Kirisurf: A Novel Censorship-Resistant Onion-Routing System

Journal:	*IEEE Security & Privacy*
Manuscript ID:	SP-2014-01-0016
Manuscript Type:	Regular
Date Submitted by the Author:	17-Jan-2014
Complete List of Authors:	Dong, Yuhao; University of Waterloo, Faculty of Mathematics
Keywords:	censorship, mixnetwork, onion-routing, privacy, proxy

Kirisurf: A Novel Censorship-Resistant Onion-Routing System

Yuhao Dong

University of Waterloo

January 17, 2014

Abstract

Nowadays, authoritarian governments such as those of the People's Republic of China and the Islamic Republic of Iran increasingly use deep packet inspection (DPI) and application-layer firewalls to censor and monitor the Internet. Unlike simple filters which only filter endpoints of connections, these advanced firewalls categorize and block by looking into their content. Not only are sensitive keywords filtered, advanced heuristics also detect protocols used by popular censorship circumvention and privacy protection software, such as Tor and Ultrasurf, and block related connections.

In this paper, we present a multi-path onion routing system, Kirisurf, with a novel node graph layout and server distribution system. Kirisurf is designed from the ground up to counter very large-scale censorship and monitoring systems, while providing higher quality of service and anonymity compared to the existing onion routers, such as the Tor network. In addition, we discuss possible attacks upon Kirisurf and further improvements that could mitigate such attacks.

1 Introduction

1.1 Censorship infrastructure of China

As China has the most advanced national internet censorship system in place around the world, we use China's internet firewall and monitoring infrastructure as our model adversary[1]. Censorship can be roughly divided into filtering of websites, and filtering for blocking certain data stream protocols used for circumvention of the firewall (such as Tor).

The censorship used to block websites is rather primitive. On the link layer, certain IP addresses are blocked, and packets destined to those IP addresses would simply be routed into a black hole. DNS requests for the IP addresses of blocked websites are often intercepted too, with fake results being returned to the client[2]. Finally, deep packet inspection looks for specific keywords in a webpage, cutting off the connection with a flood of RST packets if the words are detected [1, 4].

Understandably, filtering of anti-filter software is much more sophisticated. Using application-layer firewalls which intercept TCP streams and apply filtering before forwarding them (instead of looking at each packet on the wire individually as in naive deep packet inspection), protocol-internal metadata such as TLS certificate details can be easily gleaned and used as distinguishers of "suspicious" connections. If the metadata gives a high enough assurance that the connection in fact is used in an anti-blocking system, the connection may be reset and the associated foreign server may be blacklisted — China seems to use this measure with some success against former versions of Ultrasurf [7, 8]. If the metadata does not give a high enough assurance — *unpublished* Tor bridges using unobfuscated TLS belong to this category — active probes may be dispatched to the suspicious foreign server and attempt to communicate using the filtered protocol; if the server responds correctly, it is then blacklisted [12, 3]. There are also some unconfirmed reports of whole protocols with little use outside of tunneling, such as PPTP, being blocked [2]. Of course, primitive methods such as trying to enumerate all of the servers in an anti-blocking network and blocking all of them, or simply blocking or hijacking access to the server address distribution authority, are often used, quite effectively in the case of blocking direct Tor connections [3] or those of the once-popular Psiphon [9] software.

1.1.1 Threat model

To give a concrete goal for censorship-resistant onion routing networks in general, and Kirisurf in particular, we need to formulate a threat model, against which we must

[1]Rather amusingly, the Tor Project gives the name "China problem" [15] as an umbrella term for problems related to huge internet-controlling censors!

[2]See appendix

protect such a network. Our adversary would be an idealized combination of a national censor in the style of China, and an international passive eavesdropper in the style of the NSA.

