Onion Routing and Online Anonymity

Matt Hooks and Jadrian Miles
`mah17@duke.edu, jadrian@cs.duke.edu`
Final paper for CS182S; Patrick Reynolds and Owen Astrachan, Professors
Department of Computer Science, Duke University, Durham, NC, USA

April 30, 2006

Abstract

In this paper we present a technical description of onion routing, an anonymizing protocol that protects users against traffic analysis. Onion routing combines the concepts of proxy redirection and layered mixed-key cryptography to hide the routing of requests from every participant in the network except the originator of the request. We also analyze the performance and potential legal threats against Tor, the second-generation implementation of this system. Usability of the client software enhances the security of the network by providing more cover traffic and more redirection options for all users. Therefore we analyze the usability of both the client and server software currently released by the Tor project. We conclude with potential future developments of the onion routing concept and improvements that could be made to Tor.

1 Motivation

To define the need for anonymizing schemes such as onion routing, one must first understand the concept of traffic analysis. On a public network, there are four sources of information for each connection that can be analyzed: activity on the sender's end, activity on the receiver's end, the actual data sent and the actual data received [10]. These four sources are often used in combination by attackers to determine a model for network traffic. For example, suppose a network is mostly devoid of traffic. Knowing that sender A sent data at time X and receiver B received data at time $X + 2$ is enough to determine that the two parties are communicating. Similarly, suppose you know that sender A sent a particular data packet and receiver B received that same data packet. This would likewise be enough to determine that sender A and receiver B are communcating. This property is know as source-destination linking [10].

If one wishes to have a truly anonymous communication, one must obfuscate all four sources of information. An attacker with effectively placed sniffers will be able to tell if there is activity on both the sender's and receiver's ends. Onion routing protects against this by making the first and last hops of a connection into and out of a network where the attacker cannot follow (i.e. into a trusted onion routing network). If you combine this strategy with constant activity on either end of the communication (whether it is real activity or simply "padding"), it becomes difficult to link source and destination. You are less at risk for this type of attack if the attacker does not know where you are sending information since he won't necessarily have a sniffer set up at the end location.

The third and fourth sources of information are easy to obfuscate using public key encryption. However, using one layer of encryption is not enough. Although this would hide the actual message that is being sent, the data packets would leave the sender and arrive at the receiver in the same state and the communication would still be vulnerable to source-destination linking and therefore to traffic analysis. Onion routing solves this problem by using multiple layers of encryption to establish a link between two parties, which also helps preserve perfect forward secrecy. Onion routing is still vulnerable to some types of attacks, as we will examine later in the paper.

1.1 Why Be Anonymous?

The question remains: who needs anonymous communication? The obvious answer is that criminals do. Criminals want anonymity so that they can continue to break the law without being jailed. However, the definition of a criminal differs greatly from country to country and from culture to culture. A student in the United States curious about the current political state of Tibet will have a much easier time obtaining information than a student in China, who might be arrested for treason. Similarly, an Iraqi hacker who breaches the security of an American server might be seen as a hero in Iraq, but would be decried as a terrorist in America. There is a gray area here and it is filled with people who deserve anonymity in a global, humanitarian sense, but are not allowed it by the laws of their country. This group includes "people who live in countries where it can be dangerous to speak out against the government or express ones religious or political beliefs." [6] Onion routing is one such anonymizing scheme that could give these people the freedom of speech they desire.

For those criminals who do not fall into this gray area, however, the point is moot. Criminals, for the most part, already have a very practical anonymity. If a criminal's identity is easily known then he runs a much greater risk of being caught. Anonymizing schemes are a way for the average Joe to take back some of his own anonymity by offering selective privacy. They allow him to keep his identity protected not only from criminals, but also from marketing firms and junk mailers and also to keep himself protected from potential harrassment. For example, say Jane has a particularly embarassing disease. When she goes into an anonymous chat room where she can discuss treatment the disease with other sufferers, she doesn't expect her connection to be traced and her identity revealed [9]. Onion routing helps protect this kind of anonymity. Average Joes using onion routing also provide greater anonymity for users with a need for more anonymous anonymity by providing cover traffic or "'normal' traffic in which confidential traffic can be hidden." [6]

Governments need anonymity for many reasons. For example, the CIA and FBI do not always want to reveal that they are conducting intelligence gathering. Using anonymizing schemes is one way that these two organizations can keep that information secret without the need for extra funding. During summits and negotiations, governments could use anonymizing schemes to keep channels open, but also keep secret the amount and direction of communication. E-voting could be implemented in a way that mimics and improves upon the authenticity and anonymity of paper ballots. As far as the military and law enforcement agencies are concerned, anonymizing schemes allow comprimised open networks to be negotiated safely and they similarly allow anonymous tips to be truly anonymous.

Behind all these examples lies a more basic problem that can be fixed by anonymizing schemes. The route a packet takes and the fact that packets are being sent at all provide more information to an attacker than most people realize. The procurement patterns of research, tools, weapons, information, etc. can and often are just as valuable to governments, businesses, scientists, and consumers as the actual items. To take a concrete example, say an unscrupulous stock broker is wondering what stocks his competitor is thinking about buying. Using the power of the internet, he finds out that his competitor is making connections to the website of a small firm in Texas on a regular basis. From this information, the unscrupulous broker will be able to infer that his competitor got a hot tip on their stock and he will be able to interfere with his competitor's plans. If the competitor had used anonymizing schemes, it would have been much more difficult for the unscrupulous broker to leech off of his research [9].

2 Technical Description

2.1 Onion Routing

Goldschlag, Reed, and Syverson's 1996 paper introduced onion routing as a means to establish an anonymously redirected encrypted path through a network, with full control of routing decisions and identity disclosure left in the hands of the sender [4]. Before introducing the onion mechanism, we first examine naïve *proxy redirection* and *public-key cryptography*, which provide, respectively, limited endpoint anonymity and robust message secrecy. Onion routing combines these two concepts to full routing anonymity and message secrecy.

2.1.1 Proxy Redirection

An Internet proxy is a server that forwards data from a sender to a specified recipient and also forwards responses from the recipient back to the sender. A proxy may be used to provide limited anonymity to the sender; when the receiver checks the identity of the source of the data, it sees the proxy rather than the original sender. The proxy is a single point of failure in this system, and so an attacker may fully compromise the anonymity of the sender by gaining control of the proxy or simply analyzing traffic in and out of it. Proxy redirection may be made more robust by "bouncing" the message through multiple proxies instead of only one. If the chain of redirections is fixed, however, anonymity may once again be compromised by attacking only the proxy "closest" to the sender—the first one on the chain from sender to receiver. If the chain of redirections is chosen anew for each message, the full path must be spelled out to each proxy so that it can forward routing instructions to the next proxy on the path. In this case, compromising any one of the proxies reveals the identity of the sender. Onion routing is an adapted form of multi-proxy redirection that keeps routing information secret from every proxy along the path of communication through the application of public-key cryptography.

2.1.2 Public-Key Cryptography

In a public-key cryptographic system, every participant has a pair of "keys" that can be used to encrypt and decrypt messages: one public and one private. Public keys are easily available from online listings called key servers. The public and private keys are mathematically related to each other so that they form each others' unique cryptographic inverse: a message encrypted with a public key may only be decrypted by its corresponding private key, and vice versa. If Alice wants to send Bob a message, she encrypts the message with Bob's public key:*

$$C = M * K_u[B] \tag{1}$$

To read the message, Bob decrypts with his private key, which only he knows:

$$\begin{aligned} C * K_v[B] &= M * K_u[B] * K_v[B] \\ &= M * 1 = M \end{aligned} \tag{2}$$

In this way, Alice is guaranteed that only Bob may read her message [8].

