [16] and RShare [17]. We also develop a different attacker model, one based on a conservative set of legal restrictions.

This paper extends a preliminary version [13] of the above results in several ways. We revised the timing attack, so that it requires only one attacker instead of two, and we have updated our proof accordingly. And we show that the version of OneSwarm released after our initial publication is still vulnerable to the timing attack. We have newly included measurements of the attack on the real network, demonstrating its success. Based on these network traces, we also evaluate how variation in network roundtrip delays affects the timing attack, achieving a 0.01% false positive rate. We have revised our evaluation of the collusion attack, showing that it is significantly more successful than we and the original OneSwarm publication each reported. Finally, our evaluation of the TCP-based attack's TPR and FPR is new.

2 Overview of OneSwarm

Below, we include details of only the OneSwarm mechanisms that are relevant to our analysis. Our results are based on examination of the source code available from http://www.cs.washington.edu/homes/isdal/OneSwarm-20110115.tar.bz2, which is version 0.7, and we also tested our attacks on version 0.7.1. We note any relevant differences between the technical paper and source code.

OneSwarm is based on a dense topology of peers that discover each other through a community server. Neighboring peers can be *trusted* or *untrusted*. Trust is assigned by the user, and trusted peers see none of the delays or other mechanisms that OneSwarm introduces, as if they were standard BitTorrent peers. The content shared by trusted friends is displayed in the OneSwarm GUI. CP files are typically explicitly named (for example, with the age of the child as, "1yo" or "2yo", and descriptions of sexual or sadistic acts), which is an important aspect of our legal analysis.

OneSwarm is a previously popular p2p system, with strong communities in North American (http://oneswarm.cs.washington.edu), French (https://forum.oneswarm-fr.net), and Russian (http://oneswarm.ru) communities each with many thousands of users. More recently, the community collapsed — we are unsure if our initial publication [13] was a factor — but our analysis remains useful for future designers of privacy preserving protocols.

2.1 OneSwarm's Design

Topology Construction. Each peer has 26 neighbors. They can be added from out-of-band methods, including email or social networking sites, as trusted or untrusted friends. Peers assigned by the community server are untrusted. The simplest method of investigation is to be randomly assigned to peers by the community server, and that is the case we focus on here. When a target does have a trusted friend, all privacy controls are turned off, and therefore becoming an undercover trusted friend is an appealing method of investigation; we don't evaluate such investigations in this paper. But we do quantify the affect of trusted friends on our attacks, and we summarize the legal implications, which favor law enforcement, in Section 4.2.2.

Neighboring peers communicate via SSL over TCP. Key exchange is based on an underlying DHT. The public/private keypair used by each peer does not change and is stored on the local computer. The public keys of neighbors are also stored on the user's local computer. The user's keys and neighbors' keys are never deleted (until the application is uninstalled) and are useful corroborating evidence.

Searching for Content. OneSwarm is strongly linked to BitTorrent, and peers can search for content by flooding a query containing a text string or by a unique BitTorrent *infohash*, which is a standard method of uniquely identifying a torrent. When content is found, peers indicate they have a path to the content, without disclosing whether they are the source of the content or just a proxy to it. Pieces of the torrent are then swarmed from all remote peers that provide a path.

When a OneSwarm peer possesses queried content, it will return a *search reply* message after some delay but not forward the query any further. The delay is selected uniformly at random from 150–300ms. The choice is consistent (random but deterministic) for the matching content (by info hash). Two neighbors that query a source for the same file will see the same delays. A peer that queries for two different files from the source will see different delays, but when a query is repeated it will see a consistent delay. When intermediaries receive a search reply, they forward the message along the reverse path back to the original querier.

If the peer does not possess the queried content, it forwards the query to each of its neighbors with probability p; the choice to forward a specific query to a specific neighbor is random but deterministic. Forwarding of queries is delayed again by 150–300ms, a value chosen at random but again consistent for the specific query and the neighboring peer.

Content Transfer. Once the querier receives a reply, the content is requested and relayed through the path of OneSwarm peers using the BitTorrent protocol. There are no direct downloads between peers unless they are neighbors, but peers cannot naively identify these direct connections.

OneSwarm messages have no time-to-live (TTL) fields, as they would allow attackers to determine if a neighbor is a source of a file by falsely setting the TTL of an outgoing query to 1. Without TTLs, queries might cause congestion collapse from unlimited traffic, and so OneSwarm uses another mechanism. As search replies are returned along the reverse path to the querier, *search cancel* messages are sent to any neighbor that received the original query. These cancel messages are sent without delay and are designed to catch up to and stop the propagating query. There is no other mechanism for stopping searches, including those that fail entirely.

Like BitTorrent, OneSwarm allows a form of parallel downloading (called *swarming*) that Onion Routing implementations, like Tor, do not support well. Our analysis of OneSwarm's privacy is so that users can evaluate if the performance benefits are worth the decline in security in comparison. The reason Onion Routing does not support swarming well is that a separate multi-proxy tunnel is needed to each peer offering part of a torrent .

Other details of OneSwarm's operation do not introduce vulnerabilities that we've discovered, and we do not describe them further here.

2.2 OneSwarm Implementation

The OneSwarm implementation has some important differences from the OneSwarm paper. The source code assigns peers between 22 and 39 neighbors, and as peers quit, the community server can assign more peers to clients. In the source code, query replies are returned with a longer of delay of 170–340ms. In the case of infohash searches, the delay is chosen on the basis of the infohash; for text searches, the delay is selected based on the content that matches the query. Query reply messages are forwarded by intermediaries without delay in the source code, though the paper specifies that all OneSwarm protocol messages are delayed.

In the paper, $p = 0.5$ is a suggested value, but it is stated that "privacy-conscious users are free to decrease their forwarding probability". However, the software follows a different design. p is set to a much higher value of 0.95 and there is not yet a user-visible method to change the value of p; users must edit and recompile the source. As we show, this setting greatly reduces the privacy of the system.

We provided the details of our attacks and results to the OneSwarm developers in May 2011. In August 2011, they reported that the following changes to OneSwarm were made in response: the default value of p is set to 0.5, an unintended forwarding latency we found (described in Section 4.4) has been decreased from about 100ms to less than 10ms, and the discrepancy between text search and hash search delays have been fixed. Additionally, all PlanetLab, UMass, and Tor IP addresses were blocked (see http://forum.oneswarm.org/topic/1927).

3 Problem Statement & Model

In this section, we provide a problem statement, attacker model, and our assumptions, as well as related work.

3.1 Problem Statement

OneSwarm was designed to thwart third parties from logging the public activities peers on its p2p network. For years, p2p networks have been measured by academics [1], [15] and by companies such as DtecNet, Peer Media Technologies, and Media Defender that assist copyright holders in filing civil copyright infringement lawsuits.

For more than a decade [22], p2p networks have been an enormous venue for the distribution of child sexual abuse imagery, according to the US Dept. of Justice [21], past work by ourselves [9], [10] (showing 2.4 million unique peers in one year on Gnutella and eMule), and others [3], [8]. Law enforcement have verified to us that there is child pornography (CP) shared on OneSwarm.

Investigator Goals. The investigator is essentially an attacker, attempting to violate OneSwarm's privacy promises, though more limited in ability than is typically assumed [23]. Their goal is to *identify* a subset of all OneSwarm peers that are each sharing (or conspiring to share) child pornography, and it represents a small fraction of files shared on the network. The overriding goal of the attacker is to gather evidence sufficient for a search warrant; i.e., *probable cause*.

Probable cause is a lower standard than the *beyond a reasonable doubt* standard needed for conviction. There is no quantitative standard for probable cause, and courts have defined it only qualitatively. Accordingly, for the purposes of our study, we say a peer has been *identified* if the investigator's statistical confidence is above a sufficient level. While Isdal et al. analyzes attacks requiring 95% precision, we believe lower values are worth evaluating. We evaluate different scenarios throughout this paper.

Previous work [12] has investigated related issues in the civil context where the smallest success would likely be sufficient for subpoena — precision of 95% confidence is extraordinarily high in that context. OneSwarm users committing copyright infringement are at much greater risk for successful investigations than CP traffickers.