We assume that the adversary:

- Is able to:
  - Monitor the Internet infrastructure underlying a significant portion of the network of nodes
  - Arbitrarily cut off any node from access by users, but not other nodes
  - Monitor all connections to and from the users of the system
- Aims to:
  - Reliably identify or block users of the system
  - Obtain a comprehensive list of nodes (for monitoring or blacklisting)
  - Track onion routing paths based on a limited window into the network
  - Correlate entering and exiting traffic over entry and exit nodes under surveillance

Our threat model is slightly different from, say, that of Tor, in adding the two additional aims of identifying users and servers of the system (not considered a useful goal to the Tor adversary) and correlation of entering and exiting traffic (defending against which the Tor Project team deemed close to impossible).

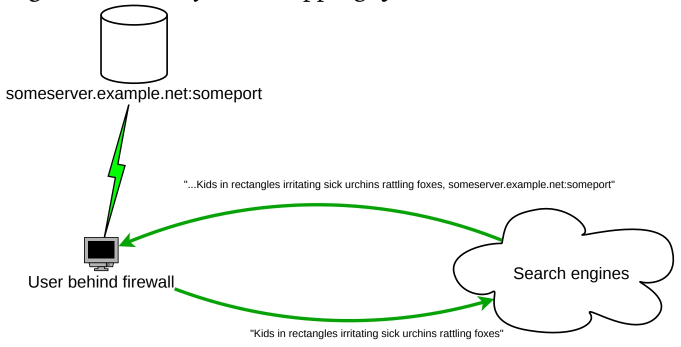
2 Solving the bootstrapping problem

2.1 Chicken-and-egg problem for metadata distribution

In any anti-censorship system, bootstrapping metadata about the network of nodes must be distributed to clients so that they know who to connect to. However, a censorship-resistant channel must exist for transporting this data, since otherwise this data could be censored as well! So we have a chicken-and-egg problem, the *bootstrapping problem*, which does not seem to have a clear solution.

The bootstrapping problem has plagued most, if not all, anti-censorship systems that have been in use in places like China. All of the current approaches to the problem have easy countermeasures by the censor and are clearly far from optimal:

Figure 1: *Kirisurf bootstrapping system*


someserver.example.net:someport
"...Kids in rectangles irritating sick urchins rattling foxes, someserver.example.net:someport"
User behind firewall
Search engines
"Kids in rectangles irritating sick urchins rattling foxes"

- Ignore the bootstrapping problem and use a fixed centralized directory server. Tor uses this method [15]; any censor on the scale of the Chinese firewall can easily defeat this system by blocking all connections to the central directory server.

- Use some sort of obscure protocol to transport the bootstrapping data. Ultrasurf uses this method, utilizing specially crafted DNS requests to obtain entry server addresses [7]. However, obtaining a copy of the client software is all that is needed to reverse-engineer the bootstrapping protocol and block it. Thus, frequent protocol changes are required; in fact the Ultrasurf metadata distribution protocol mentioned has changed in many trivial ways[3] since the publication of J. Applebaum's analysis at [7].

- Hard-code all of the server data into the software. Psiphon seems to use this method [4]. This is both inflexible (version upgrades needed if server network topology changes) and trivial to block (obtain a copy of the software and get the embedded list). Unlike transporting easily changeable data from a central server, once a node is blocked there is no real way to push a new node address to clients. Not surprisingly, Psiphon and many similar software are completely unusable in China.

2.2 Kirisurf: chickens hatched from search engines

To solve this bootstrapping problem, we observe that we have much looser requirements for the censorship-resistant tunnel used for bootstrapping than the censorship-resistant tunnel used for the actual encapsulation of onion-routing streams. In particular, the bootstrapping tunnel can have extremely low throughput, and only needs to distribute an infrequently changing piece of data instead of arbitrary streams.

[3]See appendix.
[4]See appendix

Search engine web services serve as such low-throughput secure tunnels. Censors such as China are very unwilling to block a search engine entirely, and more and more widespread HTTPS encryption in use on search engines make selectively blocking particular search phrases impossible. We thus can place bootstrapping information onto popular websites and then wait for search engines to crawl them. Kirisurf can in turn search for the bootstrapping information and obtain it from the search phrase context preview inside the search engine.

The exact protocol is as follows. The following phrase is posted onto forum signatures in various popular websites:

```
Kids in rectangles irritating sick urchins
rattling foxes, server.example.net:someport
```

where `server.example.net:someport` is the address of an authenticated directory server.