Since Bob's public key is freely available, however, he does not know by default that Alice is who she claims she is. In order to prove her identity, Alice may "sign" the message with an appended signature encrypted with her own *private* key, and then encrypt the entire package with Bob's public key:

$$C = (M + (S * K_v[A])) * K_u[B] \tag{3}$$

When Bob decrypts the data, he finds the message and the encrypted signature data, which he then decrypts with Alice's public key:

$$\begin{aligned} C * K_v[B] &= M + (S * K_v[A]) \\ (S * K_v[A]) * K_u[A] &= S \end{aligned} \tag{4}$$

Since only he can decrypt the combined message and signature, Bob knows that the signature has not been compromised, and since Alice's public key decrypts the signature into a standard, recognizable form, he knows that it must have been encrypted with Alice's private key. Since only Alice knows this private key, Bob is now guaranteed of Alice's identity [8].

2.1.3 Symmetric Keys

Because of the mathematical intricacy of public-private key pairs, the encryption and decryption operations involved in public key cryptography are relatively cumbersome. Faster encryption is possible through the use of *symmetric keys*, keys which are each their own cryptographic inverse. If Alice and Bob agree ahead of time on a symmetric key k, they may communicate with each other by encrypting every outgoing message

*In this paper, the star operator $*$ signifies encryption or decryption with a key, while $+$ indicates concatenation of messages. Standard multiplicative and additive notations are used for the respective identities and inverses of these operations.

with k and then decrypting each incoming message with k as well. The agreement stage that precedes this communication is difficult to perform securely online, as an attacker could also discover the key and thereby decrypt any intercepted messages. Symmetric keys are typically quite short relative to the messages they encrypt, however, and therefore it is practical to protect the transmission of symmetric key negotiations using public key cryptography. This negotiation procedure, sometimes called *handshaking*, is quite complex, and a description is beyond the scope of this paper. Once the symmetric key is established by way of costly public-key communications, however, Alice and Bob may be sure that only they know the value of the symmetric key, and may then pass messages between each other using lightweight, but by no means insecure, symmetric key cryptography [8]. Such *mixed-key* schemes are among the most practical and common cryptographic communication systems used online.

2.1.4 Onions

Onion routing, like mixed-key cryptography, uses a bulky but carefully secured initialization procedure to establish fast, lightweight communication via symmetric keys that only the legitimate participants in the communication know [4]. These participants include the sender, the receiver, and any subset of a network of proxy servers that forms a path from the sender to the receiver. Public-key cryptography keeps the global routing information secret from all participants except the sender, and thereby protects the path of communication from discovery by an attacker at any intermediate node. In the most basic implementation of onion routing, the sender (Alice) chooses a list of proxy servers $(P_{\phi(1)}, P_{\phi(2)}, ..., P_{\phi(n)})$ from among the set of all proxies P_i; these servers will form a chain of secure links through which she can redirect her communications. Through the mechanism described below, she sends an onion, constructed with the aid of the routers' public keys, along that path, paving the way for a *virtual circuit* made up of symmetric-key encrypted links between each pair of proxies. The final proxy on the chain communicates with the outside Internet on her behalf, communicating through ordinary channels with the receiver (Bob). Communication along the virtual circuit, once established, is fast, lightweight, and anonymous. All links between proxies and from Alice to $P_{\phi(1)}$ also maintain the secrecy of the data; the final link from $P_{\phi(n)}$ to Bob may or may not be encrypted, as it is identical to the link Alice would ordinarily make to Bob through the Internet.

An onion, O_j, is a datagram intended to be read by only one proxy server, $P_{\phi(j)}$, that establishes a link through the server from the previous node on the path, $P_{\phi(j-1)}$, to the next one, $P_{\phi(j+1)}$. Since $P_{\phi(j)}$ knows the identity of $P_{\phi(j-1)}$ as soon as it receives the onion from it, the datagram needs only to specify the identity—that is, the IP address—of the next node, $I[P_{\phi(j+1)}]$. The onion contains one symmetric key, k_j, to be used to communicate with the previous node and one, k_{j+1}, for communication with the next node. The onion also specifies a timeout period t_j for the connection that will be formed following the onion's path.

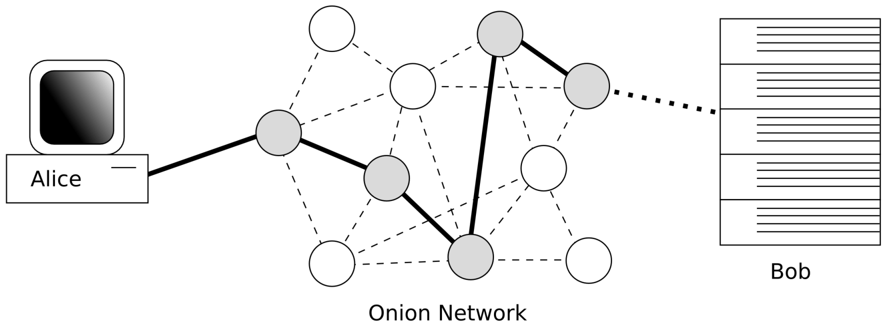


Figure 1: Alice chooses a path through the onion network to communicate with Bob. Darkened nodes are the chosen proxy servers. Thick, solid lines are encrypted links; thick, dotted lines are potentially unencrypted.

The remainder of the onion's contents is a message to be passed on to $P_{\phi(j+1)}$. To make sure that only $P_{\phi(j)}$ may read the contents of the datagram, it is encrypted with the proxy's public key, $K_u[P_{\phi(j)}]$.

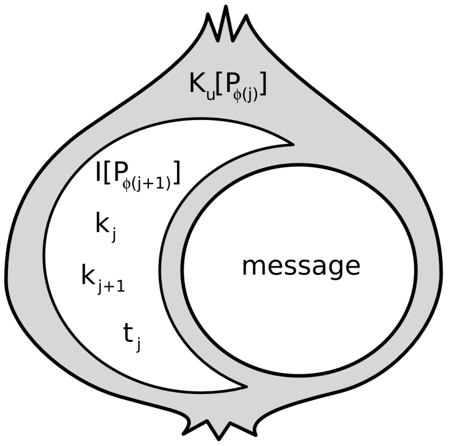


Figure 2: An onion that only $P_{\phi(j)}$ can open.

The power of onion routing comes from a recursive notion of the message contained within an onion [4]. The message within O_1 is another onion; specifically, it is O_2. Each onion's message is the onion for the next node, all the way up to O_n, whose message is empty (or, equivalently, garbage padding data). In order to build the initial onion, O_1, Alice starts with the padding data M, adds the symmetric key k_n that $P_{\phi(n)}$ needs to communicate $P_{\phi(n-1)}$ and the timeout t_n, and encrypts the combined message with $K_u[P_{\phi(n)}]$. This encrypted datagram, O_n, is the message for O_{n-1}, accompanied by $I[P_{\phi(n-2)}]$, k_{n-1}, k_n, and t_{n-1}. Layer upon layer of routing information and public-key encryption is applied in reverse order, until the initial onion is formed.

