Evading Censorship with Browser-Based Proxies

David Fifield[1], Nate Hardison[1], Jonathan Ellithorpe[1], Emily Stark[2], Dan Boneh[1], Roger Dingledine[3], and Phil Porras[4]

[1] Stanford University
[2] Massachusetts Institute of Technology
[3] The Tor Project
[4] SRI International

Abstract. While Internet access to certain sites is blocked in some parts of the world, these restrictions are often circumvented using proxies outside the censored region. Often these proxies are blocked as soon as they are discovered. In this paper we propose a browser-based proxy creation system that generates a large number of short-lived proxies. Clients using the system seamlessly hop from one proxy to the next as these browser-based proxies appear and disappear. We discuss a number of technical challenges that had to be overcome for this system to work and report on its performance and security. We show that browser-based short-lived proxies provide adequate bandwidth for video delivery and argue that blocking them can be challenging.

1 Introduction

While the Internet began as a research network open to all types of data, many nations now filter Internet traffic. The OpenNet Initiative, which tracks public reports of Internet filtering, lists a large number of countries that filter Internet traffic. Some countries block sites like YouTube and Facebook while others block access to web content containing materials they consider objectionable. The list of countries includes well-publicized examples in Asia and the Middle East as well as Australia and several European countries. Over half of the 74 countries tested in 2011 imposed some degree of filtering on the Internet [1].

Censored users try to bypass the censor by connecting to sites through a proxy. Several proxy systems have emerged to help users circumvent censorship. Most notable among these is Tor [2], which, while originally designed to provide anonymity, has also seen wide use in circumvention. Other proposals include Telex [3] and Ultrasurf [4]. The existence of circumvention systems makes the censor's job harder: The censor must block access to all circumvention tools, in addition to any resources it would ordinarily block, if it is to remain effective. Our goal is to enable access to circumvention even in the face of such blocking.

Proxy-based circumvention systems generally need to solve three problems:

1. **Rendezvous.** A rendezvous protocol lets a user in the censored region send and receive a small amount of information (a few bytes) from the circumvention system to outside the censored region, for the purpose of introducing a

user to a proxy. Rendezvous protocols are designed for low-rate traffic and are intended to be difficult to block. Tor, for example, developed rendezvous protocols to distribute the IP addresses of Tor bridges, which are relays whose addresses are not universally known so they are harder to block [5].

2. **Proxy creation.** The circumvention system relies on proxies outside the filtered region to relay traffic from the client to the desired site. In response the censor can masquerade as legitimate users to discover proxy addresses and promptly block them. One way to combat this Sybil attack is to constantly create new proxies outside the filtered region. As proxies get blocked, new ones take their place. Rapid proxy creation is the main topic of this paper.
3. **Camouflage.** Once the client has the address of a non-blocked proxy, it needs to camouflage its conversation with the proxy so that the session cannot be blocked by traffic analysis. The goal of camouflage is to make the conversation look like acceptable traffic such as an e-commerce transaction, a voice conversation, or part of a multiplayer game. Concrete proposals for camouflage include obfsproxy [6] and StegoTorus [7]. We treat camouflage as an independent layer and do not discuss it in this paper.

A complete circumvention system must also address secure client software distribution, an install system, and secure integration with a web browser. The Tor Project already handles these issues quite well and we do not discuss them here.

In this paper, we focus on Tor because it is widely deployed, with hundreds of thousands of daily users [8]. Tor consists of a network of several thousand volunteer nodes, known as relays. Clients build a three-node encrypted *circuit* through the network by selecting relays from a public relay directory. The client sends data to an *entry node*, which forwards it through an intermediate and an exit node, after which the exit node sends the data to the destination host on the public Internet. More details about the Tor design are given in [2]. The fact that relays are listed in a public directory makes them easy to block [9] – a countermeasure to this blocking is Tor *bridges*, relays whose addresses are not made universally known. Bridges, too, have been found susceptible to partial enumeration and blocking [10–12].

Our contributions. In this paper we propose a new approach to rapid proxy creation. The core idea is to use the power of the web to create millions of short-lived proxies, each proxy being active for only a few minutes. To do this, we enlist the help of volunteer web sites (e.g., personal home pages) outside the filtered region that want to support an open Internet. These volunteer web sites are unrelated to the destination web site that the censored user is trying to reach.

