

An internet protocol protecting the interests of its users

Rory Flynn

Newbridge College
County Kildare

An internet protocol protecting the interests of its users

Comments Page

Contents

2	-	Comments Page
3	-	Contents
4	-	Abstract
5	-	Introduction
7	-	Background
9	-	What's wrong with the current internet?
13	-	Existing solutions:
13	-	CJDNS: Caleb James Delisle Network Suite
14	-	Tor: The Onion Router
18	-	I2P: The Invisible Internet Project
19	-	The Phantom Protocol
21	-	New Solution
23	-	Proof of Concept
27	-	In conclusion
28	-	Acknowledgements
29	-	Appendix 1.

Abstract

The Internet is a vital tool which people use every day. Recent world events whereby the internet has been blocked, censored or monitored by authorities have demonstrated that the internet is not as free and open as many feel it to be. For this project, I identified problems with the current internet that allow this to happen. For this I researched reports from China, Libya, Egypt and Syria where the Internet had been censored, cut off, or monitored. I identified these problems as issues of access, privacy and anonymity in each of these countries. I researched existing proposed solutions to these problems and evaluated these solutions. These included CJDNS, Tor, I2P and the Phantom Protocol. My evaluation of these solutions found that each of them targeted a particular problem with the Internet, and that each had their own respective strengths and weaknesses. From this evaluation, I designed a new internet protocol that would protect the interests of its users, by incorporating and improving on the existing solutions I evaluated. The protocol proposes a completely decentralised internet solution, where all communications are encrypted from end-to-end by default, and an anonymity solution is tightly integrated. The impact of such a solution would be far-reaching, particularly in countries where the internet is heavily controlled by authorities. To demonstrate the usefulness of such a protocol, I have developed a proof-of-concept program that adds ad-hoc WiFi peering to one of the solutions I evaluated, CJDNS.

Introduction

My own interest in this topic began when the 'Arab Spring' protests broke out in early 2011. I was listening to a technology news podcast while out walking, and one of the top stories was about how the internet in Egypt had been effectively turned off. It amazed me how effectively it had been done, and how an entire nation had effectively gone dark. I had naively assumed that the internet was decentralised already, and that no-one could shut it down. I began to take an interest in how the internet worked, and learned about the internet's hierarchical structure, and why this structure is necessary. I had always been curious from a technical viewpoint about Eircom's blocking of a certain notorious pirate. Why, for example, did the URL appear to resolve correctly and the page load normally? Why did ICMP packets get intercepted differently to HTTP traffic? How was it technically feasible to effectively hijack communications to the website? I began listening to more podcasts on the topic of vulnerabilities in internet protocols we use every day.

I began bookmarking interesting articles and talks on YouTube about this topic. I decided to undertake a project on this topic as I could see that the subject of internet freedom and privacy would become very important issues in the near future. Even ignoring tin-foil hat conspiracies about government surveillance et cetera, it was clear to me that there was a large movement towards regulation of the internet around the world. Realising that the internet is not the oasis of freedom you believed it was is the first step in going about fixing it so that it can be.

I began following the [/r/darknetplan](#) subreddit on Reddit.com about a year ago. Here, people were posting links to various interesting protocols, papers and ideas on making an internet or 'meshnet' controlled by people. It was here that I learned of the existence of CJDNS, a new project which [/r/darknetplan](#) had gotten behind. I had known about better-known protocols, particularly in the area of anonymity, like Tor and I2P before, but CJDNS was taking a whole new interesting angle on it.

My personal aim in doing this project was not only to find out for myself about how these various protocols worked, but also to raise awareness about them, and why they are needed under the current internet's design.

I set the project aims as follows:

- To identify problems with the the current internet that cause it not to be free and open
- To evaluate existing solutions to these problems
- To design a new protocol based on these solutions

Background - Issues with the current internet

Case Study: China

The 'Great Firewall of China' is the name given to China's internet censorship system. Anything that the country's censors do not agree with is blocked. In Spring 2012, Google conducted tests using its search engine from mainland China. It found that when certain terms were searched for, the TCP connection was reset, and all further TCP connections to Google were blocked for around one and a half minutes ¹.

Internationally popular websites blocked in China include Facebook, Twitter, YouTube, Wordpress and Blogspot. The firewall system also scans for sensitive keywords in certain sites such as Wikipedia and blocks these communications.

Case Study: Egypt

During the Egyptian revolution of 2011 social media played a large part in the organisation of protests and rallies. The government blocked Facebook and Twitter when the protests began. Days later, the government shut down the country's official DNS servers, effectively disabling the average user from visiting any URLs. Soon after, BGP (Border Gateway Protocol) routes were withdrawn for Egyptian ISPs. This effectively shut down the internet in Egypt. One ISP, Noor, used a different international peer, Telecom Italia ², and so remained up for several days before being shut down.

¹ <https://www.youtube.com/watch?v=u2GHyVPoVms> - Observations in mainland China (English) - Google

² <http://www.telegraph.co.uk/news/worldnews/africaandindianocean/egypt/8288163/How-Egypt-shut-down-the-internet.html>

Case Study: Libya

The internet was also shut down in Libya for a period during its civil war. However, a Wired investigation after the fall of Muammar Gaddafi's regime discovered how the internet was being used as an intelligence-gathering method by the regime ³. They discovered an elaborate system, nicknamed 'Eagle', which allowed multiple operators to monitor anyone likely to be against the regime. All of Libya's internet traffic was duplicated and sent to government authorities, where it could be collated against mobile phone and other surveillance. They described how one activist "had her email hacked and her Skype conversations recorded. Both were leaked to state television and broadcast to the nation." The report detailed how Eagle operators could "call up social-network diagrams for the targets they were hunting, with the links between each suspect showing the frequency and type of communication."

Case Study: Syria

The Syrian government has been much more proactive in using the internet to fight back against activists online in the country's ongoing civil war. In March 2012, the Electronic Frontier Foundation reported that the Syrian government was using phishing attacks to gain access to activist's Facebook accounts. ⁴

Engineers at Cloudflare, a DDOS prevention service, which has data centres around the world, reported that over the space of fifteen minutes on the morning of 29th November 2012, all BGP routes were withdrawn from providers upstream from Syria. The Syrian government blamed the outage on "terrorists [targeting] the Internet lines, resulting in some regions being cut off" ⁵, and access was restored on 1st December 2012.

³ http://www.wired.com/threatlevel/2012/05/ff_libya/all/

⁴ <https://www.eff.org/deeplinks/2012/03/pro-syrian-government-hackers-target-syrian-activists-facebook-phishing-attack>

⁵ <http://www.reuters.com/article/2012/11/29/syria-crisis-internet-idUSL5E8MTFMK20121129>

What's wrong with the current internet?