In this paper, we say that evidence is *forensically valid* if gathered using techniques that are based on testable hypotheses, have a known error rate, are based on accepted scientific methods, and are peer reviewed.

Model and Assumptions. The general approach adopted by law enforcement for investigating CP trafficking is based on a series of legal restrictions depending on the country of jurisdiction. We assume our attacker is an investigator following common restrictions. Specifically, we assume that the investigation can gather only information available publicly, which includes all traffic sent to other peers (since anyone in the public can be a peer). We do not allow attackers to seize or compromise peers through privilege escalation.

4 Timing Attacks

This section describes a timing attack that is fundamental to any peer-to-peer network with the following properties, and OneSwarm in particular:

1. File search queries require message flooding.
2. Query propagation is stopped by a flooded cancelation message.

Property 1 is imposed by most networks when the content hosted by any given peer is not known by other peers a priori. The primary downside to flooding is that the number of contacted peers can grow exponentially with every hop. Property 2 is an alternative to adding a TTL — which can itself expose a privacy vulnerability — to query messages. In order to ensure that cancelation messages can overtake the original query message, it's necessary to impose a *query forwarding delay*.

As we will see, together the query flooding and forwarding delay have a profound impact on communication cost in the network. Specifically, from these properties alone it is possible to derive lower bounds on query response round trip time (RTT). We also use our theoretical bounds to uncover a weakness in the OneSwarm privacy model. This theoretical weakness is corroborated by experimental results.

4.1 Types of Timing Attacks

The purpose of a timing attack is for a single attacking peer A to establish that a file of interest f is hosted by an immediate neighbor T as opposed to another peer S multiple hops away (see Fig. 1). The attack itself is noninvasive in that the attacker uses only ICMP requests and application-level query requests.

Naive Timing Attack. Here attacker A compares the application-level response time to the network-based roundtrip time; if they are similar, then the attacker concludes that T is the source. A's reasoning is that a query that takes no longer than the network-level round trip time to T, could not have been forwarded. And since T forwards all requests that it cannot fulfill itself, T must be a source for file f.

To defeat the attack, T introduces a randomly chosen (but consistent for a given file) response delay. The delay should be long enough that the application and network level round trip times appear to vary significantly. Thus the attacker cannot easily determine if the difference is due to network jitter or if the query is actually being forwarded to other peers. OneSwarm employs this defense by introducing a response delay between of 150–300ms before answering queries, as described in Section 2.

Simple Timing Attack. While it's relatively easy to defeat the naive timing attack described above, we define a slightly more sophisticated attack that can be carried out in the event that the maximum response delay $r_{\max}$ is not chosen carefully. The attack refers of the following quantities. Each is in units of milliseconds.

- r: query response delay between $r_{\min}$ and $r_{\max}$
- q: query forwarding delay between $q_{\min}$ and $q_{\max}$
- l: one-way network-layer latency between two peers
- δ application-layer roundtrip query response latency

The simple timing attack consists of the observation that if T did in fact forward A's query on to S, then the total query response latency, δ, must be larger than the sum of network- and application-level delays associated with that path. The observation below follows directly from this reasoning, and from Fig. 1 (right).

***OBSERVATION** 1:* T is the source of file f if its query response time to A is such that

$$\delta < q_{\min} + r_{\min} + 4l \text{ ms.} \tag{1}$$

Algorithm 4.1 uses this equation to define a procedure for A to identify files hosted by T.

Algorithm 4.1: SimpleTimingAttack(T)

```
1) $A$ estimates $l$.
2) $A$ queries $T$ for $f$ and notes response time $\delta$.
3) If $\delta < q_{\min} + r_{\min} + 4l$, then $T$ is the source.
```

4.2 Simple Timing Attack Defense

Observation 1 establishes a test that A can use to decide if T hosts file f. Naturally, T would like to disguise the fact that it hosts f by delaying its query response in order to confuse A. But T is constrained by $r_{\max}$, which

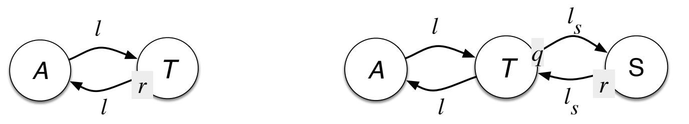


Fig. 1. The attacker attempts to distinguish two scenarios. **(Left)** Case A where peer T is the source of queried file. **(Right)** Case B, where peer S, one hop from T, is the source.

is the maximum amount of time it can wait to respond to a query request. So $r_{\max}$ should be chosen so that it's possible that the query request was forwarded to S. Specifically, the value of r chosen by T must exceed the minimum application-level query response time from T to S, which is equivalent to the bound in Equation 1 less the network delay from A to T. Based on this reasoning and Fig. 1 (left), we have the following constraint.

***OBSERVATION** 2:* In order for T to disguise the fact that it hosts f it must be the case that,

$$r \geq q_{\min} + r_{\min} + 2l \text{ ms.} \tag{2}$$

4.2.1 Attacking OneSwarm

Based on its technical description and Observation 2, the OneSwarm network is susceptible to the Simple Timing Attack. Recall from Section 2, that query response delay r and query forwarding delay q are bounded as follows.

- 150 ms $\leq q \leq$ 300 ms
- 150 ms $\leq r \leq$ 300 ms

This means that

$$r \leq 300 \text{ ms} < 300 \text{ ms} + 2l = q_{\min} + r_{\min} + 2l,$$

which violates the constraint in Equation 2.

4.2.2 Trusted Neighbors

Trusted relationships are an important aspect of OneSwarm's design. Messages sent to a "trusted friend" are direct and without added delays. The Simple Timing Attack will not distinguish between three cases: (a) the target is the source of a CP file; (b) the target is a direct proxy for a trusted friend that it knows is sourcing CP files; (c) the target is a direct proxy for a trusted friend that it does not know is sourcing CP files. In all cases, the machine contains evidence of a crime and a warrant is appropriate[1]. The Simple Timing Attack far exceeds the probable cause standard required to grant a warrant whose execution will clarify which case applies.

Case (b) occurs when the user of the target machine is a trusted friend of the possessor and views the filenames he shares. Filenames are shared by the possessor to its trusted peers by default and shown in each trusted peer's GUI. CP filenames are descriptive, typically containing the age of the child, terms describing sexual assault and sadistic acts, and slang for CP. Upon timely execution

1. See https://www.law.cornell.edu/rules/frcrmp/rule_41.

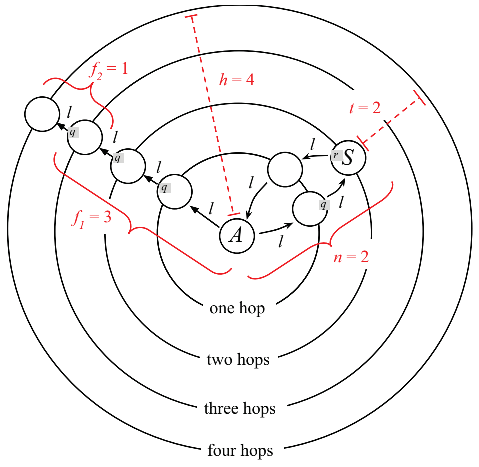


Fig. 2. Query Request Propagation.

of a warrant, an investigator's forensic examination of the target machine may reveal if CP files are shown in the GUI or other evidence of intent [5]. And though not a source of the content, in this case, a target proxying for a trusted friend is part of a conspiracy to knowingly distribute CP. Furthermore, by participating in the trusted relationship with the source, the target gains the non-pecuniary benefits of bandwidth and better performance, which can incur a greater punishment in some jurisdictions. In short, trusting a neighbor can raise significant criminal liability.

Case (c) occurs when a peer turns off the default sharing of filenames to its trusted friends (which must be done on a per-torrent basis). This case is still valuable to investigators: timely execution of the search warrant and forensic examination of the target machine will disclose a list of trusted friends, one of which can be revealed by a Naive Timing Attack as the source of CP files on the network.