A volunteer web site simply embeds a small "Internet Freedom" badge on its web pages (see Fig. 1). The badge is a tiny user interface on top of some JavaScript code. When a web browser outside the filtered region visits the volunteer site, it runs the JavaScript program, which relays traffic to and from the filtered region through the visitor's web browser. In effect, the browser visiting the volunteer's site becomes a short-lived proxy. As soon as the visitor navigates away from the page, the browser unloads the badge and the proxy disappears leaving no trace on the visitor's machine. Surprisingly, browsing the web through these ephemeral browser-based proxies, even when hopping from one proxy to another, works quite well. Our experiments in Sect. 4 show that the filtered client can sustain more than enough bandwidth to carry a Tor tunnel.

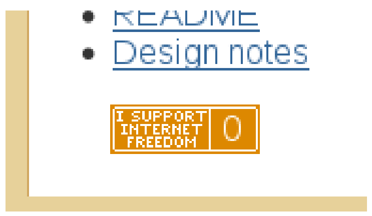

Fig. 1. Flash proxy badge on a web page. It runs in a visitor's web browser, and the (optional) counter increments for each client served. Clicking on the badge disables it.

Our flash proxies[5] are at the opposite extreme of Tor bridges. Tor bridges are intended to serve for a long time; they are created at a relatively low rate and there are only a few thousand of them. Our proxies are only active while the visitor's browser is viewing the volunteer web page – often just a few minutes – but a new proxy is created every time someone visits a volunteer page, potentially creating a pool of millions of active proxies at any given time. Building a reliable transport using these ephemeral proxies presents interesting challenges discussed in Sect. 3. For completeness we discuss new strategies for rendezvous in Sect. 6.

2 Threat Model and Assumptions

Our setting includes four key players. The **client** owns a computer in the filtered region and is trying to access a web site outside the filtered region. We assume the client has complete control of his computer and, in particular, can install arbitrary software on the computer. The **target web site** (e.g., Facebook) is located outside the filtered region and is generally oblivious to the circumvention effort. That is, it does not cooperate with nor try to prevent circumvention. The **circumvention tool** attempts to relay traffic back and forth from the client to the target host. The tool may include client software as well as network infrastructure inside and outside the filtered region. The **adversary** (censor), who tries to filter traffic, must allow acceptable Internet traffic such as banking and e-commerce, but tries to block all objectionable traffic, including possibly blocking systems used by the circumvention tool. The definition of what is acceptable and objectionable is up to the adversary.

[5] The word "flash" is meant to evoke an idea of quickness and ephemerality. Our first implementation of the system used Adobe Flash, which partly inspired the name. The current implementation uses JavaScript rather than Flash.

The adversary achieves its goal by installing hardware and software at Internet Service Providers (ISPs) in the filtered region. It can inspect all network traffic to and from the filtered region and block any packet it wishes. However, the adversary operates under the following three constraints:

Line rate. It must operate at line rate and cannot noticeably slow down legitimate traffic to and from the filtered region. In other words, it has little time to make its decision whether to block or allow packets and flows.

No control of clients. We assume the adversary does not have software installed on the client computer. This assumption matches current reality where filtering happens at the network and not at the end host. If the adversary can force users to run filtering software – either by law or by using technologies such as Trusted Computing (TCG) – then the circumvention problem becomes much harder, though not impossible. Some circumvention tools are designed for a public-kiosk environment where the client is browsing the web over a public terminal that may have filtering software installed. These tools, however, are much harder to use and therefore in this paper we assume the user is in full control of his computer.

Minimal collateral damage. To the extent possible, the adversary tries to avoid collateral damage. It tries to minimize the impact to acceptable flows, such as those used for banking and e-commerce. The adversary cannot simply shut down all Internet traffic to the filtered region, as this would halt all banking and e-commerce in the region. While some governments, including Egypt, Libya and Syria, have recently implemented filtering by literally shutting down all Internet traffic, most deployed Internet filters try to minimize collateral damage so as not to hurt commerce.

3 Rapid Proxy Creation Using Flash Proxies

Next we present an architecture for creating a large number of short-lived proxies. The flash proxy system uses browsers all over the Internet as ephemeral proxies. The design is in large part dictated by a limitation of web socket technologies, namely that they can only make outgoing TCP connections, and cannot receive connections as a normal proxy would. In the flash proxy model, the proxy connects to the client, not the other way around.

