BitTAKA: Anonymous BitTorrent

Tural Badirkhanli
turalb@mit.edu

Amrik Kochhar
amrik@mit.edu

Albert Ni
albertni@mit.edu

Kuat Yessenov
kuat@mit.edu

May 15, 2008

Abstract

The highly popular BitTorrent peer-to-peer file sharing protocol does not provide users any anonymity. We present BitTAKA, a modification of the BitTorrent protocol that provide users a degree of anonymity without an overwhelming performance sacrifice. Partly inspired by onion-based routing protocols, we analyze the theoretical anonymity of BitTAKA. We also measure the performance loss in our protocol relative to the original BitTorrent protocol using a customized simulation framework.

1 Introduction

BitTorrent[1] is one of the most popular peer-to-peer file sharing protocols on the Internet today. In fact, according to CableLabs, a North American research and development laboratory, BitTorrent currently accounts for approximately 18% of all broadband traffic [2]. However, the BitTorrent protocol does not provide users any anonymity. In particular, users can easily discover the IP addresses of other users in the network, as well as what files they are uploading and downloading. This has become a serious concern among BitTorrent users for a variety of reasons, including the risk of being sued for downloading a copyrighted material. In this paper we present the BitTAKA protocol, a modified BitTorrent protocol for anonymous peer-to-peer file sharing.

1.1 TOR

TOR[3] is an overlay network that operates on the TCP stream level and provides anonymity to its users by forwarding packets through what is called a virtual circuit. The packets are forwarded in layers of encrypted data, which is why it is called The Onion Router (TOR). TOR ensures that every hop in the virtual circuit only knows about the previous hop the packet came from and the next hop that it needs to forward the packet. No hop in the circuit has information about both the source and the destination of the packet.

1.2 BitTorrent over TOR?

An obvious question to ask is why not just use the BitTorrent over TOR? A problem with using TOR for BitTorrent traffic is that it decreases network performance substantially. One of the reasons TOR is slow is that a typical circuit consists of 3-4 TOR nodes through which a packet needs to go before reaching the destination. These nodes are located in various parts of the world and the traffic is often bottlenecked by one of these nodes. Another reason stems from the measure taken to prevent a possible attack to TOR network. The problem is that TOR nodes self-report their available bandwidths and therefore can lie about them, thus resulting in network congestion. To prevent this, TOR assumes a maximum bandwidth (currently 1.5 Mb/s) for every node independent of what the node reports [4]. Obviously, this leads to inefficiency because the bandwidths of the nodes that, in fact, have more bandwidth than the enforced upper-limit are not utilized. However, even if all the bandwidths of all the nodes were fully utilized, the TOR network would still likely be too small to handle all of the file sharing traffic. Lastly, TOR currently tries to avoid being used for BitTorrent traffic by rejecting certain ports that are known to belong to BitTorrent.

1.3 Organization

In Section 2, we describe our goals for the BitTAKA protocol, as well as the protocol itself. In Section 2.5.3, we analyze the security of the system in the face of a variety of attacks. In Section 4, we describe a simulation implemented to test various aspects of the protocol, and its results. Finally, we talk about potential improvements and conclude in Sections 5 and 7.

2 The BitTAKA Protocol

In this section we describe the BitTAKA protocol. In particular, we explain how BitTAKA can be used to share files between users while still providing those users anonymity, the concept of which is more concretely defined in Section 2.1. Terminology and notation specific to BitTAKA and this paper is explicitly covered in Section 2.1. The BitTAKA tracker is described in Section 2.2. Further specifics regarding how everything works are given in Sections 2.3.3 through 2.4.4.

2.1 Axiom of Anonymity

The fundamental idea behind BitTAKA is to make it possible for any two of its clients to share a file without either client learning any identifying information about the other. In particular, BitTAKA aims to satisfy the following "axiom of anonymity":

- Given an arbitrary user U_i of the BitTAKA network for which U_i has an IP address of IP_{U_i}, and is downloading and/or uploading a set of files F_{U_i}, any other arbitrary user U_j of the network can never associate IP_{U_i} with any element of F_{U_i}.

In other words, this says that any BitTAKA user cannot determine what files are being shared by a given IP address, and cannot determine what IP addresses are sharing a given file. In the simplest case where two users A and B are sharing a file f, since both users know of a file that the other is sharing, this implies that A and B must not be able to learn each other's IP addresses. In order to make this possible, BitTAKA requires that all traffic between A and B go through a third intermediary client F.

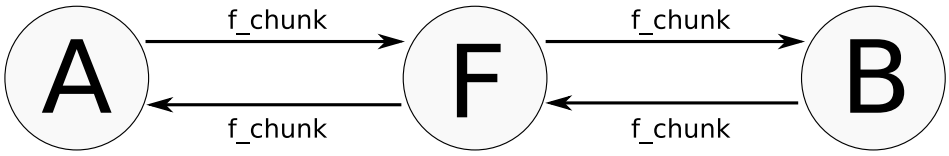

Figure 1: The simplest case, where clients A and B share parts of file f (denoted by f_chunk) without communicating directly, but rather through a forwarder, F.

In Figure 1, it is not necessary for A and B to know each other's IP addresses. However, it is necessary for both A and B to know IP_F, and similarly, it is necessary for F to know both IP_A and IP_B. Therefore, F must not know of the contents of the file f it is forwarding. Given this basic setup, three primary issues arise, each of which will subsequently be addressed within this section.

The first issue is how to give F any incentive to forward. In Figure 1, F gains nothing from facilitating the sharing of f between A and B, and loses bandwidth in the process. The second

issue is how to bring a situation such as the one in Figure 1 into existence in the first place. In the original BitTorrent, one of A and B would have contacted a tracker, and been given the other's IP address to connect to. Obviously this protocol is no longer viable. In particular, where F comes from in the very first place must be determined. Lastly, the third issue is how to keep F from learning the contents of f.

2.2 Terminology and Notation

We introduce some terminology and notation specific to BitTAKA and/or this paper, as well as formalize the notions behind some of general terms that have already been used in this section.

BitTAKA - The name of our file sharing network. May be used interchangeably with the phrase "the network".

Client - This refers to any arbitrary user, for instance a person on his/her personal computer, connected to the network to share files. "Client" is used interchangeably with "user".

Share - A client is said to **share** a file if it is either downloading or uploading that file (or both). In particular, for a given file f, every client, from one that has just started downloading it and possesses no part of f, to what is traditionally known in BitTorrent as a seeder for f (a client that already possesses the entirety of f and is staying connected to the network to help distribute f to others) is considered to be sharing f.

Peer - Two clients in the network are considered to be **peers**, with respect to a given file f, if they are currently sharing f with one another.

Forwarder - A client F in the network is considered to be a **forwarder**, with respect to a given file f and a given pair of peers A and B, if F is forwarding parts of f between A and B. In this situation, we often say "F is a forwarder for peers A and B".

Neighbor - Two clients A and B in the network are considered to be **neighbors**, with respect to a given file f, if traffic related to f is being passed between A and B in some manner. Note that this implies that A and B know one another's IP addresses. In addition, if some client F is a forwarder for some client C, this implies that F and C are neighbors, so for any given file, C's set of forwarders for that file is a subset of C's set of neighbors.