4.3 Best-Case Query Response Latency

In any peer-to-peer network with properties listed at the start of this section, it's necessary to introduce some query forwarding delay q to ensure that cancelation messages can catch up to query requests once a response has been received. Moreover, the delay q should be chosen so as to minimize the total number of hops the query propagates, h. But according to Observation 2, a requirement on the minimum value for q imposes a larger maximum value for r. So it's natural to ask how the overall query response latency δ is affected by the addition of this delay. In this subsection, we use the bounds developed in earlier sections to bound the minimum response latency $\delta_{\min}$ from below that thwarts the simple timing attack.

Suppose that a query is initiated from peer A and consider its movement through the network as depicted in Fig. 2. The query travels n hops before reaching the nearest source S. Once S receives the request, it waits for a maximum of $r_{\max}$ ms and then sends a response back to A along the reverse path. Meanwhile, because the network floods query requests, the same query has also been propagated recursively to all neighboring peers beginning with the neighbors of A. We call this movement through the network at large the query propagation *frontier*.

DEFINITION 1: The *diameter* h of the query propagation frontier is the total number of hops from A that the query propagates through the network before being overtaken by a cancelation message.

Since the query carries no TTL, the frontier continues to expand until it's overtaken by a cancelation message. Naturally a cancelation must travel faster than the query in order to achieve this. In fact, the cancelation should be fast enough that it catches the expanding frontier just a constant number of *additional* hops t after the query reaches the nearest source S. This ensures that the frontier does not grow exponentially if not necessary.

DEFINITION 2: The *cancelation penalty*, given by an integer t, is the number of additional hops the frontier continues to expand after the nearest source S receives the query request.

Taken together, the source distance n and cancelation penalty t are related to the diameter by $h = n + t$.

4.3.1 Characterizing Response Latency

Our goal is to use the quantities defined in the previous section to prove the following bound on the minimum query response latency $\delta_{\min}$.

THEOREM 1: For a query file n hops away, to ensure a cancelation penalty no greater than t, we must have

$$\delta_{\min} \geq \frac{2l(n+1)(n+t)}{t}. \tag{3}$$

PROOF: We return to the example of a query traveling from the requester A to the nearest source S. In n hops the query will reach S. By the time A receives a response from S, the frontier has simultaneously saturated all the peers in the network up to f_1 hops away.

At this point a cancelation request is flooded from A recursively to all neighbors. The cancelation request has no delay and therefore moves faster than the query request; nevertheless, the frontier continues to propagate f_2 *additional* hops before being overtaken. Note that $h = f_1 + f_2$.

We begin by establishing some fundamental relationships between query latency, forwarding delay, and response delay. Note that $\delta_{\min}$ can be expressed from two different perspectives. First from the perspective of the querier A who receives the response we have

$$\delta_{\min} = r + (n-1)q_{\min} + 2ln. \tag{4}$$

Second, from the perspective of the expanding frontier

$$\delta_{\min} = f_1(q_{\min} + l). \tag{5}$$

The frontier's diameter can also be expressed from these two different perspectives, which yields the following relationship.

$$f_1 + f_2 = n + t. \tag{6}$$

We can also derive a relationship between r and $q_{\min}$ from Equation 2 by recognizing that in order to make $\delta_{\min}$ as small as possible, we will also want r to be as small as possible. Therefore,

$$r = q_{\min} + 2l. \tag{7}$$

Finally, the cancelation penalty can be related to frontier diameter as follows.

$$(n+t)l = f_2(q_{\min} + l) \tag{8}$$

In words, this expression dictates that a cancelation message can propagate $n + t$ hops in the same time it takes a query request to propagate f_2 hops.

The result in Equation 3 can be derived algebraically from Equations 4–8. □

4.3.2 Query Response Latency in OneSwarm

Fully decentralized peer-to-peer networks like OneSwarm rely on a high degree of connectivity between individual peers in order to ensure robust end-to-end communication. This feature inadvertently imposes a practical constraint on the size the cancelation penalty t. Specifically, if each peer has many neighbors, then a query request message can quickly saturate the entire network even if a response is ultimately received from just one hop away.

The technical description of OneSwarm specifies that each peer has 26 untrusted neighbors. However, the source code allocates between 22 and 39 untrusted neighbors. In any case, the query propagation frontier will encompass at least 22^h peers before cancelation. Even if the source S is just one hop away ($n = 1$), a cancelation penalty of $t > 2$ is unacceptable since this would mean that the query reaches more than 225,000 peers. So we fix $t = 2$ and use Equation 3 to show how query response latency δ scales with the source distance n under practical conditions.

Source Distance (n)	Response Latency (δ)
1	$6l$
2	$12l$
3	$20l$

In comparison, when Onion Routing consists of a chain of 3 proxies, the delay in receiving data from a Torrent search engine is $E[t] = 6l$. Therefore, for any query, OneSwarm is never faster than OR to a search engine in terms of roundtrip delay; and for most queries OneSwarm is considerably slower than OR. Moreover, the number of nodes that receive query traffic (in the thousands) is inefficient compared to contacting a single web server over an Onion Routing circuit.

4.4 Simple Timing Attack in Practice

Above, we described and proved correct a theoretical timing attack against the OneSwarm system. Here we ask, Can this simple timing attack we identified be used in practice? In short: yes. With a small and empirically quantifiable error rate, we can differentiate peers acting as a source of a file from peers that are not a source of a file, solely on the basis of measured network- and application-layer delays.

We cannot simply apply the test derived in Theorem 1 for two reasons. First, several details of the OneSwarm implementation differ from its specification. For example, the value of r is ranges from 170–340ms for keyword queries. Queries for specific infohashes follow a series of additional steps for determining the random delay. Second, there are other undocumented and sometimes non-deterministic delays present in the implementation. For example, we discovered an presumably unintentional delay of 100ms being added to nearly all messages. It was later removed after we reported it to the OneSwarm developers. The multi-threaded nature of the code and garbage collection by the Java implementation running OneSwarm can also introduce non-deterministic delays to the system.

Despite these limitations, the insight of the simple timing attack still stands: independent of network RTT, if a peer responds with less than a certain delay, it (or its trusted friend) almost certainly possesses the file described in the response. We performed measurements on running OneSwarm peers that we controlled, querying for content that we knew the peers to either possess or not. Our results indicate that the two cases are clearly separable.

Methodology. We set up two OneSwarm peers on a LAN in the configuration shown in Fig. 1(right), with A functioning as the attacker, and T as the target. Using a LAN presents the most difficult situation for the attacker. Each peer ran a modified version of OneSwarm 0.7.1 (which includes changes made as a result of our initial publication [13]) that we modified to take measurements described shortly[2]. A was connected to T as an untrusted friend, and had no other peers. T was configured to contact several of the OneSwarm community servers. T bootstrapped connections to other peers as untrusted friends, and was allowed to maintain these connections (typically around 30) during the measurements; these untrusted peers of T provided the responses to queries from A for files that were not present on T. T was configured to share 30 unique files, each with a unique name and consisting of random data. The filenames were long random strings, unlikely to exist on the OneSwarm network.

Each measurement run proceeded as follows. First, A sent five ICMP pings to T and recorded the results. Then, A sent a keyword query to T, and logged the observed application-layer roundtrip time (RTT). Finally, A sent five more ICMP pings and recorded the results. We recorded the application-layer RTT less the mean network RTT (as determined by the pings) as the OneSwarm delay γ. This delay γ corresponds to the value $\delta - RTT$ defined in the simple timing attack.

2. We will release our measurements and the source code for these modifications.

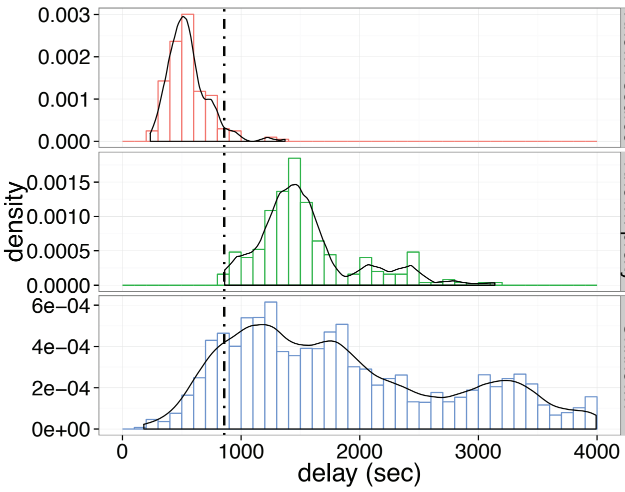


Fig. 3. The measured distribution of OneSwarm-imposed delay (γ) to nodes known to be sources and proxies, and to nodes of unknown status. The two known distributions are highly separable, implying the same holds for nodes of unknown status.