Our implementation uses the Tor pluggable transports support introduced in Tor version 0.2.2.32 [13]. Pluggable transports are designed to enable different circumvention and tunneling schemes without having to modify core Tor code. Our system requires a program called a *client transport plugin* running on the client and a *server transport plugin* running on the relay. The functioning of these pieces is described further below. The integration of pluggable transports in Tor means that using these auxiliary programs is fairly painless.

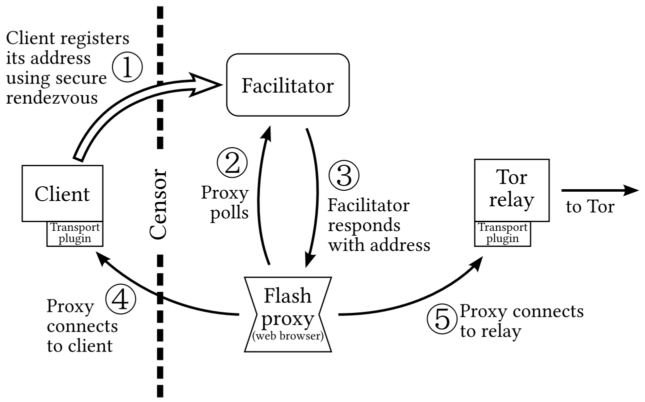

Fig. 2. Flash proxy architecture. When a client wants service, it registers with the facilitator and waits. A flash proxy appears and polls the facilitator for a client address. Once the proxy obtains a client address it connects to both the client and to the Tor relay and proxies traffic back and forth for the client.

Two of the five system components (the Tor client and relay) are the same as are used for any Tor connection. The remaining three (the proxy, facilitator, and transport plugins) are specific to the flash proxy system (see Fig. 2).

Tor client. The censored user runs a Tor client, configured to use a **client transport plugin**.

Flash proxy. The proxy itself is a small application that runs in a web browser and is hosted on a volunteer web site. We have made two implementations of the flash proxy, one using Adobe Flash and one using JavaScript with WebSocket. We have presented the proxy as a small "Internet Freedom" badge as shown in Fig. 1. Whenever someone visits the page, the badge begins running in their browser. When they navigate away the badge is unloaded leaving no trace. The badge communicates with the **facilitator** to find the addresses of clients that need a connection. Once it has a client address, it connects to the **client transport plugin**, then to the **server transport plugin** running on the **Tor relay**, and begins proxying data between them. The badge itself runs in the background and has no impact on the visitor's interaction with the volunteer site. It is important to understand that censored users do not need to see the badge – in fact web sites including the badge can be blocked by the censor – because it is visitors to the web site, not the web site itself, who provide the proxy access.

Facilitator. The facilitator keeps track of client registrations and hands them out to **proxies** when requested. When the **client transport plugin** starts it registers with the facilitator using a robust rendezvous protocol such as one of those described in Sect. 6. The facilitator may be blocked by the censor; the purpose of the rendezvous protocol is to allow the client to send its IP address – just a few bytes – to the facilitator despite blocking. The flash proxies that communicate with the facilitator do not go through the censor and are therefore not impacted by blocking the facilitator. After registration, when a browser-based flash proxy is available for service, the **facilitator** assigns it to serve a censored client.

Transport plugins. The Tor client and relay both run their own transport plugin to read from and write to the flash proxy tunnel. In the case of WebSocket, the transport plugins respond to the HTTP handshake, encode and decode binary data, and put data into WebSocket frames. The client transport plugin has the additional important duty of bridging Tor's "connect outward to a proxy" expectation and the "receive a connection from a proxy" reality of the flash proxy architecture: it receives connections from the Tor client (using ordinary TCP), and on another port receives connections from flash proxies (using WebSocket), maintaining a pool of connections of each type. Whenever there is at least one connection in both pools, the transport plugin links them up and begins to relay data traffic back and forth. When a flash proxy disappears, the transport plugin begins to use another proxy connection from the pool. The Tor client is mostly unaware that a new flash proxy is put in place.

Tor relay. Any Tor relay can be used as the entrance to the Tor network, as long as it runs the server transport plugin. The relay may be blocked from the point of view of the client.