Connection - Two neighbors in the network are always considered to have a **connection** between each other. Depending on the context, connections may be considered to be directed. They also may be file-specific or just referring to the connection through which all traffic between two neighbors is passing through.

Forwarding Connection - This is a directed, file-specific connection that is being used to forward a file f from a forwarder, who does not actually care about the contents of f, to one of its neighbors who does.

Peer Connection - This is a directed, file-specific connection that is being used to send a file from a client to a forwarder, so that the forwarder can then subsequently forward this to the intended

destination peer.

IP$_X$ **-** Stands for "the IP address of client X".

UID$_X$ **-** Stands for "the unique ID of client X". The purpose of UIDs is explained in Section 2.3.1.

Know$_X$ **-** This is the set of IP addresses and UIDs known to client X.

2.3 Tracker

In the BitTorrent protocol[5], trackers are centralized servers which serve as an initial contact point for BitTorrent clients when they first start downloading and/or uploading a file. Trackers also assist clients in establishing additional connections with other clients, and communicate with all the clients they know of periodically to aggregate statistics, update information about the state of the network, and more.

In BitTAKA, the tracker serves a much larger role than that of a BitTorrent tracker. In this section, we introduce the BitTAKA tracker and describe its role within the BitTAKA protocol as a whole. We also introduce a special piece of state maintained within the BitTAKA tracker known as the IP-UID map in Section 2.3.1. From this point forward, the term "tracker" is assumed to mean "BitTAKA tracker", and not "BitTorrent tracker".

Also, note that thus far, we have referred to the BitTAKA tracker in singular terms. This is because in this section, we make the assumption that we have but exactly one tracker, that is both *centralized* and *trusted* for everything. Readers familiar with BitTorrent may note that not only do there exist many BitTorrent trackers, but that the BitTorrent makes it possible to have distributed, decentralized tracking. We make the "one centralized tracker" assumption for two primary reasons. First, this assumption simplifies the description of the BitTAKA protocol, as it there is no ambiguity as to which tracker we are referring to at any given moment. Second, and more importantly, certain elements of the version of the BitTAKA protocol being presented in this section do indeed depend on having a centralized trusted tracker. Further discussion on modifications to the BitTAKA protocol which would remove the need for a centralized trusted tracker can be found in Section 5.

2.3.1 Responsibilities

The primary, and in some sense only, responsibility of the tracker is to establish and regulate communication between BitTAKA clients. Like in the original BitTorrent protocol, the tracker serves as an initial contact point for a client seeking to share a file. However, unlike in BitTorrent, where subsequent peer to peer communication does not necessarily have to be facilitated by the BitTorrent tracker, in BitTAKA, all communication is strictly regulated by the tracker. As we saw in Section 2.1, establishing a connection between two peers for the sharing of some file is no longer as simple as it was in BitTorrent. In particular, a forwarder is needed, and it is the tracker's responsibility to find one. In addition, the tracker cannot simply assign any arbitrary forwarder to a given pair of peers. Rules and guidelines regulating this process are detailed in Section 2.3.3.

To fulfill its duties, the tracker must also maintain two important pieces of state - the IP-UID map, and the network graph.

2.3.2 IP-UID Mapping

The IP-UID map is one of the keys to BitTAKA's anonymity. It is a two-way mapping between the IP addresses of clients in the network, and unique IDs assigned to those clients by the tracker. The purpose of UIDs is to act as a piece of identifying information for clients that does not expose any further information about the client (in particular, the client's IP address). For instance, the set of clients sharing some file will often learn many of the UIDs of the other clients sharing that file, but none of their IP addresses.

Two invariants regarding the IP-UID map are vital. First, it is absolutely crucial that no clients ever learn of *any* element in the IP-UID map. Second, no information about a client should be exposed through its UID. This means that clients should not be assigned permanent, or even long-term UIDs, and that the UID of a client should not depend on its IP address in any way. In practice, assigning clients a new, randomly generated 256-bit integer every 12 hours should suffice.

2.3.3 Network Graph

The network graph is an encapsulation of all of the connections in the network. It is not necessarily a true graph, but rather just a full picture of the network, and contains information about all of the clients, all of the connections, and all of the files being shared. It is important for the tracker to keep the network graph reasonably up to date at all times, since knowing what clients are sharing what files, and which forwarders are being used for which pairs of peers, are needed for the tracker to manage the connections of the network.

Note that it is actually not too difficult for the tracker to maintain the network graph. While in BitTorrent, users may form connections without informing any BitTorrent trackers, in BitTAKA, clients are completely dependent on the tracker for forming new connections. In addition, the BitTAKA tracker can imitate BitTorrent trackers in how they periodically communicate with the clients of the network for status updates, or to simply ensure that the client has not abruptly disconnected.

2.4 Circuits

Now that the tracker has been introduced, we can finally discuss the details of the BitTAKA protocol. To do so, let us introduce the BitTAKA **circuit**. Named in the spirit of the TOR circuit, a BitTAKA circuit consists of the clients and connections that make the sharing of one or more files possible, and is essentially the fundamental unit of BitTAKA. We have already seen an example of the simplest possible circuit in Figure 1 (shown again here for convenience along with additional labeling).

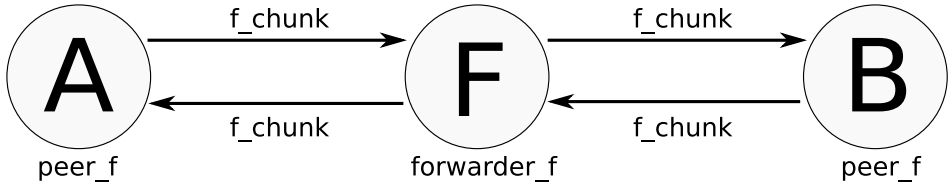

Figure 2: The simplest possible circuit.

In this section, we describe more complicated circuits, how they are used to address the three issues mentioned in Section 2.1, and how they are formed in the first place.

2.4.1 Standalone Circuits

As has been noted, the circuit shown in Figures 1 and 2 provides no incentive for the forwarder to actually perform the task required of it. Thus, let us introduce the concept of a **standalone circuit**, which is a circuit that includes incentive for the forwarders in the circuit to actually forward.

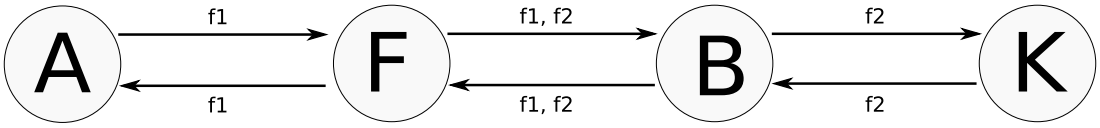

Figure 3: The simplest standalone circuit.