Recall our goal is to show γ in the case where T possesses a file is differentiable from γ when T does not. To this end, we performed measurement runs in two scenarios.

- In the first, A performed keyword queries T for each of the unique filenames known to be on T, with the goal of determining the distribution of γ for files possessed by a target.
- In the second scenario, A queried T for keywords drawn from popular music and film titles, with the goal of determining the distribution of γ for files not possessed (that is, proxied) by T.

As a final point of comparison, we configured A to query the community servers for untrusted peers and connected to them. We then repeated the queries for popular media, with the goal of determining the distribution of γ for a mixture of possessed and proxied files.

Results. The results of our measurements are shown in Fig. 3. The upper graph depicts the first scenario, where A queried T, and T is known to be the source of the responses to keyword queries (203 data points). The middle graph reflects the scenario where A queried T, and T did not possess a file corresponding to a keyword query (249 data points). In that case T proxied the response from another untrusted OneSwarm source. The bottom graph shows delays for A generated by querying a set of untrusted OneSwarm peers, which includes a presumed mixture of both source and proxied replies (9,206 data

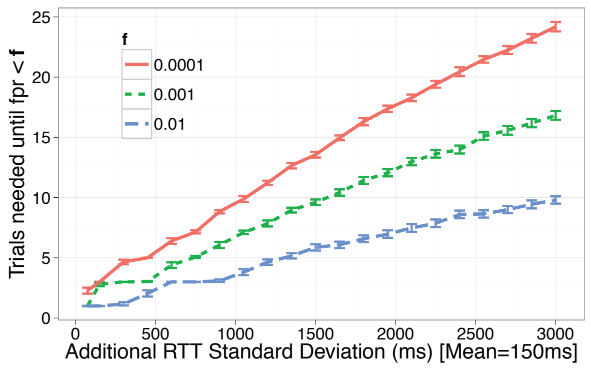


Fig. 4. Extraordinary variance in RTTs does not substantively affect the timing attack's efficacy.

points).

The results are shown as a distribution of the observed γ in each scenario. We show both estimated empirical density functions (using kernel density estimation with normal kernels), and histograms showing the observed values for γ. The vertical dot-dash line is at the minimum value observed in the known-proxy case.

Discussion. The empirical distribution of γ in the case where T is known to be the source of a file is clearly differentiable from that where it is known to be a proxy. To distinguish the two, we can use the minimum value of γ observed for the known proxy case (here, 858ms) as a cutoff: searches resulting in observed values of γ less than the cutoff can be considered sources of a file, and otherwise proxies. Here, this leads to a false positive rate of zero, and a false negative rate of 2.2%.

4.4.1 Effect of RTT Variance

If RTT delays vary in a significant fashion, then the attack is more difficult to carry out. We conducted a trace-based simulation of the effect that RTT variance has on the attack success. Our simulation performed the timing attack, drawing RTTs from the distribution observed in the traces but also adding additional delay. This additional RTT delay was drawn from a normal distribution with a mean of 150ms and a varying standard deviation. The attack was performed multiple times (that is, there were multiple *trials* per attack) with vote to determine whether the target was a source or proxy. Our goal was to find the number of trials required to have a false positive rate less than 0.0001.

Fig 4 shows the number of trials required versus the standard deviation in RTT in order to have a false positive rate less than *fpr* for values of *fpr* $\in$ {0.0001, 0.001, 0.01}. Note that even when the additional delay has an extreme standard deviation of 3000ms, only 25 trials were required to achieve a false positive rate less than 0.0001. In practice, each trial consists simply of pinging the remote host to determine the RTT, and making a single application-layer request; repeating this sequence 25 times is not difficult or time-consuming. A more stealthy RTT measurement

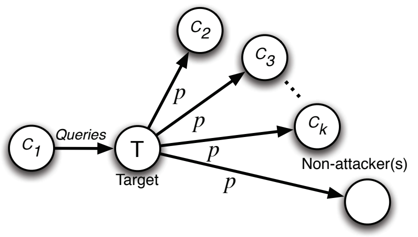


Fig. 5. Setting for collusion attack. One attacker queries for content and the remaining attacker each have a probability p of receiving the query if T does not respond as a source for the content.

than using ICMP is desirable if the attacker fears being noticed.

5 Collusion Attack Revisited

Protocols that rely on the network itself to service queries generally have a high-degree topology. which is true for OneSwarm as well as Gnutella and Freenet. The high degree is necessary to quickly propagate search queries throughout the network. By inserting herself as different identifies in the p2p network (i.e., the Sybil attack [4]), an investigator can be assured that some subset of its nodes will neighbor one or more targets. If there are sufficient neighboring Sybils, then they are capable of inferring properties of the target peer by means of statistical traffic analysis.

OneSwarm aims to thwart traffic analysis of a peer by its neighbor with a particular defense: queries are forwarded to its many neighbors only with probability p when the peer doesn't have requested content. Attacks can defeat this defense as follows: one Sybil sends a request for content and one or more colluding Sybils listen to see if the request is forwarded. This *collusion attack* gains precision as more colluders are involved. If all other attacks we present in this paper are patched, the collusion attack remains.

In this section, we demonstrate that Isdal et al. underestimates the probability of collusion attack success. Further, we quantify the attack's success given *file popularity*, which we show is a critical factor.

5.1 Attack Definition

Searches are recursively flooded to neighbors until the requested file is found some number of hops away. Consider the scenario in Fig. 5, which has k colluding Sybils $C_1, \ldots, C_k$ connected directly to target peer T. A search implementation that doesn't consider privacy would have peer T forward a query to all its neighbors whenever T itself does not posses the search file piece f. In this case, it's trivial for any two colluding neighbors C_1 and C_2 of peer T to determine if T possesses piece f.

The defense against this attack is to have T propagate the query probabilistically. In this case, if T receives the query but does not possess f, then T forwards the query

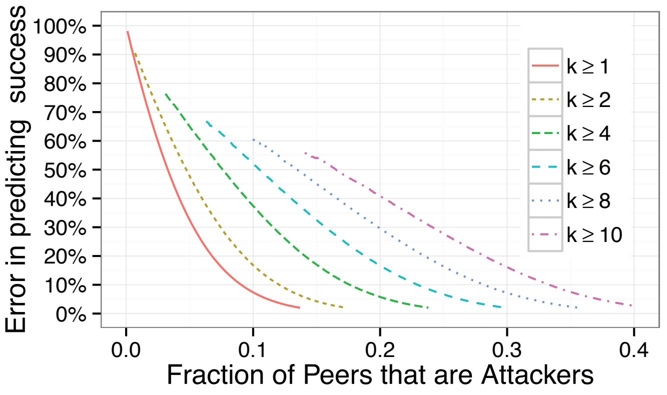


Fig. 6. A comparison between values for $P\{A \geq k\}$ calculated from binomial model (Eq. 10 from [6]) and a Monte Carlo simulation, for $N = 1000$ and $U = 39$. Error is difference divided by the value from the simulation.

to only a subset of its neighbors. This subset is chosen based upon a forwarding probability[3] p. The decision to forward is made independently for each piece of unique content and each neighbor and remains consistent.