Then, this phrase would propagate through the caches of various encryption-enabled search engines, such as Google, DuckDuckGo, and IxQuick, none of which are usually blocked in China. The Kirisurf client software subsequently searches for “kids in rectangles irritating sick urchins rattling foxes”; the preview in the search engine would complete the phrase with “server.example.net:someport”, which Kirisurf would extract and connect to to obtain metadata.

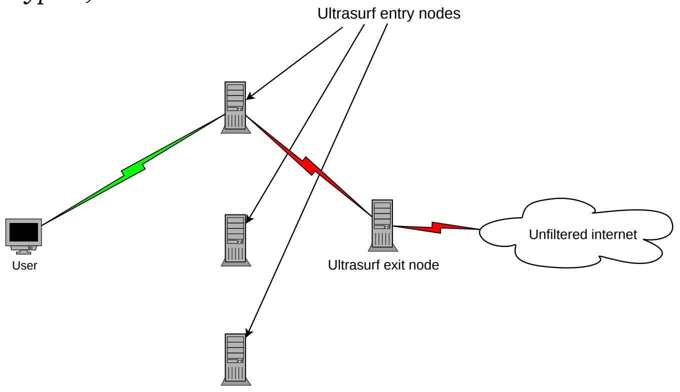
This system solves the bootstrapping problem elegantly; censors cannot simply block all encrypted search engines just to disable Kirisurf, and if censors take the time to use the protocol to discover the directory server address and block it, the directory can be quickly moved and new server addresses distributed, all without any client-side changes. One possible attack may be to massively flood search engines with bogus data pointing to fake directory servers; usage of secure authentication protocols by the directory server prevents this scenario.

3 Hiding network topology

3.1 Existing attempts at obscuring topology

One large problem with onion routers is that each client usually needs to know the addresses of all of the nodes in the network in order to build circuits. However, this allows an attacker to simply act as a client, get the addresses of all of the nodes, and add them all to a blacklist. None of the existing onion routing systems solve this problem very well; Tor does have special non-public entry nodes called “bridges”, but the rest of the Tor network topology is easily accessible, and in our threat model this would allow

Figure 2: *Ultrasurf network layout. Green links are encrypted; red links are not.*



an international eavesdropper to monitor traffic flowing between nodes, possibly doing traffic volume correlation attacks.

Of course, one solution to this topology-hiding problem is to stop using onion routing. Ultrasurf [7] (and possibly Freegate[5]) have all of the nodes controlled by the producer of the software, and each node decides the next node to forward traffic to. The client only needs to know the addresses of a few entry nodes. However, this system loses most, if not all, of the anonymity protections of onion routing, and does nothing but add pointless complexity compared to having one-hop proxies, a few of which are distributed ot the client. As all the nodes are controlled by a single entity, by compromising this entity all users could be easily tracked; in fact Ultrareach, Ultrasurf’s producer, actively gathers user usage data [7].

Another possible and easy solution to this problem, although not widely used in onion routing systems, is to give each client a subset of the nodes. This allows each client to form onion routing circuits, while preventing a single attacker from blocking all of the nodes. However, one problem is that usually, the number of clients far exceed the number of nodes. Each client needs to know a quite large fraction of the network topology; this allows an attacker to simply impersonate a small number of clients and obtain the entire topology.

In the end, we want a routing system that:

- Gives the client enough information to securely build onion-routing circuits
- Allows for low-latency and reasonably low-overhead communication
- Makes gathering information about the addresses of each node very difficult

[5]See the appendix.

3.2 Kirisurf's approach: know the addresses of only your neighbors

We observe that the graph of all possible circuits in a traditional onion-routing network of n nodes forms a n-complete graph, where s_i is connected to s_j if and only if $s_i s_j$ is allowed in a circuit $s_1 s_2 \dots s_k$. That is, all nodes are adjacent to all nodes. However, note that for circuits involving any node (so that no node goes unused) to be viable, we only need to have a *connected* graph, not a complete graph. In other words, only need to ensure that a path exists between any two arbitrary nodes.