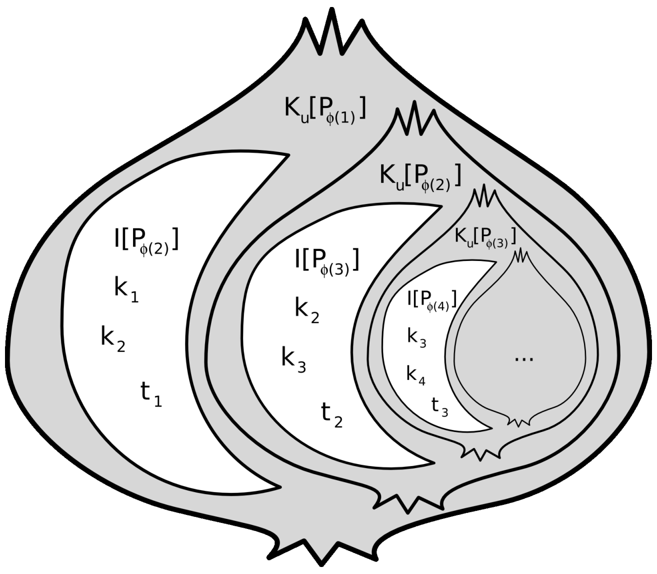


Figure 3: Onion layers: the message for each node's onion is the onion for the next node.

2.1.5 Connecting Through the Onion Network

To establish a connection to Bob through a chain of proxies, Alice builds an onion appropriate to the chain, inserts it into a `create` cell—a message with a header that tells the receiver to build a virtual circuit—and sends it to $P_{\phi(1)}$. $P_{\phi(1)}$ decrypts O_1 using its private key and may now read its personal routing commands. It selects a unique transmission ID for this connection from Alice to $P_{\phi(2)}$, stores O_1 for comparison to future onions as a protection against spoofing attacks, and keeps the connection alive for the timeout period t_1. All forward communication along this connection is decrypted using k_1 and then passed on to $P_{\phi(2)}$, and backward communication, received from $P_{\phi(2)}$, is encrypted with k_1. The message portion of O_1 is O_2, and $P_{\phi(1)}$ forwards this on to $P_{\phi(2)}$ inside a new `create` cell. Each successive proxy peels off a layer of the onion, finding within it a timeout, the identity of the next node, the symmetric keys for communicating with the previous and next nodes, and another onion. It establishes encrypted connections with the previous and next nodes and keeps them alive for the duration of the timeout.

Alice may now communicate with Bob by constructing the message, M, that would ordinarily connect her to Bob through the Internet [4]. She encrypts M with all the symmetric keys $k_n ... k_1$ in reverse order, resulting in a lightweight onion of symmetric-key layers. She sends the onion inside a `data` cell to $P_{\phi(1)}$, who peels off his layer of the onion and passes it on to $P_{\phi(2)}$ in a new `data` cell. Each proxy server along the virtual circuit peels off its layer of encryption, until M eventually arrives at $P_{\phi(n)}$, who then connects to Bob as though she were Alice. Since all the k_js are symmetric keys, this transmission is relatively inexpensive, and each proxy server knows the identities only of its neighbors in the transmission chain. No member of the transmission except Alice has global knowledge of the route; it could even loop through a proxy twice, and the proxy would not know that the two chains of which it was a part were actually portions of the same virtual circuit.

Reverse communication along the virtual circuit, from Bob to Alice, builds up layers of encryption. The response message, R, first travels from Bob to $P_{\phi(n)}$, who applies k_n to R and then passes this onion back to $P_{\phi(n-1)}$. Alice ultimately receives R wrapped in all the symmetric keys $k_n ... k_1$, k_1 being the outermost. She applies the keys in order, ultimately revealing R. In this way, Bob may communicate with Alice without knowing her identity or any of the secret keys.

2.1.6 Reply Onions

Since the virtual circuit has a relatively quick timeout, onion routing provides so-called *reply onions* as a mechanism for Bob to re-establish the virtual circuit without discovering Alice's identity [4]. This is useful, for example, if Alice uses the onion network to send email to Bob; if he wishes to respond after the virtual circuit timeout has passed, he must re-create the original circuit. A reply onion contains all the same information as the original onion that formed the circuit, but with the layers in reverse order. Rather than padding at the center of the onion, which is now wrapped in $K_u[P_{\phi(1)}]$, the reply onion contains routing information and a symmetric key for $P_{\phi(1)}$ to use to connect to Alice. By including this onion in a `create` cell, Bob may now re-establish the virtual circuit to Alice. In order to make this onion available, Alice sends it at the end of her initial transmission of data to Bob.

2.2 Tor

No large-scale deployments of Goldschlag, Reed, and Syverson's original onion routing protocol (henceforth GRS) were ever attempted outside of the research community. In 2004, however, members of the Free Haven project described a second-generation onion routing protocol, called Tor, which has since been deployed in a functional, world-wide anonymizing network [1]. Tor includes numerous refinements of the original onion routing model, the central change being the method for creating virtual circuits.

2.2.1 Building Circuits in Tor

In the GRS model, each layer of the start onion contains extra information for a specific proxy along the circuit. As the onion proceeds along the path, the information is removed, and so the onion becomes smaller. By observing the size of a `create` onion it receives, a proxy node may be able to infer its position in the virtual circuit, information which the designers intended to be unknowable. GRS therefore suggested that each node pad the onion with an amount of garbage data equal to the amount peeled off before passing it on [4]. This requirement is somewhat cumbersome and places restrictions on the types of encryption that may be used in onions, as random appended data must not change the decryption of earlier data, lest the later nodes' onions become corrupted.

Instead of conveying the path of the virtual circuit in a single onion, Tor instead grows the circuit one link at a time, passing a fixed-size `extend` cell to the final node on the circuit by way of lightweight onions. The `extend` cell includes initialization information for a Diffie-Hellman key handshake that depends upon the public key of the node to be added to the circuit. Alice and this node negotiate a symmetric key by way of anonymous messages relayed by the intermediate nodes [1].

GRS also leaves unspecified the nature of the links between proxies on the virtual circuit. Tor uses TLS, a handshaked symmetric-key protocol, to encrypt the connections between each proxy and between Alice and $P_{\phi(1)}$. TLS guarantees *perfect forward secrecy*, the cryptographic property that if an attacker compromises the key for a link at some point in time, he is still unable to decrypt previous messages that traveled along the link.

The `truncate` cell acts as the opposite of `extend`; it specifies to a particular node on the virtual circuit that all further links should be broken, and that subsequent nodes should consider the circuit closed [1]. Alice may send a message to $P_{\phi(j)}$ by building an incomplete lightweight onion with layers $M * k_j * k_{j-1} * ... * k_1$ corresponding only to those nodes up to and including $P_{\phi(j)}$ in the circuit. When a router receives a cell, it removes its layer of the onion and checks the message to see if it makes sense. If not, it must be encrypted and so the router passes it on to the next node in the circuit. If the message does make sense, however, the router follows the instructions contained within it [1].

Alice may recover from a failed node along her circuit by relaying a `truncate` cell to a node before the failed node, and then re-extending the circuit through different nodes by a series of `extend` cells. This is also useful for creating a new circuit without investing the full cost of starting one from scratch; Alice may intentionally truncate the circuit at some node and then re-extend it through a different route. Frequently creating new circuits decreases the ability of an attacker to correlate data entering the network from Alice's computer and data leaving the network through some endpoint proxy. Since the destination of traffic over a virtual circuit depends only on the `data` cells being passed to the endpoint, multiple streams may pass through one virtual circuit, and the creation or destruction of a stream does not imply a corresponding creation or destruction of the circuit. The Tor client software therefore periodically creates new circuits as a background process and sends streams through them as convenient [1].