The simplest standalone circuit consists of four clients as shown in Figure 3. In this circuit, peers A and B are sharing file $f1$, and peers F and K are sharing file $f2$. Thus, F is acting as a forwarder for A and B, and B is acting as a forwarder for F and K. Also,

$$\begin{aligned} Know_A &= \{IP_F, UID_B\}, \\ Know_F &= \{IP_A, IP_B, UID_K\}, \\ Know_B &= \{IP_F, IP_K, UID_A\}, \\ Know_K &= \{IP_B, UID_F\}. \end{aligned}$$

Of course, this construct alone does not actually provide incentive for F to forward $f1$ and B to forward $f2$. Rather, a modified version of the BitTorrent tit-for-tat upload incentive system is necessary.

2.4.2 Tit-for-Tat

In BitTorrent, a tit-for-tat based system is used to give peers incentive to upload, where a user's upload rate to a peer dictates the rate at which the user is allowed to download from that peer as

well. The most important feature of this system is that it makes it possible to punish a malicious or uncooperative user that seeks to maximize its download rate and minimize the amount of its bandwidth used for uploads.

However, in BitTAKA, not only are peers not directly connected, but the livelihood of a connection between two peers also depend on the cooperation of a forwarder which neither seeks to upload nor download the file being shared by the peers-in-question. This has a couple of important ramifications. First, given a pair of peers A, B and a forwarder F, the download rate of A from B is dependent on the cooperation of both F and B. However, if one of F and B are being uncooperative, it is impossible for A to determine which. Second, in order for a forwarder C to have any incentive whatsoever to actually forward, C's download rates for files it actually cares about must depend on more than just its upload rates for those same files, but also its forwarding rates.

Because of the aforementioned ramifications of the BitTAKA protocol, clients in the BitTAKA network use a system where blame for a poor connection is distributed between forwarders and peers. In particular, in a given circuit $\mathcal{C}$ containing a pair of peers and a forwarder, each peer effectively views the forwarder and other peer as one unit, as in Figure 4.

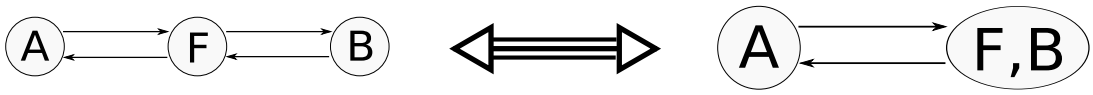

Figure 4: A circuit $\mathcal{C}$ containing peers A, B and forwarder F on the left, and how A views this circuit from a tit-for-tat perspective on the right.

Consequently, from the perspective of any given peer, a circuit involving that peer is very similar to just a standard BitTorrent connection between two users in terms of managing tit-for-tat. We now return to the simplest standalone circuit from Figure 3, and consider it from the perspective of its two forwarders.

Figure 5: The view of the simplest standalone circuit from the perspective of F on the left, and from the perspective of B on the right.

In Figure 5, from F's perspective, B and K are effectively one unit, and from B's perspective, A and F are effectively one unit. Since F wants to share a file (or files) with K, it has incentive to satisfy the wants of both elements of the (B, K) unit so that tit-for-tat will kick in in F's favor. Note that in particular, this implies that F has an incentive to satisfy the wants of B. But, since B wants to share a file with A, this implies that F will need to forward traffic from A to B. We perform a more formal analysis of the incentive structure below.

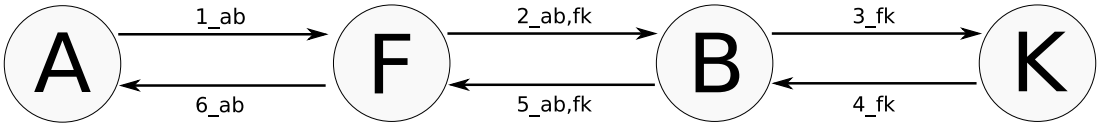

Figure 6: The simplest standalone circuit, with its connections labeled 1 through 6.

Consider the connections labeled 1 through 6 in Figure 6. Assume that the peers A and B are sharing the file f_{ab}, and that peers F and K are sharing the file f_{fk}. Since connections 2 and 5 are being used to share both f_{ab} and f_{fk}, we distinguish these cases by calling them $2_{ab}, 2_{fk}, 5_{ab}, 5_{fk}$. Now, let us examine how tit-for-tat impacts each connection, and enforces incentive for all clients involved in the circuit.

Connection 1 - This peer connection is used by A to send portions of f_{ab} to F with the intent of it reaching B. If A decreases the amount of bandwidth it allocates for **1**, this will cause B to receive parts of f_{ab} at a slower rate. In turn, since B views A and F as a unit, it will respond by decreasing the amount of bandwidth it allocates for both $\mathbf{5}_{ab}$ and $\mathbf{5}_{fk}$. The decrease in allocation to $\mathbf{5}_{ab}$ will result in A receiving parts of f_{ab} at a slower rate as a consequence of its original decrease of **1**.

Connection $\mathbf{2}_{ab}$ - This forwarding connection is used by F to forward portions of f_{ab} that F received from A through **1** to B. If F decreases the amount of bandwidth it allocates for $\mathbf{2}_{ab}$, this will cause B to receive parts of f_{ab} at a slower rate. In turn, since B views A and F as a unit, it will respond by decreasing the amount of bandwidth it allocates for both $\mathbf{5}_{ab}$ and $\mathbf{5}_{fk}$. The decrease in allocation to $\mathbf{5}_{fk}$ will result in F receiving parts of f_{fk} at a slower rate as a consequence of its original decrease of $\mathbf{2}_{ab}$.

Connection $\mathbf{2}_{fk}$, 4, $\mathbf{5}_{ab}$ - These peer connections are effectively identical to Connection **1**.

Connection 3, $\mathbf{5}_{fk}$, 6 - These forwarding connections are effectively identical to Connection $\mathbf{2}_{ab}$.

Consequently, we see that this standalone circuit does indeed stand alone in the sense that all connections within the circuit do indeed provide incentive for the clients in the circuit to maintain them. Thus, if BitTAKA is run over a network of standalone circuits, the result will be a BitTorrent-esque peer-to-peer file sharing network in which there exists incentive to upload and forward files. In the following section, we address how standalone circuits can be formed in the first place, and how the different components of a standalone circuit learn of their standalone circuit.

2.4.3 Forming Circuits

In this section, we discuss the three primary methods used by the tracker to form standalone circuits.

Bootstrapping

The bootstrapping method for forming standalone circuits is named after the fact that it does not

require any sort of preexisting network of peers, and thus can be used at any time. The tracker "bootstraps" a standalone circuit by first waiting for two pairs of peers, A, B and C, D where A and B want to share the same file, and C and D want to share some other file (not necessarily different), to contact it, as can be seen in Figure 7.