This allows each node to only know the addresses of its 3 to 5 neighbors, although it does need to know adjacency information for the entire network. Thus each node is universally identified by its public key, not by its address, and an adjacency table describing the graph of nodes is distributed to clients and nodes alike, with a very short appendix containing the specific addresses corresponding to the public keys assigned as the node or client's neighbors.

However, one possible complication is the fact that each node only has 3 to 5 neighbors, so very roughly, for a circuit path length of k, there are around 3^k to 5^k possible circuits to build by a client; this pales in comparison to the roughly n^k[6] (where n is the number of nodes) possible circuits for a traditional onion router like Tor. Tor in normal cases uses $k = 3$; this is clearly very far from adequate. Kirisurf, on the other hand, uses a random value of $k = 7$ to $k = 10$ by default (longer and shorter paths are client-configurable); this gives around $3^7 \approx 2,000$ to $5^{10} \approx 10,000,000$ possible circuits, solving the problem of too little possible circuits.

In addition, we remove the distinction between entry and intermediate nodes; this is to prevent topology identification by large-scale snooping, as with this distinction removed, the adversary would have a hard time distinguishing between incoming client connections from outside the node graph, or incoming connections from another node.

The precise mechanism of establishing a circuit is as follows:

- Obtain the public key graph, represented as an adjacency table, from the directory server, and obtain the list of your neighbors.
- Select a path in the public key graph starting from a neighbor and ending at an exit node.
- Establish the circuit starting from the neighbor; instead of requesting the node to forward to a certain address at each step, the client requests the node to forward to a specific public key.

[6]The precise number is the number of permutations of k objects selected from n objects

4 Protocol obfuscation

4.1 The authentication/identification dilemma

For any encrypted system to be secure, authentication is required. Clients must know that they are connecting to the correct servers, rather than man-in-the-middles set up by attackers, so servers need to provide identifying information. Yet this identifying information, in the form of cryptographic public keys and certificates, would also expose the fact that an anti-filter system is in use.

Tor has suffered this problem repeatedly, leading even to non-publicly listed bridges getting blocked. By sniffing the TLS handshake, the firewall can identify a TLS certificate in the format used by Tor, and then block the connection after further investigation by an automatic probe.

Older versions of Ultrasurf, which also use TLS [7], use the rather naive approach of simply ignoring authentication and using unauthenticated TLS. On one hand, this does prevent certificate-based identification of traffic; on the other hand, if the adversary does identify a connection, it can then silently eavesdrop, causing far more damage to privacy and confidentiality than simply blocking the connection.

4.2 Obfuscation as a solution

Protocol obfuscation is basically encapsulation of streams belonging to a suspicious protocol (such as Tor) into some protocol unlikely to cause suspicion, and to effectively hide metadata transmitted in the original protocol. Tor's experimental `obfsproxy` bridges, which are distributed via non-public channels, use the `obfs3` protocol. `obfs3` relies on a modified anonymous Diffie-Hellman key exchange [13] to provide an encapsulation both difficult to distinguish from random noise and immune[7] to passive listeners.

However, `obfs3`, although not currently blocked in China, does have a distinguisher:

- Near-simultaneous transmission of a huge block of random-looking data without waiting for the other party, and then pausing (as the shared secret is computed). This does not make much sense for a non-obfuscating protocol, and could possibly be used as a heuristic for detecting suspicious connections and dispatching investigation probes.

With enough resources, it would be conceivable that `obfs3` connections could be identified and blocked.