1 Access

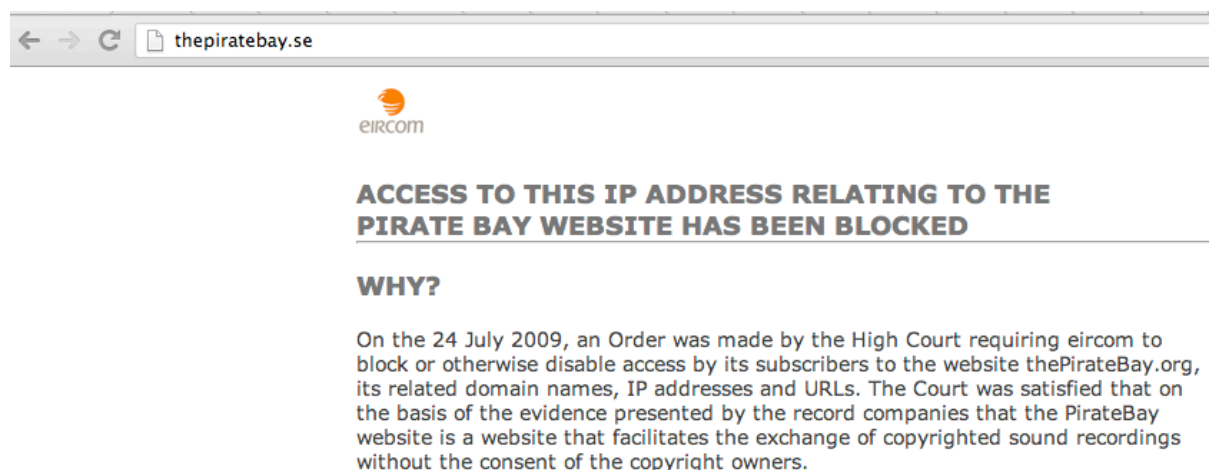
One problem is that of access. Without access to a method of communication, people cannot communicate at all. Restricting people's access to communication is an effective method of censorship.

Problems with the existing internet in relation to access are that:

- Addresses on the internet (IP addresses) are assigned by a central authority (IANA/ ICANN, ISPs on a local level). People then must rely on this central authority to grant them an IP address, which is easily revoked.
- The structure of the current Internet is hierarchical, with large ISPs controlling infrastructure used by many subscribers, allowing the ISP to become a single point of pressure for adversaries wishing to just cut off users' access altogether. (E.g. in Egypt during the Arab Spring)
- This hierarchical system means that it is easy to block access to specific IP addresses if necessary. (Eircom currently does this for all IP addresses relating to the Pirate Bay, pursuant to an order by the High Court:)

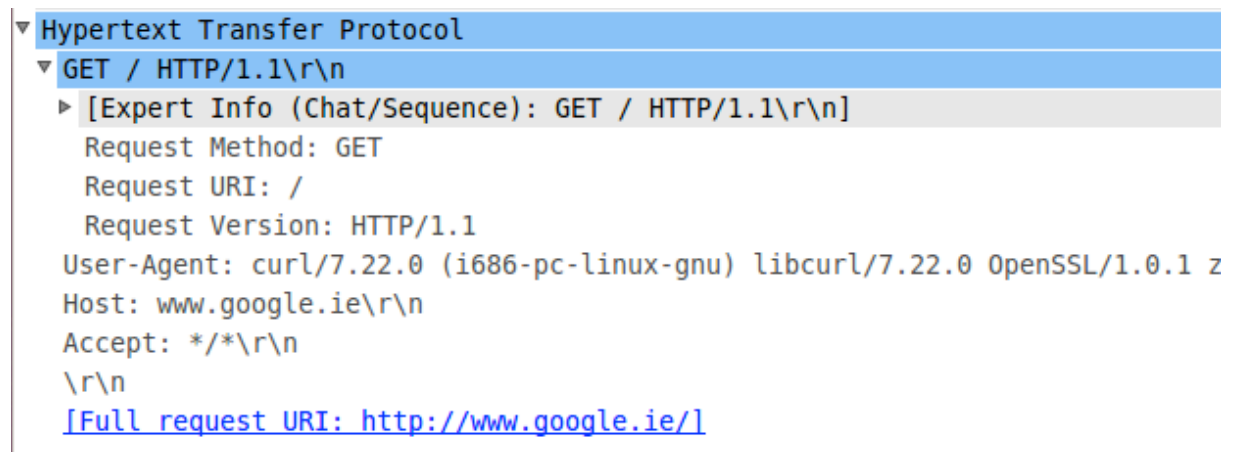
```
PING thepiratebay.se (194.71.107.15): 56 data bytes
```

```
36 bytes from ge0-2.vhg1.cwm.dublin.eircom.net (159.134.124.176):  
Communication prohibited by filter
```



2 Privacy

The Internet protocols we use today were never intended to be used by everyone in its current form. They were used in research networks, between universities, where there was no reason to implement encrypted communications. It sounds ridiculous really, sending packets in the clear (with no encryption) via routers who can read everything in your packet, who also choose the route for that packet, and can easily masquerade as the person you want to communicate with. Here's a typical HTTP request packet (request to view a website):



The screenshot shows a network packet capture interface. The top level is 'Hypertext Transfer Protocol'. Below it is a 'GET / HTTP/1.1\r\n' packet. An expanded view shows the following details:

- [Expert Info (Chat/Sequence): GET / HTTP/1.1\r\n]
- Request Method: GET
- Request URI: /
- Request Version: HTTP/1.1
- User-Agent: curl/7.22.0 (i686-pc-linux-gnu) libcurl/7.22.0 OpenSSL/1.0.1 z
- Host: www.google.ie\r\n
- Accept: */*\r\n
- \r\n
- [Full request URI: <http://www.google.ie/>]

Every single router in between the sender (us) and the recipient (google.ie) can read this, and has enough information to masquerade as google.ie (TCP sequence and ack numbers), if they so wish. They can also maliciously inject HTML into standard HTTP responses from google.ie if needs be.

2 Privacy (ctd.)

At DEFCON 16, the 16th year of the well-established hacker conference, security researchers Anton Kapela and Alex Pilosov successfully manipulated BGP route announcements to intercept the inbound traffic of the conference's backhaul remotely ⁶ (from a server in New York, 2,500 miles from the conference in Las Vegas). They were then able to filter the captured traffic to retrieve personal information of people who used the network, including all unencrypted logins. A similar method was accidentally used by a Pakistani telecom provider when trying to block YouTube only within the country. It announced a /24 of YouTube's addresses, which under BGP, wins as it was a more specific prefix than the /22 YouTube themselves had announced. Most YouTube traffic was thus routed through Pakistan, where it was null-routing (being dropped).

Users need to have privacy by default. While many sites support HTTPS (the HTTP protocol encrypted using TLS or SSL), all other types of internet traffic are not encrypted by default. It is left up to developers of applications to implement their own encryption if they so wish. Encryption is done at the highest layer. Users should not have to worry whether or not a particular application is sending data in the clear over the Internet. It should be implemented by default, at the internet layer.

⁶ <http://www.youtube.com/watch?v=S0BM6aB90n8> - DEFCON 16: Stealing The Internet - A Routed, Wide-area, Man in the Middle Attack

3 Anonymity?

Anonymity is certainly an issue that is up for debate at this point, and a difficult one at that. Journalism has anonymity at its very core, with the protection of sources. Looking at the recent WikiLeaks saga, we can see how anonymity can allow information that is in the public interest to be known to be revealed.

Anonymity can also be used for nefarious purposes, such as defamation, harassment and illegal activities.

Anonymous communications are those where you cannot trace or identify a real life person who has sent the communications. Anonymity is thus very similar to privacy, and by design most anonymous protocols use end-to-end encryption (only the recipients can decrypt each others communications). Anonymity is fast becoming a steadfast feature of the internet, and will eventually become a default feature of it.

Why the current internet is not anonymous:

- IP addresses can be traced back to individuals easily, as address assignment is centralised.