2.2.2 Directory Servers and Exit Policies

In order to choose the nodes along her virtual circuit, Alice must know, at minimum, which onion routers exist in the network and their public keys. She may also wish to know the reputation of each router or latency properties of its connection. GRS accomplishes the dissemination of this information through *broadcasting*: each node periodically floods the network with a signed status update that clients and other routers can read to determine the network state. This creates significant extra network traffic and makes it difficult to assume consistency of each node's understanding of the network. Tor approaches the problem by instead specifying a universally-known set of proxies as *directory servers*, to whom the routers pass their signed updates directly [1]. The servers compare their respective views of the network, come to a collective decision about the status of each router, and then each aggregates the signed status updates into a single message, which they then sign with their own private key. Clients and routers may then query these servers through onion-routed anonymous requests at an appropriate interval—say, every fifteen minutes.

Since status updates are aggregated and then served directly to clients, they do not have very strict size limitations and may therefore include detailed information about each node. Tor assumes that most proxies are run by individual volunteers, who may be willing to serve only certain types of traffic. Some proxies may even wish only to serve as intermediaries, never connecting on anyone else's behalf to outside servers. In order to maintain a high level of volunteer participation, Tor allows each proxy to specify an *exit policy* that is included in the status update. Clients may then use this information when planning a virtual circuit.

We see, then, that there are four different types of keys used in Tor, each existing over a different time scale [1]:

- **Long-term** public *identity* keys associated with each router, used to sign the router's status updates. As directory servers are also routers, these keys are used to sign entire directory updates.

- **Medium-term** public *onion* keys associated with each router and client, used to wrap up `create` onions and negotiate ephemeral keys between Alice and each proxy.

- **Short-term** symmetric *ephemeral* keys used to pass lightweight onions through the virtual circuit during the circuit's lifetime.

- **Short-term** symmetric *link* keys, negotiated via TLS, that encrypt the links between each pair of nodes.

2.2.3 Hidden Servers and Rendezvous Proxies

Tor's final major refinement over GRS is support for *hidden servers*. A hidden server is a webserver that is hosted on a machine that also acts as an active onion router. Multiple layers of redirection allow onion routing clients to anonymously connect to a hidden server without actually knowing that server's identity [1].

Say Bob hosts a hidden website; users will connect to him by way of an independent router called a *rendevous point* that, because of routing anonymity, knows nothing about the content, source, or destination of the traffic. In order to anonymously agree on a rendezvous point, however, Bob must use other intermediaries to redirect requests to meet at a particular rendezvous point. The first step in connecting users to his hidden website is registering a separate identity key for the site. Bob then picks out a small set of routers to use as *introduction points*, and advertises this list of introduction points on the directory servers, signed with his site's key. He then connects through the onion network to each of these routers and tells them to forward rendezvous requests for the hidden website back to him.

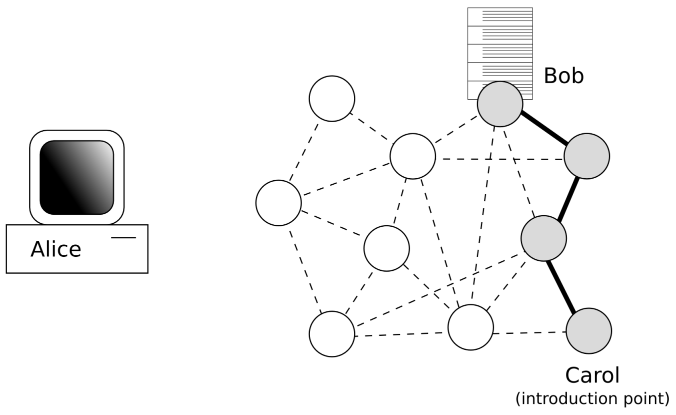


Figure 4: Bob tells his introduction points, "send any rendezvous requests back to me."

If Alice wishes to connect to Bob's website, she picks out a proxy (Dennis) to serve as a rendezvous point, retrieves the list of introduction points from the directory server, and contacts one (Carol) through the onion network. She proposes to Carol that Bob's website rendezvous with her at Dennis's router, sending her a datagram wrapped in Bob's site's public key that includes a random rendezvous key and the first half of a Diffie-Hellman handshake. She may also include a signature if she wishes to prove her identity to Bob.

Carol now forwards the rendezvous request and accompanying data to Bob using a reply onion, allowing her to send him messages without knowing his identity. Alice builds a circuit to Dennis and instructs him to connect his end of her circuit to the end of the expected incoming circuit from Bob, identifiable by the rendezvous key. If Bob decides to allow Alice to connect to his website, he builds a circuit to Dennis, sends him the rendezvous key, and follows up with his half of the Diffie-Hellman handshake. Now Alice and Bob

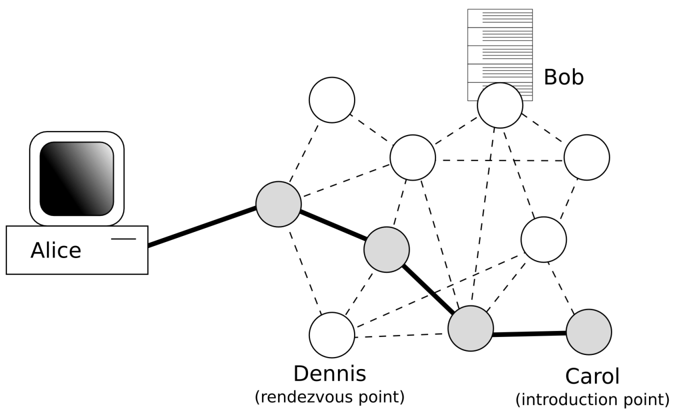


Figure 5: Alice tells Carol, "tell Bob to meet me at Dennis."

have a shared, secret symmetric with which they can encrypt their data streams, so Dennis cannot eavesdrop.

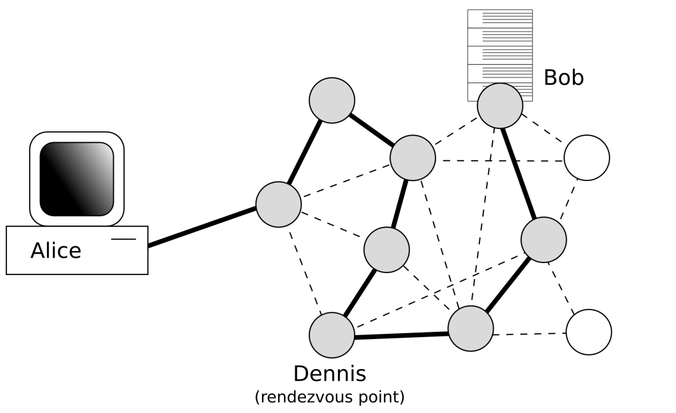


Figure 6: Dennis passes encrypted messages between Alice and Bob.

At the end of this transaction, Alice and Bob know only the identities of the nodes on their respective circuits, up to and including Dennis, and the identities of Carol and all nodes on their respective introductory circuits. Carol knows nothing about either Alice or Bob, nor does Dennis. All other intermediate nodes don't even know that they are part of a hidden server communication; to them, and therefore any eavesdroppers, the traffic looks just like any other passing through the onion network.