[7]That is, passive listeners cannot decode the encapsulated stream without solving the DH problem

4.3 Kirisurf's obfuscation

Kirisurf's obfuscation is in part based on `obfs3`, notably using Ian Goldberg's "UniformDH" modified Diffie-Hellman key exchange, but it instead works in the following way:

- Client side:
  - Generate a new UniformDH key pair, and send over the 1536-bit public key.
  - Without waiting, slowly (take around 3 seconds) send 16384 bytes of pseudorandom garbage.
  - Then read 1536 bytes of public key from the remote end, and generate the shared secret.
  - Then read 65536 bytes of pseudorandom garbage from the remote.
  - Now derive upstream and downstream 128-bit RC4 keys, and drop the first 8192 bytes of the keystream.
  - Then send the following *encrypted* message:
    - 2 bytes: random length of zeroes in little-endian, ℓ
    - ℓ bytes of zero bytes
  - Read a similar message from the remote.
  - The stream is now ready for transporting the encapsulated stream, encrypted and decrypted with the upstream and downstream RC4 keystreams.
- Server side:
  - Wait until client sends 1536-bit public key
  - Generate own public key, and send over
  - Derive shared secret, and send 16384 bytes of pseudorandom garbage slowly
  - Read 65536 bytes of garbage from the client
  - Derive RC4 keys, dropping the first 8192 bytes of the keystream.
  - Send a similar encrypted zero-filled message to the client
  - Read such a message from the remote
  - Start sending and receiving encapsulated data.

This obfuscation has the great advantage of the server not replying until the client sends its key, which eliminates the traffic distinguisher and also prevents a resource exhaustion attack in which a client opens lots of connections to a server, while sending nothing, forcing the server to repeat expensive DH operations over and over.

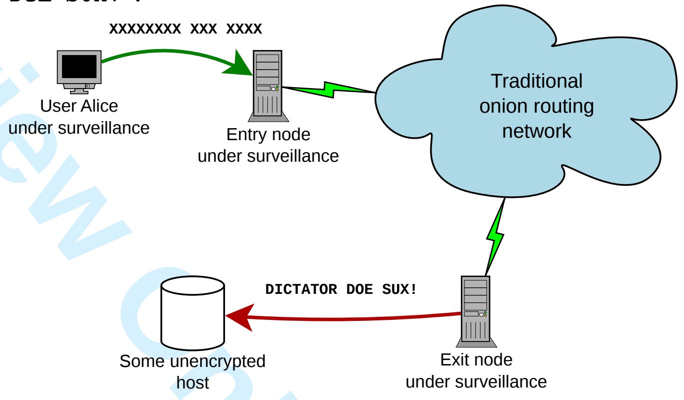
4.4 Obfuscation all the way!

Unlike Tor's case, where obfuscation is used only on select "bridges", Kirisurf universally uses the abovementioned obfuscation scheme as the underlying transport for all of its communications. This is crucial as our threat model dictates that an attacker in control of large amounts of Internet infrastructure should not be allowed to trace out the onion routing topology of the whole network. If inter-node communications were to be unobfuscated, then our attacker can deduce network topology by gathering data on observations of Kirisurf communications.

5 Demultiplexing connections

5.1 Volume correlation attacks

Figure 3: *A volume correlation attack. By looking at the volume of information sent by Alice, a monitor can guess that it is she who sent the incriminating message* "`DICTATOR DOE SUX!`".



XXXXXXXX XXX XXXX
User Alice under surveillance
Entry node under surveillance
Traditional onion routing network
DICTATOR DOE SUX!
Some unencrypted host
Exit node under surveillance

Tor and similar systems multiplex many TCP connections over several circuits, but with each connection staying on a single circuit. This has the problem of *volume correlation attacks*. If our adversary monitors an entry and exit node, and notices incriminating traffic coming into the exit node, and shortly after, traffic of the same volume going from the entry to some client, the adversary then can guess that said client requested the incriminating traffic. Conversely, if a user sends some encrypted message into an entry node, and shortly after an incriminating message of the same volume pattern emanates out of an exit node, then the eavesdropper would be able to pin the message to the user. Considering this an unsolvable problem, "Tor does not defend against such a threat model"[15].

5.2 Bandwidth bottlenecks

Another problem with limiting each connection to one circuit is that the total throughput of any given circuit is not greater than the slowest link in the circuit. This can cause serious problems in certain cases.

Consider the case where nodes A, B, and C are all very fast nodes in datacenters, each supporting, say, 100 Mbps. Now let D be some very slow node on a slow DSL connection, supporting, say 0.5 Mbps. Now, a traditional onion router would route far more circuits through A, B, C than through D, but for the unlucky few clients using circuits going through D, the fact that the client might have, say, 5 Mbps of downlink bandwidth does not matter at all; 4.5 Mbps of the bandwidth would be wasted.