3.1 Establishing Connections

Programs running in a web browser, whether they use WebSocket, Flash, or other socket-like technologies, share a limitation: they cannot open a listening socket and wait passively for connections; they can only initiate new outgoing connections. This forces an inversion of the usual proxy model: It is the client (with a full complement of socket operations) that listens for a connection, and the flash proxy (limited by running in a web browser) that connects to it. This inversion is the source of most of the complexity of the model.

A less important restriction is that web browser security policies generally prevent programs from making connections to arbitrary destinations. The browser requires some positive cooperation from the destination that indicates that the connection should be allowed. For us this poses no problem as the three destinations the proxy connects to – the facilitator, the client transport plugin, and the server transport plugin – are cooperative. What this means in concrete terms for the WebSocket implementation is that the facilitator must allow cross-origin resource sharing (CORS) [14] by sending an Access-Control-Allow-Origin header field, and that the client and server must be able to answer the WebSocket handshake and proxy the tunneled data to Tor. For Flash it means that the endpoints must serve a crossdomain policy [15], a small chunk of XML specifying which connections should be allowed.

The fact that the flash proxy may not receive connections has one important advantage. The flash proxy operator (i.e., web browser) may be behind network address translation (NAT) or a firewall that doesn't allow incoming connections, and it doesn't affect the architecture. The unfortunate corollary is that clients must be able to receive connections, which generally means not being behind NAT or else being able to configure port forwarding. We feel that this is at least the right way to allocate the burden, if it must fall somewhere. We expect flash proxies to greatly outnumber censored clients; running a proxy should be as simple as possible while clients are already motivated to take technical steps for secure communication. Ideally flash proxies would be usable even behind NAT without any special configuration using NAT punching techniques; Sect. 5.2 discusses difficulties and solutions.

Figure 2 illustrates the components operating in sequence in a sample session:

1. The client starts Tor and the client transport plugin, and sends a registration to the facilitator using a secure rendezvous mechanism. The transport plugin begins listening for a remote connection.
2. A flash proxy comes online and polls the facilitator.
3. The facilitator returns a client registration, informing the flash proxy where to connect.
4. The proxy makes an outgoing connection to the client, and this connection is received by the client's transport plugin.
5. The proxy makes an outgoing connection to the transport plugin on the Tor relay, and begins sending and receiving data between the client and relay.
6. Sooner or later, the flash proxy disappears and breaks the connection between the the client and the relay. The client's transport plugin then switches immediately to another available proxy. In the unlikely event that none are available, the transport plugin waits until one becomes so. In this process, existing tunneled TCP streams are broken, but see Sect. 5.2 for ideas on transparently keeping TCP connections intact.

The client transport plugin maintains a pool of up to five live proxy connections, in order to make switching between them faster. Only one of the proxies will be used at a time, but when the one being used disappears, the socket handshake is already finished with one of the reserves, for a lower delay in establishing a new connection. The transport plugin discards reserve proxies as they go offline, and the facilitator replenishes the reserve as long as there is extra capacity, so there is no danger of uselessly switching to a defunct proxy.

The flash proxy has features for quality of service and good network behavior. A single proxy limits itself to 10 simultaneous client–relay pairs. Its built-in adjustable rate limit avoids using too much of the operator's bandwidth. The proxy checks the platform it's running on, and disables itself if it is on a mobile phone or similar device where bandwidth that may be limited or expensive.

The goal of the facilitator is to fairly allocate proxies to clients and attempt to provide good service for all. The facilitator also seeks to even the load carried by each proxy whenever there is spare proxy capacity. The facilitator keeps track of the number of clients given to each proxy. When a new client arrives, it is given to the proxy with the least load, with ties being broken randomly. (In the usual case we expect most proxies to be idle most of the time, so this will be a random selection from among all the idle proxies.) While there is unused proxy capacity, the facilitator seeks to provide each client with a small number of redundant proxies for faster switching. When one of the proxies goes offline (which can be detected because the proxy ceases polling), the facilitator will attempt to assign a new proxy to the client that it had been serving. As we discuss in Sect. 6, client communication with the facilitator is low-bandwidth and infrequent; therefore there is no explicit message for a client to "unregister" itself. Instead, the facilitator estimates that a client no longer needs service when a proxy reports that it has been unable to connect to a client, and a reasonable timeout elapses. (The timeout is to allow other proxies an attempt to serve the same client, in case the failed proxy is experiencing network problems.)