2.2.4 The Tor User Client

Apart from the technical refinements outlined above, the Tor project focuses on improving the anonymity properties of its network by recruiting the participation of many onion routers. This is a social, rather than technical, task, but is still essential for the proper functioning of the network. One social approach to assuring user participation is maintaining a good reputation for the network; the Tor administrators prefer that the network be used to anonymize "legitimate" traffic, rather than hiding illegal P2P users or criminals [1]. Of course, there is no way for anyone to know who is using Tor, but informal surveys indicate that a significant number of legitimate users are both accessing the network as clients and participating as proxy routers.

The other way to encourage legitimate traffic is to seamlessly integrate into ordinary users' familiar computer workflow. The Tor project provides a user-friendly, open-source GUI client that acts like a firewall on the user's computer, transparently intercepting outgoing TCP traffic and redirecting it through the onion network. The client software is also onion router server software, and users are encouraged to participate in the network as proxies. They are free to choose whatever exit policy they want, however, or to opt out of being a proxy entirely. Of course, claims of user-friendliness in encryption software must be taken with a grain of salt [11], so we examine the usability of the Tor client in section 3.3 below.

3 Real-World Deployment

Now that we've covered the theoretical description of onion routing and its contemporary realization in Tor, we must examine the interaction of the network with the real world. While Tor is a robust routing anonymity protocol, the Internet and the security community provide fierce challenges to the network's performance and anonymity goals.

3.1 Technical Analysis

3.1.1 Overhead and Latency

The most important thing to remember when speaking about onion routing's latency is the fact that almost none of it comes from encryption overhead. The heavy lifting of virtual circuit creation via public key cryptography is continually performed as a background process. Connections through virtual circuits are encrypted and decrypted using symmetric stream cyphers, which introduce almost no latency and are typical in ordinary secure connections through the Internet. There is significant network latency, on the other hand, inherent in creating a multi-hop proxy redirection route that effectively masks the routing information of a connection. Also, according to the Tor design paper, the "end-to-end congestion control algorithm focuses on protecting volunteer servers from accidental DoS rather than on optimizing performance." [1] This leads us to believe that the congestion control algorithm has room for improvement. According to our own practical tests, an anonymous connection takes approximately 8–10 times longer than an ordinary connection for small files such as web pages. This is confirmed by data gathered by the Tor designers, who observed a direct connection to cnn.com taking 0.3 seconds versus anywhere from 0.4 seconds to 5.3 seconds over an anonymous connection [1]. We did not test downloads of long files, but the design paper shows us that large files have a smaller, but still considerable overhead on them.

In addition to the obvious latency inherent in multiple proxy redirections, the Tor network is spread across the globe, so the exit node for a given virtual circuit may be anywhere in the world. This results in unexpected behaviors like retrieving localized websites (`google.ca` versus `google.com`) when a general site is requested. Most users, especially Americans, tend to browse websites hosted within their own country, and may access cache servers even closer to them geographically than a website's main server. When one requests an American website from an exit node in the Czech Republic, however, it is unsurprising that significant latency results just from retreiving the webpage.

3.1.2 Network Attacks

Low-latency anonymous networks are subject to *timing attacks*, which correlate packets entering one part of the network with those exiting by observing their temporal proximity [1]. Compromising intermediate nodes in the onion routing network gives little information to an attacker (though creative techniques do exist to leverage this information [5]), but observing the connections of a single victim and several target websites may give sufficient information about the victim's access patterns. While Tor is technically low-latency, the wide variation and order-of-magnitude maximum latency through the network indicate that timing attacks may be more difficult in practice than theoretical analysis indicates. End users may also further protect their traffic from analysis by participating as proxy nodes in addition to connecting to the network as clients. To an external observer, it is impossible to distinguish requests originating at the client's computer from those that are simply being bounced. The higher the volume of redirected streams from the network, the greater confusion in which the user's own request may hide.

An attacker may also guess what websites a particular victim of interest is visiting by creating *fingerprints* of target websites [1]. A fingerprint is a log of the packet volume that results from a request to the website. By observing packets incoming to the victim's computer, the attacker may correlate this traffic to a particular website fingerprint, concluding that the user was connected to that website. Once again, acting as a proxy in addition to a client helps protect users from this sort of attack. In addition, modifications to the Tor protocol allowing larger `data` cells and cell padding would reduce the granularity of the attacker's available fingerprints, making comparison less precise.

If an attacker does not have a specific victim in mind but wishes to identify potential victims, he may observe the frequency with which new virtual circuits are created and used [1]. Tor allows each client to specify her own virtual circuit refresh rate (with a typical setting of about a minute), but users who are interested in increased anonymity may choose to create circuits more often. These users would therefore have an externally identifiable network profile that the attacker could use to target them individually. The tradeoff between increased anonymity through fresh virtual circuits and increased targetability is a choice the individual user must make.

An attacker obviously may gain near-absolute control over circuits passing through an onion router if he is able to compromise the router's private onion or identity keys. This is an insecurity inherent in every public-key cryptosystem, and is therefore not an especial concern for Tor. Similarly, compromising TLS link keys gives an attacker temporary control of the link over the lifetime of a single circuit, but perfect forward secrecy and the encryption layers added by other nodes in the circuit limit the damage possible from this control. If the compromised link connects to the last node in the circuit, however, it may be possible to tamper with the content sent back to the initial sender.

A different style of attack is *intersection*, which determines the path of a virtual circuit through compromising individual routers [1]. By bringing down a node via DOS or direct compromise, an attacker may determine which connections pass through the node by observing which endpoints go dead. Additionally, an attacker may degrade the service offered by topologically nearby nodes in the network to force traffic through a desired node. Performing this action sequentially on several different nodes allows the attacker to create an intersection set of endpoints who connected through all the targeted nodes. This information could ultimately lead to a full trace of a virtual circuit through the network. However, since virtual circuits time out after only a short period, this is a difficult attack to execute and offers little lasting information.

3.2 Potential Legal Entanglements

3.2.1 JAP and Germany

After a lengthy search, we did not come upon any legal precedent specific to onion routing. We also did not come upon any legal precedent specific to the practicalities of anonymous use of an open network i.e. the Internet. In [1], there is a small reference to a legal battle between the developers of the Java Anonymous Proxy, also known as JAP, and the German Federal Bureau of Criminal Investigation (FBCI). Although JAP uses an anonymizing scheme based on mixes, we can assume that the legal precedent created by this case will be applicable to any legal issues surrounding onion routing, and more specifically Tor.

The developers of JAP were forced by the FBCI to reveal their "protocol data record" after a search warrant was handed out by the Lower District Court of Frankfurt / Main. Later, the creators of JAP revealed that a "new version of the mix software [included] a function by use of which the access to a particular web server can be recorded." [7] They qualified this by stating that the function recorded only the "IP address of the requesting user, the request, [the] date and [the] time" that pertained to communication with a single IP address. Furthermore, they stated that they would not record any future accesses unless, as was true in this case, they received a binding judicial instruction that pertained to a specific IP address, as they were—and should be—committed to law and order.