Existing Solutions

CJDNS: Caleb James Delisle Network Suite

CJDNS aims to create a completely decentralised internet, with end-to-end encryption for complete privacy. The CJDNS project is maintained by its creator, Caleb James Delisle, as an open source project on github ⁷. Routing is based on the Kademlia DHT (Distributed Hash Table) and so is completely decentralised. In CJDNS, privacy and decentralisation go hand in hand. Node IDs for the distributed hash table to work are the first 128 bits of the double SHA-512 of the node's public key. 128 bits, of course, is the length of an IPv6 address. Public keys are brute forced until the double SHA-512 address is a valid FC00::/8 private address. In this way, when the rest of the internet switches over to IPv6, CJDNS addresses will not collide with globally routable IPv6 addresses. This means that there are 2^{120} possible CJDNS addresses (the address will always start with the byte 0xFC). Presuming uniform distribution of the SHA-512 hashing algorithm, it would be cryptographically impossible to intentionally create a collision for a specific address.

CJDNS also uses Elliptic Curve Cryptography, which has smaller key sizes for the equivalent in RSA strength. This makes communication using Public-Private Keys more efficient, which means that CJDNS can use them for all communications with a node. It also uses session keys, however, for forward secrecy (if the private keys are compromised).

Strengths of CJDNS:

- **Access:** Completely decentralised, including addresses
- **Privacy:** End-to-end encryption by standard (address is public key fingerprint)
- Appears as a layer 3 TUN interface (can be used with any application supporting IPv6 addresses), extremely versatile

Weaknesses of CJDNS:

- **Anonymity:** Connections are not necessarily anonymous
- Address lookups can be slow due to the distributed design
- No GUI, very developer orientated at this point, also knowledge of basic shell commands required for setup.

⁷ <https://github.com/cjdelisle/cjdns/> CJDNS project Github page

Tor: The Onion Router

Tor is one of the best established anonymity protocols to date, with ~500,000 daily users as of present. Tor uses a concept known as Onion Routing to securely and anonymously route TCP connections across the open Internet.

Tor users download information about participating nodes from directory servers, and then create a circuit through a selection of these to an exit node. These nodes can be selected based on factors such as bandwidth and probability of an adversary controlling all nodes in the circuit.

Exit nodes are those that agree to be the end node in a Tor circuit, that is to have the anonymised node's traffic routed through their own connection over the internet as if it were coming from their own connection.

In order to maintain the originating node's anonymity, Tor uses a concept known as onion routing. Symmetrical session keys are negotiated for each node in the circuit (entry, relay and exit node) using the public keys of these nodes, which enables further encrypted communications to be a lot more efficient than using public key encryption all the time. Currently 2048-bit RSA asymmetric public keys are used just to exchange session keys. The data intended for the last node (e.g. a HTTP request for the exit node to send over the open internet), is encrypted with the negotiated symmetrical session key for the exit node. A label with instructions to the relay (middle) node to forward the message to the exit node is added to the encrypted message and encrypted using the relay node's symmetrical key. The same process is followed for the entrance (first) node, except it is instructed to forward the packet to the relay node. Figure 1, overleaf, illustrates this encryption method.

Tor: The Onion Router (ctd.)

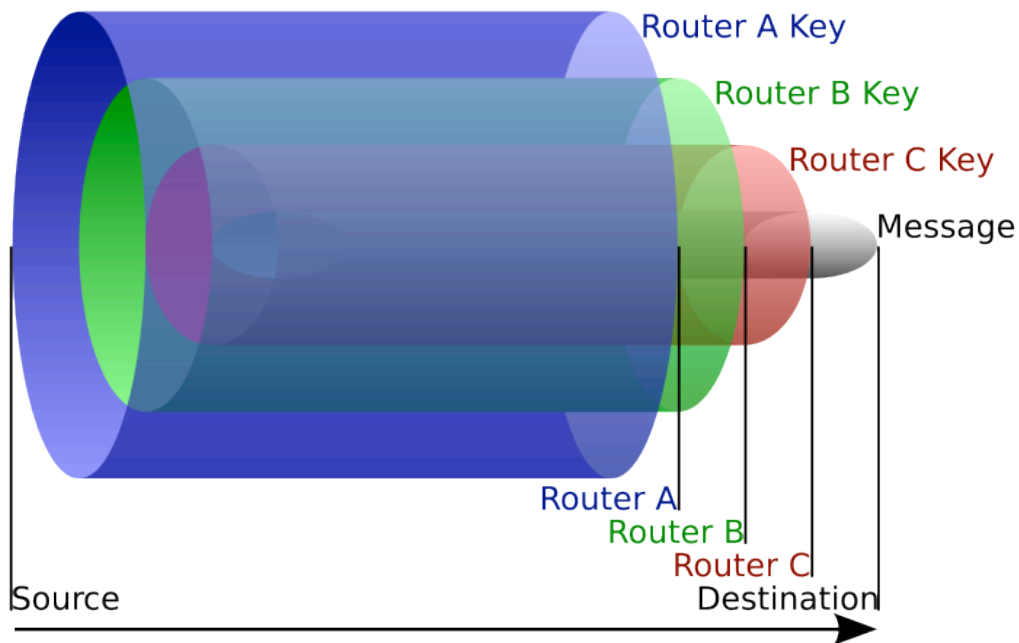


Figure 1. Diagram created by English Wikipedia user HANtwister

The originating node will change its circuit completely about every ten minutes, for added anonymity.

Tor also has a feature known as hidden services. This allows for any server (such as a web or IRC server) to hide its real IP address, remaining anonymous, and to allow Tor users to communicate with the server anonymously through the Tor network. Hidden services then have a 16-character name within the internal .onion top level domain, which is derived from the hidden service's public key. The hidden service publishes its public key, .onion domain and a number of Introduction Points, which are nodes in the Tor network, in a Distributed Hash Table (via a Tor circuit), signed by the hidden service's private key. This means that people can verify that those Introduction Points were in fact chosen by the hidden service, as they can only be decrypted by the hidden service's public key. The hidden service then establishes Tor circuits (anonymous) to the introduction points it has specified in the DHT. The process for a client wishing to connect to a hidden service is this: It looks up the 16-character domain name in the DHT (via a Tor Circuit), and gets the public key for the hidden service, which it can verify, as the 16-character domain is derived from the public key. It then

Tor: The Onion Router (ctd.)

verifies the Introduction Points as being chosen by the hidden service by decrypting their descriptors using the public key it just got.

The client then randomly picks a node in the Tor network to act as a rendezvous point between it and the hidden service, and builds a Tor circuit to it, telling it a one-time secret (random unique data).

The client also builds a Tor circuit to one of the Introduction Points, and asks to send a message to the hidden service containing the client's own public key, one-time secret (random unique data) and the rendezvous point's details to the hidden service, encrypted to its public key. The hidden service then builds a Tor circuit to the rendezvous point and sends it the one-time secret it has received. If the rendezvous point matches the two secrets, communication between the client and the hidden service is allowed through the rendezvous point. The reason for not just using the Introduction Points for communications between client and hidden service is that no single Tor node should be associated with a particular hidden service. Introduction Points do not learn the identity of the hidden service unless they know the .onion name of the service and find that they are acting as introduction point, or are told by someone else that they are listed in the DHT as an introduction point for a particular .onion.

This may all sound confusing, so I have attached diagrams from the Tor Project's website which do a much better job of explaining the hidden service feature. (see Appendix I.).

Tor: The Onion Router (ctd.)

Strengths of Tor:

- **Anonymity:** Well established protocol with proven technique (onion routing)
- Low latency due to (somewhat) centralised directory

Weaknesses of Tor:

- **Access:** Directories are centralised (although protections are in place to prevent manipulation of the directory servers). Relies on existing internet for communication between nodes.
- **Privacy:** The exit node can see the data sent from the originating node in cleartext. While the identity of the originating node remains anonymous, the exit node can still read any traffic sent and received through it. Although hidden services solves this, they are very slow due to the complex set-up process
- Making running an exit node optional means that there is limited bandwidth available. Most Tor exit nodes are run by volunteers.
- Exit nodes are open to prosecution by authorities, as Tor users' actions that route through them appear to come from the exit node's IP address.