Furthermore, an adversary can flood the network with very slow nodes advertised as high-bandwidth to the directory server. Circuits would be herded to these nodes, causing massive congestion, greatly decreasing QoS, and leading people to abandon the network. Such attacks have in fact been revealed to be used by the NSA in cooperation with GCHQ to disrupt Tor usage [16].

5.3 Kirisurf: packet-switched over multiple circuits

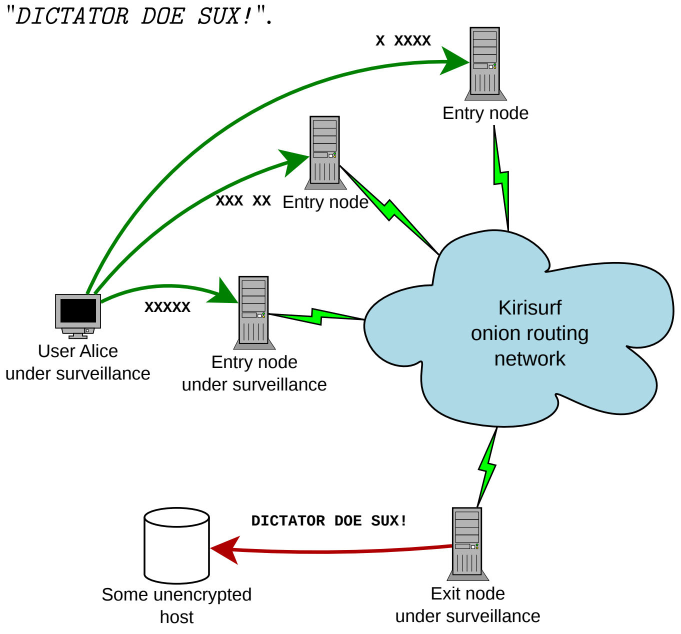
Similar to the concept of "Multipath TCP", Kirisurf *demultiplexes* one multiplexed TCP stream over several circuits. There is essentially no correlation between circuits and connections. After multiplexing our connections into one stream, we packetize this stream and add sequence numbers. We then establish around 10 to 20 circuits with the same exit node, and send each packet to a randomly selected circuit. Returned packets are then reordered with their sequence numbers and reassembled into one contiguous return stream. A memorable name for this whole assembly of onion-routed circuits might be an "onion stew". The exact format of the packets are:

- 64 bytes: onion stew number
- 8 bytes: sequence number
- 2 bytes: body length ℓ
- ℓ bytes: body

One thing to note is that the packets are not actually sent down completely randomly selected circuits; instead, each selected circuit for the packet is more likely to be the last selected circuit than random selection would suggest — an inter-selection correlation factor is in place. Kirisurf can also add and remove circuits at will to and from an onion stew.

Both of these rather pointless-sounding measures defeat volume correlation attacks. The first measure means that if an attacker were to, say, observe 200 bytes of incriminating information being requested through an exit node,

Figure 4: *If Alice uses Kirisurf, the censor would only see a random portion of the traffic, and is unable to correlate the encrypted information emanating from Alice with* "`DICTATOR DOE SUX!`".


x xxxx
Entry node
xxx xx
Entry node
xxxxx
User Alice under surveillance
Entry node under surveillance
Kirisurf onion routing network
DICTATOR DOE SUX!
Some unencrypted host
Exit node under surveillance

and then observe 20 bytes be downloaded from some entry node onto a client, it has no way of knowing whether those 20 bytes are packetized fragments of the 200 bytes of incriminating information, as the correlation factor means that the attacker cannot simply divide 200 by the number of circuits to get the number of bytes to expect on a single circuit. The second measure means that the same long-running onion stew would constantly shift between circuits, making long-term traffic analysis of a particular stew very difficult, if not impossible.