Suppose that after C_1 sends an initial query message for f to T that none of the Sybils $C_2, \ldots, C_k$ receive a query message from T. The probability that T possesses f can now be bounded by observing that each Sybil has independent probability p of receiving a query when T *does not* possess f. Therefore, the chance that T will forward the query to at least one of the $k-1$ colluders when it does not have f (i.e., the attack *precision*) is

$$1-(1-p)^{k-1}. \tag{9}$$

5.1.1 *Measuring Attack Precision*

It's natural to seek to measure the attack precision given a certain number of Sybils. Isdal et al. [6] calculated that with $p = 0.5$, achieving 95% precision requires at least $k = 6$ attackers (a querier and 5 colluders) to be directly connected to target T. But what is the probability that a peer will have 6 colluding neighbors? Let N be the total number of peers in the network and C the total number of Sybils. Suppose that each peer is assigned exactly U neighbors uniformly at random by a process similar to drawing balls from an urn. Isdal et al. argued that the probability follows a binomial CDF, and that when $C = 30$, $U = 39$, and $N = 1000$ we have $P\{A \geq 6\} = 0.01$. Specifically, a binomial assignment model has the following form, where A is a random variable denoting the number of Sybils neighboring T.

$$P\{A \geq k\} = \sum_{i=k}^{U} \binom{U}{i} \left(\frac{C}{N}\right)^{i} \left(1-\frac{C}{N}\right)^{U-i} \tag{10}$$

The binomial model in Equation 10 is not appropriate in this context, however, because *i)* neighbors are actually drawn without replacement and *ii)* the probability that peer T_1 sees k colluders is dependent on the probability that peer T_2 sees k colluders.

The lack of independence between peer outcomes increases the difficulty of modeling this scenario; however, deriving a closed form expression for $P\{A \geq k\}$ is not necessary for understanding how the attack efficacy relates to colluder presence. We implemented a simulation[4] that provides Monte Carlo estimates for $P\{A \geq k\}$ for fixed values of N and U, and varying numbers of colluding Sybils C. For each value of k and C, we ran 1,000 trials. For example, for $C = 30, U = 39$, and $N = 1000$, the average probability calculated by the simulation is much higher at $P\{A \geq 6\} = 0.07$.

Fig. 8 plots the results of the simulation; and Fig. 6 plots a comparison of the binomial model against the averages from our simulation, in order to demonstrate the practical disagreement between the two models. For most configurations, the binomial is a fraction of the value calculated by simulation. The figure plots the error as difference of the simulation and binomial means, normalized by the simulation mean; e.g., for 10% attackers, the binomial model predicts $P\{A \geq 4\} = 0.35$ whereas the simulation's mean is 0.56; an error of (0.56-0.35)/0.56=38%.

As a statistical comparison, we compared the sample mean to a hypothesis based on the binomial mean, based on beta prior and two-sided 95% confidence interval from a beta posterior. For attacks where $k \geq 2$ attackers are present, the means of the binomial and simulation are significantly different for all fractions of attackers where $C/N > 0$; for $k \geq 4$, the models don't agree past $C/N \geq 0.017$; for $k \geq 6$, there is no agreement past $C/N \geq 0.042$ attackers; and so on.

5.2 Accounting for File Popularity

Thus far we have assumed that for a given peer T all file pieces appear in T's collection with equal probability. In fact, because some files are more popular than others, any given piece f will tend to appear with varying frequency. It turns out that as f's popularity increases, fewer colluders are actually necessary to achieve the same attack precision. We next derive the relationship between attack precision and file popularity, and go on to show how it influences the efficacy of attack.

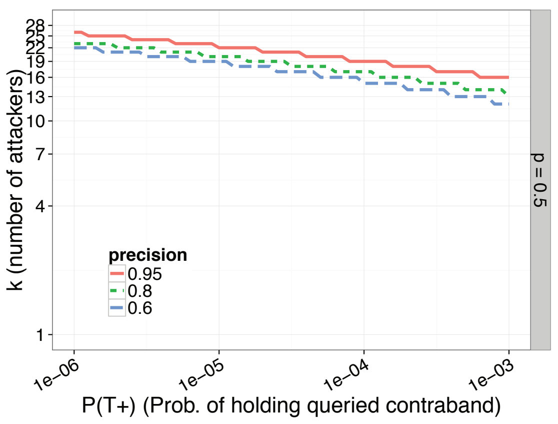
Let T^+ denote the event that the target T has content that was searched for, and let F denote the event that one of the $k-1$ colluders were forwarded the search query issued by C_1. From Bayes Theorem, we can define the precision of the collusion attack as

$$P(T^+|\neg F) = \frac{P(\neg F|T^+)P(T^+)}{P(\neg F|T^+)P(T^+) + P(\neg F|\neg T^+)P(\neg T^+)} \tag{11}$$

The likelihood $P(\neg F|T^+)$ must always be 1 because T never forwards a query when f is in its possession.

3. Here $p = 1 - p_f$, where p_f is Isdal et al.'s notation for the probability of *not* forwarding a query.

4. We will release the short Java program and statistical analysis code with the paper.



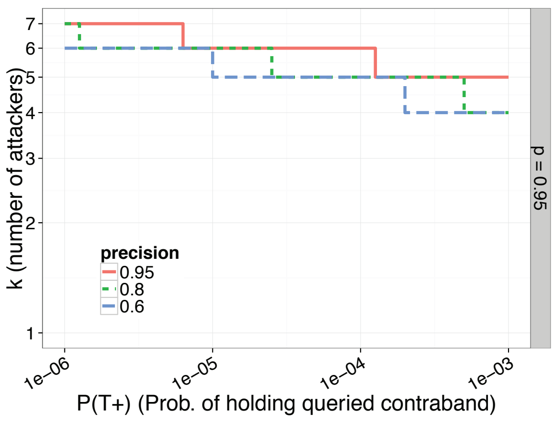


Fig. 7. **(Top)** For a fixed probability of forwarding $p = 0.5$ (the value suggested in the paper), the plot shows the required attackers k given the the popularity of content $P(T+)$ from Eq. 12. Each line is a different precision $P(T^+|\neg F)$; **(Bottom)** The same plot for fixed probability of forwarding $p = 0.95$ (the value hardcoded in software).

Quantity $P(T^+)$ gives the prior probability that a peer possesses file piece f, which we interpret as the popularity of f. Finally, from Eq. 9, it follows that the false positive rate $P(\neg F|\neg T^+)$ is equivalent to $(1-p)^{k-1}$. Given the last expression, we can solve Equation 11 for k, which yields the following result.

$$k = 1 + \frac{\log \frac{P(\neg F|T^+)P(T^+)(1-P(T^+|\neg F))}{P(T^+|\neg F)P(\neg T^+)}}{\log(1-p)} \tag{12}$$

Fig. 7(top) plots Eq. 12 for $p = 0.5$ (the value suggested by Isdal et al.) showing the minimum k value required for different precision levels with prior $P(T^+)$ as the independent variable. (Because it can be only an integer, we plot the ceiling of k.) Fig. 7(bottom) shows the same equation for $p = 0.95$ (the value that was actually hardcoded in the OneSwarm software). Each plot shows three lines corresponding to precision values $P(T^+|\neg F) = 95\%, 80\%,$ and 60%. The first is the value used in Isdal et al., the second is conservative for probable cause, and the last is weaker evidence for probable cause. (Note that the value of k is independent of U.)

For example, when $p = 0.5, P(T^+) = 0.1$, and $P(T^+|\neg F) = 0.95$, then the attack requires $k = 8$ attackers (rather than 6); however, the more important point is that the value of k varies considerably with file popularity. Roughly, as content is an order of magnitude more popular, k typically halves in size. Comparison of the two plots makes the obvious point that p also has a strong influence on k. Releasing the software with a higher value for p than documented in the paper reduced the required number of attackers by 60–75% in all cases. Finally, when $p = 0.95$, the plots demonstrate that values of $k \geq 4$ are sufficient for probable cause for content that is sourced at only one in a thousand peers.

Forwarding Probability and the False Positive Rate. We argued above that $P(\neg F|\neg T^+) = (1-p)^{k-1}$ is the false positive rate (FPR) for the collusion attack, because it gives the probability that the target does not forward the query to any of the colluding attackers given that T is not the source of the file. This quantity is less than 0.0025 when $k \geq 3$ and $p = 0.95$. Thus investigators are at a very low risk of misplaced suspicion when the forwarding probability is high. On the other hand, when $p = 0.5$ the false positive rate is less than 1.6% only when $k \geq 7$; note that for less popular files (i.e. $P(T^+) \leq 0.06$), the same bound of $k \geq 7$ holds for precision values of 95% (see Fig. 7(top)). Regardless of the difference, the requirement of a low FPR is not a significant issue for the investigator. However, querying for multiple files of interest will affect the FPR, as we discuss in Section 5.3.