One unusual feature of this architecture is that the client using the proxy does not know in advance where it is connecting to. In fact, the client transport plugin accepts a dummy destination address only to comply with the pluggable transports protocol, then ignores it and tunnels to the proxy's other endpoint, namely the Tor relay. In effect, the first hop of a Tor circuit is made blindly, and then the client may choose the second and third hop from the directory of relays. Section 5.1 explains why this does not affect security (briefly, the use of Tor allows authenticating the destination after the connection is made). We also note that this is the same situation a client finds itself in when using a bridge address it has not seen before.

As a practical matter, there may be more than one facilitator, to diffuse the risk of one's being breached, and to distribute load. However we do not rely on there being multiple facilitators for the purpose of evading blocking; we assume that all facilitators will be permanently blocked by the censor. The censored clients send data only over a secure rendezvous channel whose characteristics – being very low-bandwidth, write-only, and infrequently used – make it much harder to block. Section 6 discusses potential rendezvous protocols in more detail.

4 Experimenting with Ephemeral Flash Proxies

4.1 Throughput

We measured the maximum throughput of a single proxy, independent of Tor and any other network bottleneck, by running transport plugins, facilitator, proxy, and web server all on the local host. We then started between 1 and 50 simultaneous HTTP downloads of a 10 MB file directly over TCP, through a WebSocket proxy, and through a Flash-based proxy. The test was run on the Linux kernel version 3.2 in a QEMU instance with a 2.27 GHz CPU. The proxies ran in Firefox 8.0.1 with Flash Player 11.1 r102. The proxies' bandwidth and connection limits were disabled. Figure 3 shows the time taken for each of the simultaneous clients to download the file. Each column contains multiple dots, but in the cases of Flash sockets and direct download the dots approximately coincide.

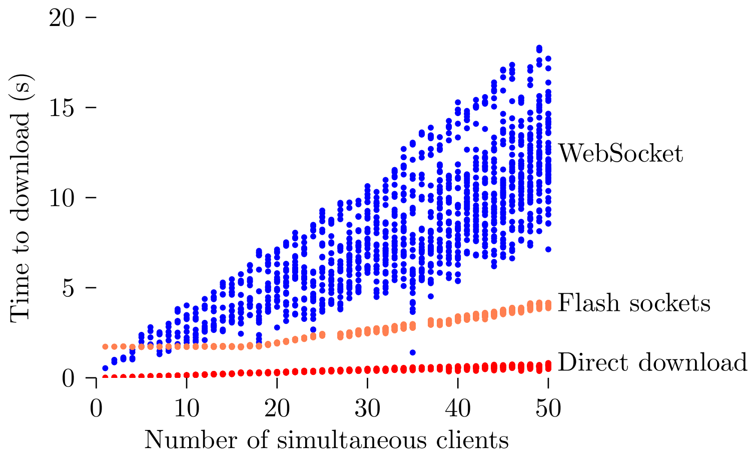

Fig. 3. Time taken to download a 10 MB file through many simultaneous connections.

Both flavors of proxy are slower than a direct download. The WebSocket measurements show download times being roughly proportional to the number of clients. There is much variance, perhaps due to unfairness in the way that the browser schedules WebSocket message events.

A surprising effect is seen with Flash sockets: the time taken to download per client is almost constant, up to about 16 clients. We suspect some kind of internal limit within Flash Player that restricts individual connections to no more than about 6 MB/s. Because of this restriction, WebSocket is faster than Flash sockets for small numbers of clients. Eventually natural limits on bandwidth become more restrictive than this artificial limit, and the download time becomes approximately linear.

The WebSocket API does not expose a "read" procedure to read from the socket; rather, one registers a "message" callback function that is called whenever data have already been read. Received packets not immediately handled by the application are buffered. At high data rates this buffer can grow without bound, possibly eventually crashing the proxy. This issue is not specific to flash proxies, and has not been a problem at Tor rates, but it leaves proxies somewhat exposed to attacks by malicious clients or relays. WebSocket provides a way to control the size of the write buffer; a complementary mechanism for the read buffer would be sufficient to solve this. Flash sockets don't have this problem as they buffer data at the operating system kernel level, so the TCP window prevents too much data from being received at once.