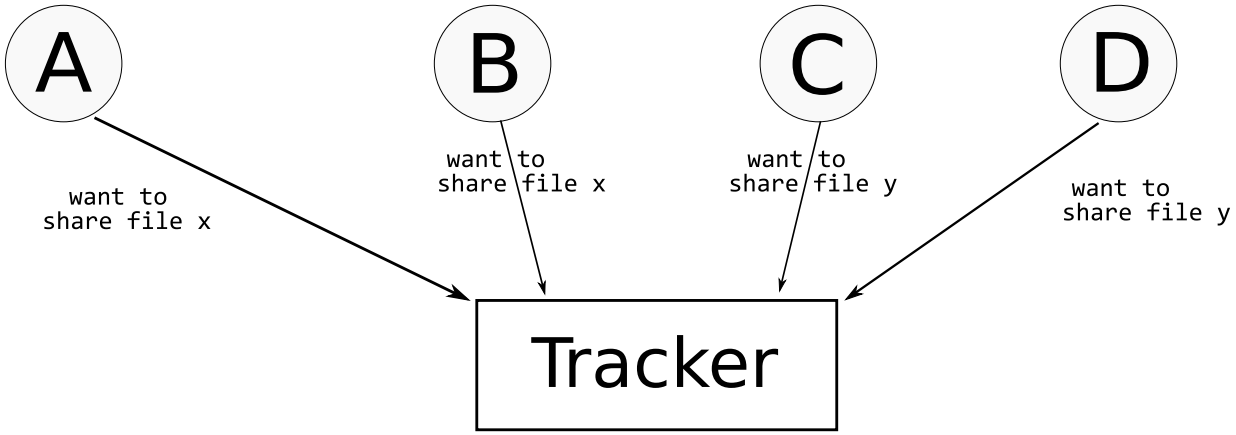

Figure 7: Step 1 in the bootstrapping process - clients A, B, C, D contact the tracker informing it of the file they each want to share.

Once this happens, the tracker sends each client the information necessary for forming a circuit. Each client is told which IP addresses it needs to form connections which, the UIDs of the client they will become peers with, and whether or not they need to act as a forwarder as well, as can be seen in Figure 8.

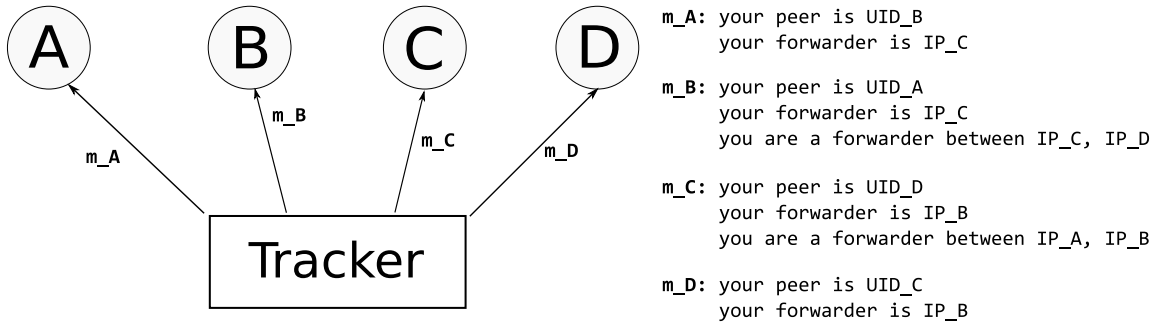

Figure 8: Step 2 in the bootstrapping process - the tracker sends clients A, B, C, D the information necessary to form a circuit, as well as their assignments within the circuit.

The end result is the circuit seen in Figure 9.

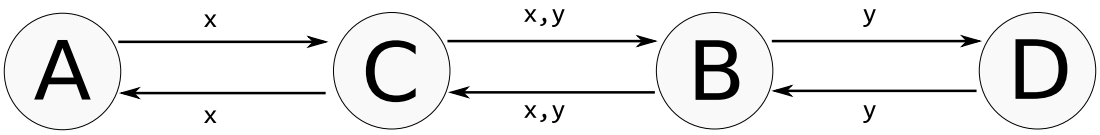

Figure 9: The result of the bootstrapping process.

Thus, we can see that only two pairs of peers are needed to establish a standalone circuit.

Integration

In the original BitTorrent protocol, if possible, users will connect with multiple other users for the same file. Fortunately, it is easy to do this while still using standalone circuits, using the integration method for forming circuits. The integration method can be used with clients that are already involved in some circuits, but would like to form more, presumably to download their desired file at a faster rate. The integration method might also be used for a client that wishes to switch circuits, perhaps because the peer/forwarder pair in a current circuit is misbehaving, or simply because the client would like to try to find a new, better connection.

First, a client that is already part of one or more circuits contacts the tracker, as can be seen in Figure 10.

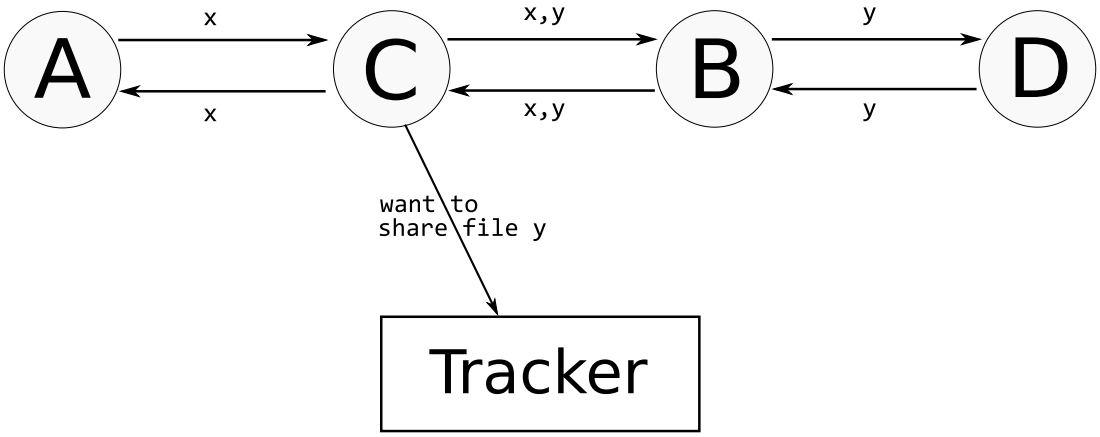

Figure 10: Step 1 in the integration process - client C informs the tracker that it would like to find another peer to share file y with.

Upon receiving this message, the tracker then finds another client in the network also seeking to share the file in question, and sends the relevant clients the information necessary for forming a circuit, as can be seen in Figure 11.