I2P: The Invisible Internet Project

I2P is an anonymity network similar to Tor. However, unlike Tor, I2P is completely decentralised, using the Kademlia DHT design. I2P is completely sealed off from the internet (there are no exit nodes). Users advertise their inbound tunnel gateway(s), i.e. end nodes in the DHT. Users then create their own outbound tunnel for the purpose of connecting to other nodes' inbound tunnels. I2P uses 'garlic routing'; an extension of onion routing. The layered encryption process of Tor is still used for all nodes in the tunnel. Multiple messages are encrypted to the 'exit node' (last node in the outbound routing tunnel) which then can carry out instructions in these messages, for example to delay a certain message to the first node in the inbound tunnel of a node the originator wishes to contact by an arbitrary amount of time, to protect against timing attacks. This system makes it more difficult to correlate data coming through the outbound tunnel with data received in the inbound tunnel.

Strengths of I2P:

- **Access:** Completely decentralised, including addressing
- **Anonymity:** Garlic Routing adds further anonymity in addition to Onion routing.
- **Privacy:** End-to-end encryption

Weaknesses of I2P:

- **Access:** Relies on existing Internet for communication between nodes.
- Not very versatile; addresses are internal and developing services to run on the I2P network requires libraries
- Lookups of addresses can be slow due to the decentralised design.
- Not well peer reviewed and limited documentation exists.

The Phantom Protocol

The Phantom Protocol is a design first unveiled by Swedish security researcher Magnus Bråding at DEFCON 16 in 2008. It aimed to be a '*Generic, Decentralized, Unstoppable Anonymity*' solution. Phantom, like I2P, is sealed off from the open internet. Users create routing paths (as in I2P) but they are bi-directional. It uses two separate DHTs, one for AP addresses (anonymous addresses) and one for IP addresses of nodes participating in the Phantom network, with the principle that the two tables stay separate. The DHT is also based on Kademlia.

To create a routing path, the originating node first selects random nodes from the IP address DHT. It creates a tunnel using a technique of X and Y helper nodes in a specific circular messaging arrangement that speeds up the tunnel setup process while maintaining the anonymity of the originating node. The uniqueness of this process means that no node can explicitly say if they are next to the originating node or not. For example no 2 X nodes are beside each other in the path during set up. X nodes are then instructed by another circular message to connect to other X nodes as well the originating node, forming the tunnel to be eventually used. They do this using SSL keys they were sent in the circular message. (See Figure 2, below).

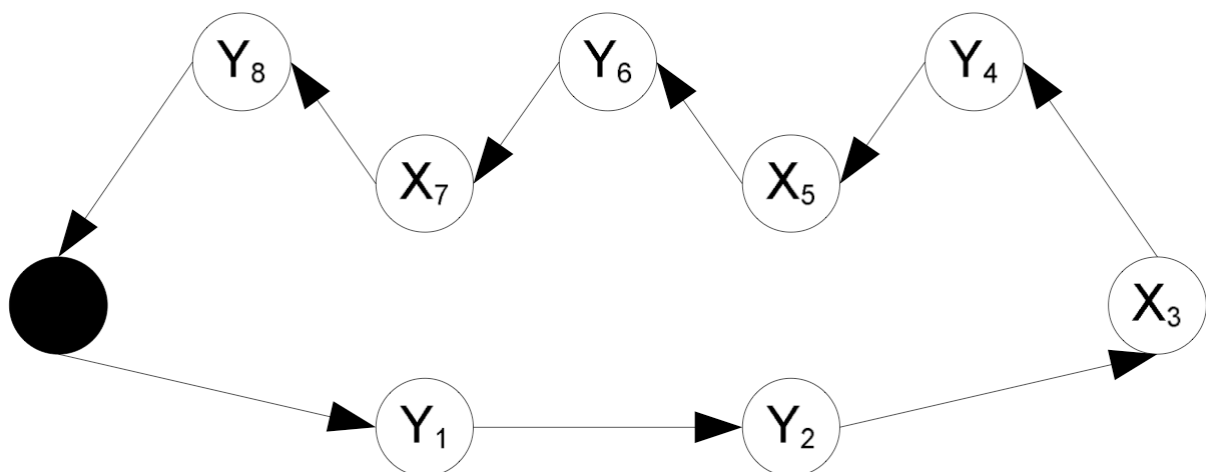


Figure 2. From the Phantom Protocol Whitepaper, Magnus Bråding, 2011

You may notice that at one end of the circular path, there are two Y nodes next to each other. The the Phantom Protocol design paper states that the reason for this is to prevent as much as possible, the linking of the gateway (end or exit) node and the originating node. Thus, the number of Y nodes between the end node and the originating node should be equal to the

Phantom Protocol (ctd.)

number of chosen X nodes minus one, ensuring an appropriate distance is kept between them.

This is much better than the Tor 3- hop circuit extending routine in protecting the identity of the originating node. Once the tunnel has been created, the originating node anonymously publishes its gateway node in the AP DHT, along with its chosen AP address.

To connect to another AP address, the originating node sends the AP address to its gateway node, which looks up the AP address' gateway node. The originator's gateway node then connects to the gateway node of the requested AP address, and initialises a connection. If the connection is accepted, the two nodes can now communicate over this virtual tunnel.

Strengths of Phantom:

- **Anonymity:** Can be better than Tor. Users can specify as many hops as they like. If a server wishes to allow clients to connect anonymously to it, it can create a 0-hop routing tunnel to which clients, who wish to remain anonymous, can connect to with their own n-hop tunnels.
- **Privacy:** End-to-end encryption by default (SSL) + SSL between nodes in the routing path
- **Access:** Overlay network is completely decentralised

Weaknesses of Phantom:

- **Access:** Relies on existing Internet for communications between nodes.
- Can be slower than Tor due to distributed design
- There has been no activity in the development of the project since 2011.

New Solution

All of the existing solutions outlined have their respective strengths and weaknesses. A better solution would use different solutions together to negate these weaknesses.

The aim of this section is to try to specify a new internet protocol that protects the interests of its users. Based on my research, the ideal requirements of such a solution are that:

- 1) The solution is completely decentralised.
- 2) Communications are completely private.
- 3) Communications can be anonymised, if necessary.
- 4) The solution is easy to use, with a GUI preferably.

Incorporating and improving existing solutions

The base solution should be CJDNS. CJDNS is a very versatile platform on which to build. It is completely decentralised, thanks to its adaptation of the Kademlia DHT. It uses fast Elliptic Curve Cryptography for asymmetric keys whose double SHA-512 hashes are used as IPv6 valid private addresses, which means that users' identities can be confirmed easily. CJDNS also kills NAT, finally, which fulfills the original vision of end-to-end connectivity on the Internet (NAT is a way of using a single IP address for multiple users, used currently because we are running out of IPv4 addresses, as there are only 2^{32} of these possible). CJDNS' innovative routing label system (based on MPLS) also makes it very fast at routing packets in the network.

Things that could be improved on in CJDNS include the fact that initialising communication with another address can often be quite slow, due to the nature of the Kademlia findnode process. This can be improved by caching of frequently contacted nodes and their routes.

Another thing that could be improved on in CJDNS is the developer orientated interface that exists at present. The CJDNS developers have made it very easy to build tools for CJDNS with a comprehensive network socket-based API. A GUI can easily be built using Python or other languages.