Comparison Against Onion-style Routing. Isdal et al.'s paper does not quantitatively compare OneSwarm directly against any other privacy mechanism. As a basic comparison we analyze the following simple OR attack. Peers hiding behind an OR circuit can be deanonymized if attackers are selected at random to be in the first and last positions. Selection of these nodes occurs without replacement. Once in these positions, attackers can use a well-known attack of sending a specific sequence of duplicate packets to determine if they are on the same path [20], essentially with a precision of 1 and FPR of 0. Therefore, the chances of a circuit being compromised in a OR network of N peers where C are attackers is

$$P\{A = 2\} = (\tfrac{C}{N})(\tfrac{C-1}{N-1}). \tag{13}$$

Mechanisms such as *guard nodes* [25] can make this attack on OR more difficult. In general the simplicity of the model in Eq. 13 prevents us from comparing OneSwarm to Tor directly.

We use this passive attack because it's a tractable, conceptually simple presentation of the collusion attack on the OR architecture, allowing a straightforward comparison with OneSwarm's architecture. A more accurate analysis of attacks upon Tor would consider a more sophisticated model of Tor's circuit creation algorithm in the presence of various classes of attackers. As shown by Johnson et al. [7], Tor's vulnerability to such attackers is greater than one might expect from this simple model, though it does not change the relative comparison, that

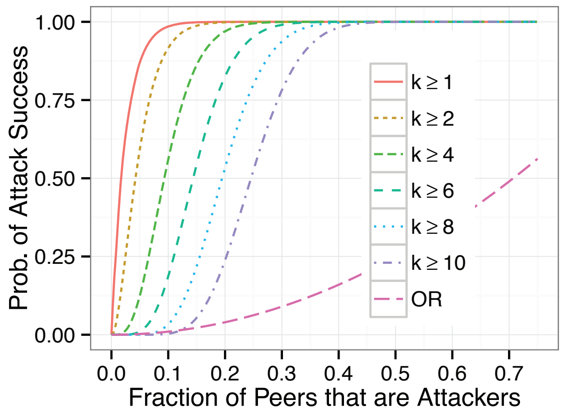


Fig. 8. Collusion attack success (from simulation where U=39) on OneSwarm given a required minimum value of k, and first-and-last attack success against OR (Eq. 13).

is, that OneSwarm is substantially more vulnerable to a comparably resourced adversary than Tor.

Fig. 8 compares effectiveness of these attacks against OneSwarm and OR. The plot is independent of a chosen value for p; to determine the required value of k, we first choose a value of p and consult Fig. 7(top or bottom). There are a number of implications to note. First, there is a non-linear, sharp increase in attacker success as investigators increase their proportion of the OneSwarm network. When the attack requires only that $k \geq 4$, the chances of success are 98% as attackers comprise 25% of peers in the network. Note that $k \geq 4$ is sufficient for even unpopular content when $p = 0.95$, which was the hard coded value in the released software. However, even if $p = 0.5$, requiring larger values of k, the effectiveness of the attack on OneSwarm grows far more quickly than the simple attack on OR.

The previous analysis is very conservative in that it assumes that the attackers join the network once. As such, they can only investigate peers to whom they've attached sufficient colluders. But attackers can repeatedly quit and rejoin the system with new identities, thereby investigating more and more peers over time. When investigators comprise 5% of the network, the attack's success for content that requires $k \geq 4$ attackers is 4%. By the geometric distribution, the expected number of times investigators must rejoin the network until success is 25 times. OneSwarm offers no Sybil attack protections and rejoining is quick and trivial to execute. Churn in the OneSwarm population will force the community server to assign and re-assign new untrusted relationships.

Trusted Neighbors. Our attacks are worst case in that we assume the peer has 39 neighbors. To give a point of comparison, if the node has only $U = 20$ neighbors from the community server (and 19 trusted friends) then the investigator's job is harder. For example, the $k \geq 4$ line in Fig. 8 shifts roughly to almost where $k \geq 8$ is in the figure. Again, attackers can execute this attack as many times as necessary.

5.3 Practical Considerations

The Multiple Comparison Problem. A naive version of the collusion attack would see investigators joining the network as described above. The investigator would then test each connected peer for possession of all known files of interest, and claim probable cause for each peer which was determined to have at least one file. This naive version is susceptible to the well-known *multiple comparison problem* of testing sets of hypothesis at once. We elide the analysis here, but we note the false positive rate quickly grows unusably high as the number of files tested per peer increases.

There are several ways to limit the effect of this problem, each of which centers around limiting the number of different files investigators check a peer for possession of. One such method is as follows. First, investigators survey the network for CP files by searching for well-known keywords. In other p2p networks such as Gnutella and eDonkey2000, these files are typically named with explicit and descriptive names; our law enforcement partners confirm the same appears to be true of OneSwarm. The files are rank ordered by their apparent popularity in the system. Investigators then sweep through the network by performing the collusion attack for the top n files, where n is chosen to keep the FPR acceptably low. Only once these peers have been fully investigated, including offline searches and the subsequent legal process, do investigators repeat their survey to determine the next rank-ordered list to investigate.

Post-Warrant Confirmation. OneSwarm peers connect to one another using SSL bootstrapped by their RSA key pairs. A OneSwarm peer stores keys of neighbors and its own key on the local file system. Once a search warrant is executed and a machine seized, the investigator can tie that machine to specific network traffic based on the recovered key file. Even if the content has been deleted locally, the mechanism can confirm with cryptographic precision if this machine is the machine that transferred it. BitTorrent clients are subject to *forensic tagging* [10], in which investigators offer remote machines nonces to store. These nonces can later be recovered to confirm that the correct machine was seized. However, OneSwarm makes this task redundant since all outgoing traffic is signed and the keys are stored persistently. All OneSwarm peers cryptographically sign a large amount of evidence that can be used against them.

6 TCP-Based Attacks

In this section, we demonstrate a novel adaptation of a known TCP-based attack [18] that can identify whether a OneSwarm peer is the source of data or a proxy. Peers that do not rate limit outgoing traffic are vulnerable. OneSwarm happens to turn on rate limiting by default to 80% of a test transfer to a central server, but nothing prevents eager users from turning off the rate limiting. A more

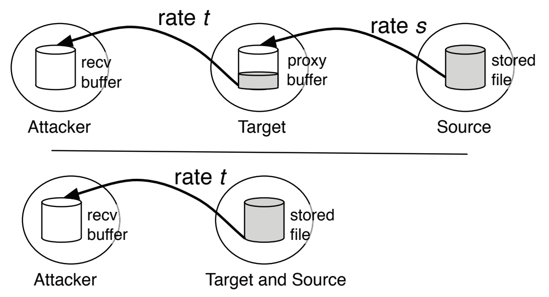


Fig. 9. The TCP-based attack increases the rate of transfer, t, from the target so that it is greater than source's rate, s. The buffer at the proxy should empty before the transfer is completed. In the case where the target is the source, for any value of t, the transfer should complete.

robust defense without rate limiting is to probabilistically drop outgoing packets and audit incoming acks. In this section, we detail the attack and its limitations, show experimental results for an implementation of the attack executed on a simple non-OneSwarm transfer, and we discuss defenses to the attack.

The attack leverages *optimistic acking* [18], where a receiver sends TCP acknowledgements for data before it is received, increasing throughput. Sherwood et al. [19] leverage the same mechanism to perform denial-of-service attacks against a server. Our contribution is in showing that the same mechanisms can be used to distinguish proxies from sources. The attack is not specific to OneSwarm; all anonymous file-sharing protocol designers should be aware of the attack.

Fig. 9 illustrates our TCP attack scenario. An attacker requests a file from a target. The attacker induces a higher-bandwidth connection between itself and target than between the target and a potential source of data. If t can be made greater than any potential s and the target is not the source, it will stall out. The stall occurs because the target's application level buffers will run out before the actual source can fill them.