This legal battle is quite pertinent because mixes are a precursor and a peer to onion routing. In fact, the developers of JAP recently (May 11, 2005) released a new version of JAP that is compatible with Tor. Obviously, an onion router that is compromised (legally or otherwise) opens the onion routing network up to traffic analysis. Onion routing is, however, less susceptible to this sort of attack for two reasons. First, building onions that travel across the globe and cross multiple jurisdictions makes it harder for legal entities to exert control over an intermediate or over the developers of the software. This is defined as "jurisdictional arbitrage" by the Tor design paper [1]. Since Tor's onion routing network does in fact have nodes all over the globe, it is conceivable, if need be, to build a virtual circuit using nodes that traverse the globe for every connection made. The only price you pay for doing such a thing is increased latency, and that is a small price indeed. The second reason onion routing is less susceptible to a jurisdictional attack is that savvy operators and users running the Tor client can configure their software not to use known compromised nodes. This is similar to creating a soft blacklist, although the effect is less chilling, as users who do not feel threatened by a legal attack will still be able to use the compromised intermediary with impunity.

3.2.2 PGP

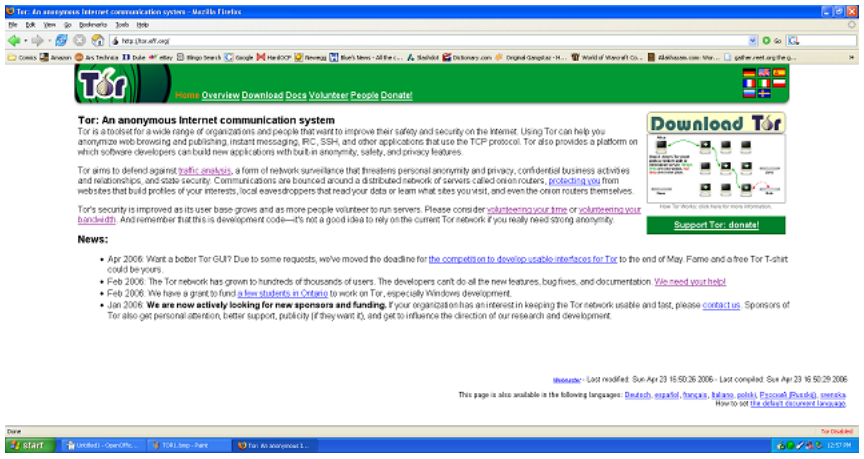
Another legal battle pertinent to onion routing is the three-year investigation of Philip Zimmerman following the freeware release of PGP in 1991. Zimmerman was accused of "violating ITAR, the U.S. government's International Traffic in Arms Regulations" under the assumption that Pretty Good Privacy (PGP), his encryption program, could be classified as a munition [3]. While the case against Zimmerman was dropped in 1996, no reason was given and no change in policy was made. This leaves the door open for the government to go after similar programs and services. Onion routing definitely falls under this umbrella. Although many have speculated on the reason that the Justice Department withdrew their case against Zimmerman (e.g. the NSA found a way to break PGP), it is not far-fetched to think that one of the main reasons they gave up was because PGP was already widespread by the time they noticed it. If this is the case, then onion routing has little to fear. Tor proxy servers are already deployed across the globe. While it is not as popular as PGP—as the perceived need for anonymity is perhaps less than the actual need for encryption—Tor is likely too widely spread for the government to suddenly take offense to it.

3.3 Usability Analysis

One of Tor's selling points is that it is easy to set up and use. By the nature of Tor's design, the more people that use the network, the better the anonymity it provides. This means that the usability of the Tor software is also a security feature. We wanted to conduct a usability analysis to see how easily a non-savvy user concerned with his anonymity and privacy could set up the Tor client, to establish his anonymity, and the Tor server, to improve his anonymity and help others anonymize.

3.3.1 Client

We began our usability analysis by going to Tor's web page (tor.eff.org). The web page is easy to navigate and filled with useful diagrams, documents and instructions.


Tor: An anonymous Internet communication system - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://tor.eff.org/
Home Overview Download Docs Volunteer People Donate!
Download Tor
Support Tor: donate!
Tor Disabled
12:57 PM

Tor: An anonymous Internet communication system
Tor is a toolset for a wide range of organizations and people that want to improve their safety and security on the Internet. Using Tor can help you anonymize web browsing and publishing, instant messaging, IRC, SSH, and other applications that use the TCP protocol. Tor also provides a platform on which software developers can build new applications with built-in anonymity, safety, and privacy features.

Tor aims to defend against traffic analysis, a form of network surveillance that threatens personal anonymity and privacy, confidential business activities and relationships, and state security. Communications are bounced around a distributed network of servers called onion routers, protecting you from websites that build profiles of your interests, local eavesdroppers that read your data or learn what sites you visit, and even the onion routers themselves.

Tor's security is improved as its user base grows and as more people volunteer to run servers. Please consider volunteering your time or volunteering your bandwidth. And remember that this is development code—it's not a good idea to rely on the current Tor network if you really need strong anonymity.

News:

- Apr 2006: Want a better Tor GUI? Due to some requests, we've moved the deadline for the competition to develop usable interfaces for Tor to the end of May. Fame and a free Tor T-shirt could be yours.
- Feb 2006: The Tor network has grown to hundreds of thousands of users. The developers can't do all the new features, bug fixes, and documentation. We need your help!
- Feb 2006: We have a grant to fund a few students in Ontario to work on Tor, especially Windows development.
- Jan 2006: **We are now actively looking for new sponsors and funding.** If your organization has an interest in keeping the Tor network usable and fast, please contact us. Sponsors of Tor also get personal attention, better support, publicity (if they want it), and get to influence the direction of our research and development.

Webmaster - Last modified: Sun Apr 23 16:50:26 2006 - Last compiled: Sun Apr 23 16:50:29 2006

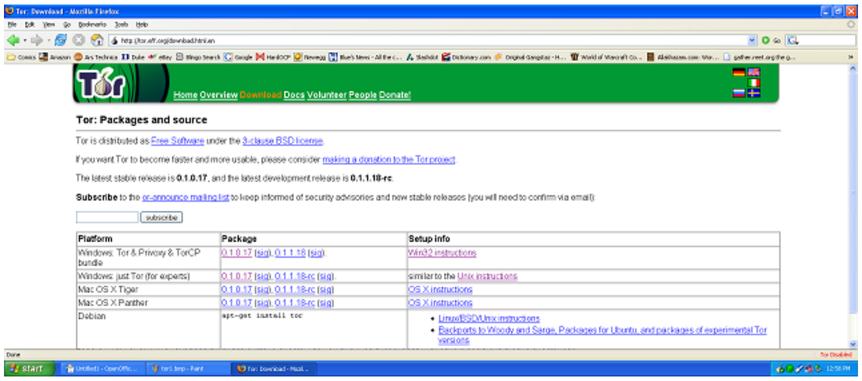
This page is also available in the following languages: Deutsch, español, français, Italiano, polski, Русский (Russkij), svenska.
How to set the default document language.

Figure 7: The Tor website.

Next we went to the download page (figure 8). We decided to use the Microsoft Windows package, since most users will be using Windows. The Windows downloads are separated into two packages based on what software is already installed on the computer. The "novice" package comes with Tor itself, the Tor Control Panel (TorCP), and Privoxy, a header-scrubbing proxy that minimally anonymizes some traffic that Tor can't intercept. The expert package comes only with Tor and allows you to set up your own control panel and proxy. The instructions for setting up the Tor client and server are easily accessible from the download page.

We then began to follow the instructions. The first step is to download and install the software. Installation is simple—if you leave everything at its default, the process is more or less automatic.