These results show that once connected, a flash proxy can provide more bandwidth than is commonly used by a Tor circuit (which is in the range of hundreds of kilobytes per second [16]), and even enough for online video sites. With few clients, browser-based proxies are stable and have predictable performance.

4.2 Switching Between Proxies

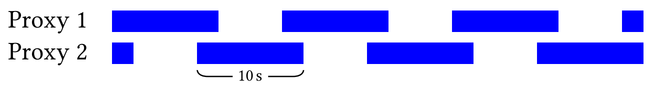

Fig. 4. Alternation of proxies. Solid bars indicate when a proxy is operating.

It is expected that a Tor client will have to switch between flash proxies frequently as they go offline. To measure the effect this has on performance, we set up two browsers running the flash proxy program, driven by a script that turned each proxy on for 10 seconds and then off for 6, so that the two proxies' periods of operation would overlap by 2 seconds in a cycle (see Fig. 4). There was always at least one proxy available, so any delay was caused purely by the need to switch between them. We did a test over the public Tor network, retrieving the same 5 MB file used by the torperf measurement tool [17]. (It is not possible to use torperf directly because it does not retry downloads.) This table shows the difference between normal Tor, an uninterrupted flash proxy, and alternating flash proxies. The direct Tor connection was configured to use the same entry bridge that the flash proxies use. Because the performance of the Tor network changes over time, all the tests were run in direct succession.

The switching experiment was run 20 times. Figure 5 shows the measurements for each test. Switching between proxies causes a visible increase in the mean time to download, but most of the variance is caused by Tor itself.

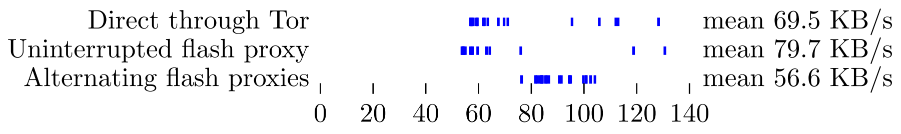

Fig. 5. Time taken for each trial in the switching experiment. Each data set contains 20 measurements.

In these results we see that a flash proxy that stays online gives performance roughly the same as connecting directly to Tor. However, overall speed is 20 to 40 percent lower when using proxies that are unreliable, because of the overhead of building new Tor circuits. The results of the throughput experiments show that a flash proxy can provide more bandwidth than Tor can use, so it is not surprising that downloading though a flash proxy is as fast as downloading over Tor. On the other hand, reestablishing a broken connection is more expensive.

In this test, the time taken to restart the download was variable, but commonly left only 3–6 seconds of useful downloading time out of each 8-second cycle.

4.3 Capacity

The flash proxy badge has several properties that, we believe, make it easy to adopt. It can be installed just by pasting an HTML snippet. It does not require any special access or configuration on the web server, nor cooperation from ISPs or the servers being proxied to. Web pages displaying the badge do not have to be in any particular network position. The sites displaying the badge may even be blocked by the censor, because it is the viewers of the site, not the site itself, that provide a circumvention bridge. In this section we seek to predict how many censored users can be helped, given a certain number of flash proxies.

Here we make some simplifying assumptions. First, we approximate the pattern of visits to a web page as a Poisson process, with different arrivals being independent, and the times between arrivals being exponentially distributed. This is not a very strong assumption; there is evidence that the request arrival process for a single web page is Poisson [18], even though the process for individual packets is not [19]. Second, we assume that traffic has reached a steady state, without edge effects from web pages appearing or disappearing, and without variability due to time of day or other factors.

Under the Poisson arrival approximation, the traffic to a single web page is governed by two parameters: λ, the mean number of arrivals per unit time; and μ, the mean visit duration. The times between arrivals are exponentially distributed with density function $\lambda e^{-\lambda t}$. The mean inter-arrival time is $1/\lambda$. We do not assume anything about the distribution of visit durations (which correspond to proxy lifetimes), other than that a mean exists and that the process generating them is strictly stationary (unchanging over time).