Note that the tracker makes use of its up-to-date network graph in both finding another client

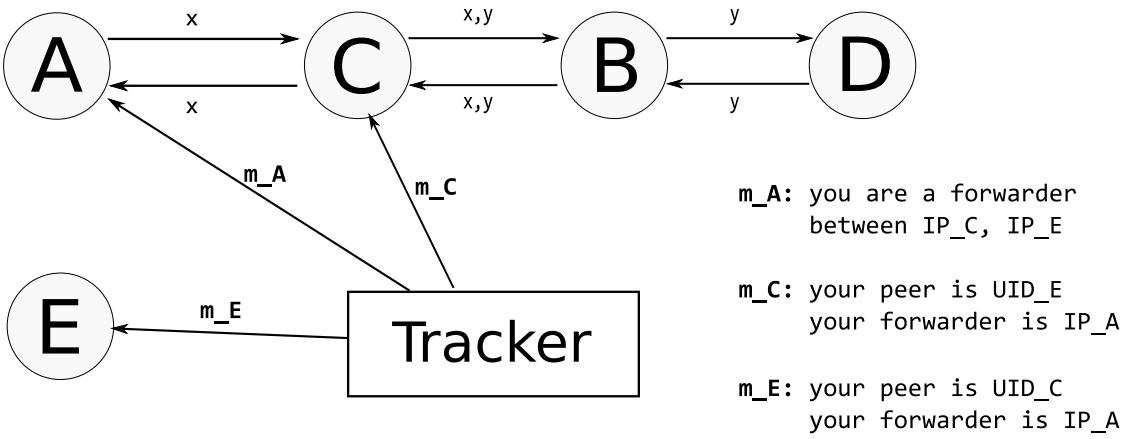

Figure 11: Step 2 in the integration process - the tracker finds client E who is also sharing file y, and sends clients A, C, E the information necessary to form a circuit, as well as their assignments within the circuit.

that is sharing the same file as C, and in knowing that C is already part of a circuit involving A. Additionally, C could have included information about all of the circuits it is currently a part of when originally contacting the tracker about finding a new peer. The end result is seen in Figure 12.

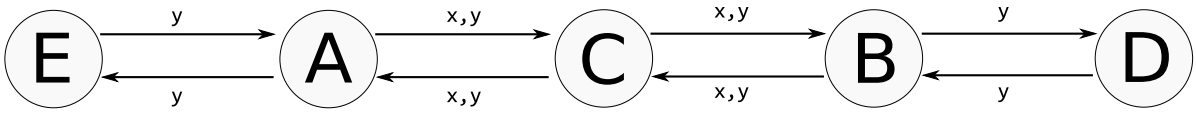

Figure 12: The result of the integration process.

Note that the tracker took advantage of the fact that C was already acting as a forwarder for one of its neighbors, namely A. Consequently, A would have incentive to properly forward traffic between C and its new peer E.

2.4.4 Addressing The Weakest Link Effect

Having traffic sent between peers in BitTAKA going through a forwarder causes an overall decrease in performance for three reasons. The first reason is that circuits obviously double the number of hops required to send anything to its desired destination. The second reason is that the bandwidth of clients is used not only to upload and download files that the client is interested in, but also to forward files that the client is not interested in. The third reason is that the effective bandwidth between two peers is limited by the weakest link out of the four connections involved in a simple peer-forwarder-peer circuit.

While the first two reasons cannot be easily avoided while preserving the original intentions of BitTAKA, the third can be alleviated. Namely, the most unfortunate situation is one where two

peers both have excellent bandwidth that they are willing to use to share a file, but the forwarder between them has poor bandwidth. In such a scenario, it would be nice if the peers could switch to another forwarder with better bandwidth, or add a new forwarder. Here, we list two such techniques to improve the likelihood of such a course of action.

Bandwidth Declaration

One way two peers are able to realize that the forwarder between them may be acting as a bottleneck for their bandwidth is to send each other a message (through their forwarder) about their respective bandwidths. If both peers determine that a different or additional forwarder may be useful, they agree to contact the tracker to establish this new circuit.

Optimistic Forwarder Unchoke

In BitTorrent, users contact the tracker and receive a list of potential peers, but do not connect to all of them simultaneously. Instead, users occasionally perform what is known as an **optimistic unchoke**, where they switch or add a new connection in hopes of finding a connection with better bandwidth.

Recall from Section 2.4.1 that in a given circuit between peers A and B with forwarder F, A views its peer and forwarder as a unit (F, B). Thus, BitTAKA also naturally incorporates optimistic unchoking between peers by having A apply the same principles used in BitTorrent to the unit (F, B). However, in the case where two peers have good bandwidth but the forwarder between them does not, this is not the desireable course of action, because it breaks up the pairing of two peers with good bandwidth. Instead, peers also occasionally perform an **optimistic forwarder unchoke**, where one or both peers in a circuit contact the tracker and request that the tracker help find them a new forwarder.

2.5 File Encryption

Thus far, the focus in this section has been on preventing peers from discovering each other's IP addresses. However, since a forwarder learns the IP addresses of both peers in a circuit, it is necessary that a forwarder does not learn anything about the contents of the file it is sharing. In addition, malicious forwarders could hypothetically observe and alter the contents of the messages and files they forward. In this section, we describe how we apply cryptographic techniques in order to remedy these issues.

2.5.1 Public Key Encryption

In this section, we describe how BitTAKA applies public key encryption to hide information about the header and contents of traffic between peers being sent through a forwarder.

Assume each client X has a public key PK_X and a private key SK_X (how exactly keys are distributed is discussed in Section 2.5.2). Then, given a circuit connecting peers A and B through

forwarder F, messages m from A are encrypted using the public key of B resulting in

$$m' = E_{PK_B}(m).$$

Next, m' is sent to F, which then forwards it to IP_B. The receiving peer B can decrypt the content of the message using its private key SK_B and read the message. This is similar to techniques used in onion-routing protocols. However, unlike TOR, BitTAKA does not apply encryption at every step. Instead, BitTAKA's technique is closer to applying end-to-end encryption to files as used in the BitTorrent protocol. As for the actual encryption scheme, RSA appears to be a good candidate for use.

2.5.2 Message Authentication Codes

In addition, messages are also paired with message authentication codes to prevent altering of their content by a malicious forwarder. Since MACs are widely used and there exists much literature on them, we do not explore this part of the protocol in detail here.

2.5.3 Key Distribution

Distribution of keys is performed by the tracker. When a client first connects to the BitTAKA network (for instance by starting up his/her BitTAKA software), the tracker generates a public and private key for that client. The private key is sent to the client, while the public key is made available for all clients in the network to view. Note that communication with the tracker can be secured by using a known public key of the tracker and a secret session key.

Diffie-Hellman key distribution scheme is applicable to the BitTAKA protocol [6]. The important assumption of this scheme is that the key distribution center is trusted, and the generated prime number and the primitive root are trusted to be correctly generated. In this scheme, peers of a circuit agree on a secret symmetric key which is known only to them and not their corresponding forwarder.

3 Security Analysis

In this section we analyze security of our system against multiple attacks. One of the crucial invariants of our system is the following:

Invariant 1. At no point in time, can a peer in the network learn the mapping between the names and the actual network addresses of other members except himself.

This invariant is too strict to be guaranteed by any real-life mixnet-like system. Therefore, we discuss different variants of relaxation of the invariant.

3.1 Attacks

We present a list of possible attacks on the system and discuss their practicality and security risks to our system. Since we cannot hope to cover all possible attacks described in literature [7], we only show the most significant attacks, roughly in the order of the magnitude of their risks.

3.1.1 Malicious Tracker

Our system does not withstand attacks on the tracker in the sense that if the tracker has full access to the tracker or its state, or can eavesdrop all communication between the tracker and the peers, our invariants are violated. Moreover, the system relies on the assumption that the tracker is trustworthy. For example, if the tracker generates names that are not truly random and can be statistically correlated to addresses, then an adversary can still learn something about the addresses from the names. In section 5, we consider the possibility of making the tracker decentralized and how it effects trustworthiness requirement.