At present, CJDNS communicates with peers over UDP. In an ideal decentralised network, there should be no need to have a routable IPv4 address. Although there is experimental

layer 2 support (peering via MAC addresses), in practice I have not been able to get this to work. Any solution should make it easy to automatically connect with peers around you via ad-hoc WiFi, the standard for mobile Internet connection.

Anonymity is also not offered by CJDNS. To solve this, Phantom would integrate well here, as it is also based on the Kademlia DHT. The 'IP table' referred to in the design specification of Phantom could easily be integrated into CJDNS' existing modified version of Kademlia. The routing path set up process of Phantom would also benefit, in turn, from not having to use SSL for connections between nodes. (ECC is efficient enough that symmetric keys are only required as session keys for forward secrecy.) A similar brute-force address generation technique could be used to distinguish AP addresses from IP addresses, for example a different /8 prefix.

Integration with existing Internet infrastructure

ISPs have invested heavily in Internet infrastructure. It would be counter-productive to simply abandon this infrastructure. The existing ISP-owned routers of the Internet could be used equally as CJDNS routers. A customer could sign up for service from an ISP, and the ISP would act as a peer to the customer. The ISP is a superpeer, in that its purpose is to remain up 24/7 and to collect as many nodes as it can. In order to become the most trusted and weighted peer, the ISP must be able to provide the customer with better Kademlia lookup responses than other open peers (i.e. neighbours). Therefore it is beneficial for the ISP to catalogue as many nodes as it can and be able to route the customers packets efficiently. In this way, if suddenly an oppressive government decides to shut down all ISPs, their customers will automatically begin trusting and weighting their peers (neighbours) more, as they provide them with faster and more numerous Kademlia lookup responses (non-zero).

Strengths of the new solution:

- **Access:** Completely decentralised, including addressing, with easy peering via ad-hoc WiFi, doesn't rely on existing Internet.
- **Privacy:** End-to-end encryption, with forward secrecy
- **Anonymity:** Phantom available to use

Weaknesses of the new solution:

- The distributed design can be slow to lookup a node. However the suggested solution of using ISPs as 'superpeers', should mitigate this.

Proof of concept

I have developed a proof of concept program to show how a new solution would work. This uses wireless ad-hoc networks to find CJDNS peers and to connect to them. The user simply chooses a channel (frequency) , an essid (effectively a network name) and the code will automatically broadcast the node's CJDNS public key and password for connection to all nodes in the network. In hidden mode, the node does not broadcast anything, but other nodes can connect to it as I will explain further on.

The code itself is written in Python, as it is the language I am most familiar with and a very simple language to use. A proper solution would operate on layer 2, and in the kernel itself.

The proof of concept code, which we shall call DARA, uses the SCAPY python packet capture and injection tool to send and receive broadcast packets. This allows for automatic peering with anyone in WiFi range of the node. The concept uses the CJDNS API to add the nodes in the network's IPv4 addresses as peers. This is basically a hack as I have been unable to get the experimental Layer 2 support (i.e. communication using MAC addresses) in CJDNS to work. It also important to note that the broadcasting of a node's CJDNS peering details effectively makes public the fact that the node is running CJDNS. If necessary, this broadcast feature can be turned off, allowing peering to take place simply by knowing the nodes peering details beforehand - thus CJDNS is encrypting peering requests using the peer's public key. All anyone listening on the network will see is encrypted traffic between two random nodes. It should be noted that in a full implementation of peering via ad-hoc WiFi, some sort of identity verification would need to be in place to prevent false advertisement broadcasts. This way, nodes can develop a better trust mechanism of previously peered with nodes to prevent adversaries from flooding the network with bad nodes.

One method of identity validation would be to require peering detail broadcasts to have a current date and time signed by the private key of the node. This way it can be proven to other nodes in the network that that particular node is actually there. The code itself is a bit messy, using system shell commands to set up the ad-hoc network necessary for communication among nodes.

Selectable options are the WiFi network interface to use, the ESSID (network name), the channel to use and an IPv4 address to use:

```
root@rory-virtual-machine:/home/rory# python DARAscapy.py
WARNING: No route found for IPv6 destination :: (no default route?)
04pbb3np3h7btn7gvqq73utbmrn45n4tcq096tnwyrwdh6zq50.k
1357639240 INFO cjdroute2.c:447 Forking angel to background.
1357639240 DEBUG AngelInit.c:279 Initializing angel with input [7] and output [10]
1357639240 DEBUG AngelInit.c:280 Getting pre-configuration from client
1357639240 DEBUG AngelInit.c:286 Finished getting pre-configuration from client
1357639240 INFO AngelInit.c:318 Initializing core [/home/rory/cjdns/cjdns]
1357639240 DEBUG AngelInit.c:322 Sending pre-configuration to core.
1357639240 INFO PipeInterface.c:372 Creating new PipeInterface [0xb8384d3c]
1357639240 INFO PipeInterface.c:372 Creating new PipeInterface [0xb8b2b4cc]
1357639240 DEBUG PipeInterface.c:234 [0xb8384d3c] Established Synchronization
1357639240 DEBUG PipeInterface.c:234 [0xb8b2b4cc] Established Synchronization
1357639240 INFO Configurator.c:89 Checking authorized password 0.
1357639240 INFO Configurator.c:101 Flushing existing authorized passwords
1357639240 INFO Configurator.c:107 Adding authorized password #[0].
Interface for ad-hoc wifi: wlan1
channel: 9
ESSID: daraineternet
IP: 10.0.0.1
```

An IPv4 address is required to allow CJDNS to connect to peers over UDP; ideally it would just use MAC addresses (layer 2) which are supposed to be globally unique.

The code then begins broadcasting an Ethernet frame (layer 2), to advertise its presence, and how to connect to it: (Its public key, password and UDP port on which CJDNS is running)

173	22.073898	Alfa_58:6d:57	Broadcast	0x0fff	141	Ethernet I
▶ Frame 165: 141 bytes on wire (1128 bits), 141 bytes captured (1128 bits)						
▼ Ethernet II, Src: Alfa_58:6d:57 (00:c0:ca:58:6d:57), Dst: Broadcast (ff:ff:ff:ff:ff:ff)						
▶ Destination: Broadcast (ff:ff:ff:ff:ff:ff)						
▶ Source: Alfa_58:6d:57 (00:c0:ca:58:6d:57)						
Type: Unknown (0x0fff)						
▼ Data (127 bytes)						
Data: 7b274950273a2731302e302e3227202c20277075626c...						
[Length: 127]						
0000	ff ff ff ff ff ff 00 c0	ca 58 6d 57 0f ff 7b 27XmW..{			
0010	49 50 27 3a 27 31 30 2e	30 2e 30 2e 32 27 20 2c	IP':'10. 0.0.2' ,			
0020	20 27 70 75 62 6c 69 63	4b 65 79 27 3a 27 70 63	'public Key':'pc			
0030	6d 70 63 73 73 75 32 76	6c 30 71 62 64 36 35 70	mpcssu2v l0qbd65p			
0040	68 67 34 6e 6c 79 38 63	67 62 30 75 73 33 76 66	hg4nly8c gb0us3vf			
0050	6c 6a 79 7a 68 39 6b 33	73 66 33 6d 36 32 34 70	ljyzh9k3 sf3m624p			
0060	78 30 2e 6b 27 2c 27 70	61 73 73 77 6f 72 64 27	x0.k', 'p assword'			
0070	3a 27 70 61 73 73 77 6f	72 64 27 2c 20 27 70 6f	:'passwo rd', 'po			
0080	72 74 27 20 3a 20 27 38	30 39 30 27 7d	rt' : '8 090'}			