Once the software is installed it appears on your taskbar in the system tray. By default it is set up to start up with your computer. Right clicking on either icon allows you to stop and start the services they



Tor: Packages and source

Tor is distributed as Free Software under the 3-clause BSD license.

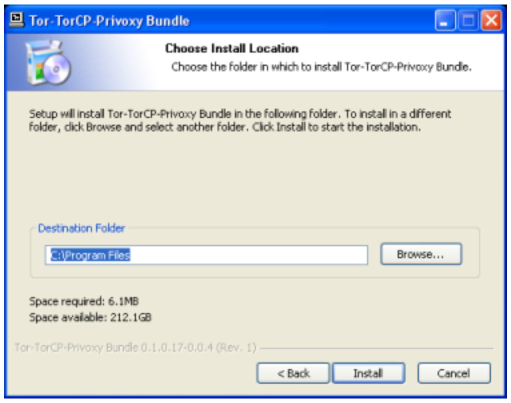
If you want Tor to become faster and more usable, please consider making a donation to the Tor project.

The latest stable release is **0.1.0.17**, and the latest development release is **0.1.1.18-rc**.

Subscribe to the or-announce mailing list to keep informed of security advisories and new stable releases (you will need to confirm via email):

Platform	Package	Setup info
Windows: Tor & Privoxy & TorCP bundle	0.1.0.17 (sig), 0.1.1.18 (sig).	Win32 instructions
Windows: just Tor (for experts)	0.1.0.17 (sig), 0.1.1.18-rc (sig).	similar to the Unix instructions
Mac OS X Tiger	0.1.0.17 (sig), 0.1.1.18-rc (sig)	OS X instructions
Mac OS X Panther	0.1.0.17 (sig), 0.1.1.18-rc (sig)	OS X instructions
Debian	`apt-get install tor`	• Linux/BSD/Unix instructions • Backports to Woody and Sarge, Packages for Ubuntu, and packages of experimental Tor versions

Figure 8: The Tor download page.



Choose Install Location

Choose the folder in which to install Tor-TorCP-Privoxy Bundle.

Setup will install Tor-TorCP-Privoxy Bundle in the following folder. To install in a different folder, click Browse and select another folder. Click Install to start the installation.

Destination Folder: C:\Program Files

Space required: 6.1MB
Space available: 212.1GB

Tor-TorCP-Privoxy Bundle 0.1.0.17-0.0.4 (Rev. 1)

Figure 9: The Tor/Privoxy installer.

provide, as well as access the connection logs.

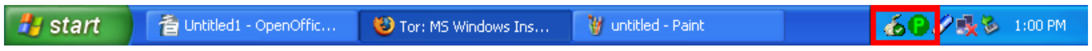
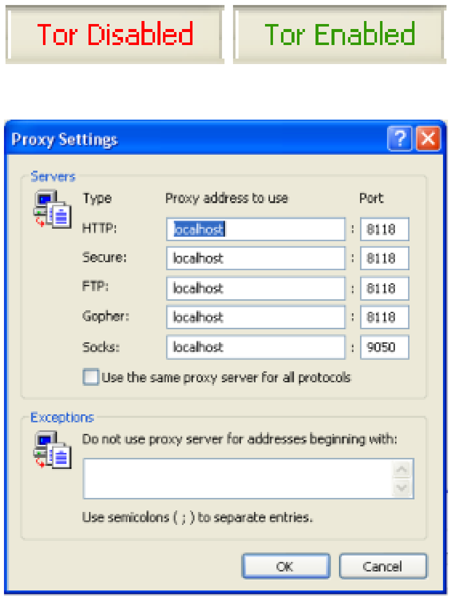


Figure 10: Tor in the Windows system tray.

The next step is to set up your web browser to access Privoxy, the proxy server installed with the Tor package. Firefox has a very easy to use plugin that installs a button in the status bar that allows you to turn TOR on and off with a single click. Internet Explorer doesn't have a Tor plugin, so you have to set up the proxy manually.

Once the proxy is set up, you can browse. Before you do that, the instructions recommend that you verify that Tor is working by visiting a Tor detector (`http://serifos.eecs.harvard.edu/cgi-bin/ipaddr.pl?tor=1`). By meticulously following all the included steps, Tor was working correctly on our machine.

The next step in the instructions is to set up Tor as a server in order to improve your anonymity and to provide more anonymity for others. Before we continued to this step, however, we did a quick test to



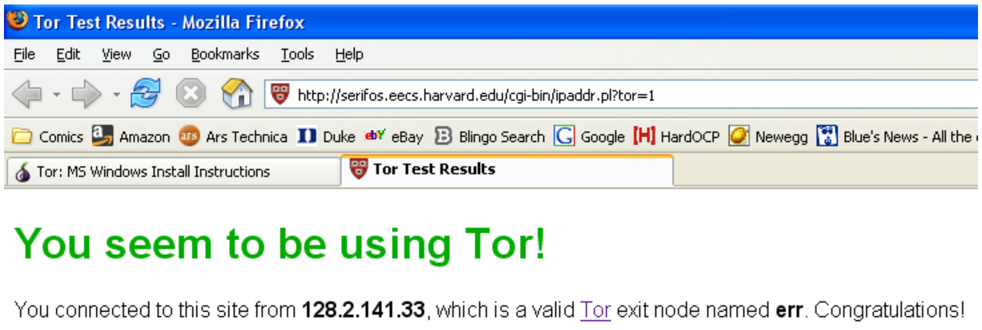
Type	Proxy address to use	Port
HTTP:	localhost	8118
Secure:	localhost	8118
FTP:	localhost	8118
Gopher:	localhost	8118
Socks:	localhost	9050

☐ Use the same proxy server for all protocols

Do not use proxy server for addresses beginning with:

Use semicolons (;) to separate entries.

Figure 11: Top: The Tor Firefox plugin in action. Bottom: Setting up a proxy for Internet Explorer.


Tor Test Results - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://serifos.eecs.harvard.edu/cgi-bin/ipaddr.pl?tor=1
Comics Amazon Ars Technica Duke eBay Blingo Search Google HardOCP Newegg Blue's News - All the
Tor: MS Windows Install Instructions
Tor Test Results

You seem to be using Tor!

You connected to this site from **128.2.141.33**, which is a valid Tor exit node named **err**. Congratulations!

Figure 12: The Tor detector: looks like it worked!

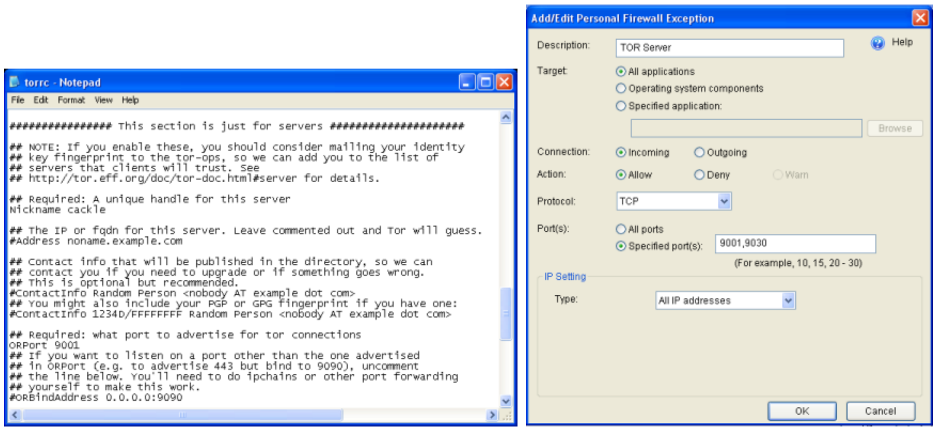
determine the practical, everyday overhead of running a Tor client. This is the main concerns of most non-savvy users who are interested in keeping their identity safe online, but do not have stringent privacy needs. Does using Tor slow down normal web site browsing? By visiting several sites, we were noticed an overhead on the order of 3–10 seconds per connection with Tor versus 1–2 seconds without Tor. Although this may seem restrictive, Tor is very easy to turn off when you don't need it. As is mentioned above, Firefox has a very nifty, easy-to-install plugin that allows Tor to be turned on and off with a single click. In order to switch Internet Explorer back to normal non-anonymous browsing, a user has to go into his preferences and uncheck a single box, so it is only very slightly more complicated.