3.1.2 Collusion of Peers

A set of different attacks on the system uses a group of malicious peers with coordinated actions. For example, in the simple system we presented if the adversary has both one of the end points of the circuit as well as all intermediate forwarders in between, then the group as a whole can learn the mapping between the name and address of the other end point of the circuit. There are two measures of techniques that help reduce the risks associated with this form of the attack:

1. Forwarders are selected non-deterministically; this way the probability that a newly formed circuit is under control of an adversary is optimal assuming the tracker does not infer which peers are malicious.
2. Circuits can consist of larger number of forwarders. This way the probability that a complete circuit can be taken over by a malicious adversary decreases exponentially.
3. Forwarders can have rankings of trustworthiness. Since a particular form of the attack in this category uses a large group of peers who join the network together in the hope of some of the peers to be connected together through a circuit, we can tolerate this kind of attack by assigning less probability of selection to recent peers rather to long-established forwarders. Ideally, a collection of dedicated trusted forwarders can greatly reduce the risks of the attack, although the existence of such dedicated peers is not required in our design but is certainly useful as a practical way of improving anonymity.

3.1.3 Malicious Forwarders

There are two kinds of attacks that a malicious forwarder might undertake. One is eavesdropping the forwarder messages, and the other kind is misbehaving in the protocol itself, i.e. not forward-

ing traffic. We have already discussed how encryption can be used against the first kind of an attack. Essentially, it will reduce all information that the adversary can collect to the IP address of the source of the message as well as the corresponding destination address assigned by the tracker. While there is an opportunity of correlating emmitted messages with names inside the messages, it would require significant proportion of the forwarders under the control of a single malicious entity.

The other attack is violating protocol specification for forwarding in order to "free-ride", ie. get data without contributing to the network, or otherwise degrade performance. We have mentioned in the protocol discussion a technique of pairing forwarders so as to enforce incentives for forwarder via distributing blame between the peer and the forwarder it is connected to. This scheme ensures robustness of the system even with some forwarders misbehaving.

3.1.4 Malicious Peers

Our tit-for-tat algorithm mimics the Pareto efficiency of the original BitTorrent protocol. The choking algorithm ensures that peers who download excessively and do not reciprocate are "choked", ie. stopped from continuing to download, thus providing a strong incentive for peers to continue to upload to their own peers.

4 Simulation

One of the primary motivations of our project was to understand the trade-off between anonymity in large and efficient file-sharing protocols and their performance. We have constructed a simulation framework modelling a network of peers operating BitTAKA protocol. In this section, we present the description of the simulation enviornment and discuss results of various experiments comparing performance of BitTAKA protocol relative to BitTorrent protocol.

4.1 Model

The framework is implemented fully in Java and uses non-blocking I/O for network communication. Each peer and the tracker are represented by threads in JVM and their associated TCP sockets. The network delay is emulated artificially by delaying packets by random amount of time approximately equal to real-life Internet latencies. Since the experiments were primarily meant to be run on a single machine, the files transmitted over the network are artificial. They are partitioned into pieces and chunks in the same way as in BitTorrent protocol, but the actual data within chunks is truncated. In addition, a set of time-outs and sleep instructions govern peers so that the performance of the system is network-bound rather than having a bottleneck in CPU or memory.

The results are produced on the following machine:

- Intel Core2 Duo CPU E6750 2.66GHz

- 2 GB of RAM
- Ubuntu Gutsy 7.10, kernel version 2.6.22-14 server

4.2 Protocol Implementation

We implemented the essential subset of BitTorrent protocol specification[5]. Since BitTAKA can be thought of as an extension to the underlying network layer of BitTorrent, the same algorithms are shared between both protocols. These include algorithms of piece selection, peer selection, and choking. The key differences between our implementation and the real-world BitTorrent clients are outlined below:

- Our files consist of 20 pieces, and each piece is partitioned into 32 chunks. The size of the chunk is chosen to be 100 bytes so as to be sufficiently small to fit into a single TCP packet. While the size of the file does not fully reflect the real-life files, the number of messages needed to be sent between peers is an accurate representation of the number of messages needed to be sent in real-life circumstances.
- The desired pieces are selected at random. In the original specification of BitTorrent, there is a multitude of strategies applied in selecting next pieces to be downloaded such as rarest-first, and end of game strategy. Our implementation primarily uses random selection of a piece.
- The upload rate of peers is measured as a moving average of the number of messages sent in the last 20 seconds, similar to the original design decision of evaluating transfer rates although potentially differing from other implementations[1].
- Communication with tracker differs considerably from BitTorrent since it is not done over HTTP. However, the sessions with the tracker are established infrequently, and whether it is done over UDP or TCP did not significantly alter performance in out simulation environment.
- Individual constants are chosen so as to fit the scale of our simulation set-up. For example, in the original implement it is suggested that the tracker reports 30 neighbor peers at the announce request. We reduced this number to less than 10 since we do not maintain as many peers at the same time as encountered in the real-life setting. Some other parameters, such as the number of parallel download sessions, and the number of parallel chunk requests are also changed.

Some of the steps of BitTAKA protocol are omitted in our implementation. Encryption on both end points of the circuit is not used. In addition, circuits used for active download sessions are not reset periodically and maintained for the duration of the whole session. However, circuits are assigned by the tracker and chosen uniformly at random, so that the duty of forwarding packets is still distributed uniformly among peers.

The simulation was primarily focused on evaluating network utilization and performance drawback due to excessive number of messages used for forwarding. Therefore, some other perfor-

mance factors such as public key encryption impact on CPU or the task of coordinating peers on the tracker side are ignored in our performance statistics.

4.3 Results

We outline some of the experiments performed with the help of our simulation framework. These experiments attempted to reflect real-life usage of the protocol for file-sharing.

4.3.1 Uniformly Distributed File

In this experiment we set up a network with peers who have a fairly random but uniform distribution of pieces between them and observe how long it takes for a peer to be able to complete his own file. This gives us a direct view of how much less efficient the BitTAKA forwarding scheme is for a given peer at a given file completion level.

As we can see in Figure 13, forwarding has a significant effect on overall network efficiency, slowing down peer downloads roughly 30% in the 20 peer case. The effect is much more pronounced in the 50 peer case, since forwarding severely restricts the parallelism that BitTorrent provides; although there is larger variance under BitTAKA a few "stragglers" take considerably longer than the rest to complete the file.