A SCAPY sniff function is then employed to sniff for the broadcast packets of other nodes. When it detects a new node, DARA sends the peers connection details to the CJDNS API, which authenticates with the peer. Above you can see CJDNS's routing table being dumped via the API. The new node is connected to by CJDNS at the line {'error': 'none'}. We can then see the node being added to the CJDNS routing table. (The first entry already in the routing table is our own node.)

```
Starting broadcast...
Scanning for peers
{'routingTable': [{'ip': 'fc3d:2386:c1d8:9ccb:0455:4776:4a29:becd', 'version': 1, 'link': 4294967295L, 'path': '0000.0000.0000.0001'}]}
{'error': 'none'}
WARNING: Mac address to reach destination not found. Using broadcast.
.
Sent 1 packets.
{'routingTable': [{'ip': 'fc3d:2386:c1d8:9ccb:0455:4776:4a29:becd', 'version': 1, 'link': 4294967295L, 'path': '0000.0000.0000.0001'}, {'ip': 'fce8:5e30:18f9:287c:95b5:8bf7:ec20:5e1f', 'version': 1, 'link': 1, 'path': '0000.0000.0000.0015'}]}
WARNING: Mac address to reach destination not found. Using broadcast.
.
Sent 1 packets.
{'routingTable': [{'ip': 'fc3d:2386:c1d8:9ccb:0455:4776:4a29:becd', 'version': 1, 'link': 4294967295L, 'path': '0000.0000.0000.0001'}, {'ip': 'fce8:5e30:18f9:287c:95b5:8bf7:ec20:5e1f', 'version': 1, 'link': 1069437744, 'path': '0000.0000.0000.0015'}]}
WARNING: more Mac address to reach destination not found. Using broadcast.
```

We can then communicate with the node:

```
rory@rory-virtual-machine:~$ ping6 fce8:5e30:18f9:287c:95b5:8bf7:ec20:5e1f
PING fce8:5e30:18f9:287c:95b5:8bf7:ec20:5e1f(fce8:5e30:18f9:287c:95b5:8bf7:ec20:5e1f) 56 data bytes
64 bytes from fce8:5e30:18f9:287c:95b5:8bf7:ec20:5e1f: icmp_seq=1 ttl=64 time=2.78 ms
64 bytes from fce8:5e30:18f9:287c:95b5:8bf7:ec20:5e1f: icmp_seq=2 ttl=64 time=2.02 ms
64 bytes from fce8:5e30:18f9:287c:95b5:8bf7:ec20:5e1f: icmp_seq=3 ttl=64 time=2.29 ms
64 bytes from fce8:5e30:18f9:287c:95b5:8bf7:ec20:5e1f: icmp_seq=4 ttl=64 time=2.30 ms
^C
--- fce8:5e30:18f9:287c:95b5:8bf7:ec20:5e1f ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3005ms
rtt min/avg/max/mdev = 2.020/2.352/2.787/0.282 ms
rory@rory-virtual-machine:~$
```

And the node can communicate with us (other node's screen):

```
rory@ubuntu:~$ ping6 fc3d:2386:c1d8:9ccb:0455:4776:4a29:becd
PING fc3d:2386:c1d8:9ccb:0455:4776:4a29:becd(fc3d:2386:c1d8:9ccb:455:4776:4a29:becd) 56 data bytes
64 bytes from fc3d:2386:c1d8:9ccb:455:4776:4a29:becd: icmp_seq=1 ttl=64 time=2.03 ms
64 bytes from fc3d:2386:c1d8:9ccb:455:4776:4a29:becd: icmp_seq=2 ttl=64 time=1.88 ms
64 bytes from fc3d:2386:c1d8:9ccb:455:4776:4a29:becd: icmp_seq=3 ttl=64 time=2.28 ms
64 bytes from fc3d:2386:c1d8:9ccb:455:4776:4a29:becd: icmp_seq=4 ttl=64 time=1.44 ms
^C
--- fc3d:2386:c1d8:9ccb:0455:4776:4a29:becd ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3005ms
rtt min/avg/max/mdev = 1.441/1.910/2.281/0.309 ms
rory@ubuntu:~$
```

The other node can also communicate with any other nodes the original node has. For example, here the other node is streaming a video from a remote server that is directly connected to the original node:



So the path the video takes is:



All communications are encrypted in this program with the exception of the broadcast packets, which are only required if you want to peer with strangers.

Again, this is a proof of concept, so, for example, it does not protect against ARP (address resolution protocol) poisoning, and adds peers as IP addresses on a first-come-first-served basis. However, ARP poisoning would have limited effect other than to disrupt peering where you do not know the node's peering details already.

In conclusion

The solution I have proposed would offer freedom of access, complete privacy and anonymity online if it were implemented. The proof of concept ad-hoc WiFi program shows the practical application of the solution for the end user.

In my opinion the impact of such a solution would be far-reaching for freedom of communication.