3.3.2 Server

Once we set up the client, we were interested in determining the ease of setting up a Tor server. The benefit of setting up a server is defined on the Tor website: "Having servers in many different places on the Internet is what makes Tor users secure. You may also get stronger anonymity yourself, since remote sites can't know whether connections originated at your computer or were relayed from others." We followed a link from the client instructions to the server configuration guide. Tor has several features built in that make it convenient to donate your machine as a server without restricting your own bandwidth. They are rate limiting, hibernation, exit policies, and automatic server advertisement based on the bandwidth that you would like to offer.

The first step to setting up Tor as a server is editing the configuration text file. The setup is fairly self-explanatory once you find the right part of the text file, but it requires that you know a little bit about editing configuration files. We uncommented several lines, set up our server's nickname and left everything else that was required at the default setting.

Next, we opened a small hole in our firewall that will forward the two ports that Tor uses. Depending on the firewall being used, this could be very complicated or very simple. Most firewalls have a decent user interface for setting up port forwarding, but this step will leave many novice users scratching their heads.



```
################ This section is just for servers #####################

## NOTE: If you enable these, you should consider mailing your identity
## key fingerprint to the tor-ops, so we can add you to the list of
## servers that clients will trust. See
## http://tor.eff.org/doc/tor-doc.html#server for details.

## Required: A unique handle for this server
Nickname cackle

## The IP or fqdn for this server. Leave commented out and Tor will guess.
#Address noname.example.com

## Contact info that will be published in the directory, so we can
## contact you if you need to upgrade or if something goes wrong.
## This is optional but recommended.
#ContactInfo Random Person <nobody AT example dot com>
## You might also include your PGP or GPG fingerprint if you have one:
#ContactInfo 1234D/FFFFFFFF Random Person <nobody AT example dot com>

## Required: what port to advertise for tor connections
ORPort 9001
## If you want to listen on a port other than the one advertised
## in ORPort (e.g. to advertise 443 but bind to 9090), uncomment
## the line below. You'll need to do ipchains or other port forwarding
## yourself to make this work.
#ORBindAddress 0.0.0.0:9090
```

Description: TOR Server

Target: [✓ All applications] Operating system components Specified application:

Connection: [✓ Incoming] Outgoing

Action: [✓ Allow] Deny Warn

Protocol: TCP

Port(s): All ports [✓ Specified port(s):] 9001,9030

(For example, 10, 15, 20 - 30)

Type: All IP addresses

Figure 13: Configuring the server and letting Tor traffic through the firewall.

Once we set up the config file and the firewall port forwarding, we restarted Tor and waited less than a minute for our log to tell us that we've been recognized as a server. We checked our name against the current server list (`http://belegost.mit.edu/tor/status/authority`) to verify that we are being seen as a server.

The last step is to e-mail the Tor operators at `tor-ops@freehaven.net` to register your server so that your nickname is unique and also so they can contact you if any problems arise on your end or theirs. There are some more complicated optional steps, but if you've gotten this far you're probably a savvy enough user to handle them.

3.3.3 Conclusion

By following all the instructions found on the Tor website, we were able to set up a Tor client and server in less than an hour. We have faith that regular, non-savvy users will also be able to follow these instructions and, at the very least, set up the Tor client. The server is a bit more complicated to set up, but also much less necessary. The conclusion is that Tor is, in fact, very easy to use and its usability is not only a selling point but also a security benefit. We would have liked to see instructions included on the Tor website for setting up common mail clients to route through Tor as well as more abstact tutorials for various programs that do not normally use proxies.

4 Future Applications

- **Usability and Integration:** In our experience, Tor is already quite easy to use. However, integration with a Tor-specific system-wide proxy, with a firewall, or just with a unified GUI would improve the user experience.

- **Attack-Specific Improvements:** Padding is proposed as a deterrent to several types of attacks on Tor users, but it does not seem to have been implemented. Artificially inflating network traffic by circular or otherwise meaningless requests may provide protection against timing attacks, and a larger population of clients and onion routers would make intersection attacks more difficult by providing less meaningful results to the attacker.

- **Garlic Routing** is a theoretical parallelization technique that expands upon the concept of routing onions [2]. A garlic bulb contains multiple cloves, each of which indicates a different branch to be constructed in the virtual circuit. Requests are sent in parallel to multiple exit points in the network, giving higher reliability to the connection and validation of multiple copies of the retrieved data. This technique could be useful in mission-critical applications with highly unstable networks, such as anonymous communication in a war zone.

- **BitTorrent and Garlic Routing:** We also propose borrowing the distributed file retrieval of Bit-Torrent and applying it to the network redundancy of garlic routing. Rather than asking the parallel connections to retrieve the same data, this scheme would partition the desired file into chunks and ask each exit node to retrieve a different one. Obviously this would introduce enormous traffic saturation to the network, which could potentially help protect against some attacks against anonymity. There are likely more efficient designs possible for anonymous P2P networks, but this idea could merit further exploration.

References

[1] R. Dingledine, N. Mathewson, and P. Syverson. Tor: The second-generation onion router. *USENIX Security Symposium*, 2004.

[2] Michael Freedman. Design and analysis of an anonymous communications channel for the free haven project. Online: http://www.freehaven.net/doc/comm.ps, 2000.

[3] Charles Gimon. Phil zimmerman investigation dropped. Online: http://www.skypoint.com/ gimonca/philzim2.html, 1996.

[4] D. Goldschlag, M. Reed, and P. Syverson. Hiding routing information. *Workshop on Information Hiding*, 1996.

[5] Steven Murdoch and George Danezis. Low-cost traffic analysis of tor. *IEEE Symposium on Security and Privacy*, May 2005.

[6] CMU Usable Privacy and Security Laboratory. Foxtor: A tor design proposal. Online: http://cups.cs.cmu.edu/pubs/TorGUIContest113005.pdf.

[7] Press Release. An.on still guarantees anonymity. Online: http://www.datenschutzzentrum.de/material/themen/presse/anonip_e.htm, Aug 2003.

[8] Bruce Schneier. *Applied Cryptography*. John Wiley & Sons, 2 edition, 1979.

[9] Paul Syverson. Making anonymous communication. Online: http://www.onion-router.net/Publications/Briefing-2004.pdf, 2004.

[10] Paul Syverson, Gene Tsudik, Michael Reed, and Carl Landwehr. Towards an analysis of onion routing security. *Workshop on Design Issues in Anonymity and Unobservability*, Jul 2000.

[11] Alma Whitten and J.D. Tygar. Why johnny can't encrypt: A usability analysis of pgp 5.0. *USENIX Security Symposium*, 1999.