Different web pages have different traffic characteristics, but we may treat a group of pages uniformly using the same two parameters, for the following reasons. The exponential distribution has the property that the minimum of several exponentials with rates $\lambda_1, \ldots, \lambda_k$ is also exponentially distributed with rate $\lambda_1 + \cdots + \lambda_k$. The minimum time to the next arrival is exactly what inter-arrival time is, so the inter-arrival times from several web pages come from their own exponential distribution. The aggregate mean visit duration across several web pages is just the mean of all of their individual visit durations. The flash proxy is configured to serve up to 10 clients at a time; this may be handled by multiplying the arrival rate by 10. In this section we think of each proxy as handling only one client at a time.

If proxy badges collectively provide service with parameters λ and μ, then the expected number of operating proxies (and hence the number of clients that can be served) at any time is $\lambda\mu$ (which is Little's law [20]). For example, if 3 proxies come online every second ($\lambda = 3$) and each lasts 60 seconds on average ($\mu = 60$), then we expect 180 proxies to be operating at once on average.

Let us substitute some measured numbers into this formula. We wrote a program to record the visits of viewers capable of running Flash, and installed it on a personal home page for about two weeks. For each visitor, the program recorded (*start-time*, *end-time*, *bandwidth*, *latency*). The program recorded 784 visits. we discounted the two longest-lived connections of approximately 1.9 and 3.1 days (the next longest connection after those was 15 hours). We also ignored 28 connections that had a measured bandwidth of 0, and an apparent outlier that came much later than any other connection. This left 753 entries.

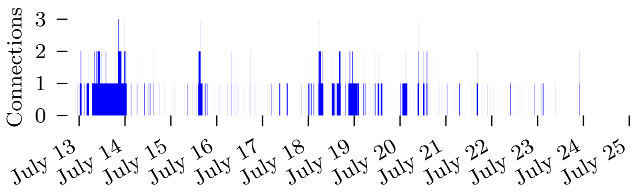

Fig. 6. Measured simultaneous connections on a personal home page.

Figure 6 shows the number of simultaneous visitors to the web page for each day of the experiment. There were at times as many as three visitors viewing the page at once, but only about 17% of the time was there even one visitor. This web page acting alone would not be able to provide good proxy service, because of the long periods during which there are no visitors and hence no flash proxies. It would take several such pages working together to fill in the gaps and provide continuous service with high probability.

We estimate the quantities λ and μ by taking the sample means of inter-arrival time and visit duration, respectively:

Mean inter-arrival time	$1/\lambda$	1407.6 s
Mean arrival rate	λ	0.00071/s
Mean duration	μ	285.8 s

If 100 web pages like the one we tested were to install the flash proxy badge, then we estimate an overall arrival rate of $\lambda' = 100 \cdot 10 \cdot \lambda \approx 0.71$ per second (recall that one proxy can handle 10 simultaneous clients). The combined mean visit duration is unchanged: $\mu' = \mu$. The expected number of clients that these 100 web pages can support is $\lambda'\mu' \approx 203$.

The number of clients scales linearly with the number of proxies, so 1,000 web pages with similar traffic characteristics would be able to support 2,030 clients on average, with the same expected duration. Pages with more visitors (lower inter-arrival time) or longer visit durations will provide better service.

4.4 Field Testing

We tested connecting to the Tor network from within China over flash proxies in December, 2011. We were running one proxy, and a few others were run by unknown web users. The proxies worked as expected, and we could use Tor despite its well-known blocking by China. This test used only a simple HTTP-based rendezvous, and not any of the advanced rendezvous methods from Sect. 6, so it could have been blocked by IP address. Nevertheless, the test shows that the proxies work as they are supposed to, once the rendezvous step is completed.

5 Discussion

5.1 Security and Privacy

Flash proxies can in principle be used to reach any kind of tunnel, such as an HTTP proxy or SSH tunnel. Tunneling through Tor, however, brings attractive security features. The most obvious is enhanced anonymity: It is not easy for a network observer, including the flash proxy itself, to know the final destination of traffic. Users who would like to use Tor for circumvention, but cannot because Tor relays and bridges are blocked, therefore do not have give to up anonymity when using a flash proxy transport.

A second feature provided by Tor is encryption. Once a proxy has connected to its two endpoints, it sends and receives only ciphertext. This is important not only for the client, but for the temporary operator of the proxy. It would not be friendly toward proxy operators to allow them to send plaintext that could potentially run afoul of a corporate firewall, for example. Note, however, that encryption can also be obtained by directing flash proxies to forward traffic to an SSH server in the open Internet, in which case the client runs an SSH client instead of a Tor client.