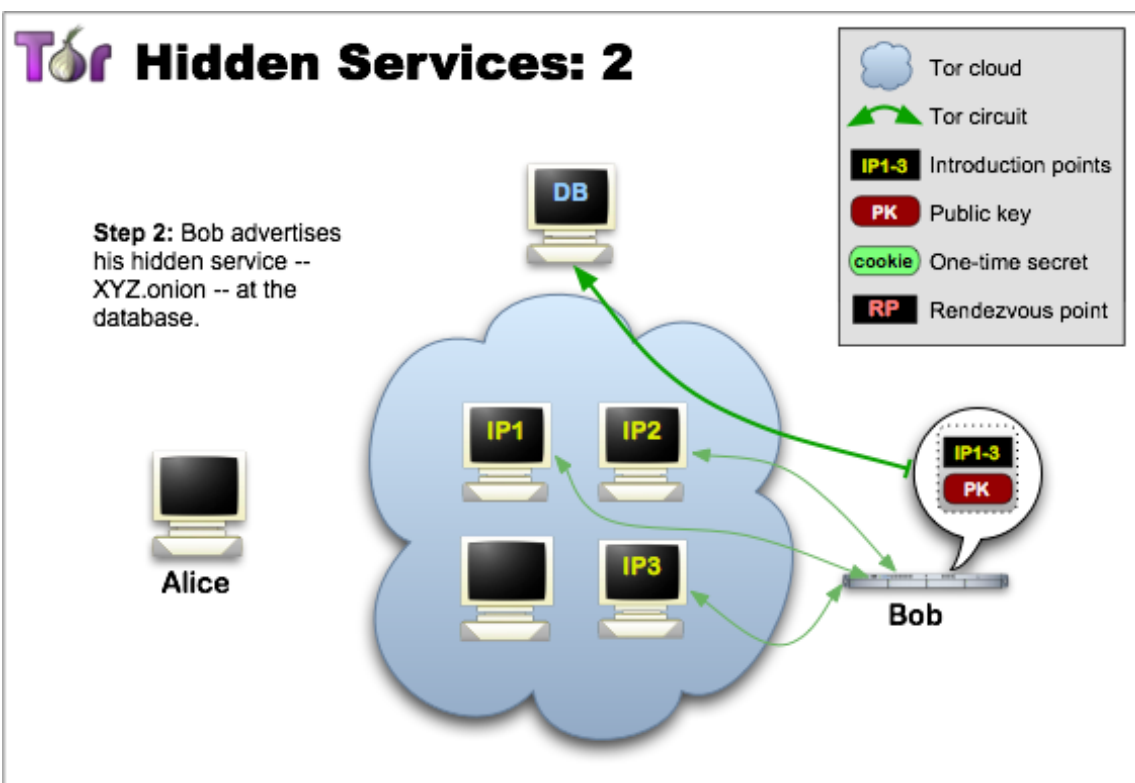
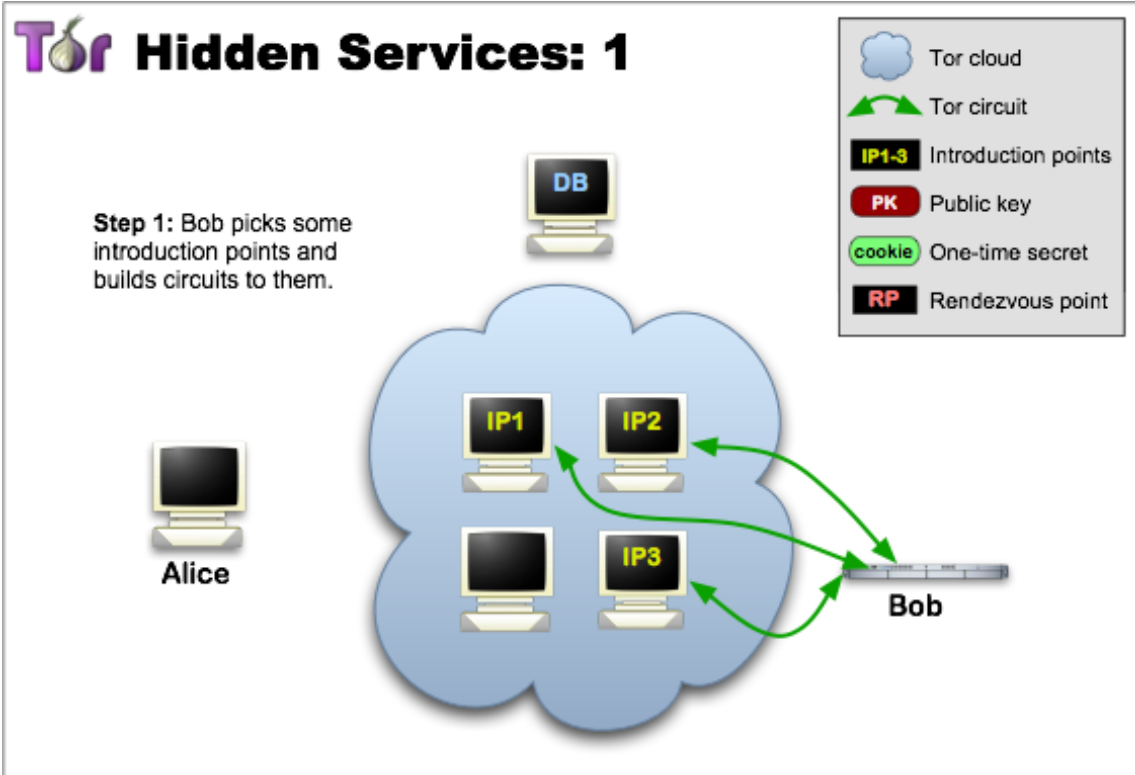
Acknowledgements

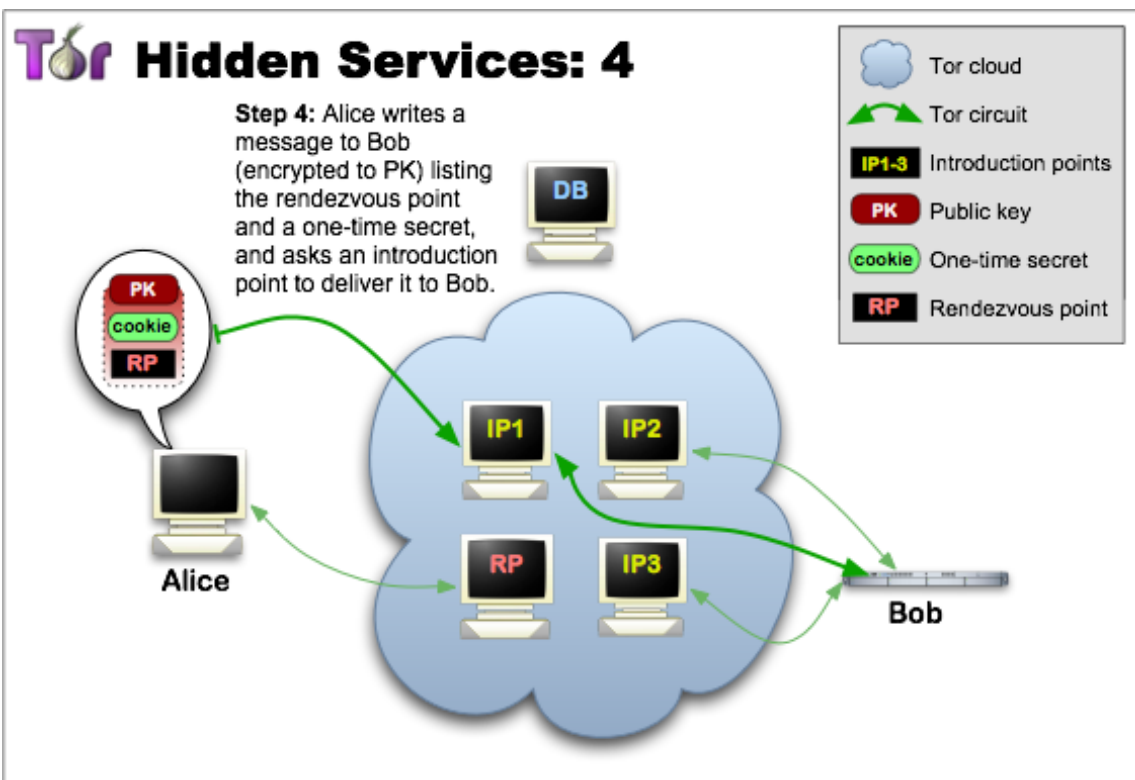
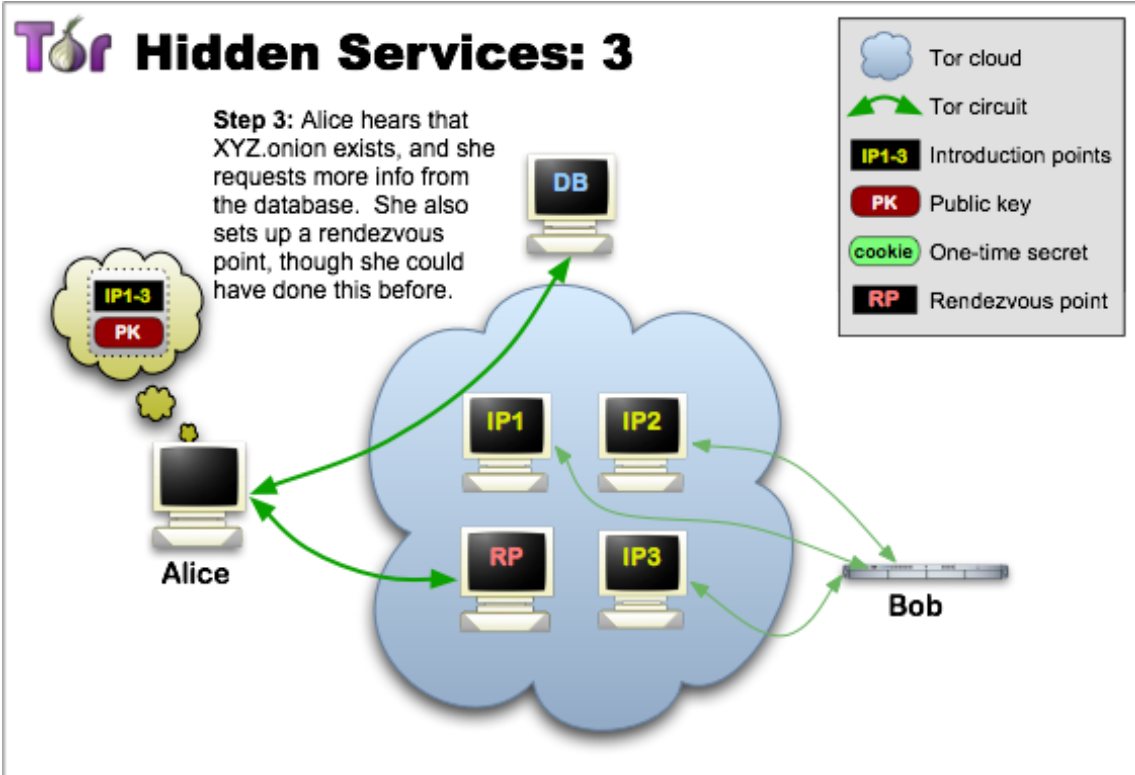
I would like to acknowledge Mr. Walsh, who kept this project on track through the year.