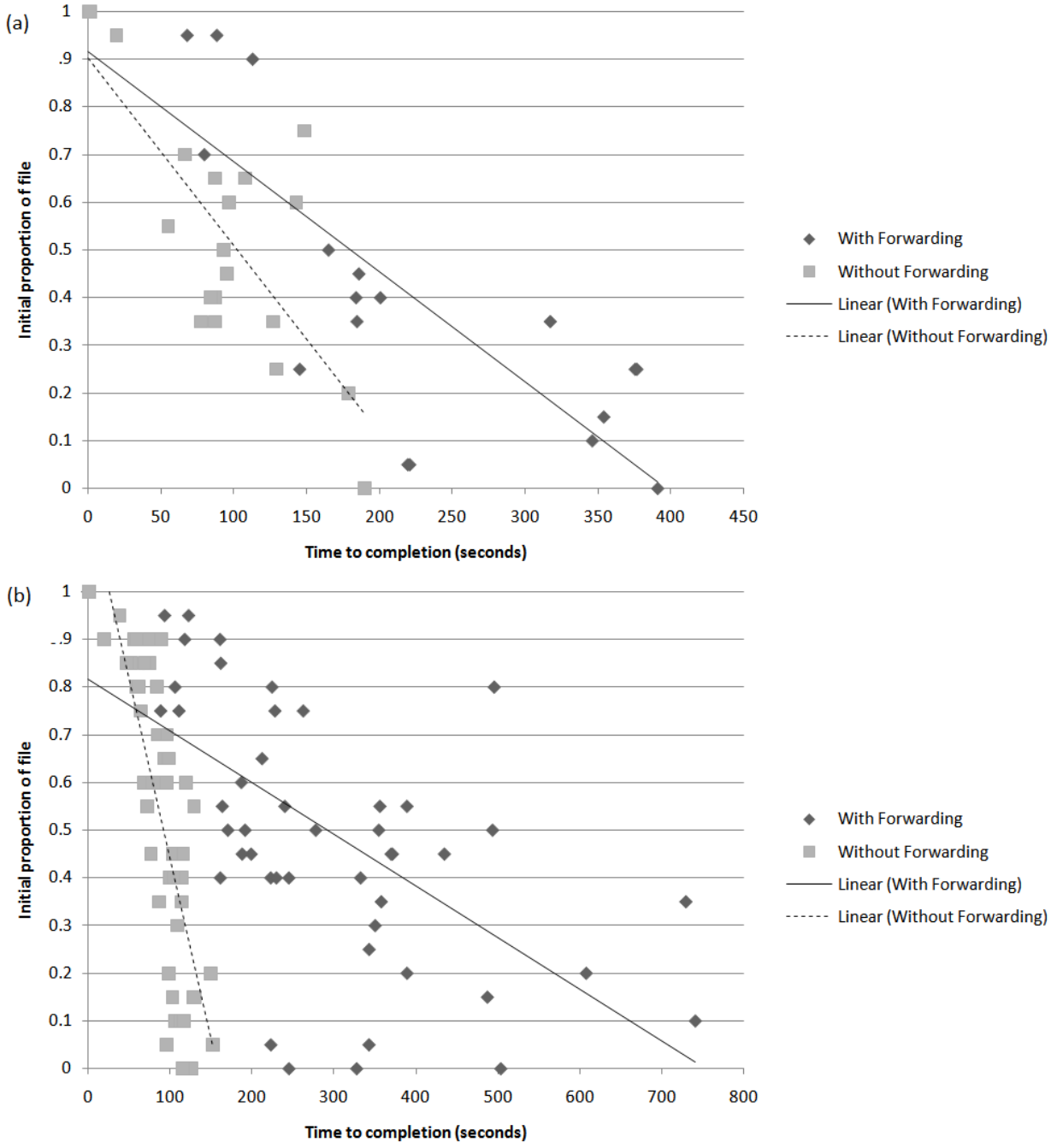

Figure 13: Completion time for a given starting file fraction in a BitTAKA network with randomly distributed chunks for (a) 20 peers, and (b) 50 peers

4.3.2 Seeders and Leechers

In this setting, the network consists of 45 leechers, peers with no pieces initially, one seeder which owns the whole file, and four additional peer which only contain randomly selected parts of the file. This simulation reflects behavior of the network at the initial release of the file. Since in this test, the bandwidth of the seeders becomes the crucial resource on the network, the experiment shows the behavior of the system in bandwidth-bound scenario.

As before, we performed the same experiment with BitTorrent protocol and BitTAKA protocol. The results are shown in figure 14.

BitTorrent	
average time to completion (milliseconds)	267,219.37
standard deviation (milliseconds)	118,919.83
BitTAKA	
average time to completion (milliseconds)	765,356.29
standard deviation (milliseconds)	429,483.39

Figure 14: Performance of protocols with 45 leechers, 1 full seeder, and 4 partial seeders

The large variance is most likely due to some peers receiving far more than their fair share of forwarding traffic and thus become slowed down, and thus there are peers that get significantly slowed down as well since they are dependent on this fowarder for a certain period of time. This leads to a larger variance than in BitTorrent.

4.4 Conclusion

BitTAKA, like other anonymous and pseudonymous peer-to-peer networks, offers a significant tradeoff between anonymity and performance when compared to standard BitTorrent.

5 Potential Improvements

5.1 Decentralized Tracking

One possible improvement to the security of the system is to decentralize tracking responsibilities. This means that the querying and circuit building functionality of the tracker has to be supported in a distributed way, and to be able to do this we will need to relax the restriction that users do not learn the UID-IP map. Instead of having our condition of anonymity, we will instead focus on the condition of *deniability* which we define as follows.

Definition 1. (Deniability) Peers who forward traffic to you don't know what is being forwarded due to the encryption, and since there is more than one file on the network they can deny knowledge of transmitting copyrighted information, they were just forwarding traffic, which has no legal precedent for prosecution.

5.1.1 Finding pieces

To achieve this, we leverage our existing BitTAKA protocol, noting that it can be used as an overlay network. We can then construct a distributed data store, by using a distributed hash table (DHT) similar to that used in BitTorrent itself as an extension, to help peers find pieces of files that they want [8]. Instead of storing IP addresses however, we store the names of peers that have pieces we want and use the bitTAKA overlay network to contact them. Pieces of a file are stored on users computers in a cache in encrypted form, and the decryption keys can be distributed separately over another anonymous network such as Tor. Thus users who are discovered with pieces of the file on their computer can again claim deniability. This is a strategy similar to that seen in the Japanese P2P program Perfect Dark [9].

Now since bitTAKA hides the IP address origin of forwarded packets, we inherit our notion of security by treating our overlay network as a mixnet [10]. The only way to defeat end-to-end security is to compromise all the forwarders in the chain from sender to receiver. By randomizing the choice of forwarders, and by increasing the minimum number of forwarders in the chain, we can make this extremely difficult for an attacker.

5.1.2 Finding routes

To be able to get pieces from one peer to another, we need to construct a chain of forwarders between them. We can do this by using a simple algorithm where clients on the network advertise routes to other peers. Those who get these advertisements propagate them as their own route for the target. This algorithm does not scale very well, however since to contact one particular node you might end up having to do a broadcast search of the entire network. We can borrow the algorithm used by I2P to build its routes, by creating a chain of forwarders called a "tunnel" of a certain minimum length on each side, and then informing everyone of the end of the tunnel as a way to contact them as seen in Figure 15.

To avoid getting routes with bad forwarders, peers can publish profiles about other peers with data collected such as speed, capacity and failure rate [11]. Data can be signed by the peer to ensure integrity. In the I2P instance the routes are asymmetric ie. incoming and outgoing tunnels are not the same, but for bitTAKA to work correctly we will need them to be to enforce tit-for-tat; this can cause a reduction in the anonymity of the network and to mitigate this effect we can accelerate the time T between switching tunnels.