Multiple circuits per multiplexed connection also fixes the problem Tor has with bandwidth bottlenecking. Using our original nodes A, B, C, D, each circuit only needs 0.25 to 0.5 Mbps of bandwidth for the total sum of 10 to 20 circuits to fully utilize the 5 Mbps connection of the client. Thus, having one circuit go through the slow node D would not be an issue.

Finally, the impact of flooding the network with slow nodes advertised as fast is somewhat mitigated, as the Kirisurf client, not the directory, would be able to gradually gathers data by trying out different combinations of circuits, and eventually learns what circuits are slower and would use them less.

6 Conclusion and further work

In this paper we have described a novel multipath onion-routing system which provides much better resistance to monitoring and censorship, while having fairer distribution of throughput which eliminates bottlenecks, and its implementation in the experimental Kirisurf anti-blocking software. Additionally, we describe the specific attacks, to which previous onion routers often fall victim to, namely protocol fingerprinting, topology discovery, and volume correlation attacks, and protections against these attacks used in Kirisurf.

One problem not solved by Kirisurf is the secure, censorship-resistant distribution of the client software. Unlike the case with bootstrapping small amounts of directory addresses, where the censorship-resistant tunnel can be very low-bandwidth, distributing a client software requires large amounts of bandwidth. Unfortunately, no public file deposits of the "too-big-to-block" scale seem to support HTTPS; peer-to-peer systems might be investigated, although systems such as BitTorrent simply kick the can down the street by introducing yet another datum to distribute (infohash).

References

[1] Richard Clayton, Steven J. Murdoch, Robert N. M. Watson (2006) "Ignoring the great firewall of china" doi:10.1.1.140.9889

[2] Lam, O. (2011) "China: PPTP and L2TP VPN protocols blocked" http://advocacy.globalvoicesonline.org/2011/03/20/china-pptp-and-l2tp-vpn-protocols-blocked/

[3] Phillipp Winter, Stefan Lindskog (2012) "How the Great Firewall of China is blocking Tor" http://www.cs.kau.se/philwint/pdf/foci2012.pdf

[4] Xueyang Xu, Z. Morley Mao, J. Alex Halderman (2009) "Internet Censorship in China: Where Does the Filtering Occur?"

[5] Global Internet Freedom Consortium. http://internetfreedom.org/

[6] 『动态网软件发布纪录』 (Dynamic Internet Technologies software version history), http://dongtaiwang.com/loc/news.php

[7] Applebaum, J. (2012) "Technical analysis of the Ultrasurf proxying software", https://media.torproject.org/misc/2012-04-16-ultrasurf-analysis.pdf

[8] 『如果无界连不上，请关闭所有破网工具15分钟，再试』 (If Ultrasurf does not work, shut down all anti-blocking tools for 15 minutes, and retry) http://forums.internetfreedom.org/index.php?topic=16464.0

[9] Psiphon 3 download page. https://s3.amazonaws.com/0ubz-2q11-gi9y/en.html

[10] OpenVPN Security Overview http://openvpn.net/index.php/open-source/documentation/security-overview.html

[11] "Iranian Man-in-the-Middle Attack Against Google Demonstrates Dangerous Weakness of Certificate Authorities" https://www.eff.org/deeplinks/2011/08/iranian-man-middle-attack-against-google

[12] Roger Dingledine, Jacob Appelbaum (The Tor Project) "How governments have tried to block Tor" https://svn.torproject.org/svn/projects/presentations/slides-28c3.pdf

[13] Obfs3 protocol specification. The Tor Project. https://gitweb.torproject.org/pluggable-transports/obfsproxy.git/blob/HEAD:/doc/obfs3/obfs3-protocol-spec.txt

[14] Olaf Titz (2001) "Why TCP Over TCP Is A Bad Idea" http://sites.inka.de/bigred/devel/tcp-tcp.html

[15] Tor FAQ. The Tor Project. https://trac.torproject.org/projects/tor/wiki/doc/TorFAQ

[16] National Security Agency (published by the Guardian): "Tor Stinks" http://www.theguardian.com/world/interactive/2013/oct/04/tor-stinks-nsa-presentation-document