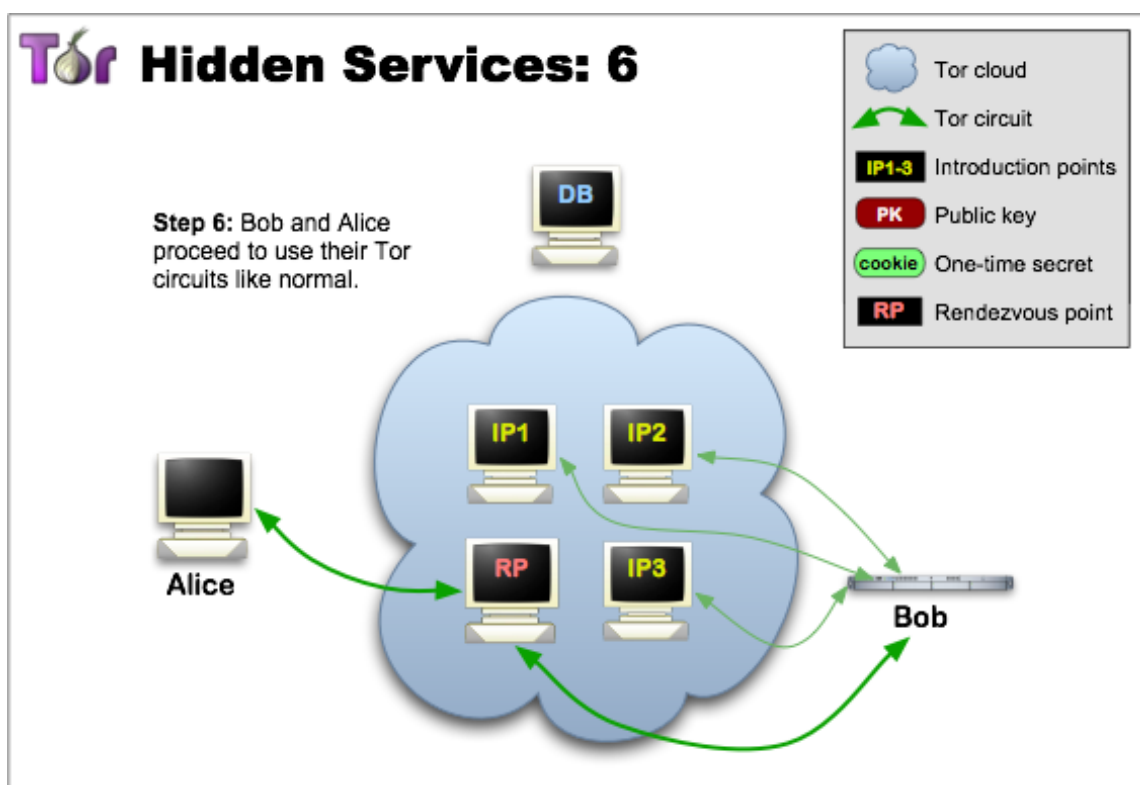
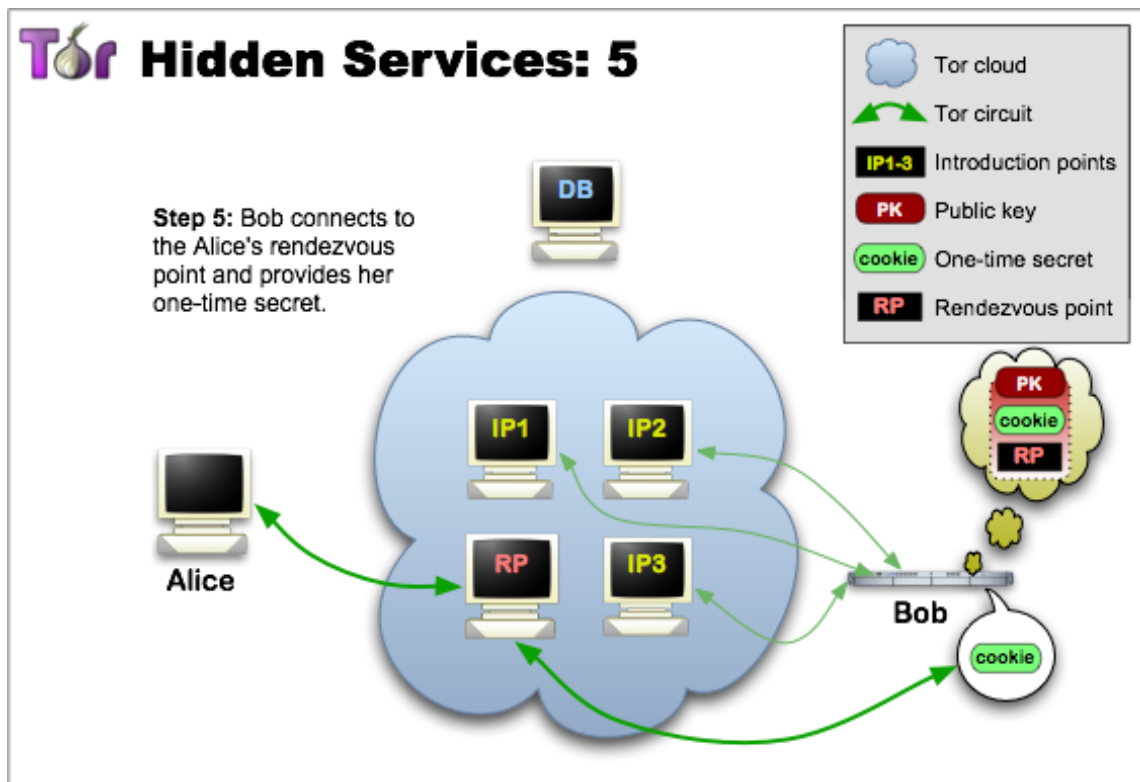
I would also like to acknowledge the people behind the CJDNS project, particularly the creator and lead developer Caleb James Delisle, who was always helpful on IRC in explaining design decisions behind CJDNS.

I would also like to acknowledge the tireless work of the entire open-source community, who spend their time making amazing software with little reward. Most of the protocols I researched for this project are open-source and are truly brilliant creations.

Appendix I.







Hidden Service explanation from <http://www.torproject.org/docs/hidden-services.html.en> by the Tor foundation.