In our evaluations, we show that the attack can succeed in practice with a TPR of 89% and FPR of 4%. In sum, OneSwarm is vulnerable (when rate-limiting is off) because it defends against only application-level timing and traffic attacks, and does not defend against an attacker breaking the underlying network abstraction.

Trusted Peers. When a peer has trusted peers (that are not undercover investigators), they reduce the chances that an investigator has an opportunity to execute the TCP attack, but do not affect the success of the attack itself. Vulnerable peers are those that have at least one untrusted relationship to an investigator; i.e., set $k = 1$ in Fig. 8.

6.1 Attack Details

To conduct the attack, the investigator incorrectly acks packets that were lost and optimistically acks packets that have not yet been received. The investigator also always advertises a large TCP flow control window so as not to inhibit the sender. The attack uses unsupervised clustering (there is no training) to classify downloads as from a proxy or from an original source. The specific steps of the attack are as follows:

1) The investigators starts with m files to download from the p2p network and queries for their availability from its neighbors. From the m files, a neighbor A responds that it can used to download $n > 1$ of the files. A is now the subject of the attack.
2) The investigator downloads each of the n files (or portions of each) 5 times, each a separate TCP connection. The download takes place with normal TCP or our aggressive TCP tool. The mean bandwidth of the 5 downloads is recorded. (Hence, $5n$ downloads total, n means total.)
3) The investigator calculates a *threshold* as the mean of the n means. A is classified as the source of a file when the mean download speed is above the threshold, and a proxy otherwise.

This attack is not unsuccessful when normal TCP is used. However, as we show, the attacks success is much greater when our aggressive TCP tool is used.

Aggressive TCP tool. Our tool does not attempt to download the data correctly. Instead, whenever a packet is received, the attacker sends an ack for the highest sequence number ever received from the sender, regardless of whether earlier bytes in the TCP flow were lost. Because acks are cumulative in TCP, the skipped over byte sequences are not a concern of the sender. Additionally, the receiver optimistically acks *extra* segments it has yet to receive but are likely on the way. The value of *extra* starts at 0 and the receiver increases it by *rate* bytes (and rounded to segment sizes) for each packet received, regardless of loss or duplication, until a given *maxExtra* value is reached.

The reason to grow this optimistic acking slowly is that the sender will silently drop acks for packets beyond what it has sent. If a sender receives overly optimistic acks, it will not close the connection, as TCP was designed to manage the occasional odd error. Accordingly, thoughtless optimistic acking by the attacker will have no effect, and the attacker must grow the window commensurately with the sender's values. As the window grows, heavy packet loss will occur.

If attacker grows the sender's window too aggressively, the RTT calculation at the sender can become mis-estimated as a very small duration, making a timeout exceedingly easy. When a timeout occurs, the sender will back off, cutting the rate drastically, and will resend old data, which is very bad for the attack. Therefore, if the receiver doesn't receive data by some timeout (250ms in our implementation), an ack packet is sent to the sender for the *latest* packet received, rather than the highest received. In our tests, this quickly re-initiates the data flow, and the attacker can return to acking the highest byte sent.

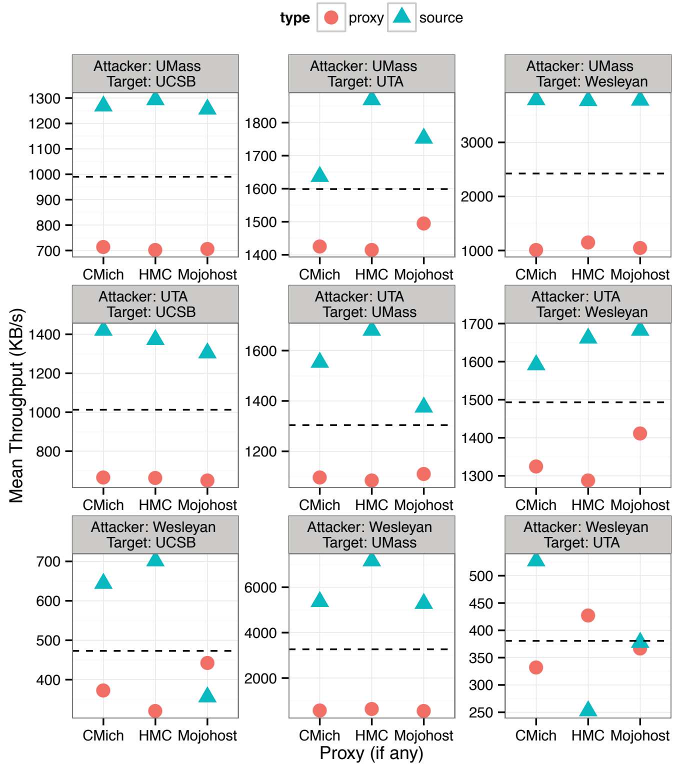


Fig. 10. A visualization of traces from our aggressive TCP program and the attack results. Points represent the mean of 5 downloads. The dotted line is the threshold, attempting to place sources above the line, proxies below. TPR=24/27=89% and FPR=1/27=4%. With normal TCP, TPR=17/27=63% and FPR=10/27=37%. The pairs of TPRs and FPRs are statistically different, respectively.

The file downloaded must be large enough to get out of slow start. Fortunately, even for small torrent pieces in OneSwarm, BitTorrent-style pipelining [2] prevents the attack from stalling.

6.2 Attack Experimentation

We implemented the attack in about 700 lines of C++ to test its feasibility. The attack implementation and details of our measurement experiments (including packet header logs) are available from http://traces.cs.umass.edu. Our prototype implementation attacks the HTTP protocol rather than the BitTorrent protocol, as HTTP has fewer implementation details to manage. In principle, the same attack will work on pipelined BitTorrent requests. Further, our implementation does not implement the SSL handshake between peers that OneSwarm requires.

Methodology. We configured a network as shown in Fig. 9. We could not use PlanetLab for this experiment since it enforces a bandwidth cap, which it splits among all virtual hosts on each node [11]. The attacking machines were located at UMass, Wesleyan (CT), and UT Arlington (TX); the targets were a superset of the attacking machines and also included UCSB (CA); and the proxied sources at Central Michigan University, Harvey Mudd College (CA)

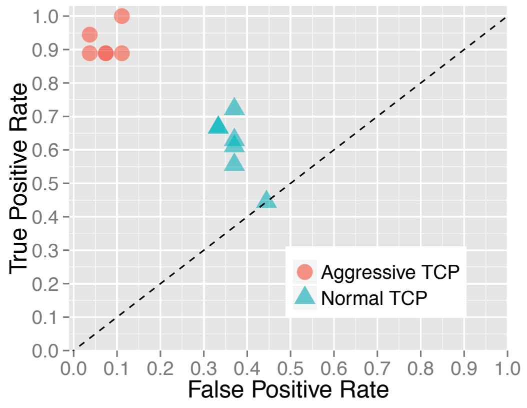


Fig. 11. Fig. 10 assumes a balance of sources and proxies in each test. This plot shows the FPR and TPR for a series of experiments: 3 files downloaded via proxies and 1 directly from a source; 3 proxied and 2 direct; 3 proxied and 3 direct. In all cases, using our aggressive TCP tool is highly accurate and better than using normal TCP for the same attack. (Some points are over-plotted.)

and Mojohost, Inc. (FL). We chose these proxies to model relatively close (UCSB and HMC are both in CA) and distant (UCSB to Mojohost crosses the country) peers. We served the file directly through an httpd daemon from the targets, and we used netcat to act as an application-level proxy to the other sources. Our target file was a 10MB ISO image.

We ran trials with both our aggressive TCP implementation and wget. We fixed the parameters of our attack implementation at relatively conservative values. We set the $maxExtra$ value to 50 TCP segments for all connections, though in our experience this value should vary per bandwidth of the connection to the target. We ran measurement experiments across 9 combinations of attacker, target, and 3 proxies; for each we recorded measurements without the proxy. Each measurement was repeated five times with each tool. Hence, we recorded 9*3*2*5*2 = 540 traces total. For our aggressive TCP tool, we recorded the max bandwidth achieved since its behavior can cause stalling. For wget we recorded the mean bandwidth of the download.