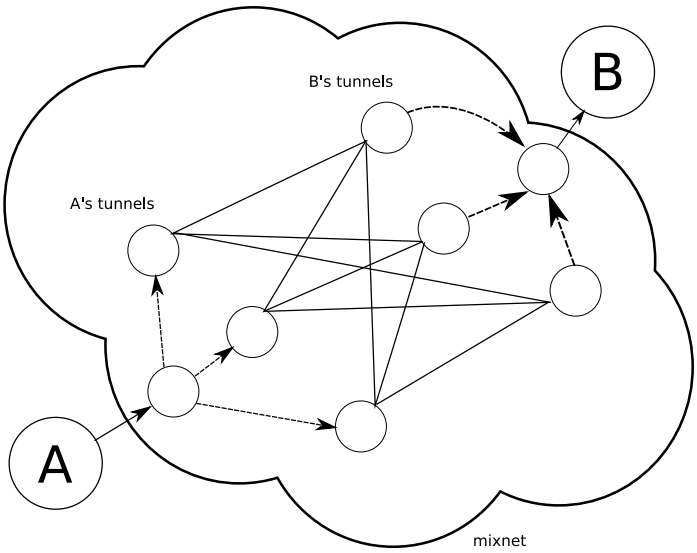

Figure 15: Tunnels can be constructed and used to provide routing.

6 Related Work

6.1 TOR

TOR has been increasingly used to anonymize the BitTorrent traffic. Azureus is one of the popular clients that implement the SOCKS interface for the TOR network. However, as mentioned earlier, the TOR traffic is slow and prevents the users from utilizing all of the efficiency BitTorrent can provide. In addition, the ports known to belong to BitTorrent are blocked by the TOR since volunteers are unwilling to donate their limited available bandwidth for filesharing purposes.

6.2 I2P

I2P is an overlay network for anonymous peer-to-peer communication designed to work for any traditional internet application. I2P uses what is called garlic routing to route messages. The idea is that the sender decides on the four nodes the message is going to be transmitted through, puts one or more messages in a four-layered encryption and sends it to be delivered to its destination. The main difference between TOR and I2P is that TOR has a centralized view to manage the network while I2P uses distributed database and peer selection. However, TOR is known to be more efficient with memory usage and the TOR nodes have less bandwidth overhead. Also the I2P developers claim that their implementation has not been completed and may have some bugs, and thus should be used for anonymity yet. However, the BitTorrent client, Azureus, and some other peer-to-peer file sharing protocols such as Imule, I2phex use I2P network [12].

6.3 Freenet

Freenet [13]is another peer-to-peer file sharing protocol that tries to achieve anonymity. The network functions like a distributed hash table, where each file is associated with a key and is distributed over many connected nodes. The files stored on individual nodes are encrypted and the claim is that it is hard for anyone to discover who is storing what file. This is true even for the peers that have a file on their computers. Files may even be broken into pieces and distributed over many nodes, which makes the anonymity claim even more solid. The routing is done using friend-to-friend topology, where a user only talks to the peers he trusts, that is, friends, and talks to other peers through the friends. However, this is a significantly different system from the BitTorrent and BitTAKA protocols as there is no incentive for the peers to upload data as they download it. In fact, this system was not built to surve the purpose of efficient peer-to-peer file sharing but rather the freedom of speech, which the authors believe to be possible only through a perfect anonymity.

6.4 Closed Source: Winny and Share

There exist a few closed-source implementations of anonymous peer-to-peer file sharing protocols as well. These include Winny, which is claimed to be inspired by the Freenet design, and Share, which is a continuation of Winny after the arrest of the Winny's main developer. Both of these protocols were developed in Japan, where the IP-layer networking has much better performance than that in Europe and North America. This is usually attributed to geographical proximity and better network infrastructure. Thus, these protocols focused strongly on anonymity rather than performance.

7 Conclusion

The BitTorrent protocol has become one of the most popular peer-to-peer file sharing protocols on the internet, however, it does not provide anonymity. In this paper, we introduced the BitTAKA protocol, which is fundamentally similar to BitTorrent protocol but uses onion routes to anonymize the traffic. Having simulated our network versus the conventional BitTorrent protocol, we measured only a 30% loss in network efficiency with a moderate number of peers. However as the number of peers increases, regular BitTorrent's better parallelization allows it to scale better, and our relative network efficiency decreases.

References

[1] Bram Cohen. Incentives build robustness in bittorrent, 2003. Available from World Wide Web: `citeseer.ist.psu.edu/cohen03incentives.html`.

[2] Leslie Ellis. Bittorrents swarms have a deadly bite on broadband nets, 2004. Available from World Wide Web: http://www.multichannel.com/article/CA6332098. [Online; accessed 14-May-2008].

[3] R. Dingledine, N. Mathewson, and P. Syverson. Tor: The second-generation onion router, 2004. Available from World Wide Web: citeseer.ist.psu.edu/dingledine04tor.html.

[4] Nikita Borisov Robin Snader. A tune-up for tor: Improving security and performance in the tor network. 2008.

[5] Bram Cohen. Bittorrent protocol specification, 2008. Available from World Wide Web: http://www.bittorrent.org/beps/bep_0003.html. [Online; accessed 14-May-2008].

[6] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–654, 1976. Available from World Wide Web: citeseer.ist.psu.edu/diffie76new.html.

[7] Jean-François Raymond. Traffic analysis: Protocols, attacks, design issues and open problems. In H. Federrath, editor, *Designing Privacy Enhancing Technologies: Proceedings of International Workshop on Design Issues in Anonymity and Unobservability*, volume 2009 of *LNCS*, pages 10–29. Springer-Verlag, 2001. Available from World Wide Web: citeseer.ist.psu.edu/454354.html.

[8] Andrew Loewenstern. Bittorrent dht protocol, 2008. Available from World Wide Web: http://www.bittorrent.org/beps/bep_0005.html. [Online; accessed 14-May-2008].

[9] Wikipedia. Perfect dark (p2p) — wikipedia, the free encyclopedia, 2008. Available from World Wide Web: http://en.wikipedia.org/w/index.php?title=Perfect_Dark_%28P2P%29&oldid=211069285. [Online; accessed 14-May-2008].

[10] David L. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Commun. ACM*, 24(2):84–90, 1981.

[11] J Random. I2p: How peer selection works, 2008. Available from World Wide Web: http://i2p2.de/how_peerselection. [Online; accessed 14-May-2008].

[12] I2P. I2p compared to other anonymous networks, 2008. Available from World Wide Web: http://www.i2p2.de/how_networkcomparisons. [Online; accessed 14-May-2008].

[13] Ian Clarke, Oskar Sandberg, Brandon Wiley, and Theodore W. Hong. Freenet: A distributed anonymous information storage and retrieval system. *Lecture Notes in Computer Science*, 2009:46–??, 2001. Available from World Wide Web: citeseer.ist.psu.edu/clarke00freenet.html.