Results. Fig. 10 shows the results of using our aggressive TCP tool to carry out the attack. The figure shows both the means of 5 downloads and the chosen threshold. These 54 experiments are comprised of 27 positives and 27 negatives. The TPR is 24/27=89%. The FPR is 1/27=4%. When the same test is be performed with normal TCP, then TPR=17/27=63% and FPR=10/27=37%. The two TPRs and FPRs are significantly different according to a one-sided permutation test of proportions (TPR comparison p-value: 0.018; FPR comparison p-value: 0.002).

We also tested the attack by limiting the number of sources. Figure 11 shows the FPR and TPR for a series of experiments: 3 files downloaded via proxies and 1 directly from a source; 3 proxied and 2 direct; 3 proxied and 3 direct. In all cases, using our aggressive TCP tool is highly accurate and better than using normal TCP for the same attack.

Limitations. We note that the attack depends upon the attacker being able to force the target to serve files faster than they can be retrieved from potential sources when proxied. This assumption does not always hold. The attacker could have poor connectivity to the target; this is easily detected in advance, and attackers can acquire high-bandwidth connections. Less avoidably, there could be rate limiting or traffic shaping in place, or the target, acting as a proxy, may have an extremely high bandwidth connection to the true source. In the latter case, the machines may be co-located, which is acceptable for a search warrant, or the owner of the machine may choose to cooperate with law enforcement in investigation of a crime. This type of cooperation is typical in practice, such as when investigations of non-anonymous systems lead to an innocent owner of an open Wi-Fi base station whose neighbor is using the connection illicitly.

Defenses. To detect this attack, nodes can purposefully drop outgoing packets from their TCP stream and determine if the remote peer requests the missing data or acknowledges receiving it. There is a patch (https://www.kb.cert.org/vuls/id/102014) to an old version of the Linux kernel that addresses this vulnerability, but any such fix results in a non-standard TCP implementation and is unlikely to be deployed on a wide scale. To defend against the attack without detecting it, OneSwarm can force a bandwidth cap on peers that can't be turned off, which isn't done currently. Since OneSwarm is an open source project, investigators will know if and when such defenses are deployed.

7 Conclusions

We showed that publicly available data can be gathered supporting at least a probable cause standard against OneSwarm peers that share child abuse materials. Each of the attacks we presented adhere to the restrictions of a constrained generally applicable criminal procedure. We quantified the effectiveness of three independent attacks: one using timing information; one using information about query forwarding; and a third comparing TCP throughput. We also introduced a new threat model, based on digital forensics and computer crime law.

References

[1] J. Chu, K. Labonte, and B. N. Levine. Availability and Locality Measurements of Peer-to-Peer File Systems. In *Proc. ITCom*, volume SPIE 4868, pages 310–321, July 2002.

[2] B. Cohen. Incentives Build Robustness in BitTorrent. In *Intl. Workshop on Peer-to-Peer Systems (IPTPS)*, February 2003.

[3] L. Denoyer and M. Latapy. Measurement and Analysis of P2P Activity Against Paedophile. *Journée Données et Apprentissage Artificiel (DAPA)*, March 2009.

[4] J. Douceur. The Sybil Attack. In *Proc. Intl Wkshp on Peer-to-Peer Systems (IPTPS)*, 2002.

[5] T. Howard. Don't Cache Out Your Case. *Berkeley Technology Law Journal*, 19:1157–1575, Fall 2004.

[6] T. Isdal, M. Piatek, A. Krishnamurthy, and T. Anderson. Privacy-preserving P2P data sharing with OneSwarm. In *Proc. ACM SIGCOMM*, pages 111–122, August 2010.

[7] A. Johnson, C. Wacek, R. Jansen, M. Sherr, and P. Syverson. Users get routed: Traffic correlation on Tor by realistic adversaries. In *Proc. ACM CCS*, pages 337–348, November 2013.

[8] M. Latapy, C. Magnien, and R. Fournier. Quantifying paedophile activity in a large P2P system. In *IEEE Infocom Mini-Conference*. http://antipaedo.lip6.fr, April 2011.

[9] M. Liberatore, R. Erdely, T. Kerle, B. N. Levine, and C. Shields. Forensic Investigation of Peer-to-Peer File Sharing Networks. In *Proc. DFRWS*, August 2010.

[10] M. Liberatore, B. N. Levine, C. Shields, and B. Lynn. Efficient Tagging of Remote Peers During Child Pornography Investigations. *IEEE TDSC*, 11(5):425–439, 2014.

[11] L. Peterson, A. Bavier, M. E. Fiuczynski, and S. Muir. Experiences building PlanetLab. In *Proc. USENIX OSDI*, pages 351–366, 2006.

[12] M. Piatek, T. Kohno, and A. Krishnamurthy. Challenges and directions for monitoring P2P file sharing networks. In *Proc. USENIX Hot Topics in Security (HotSec)*, pages 12:1–12:7, July 2008.

[13] S. Prusty, B. N. Levine, and M. Liberatore. Forensic Investigation of the OneSwarm Anonymous Filesharing System. In *Proc. ACM CCS*, pages 201–214, October 2011.

[14] M. Reed, P. Syverson, and D. Goldschlag. Anonymous connections and onion routing. *IEEE Journal on Selected Areas in Communications*, 16(4):482–494, May 1998.

[15] M. Ripeanu, I. Foster, and A. Iamnitchi. Mapping the Gnutella Network. *IEEE Internet Computing Journal*, 6(1):50–57, 2002.

[16] J. Rorher. MUTE. http://mute-net.sourceforge.net.

[17] RShare. http://www.stealthnet.de/en_index.php.

[18] S. Savage, N. Cardwell, D. Wetherall, and T. Anderson. TCP congestion control with a misbehaving receiver. In *Proc. ACM SIGCOMM*, pages 71–78, October 1999.

[19] R. Sherwood, B. Bhattacharjee, and R. Braud. Misbehaving TCP receivers can cause Internet-wide congestion collapse. In *Proc. ACM CCS*, pages 383–392, October 2005.

[20] P. Syverson, G. Tsudik, M. Reed, and C. Landwehr. Towards an Analysis of Onion Routing Security. In *Intl. Wkshp. on Designing Privacy Enhancing Technologies*, pages 96–114, July 2000.

[21] U.S. Dept. of Justice. National Strategy for Child Exploitation Prevention and Interdiction: A Report to Congress. http://www.projectsafechildhood.gov/docs/natstrategyreport.pdf, August 2010.

[22] U.S. General Accounting Office. File-Sharing Programs. Child Pornography Is Readily Accessible over Peer-to-Peer Networks. GAO-03-537T. Statement Before Congress; Linda D. Koontz, Information Management Issues, March 2003.

[23] R. J. Walls, B. N. Levine, M. Liberatore, and C. Shields. Effective Digital Forensics Research is Investigator-Centric. In *Proc. USENIX Workshop on Hot Topics in Security (HotSec)*, August 2011.

[24] J. Wolak, D. Finkelhor, and K. J. Mitchell. Child-Pornography Possessors Arrested in Internet-Related Crimes: Findings From the National Juvenile Online Victimization Study. Technical report, National Center for Missing & Exploited Children, 2005.

[25] M. Wright, M. Adler, B. N. Levine, and C. Shields. Defending Anonymous Communication Against Passive Logging Attacks. In *Proc. IEEE Symp. on Security & Privacy*, pages 28–41, May 2003.

Acknowledgement. Supported in part by NSF award CNS-1018615. We thank Elizabeth Belding, Norman Danner, Danny Krizanc, Clay Shields, and Matthew Wright for providing resources for our experiments. We thank Robert Walls and Brian Lynn for their insightful comments.

George Bissias is a Postdoctoral Researcher in the College of Information and Computer Sciences at UMass Amherst. His research concerns information privacy and p2p networks.

Brian Neil Levine is a Professor in the College of Information and Computer Sciences at UMass Amherst. His research focuses on privacy, digital forensics, mobile networks, and the Internet.

Marc Liberatore is a Research Scientist in the College of Information and Computer Sciences at UMass Amherst. His research interests include digital forensics, privacy, and p2p systems.

Swagatika Prusty is a Software Engineer and Technical Lead at Oracle. She received her MS in Computer Science from UMass Amherst.