The third feature is authentication. Even though a malicious flash proxy can connect to any endpoint it wants, it cannot in this way trick a client into connecting somewhere unexpected. The most it can do is deny service. If the proxy connects to something that is not a Tor relay, the Tor client will fail to make a connection and show error messages. Once a connection is made to the first relay, the proxy cannot interfere with the client's circuit construction because it is already within a layer of encryption. A flash proxy operated by a malicious adversary gets to choose the first relay, so it could always choose a relay it controls, and thereby always see the first hop of a circuit. However, such an adversary can already do something similar simply by running its own bridges and waiting for connections from ordinary bridge users.

How might the flash proxy system be attacked? In our threat model, the adversary "wins" if it is able to prevent access to sites that it would block normally. The adversary can also control some fraction of all flash proxies and relays. Here we list a number of attacks and possible mitigations.

Client enumeration. The censor can query the facilitator and get a list of IP addresses of presumed circumventors. This is a consequence of the fact that flash proxies connect to censored clients, instead of the other way around. Note however that the adversary is already in a position to learn the addresses of users of a circumvention tool, just by sniffing at the firewall, but having a centralized facilitator makes it easier. One possible mitigation of this attack is sheer numbers. If there are many more proxies than there are clients, then most proxies will not have any client to serve at all. (The facilitator will just return “no client address” when queried.) The adversary will be competing against all legitimate flash proxies in querying the facilitator to learn client addresses, and most attempts will be fruitless. With successful deployment we expect to have millions of available proxies at any given time meaning that most, including many of the adversary’s, will be idle.

Flooding client registrations. The facilitator accepts client registrations from anywhere as a consequence of the indirect rendezvous mechanism. An adversary can flood the facilitator with fake client addresses. When a legitimate proxy retrieves one of these fake client addresses, it will waste time trying to give service to that presumed client. This temporarily removes an otherwise useful proxy from the system. It should be noted, though, that legitimate registrations will still get through and will eventually be picked up by a proxy. Mitigating this attack requires enough proxies to absorb the busywork created by the adversary. It is also possible to limit the number of registrations accepted over a period of time from a given source address.

Exhausting client registrations. The adversary can pose as multiple flash proxies to the facilitator, and ask for addresses of clients, which it then ignores. The aim is to cause the facilitator to think that the client address has been given to enough proxies that it need not be given out any more, and prevent legitimate proxies from seeing the address. This too is mitigated by numbers. If there are enough legitimate proxies, the adversary will have difficulty claiming all the registration “slots” for a particular client. As long as one of the proxies is legitimate, the client will be able to get service.

Protocol fingerprinting. The fact that they run through a browser means that flash proxy tunnels look different from ordinary TCP connections at the network level. With WebSocket, there is the HTTP handshake to begin the connection, followed by data in a structured framing format. Flash sockets are distinguishable because they begin with a crossdomain policy request. Even if obfuscation is used to hide the Tor protocol, it must all happen within the framework provided by browser sockets. The unblockability of the system rests in part on the type of connections used by the proxies (e.g., WebSocket) also carrying enough ordinary traffic that the censor will be reluctant to block that type of connection wholesale. It remains to be seen whether WebSocket will have this level of popularity, but support for WebSocket in major browsers is a promising trend.

5.2 Usability

Usability is listed as a security requirement in the design of Tor [2]. In this section we examine how much additional effort is required to use flash proxies on the part of users and relay operators, and how this affects usability.

Relay operators. On the part of Tor relay operators, the only additional requirement is to run the flash proxy server transport plugin. This is a matter of installing the plugin and adding a line to the relay's configuration file.

Clients. Our programs are designed to work with the Tor pluggable transports design, so configuration is fairly easy. A user must run the transport plugin, add a few lines to the Tor configuration file, and then be able to register with the facilitator. Our transport plugin is written in Python, which is an additional requirement beyond plain Tor. If the user is not able to receive direct TCP connections, the more technical step of enabling port forwarding must be taken. A Windows installer can automate this process and we plan to build one as interest increases.

Limitations. The fact that proxies are expected to disappear causes a qualitative difference in network behavior when using flash proxies. These differences vary in importance depending on the application being used. Basic web browsing, with relatively short-lived connections, works quite well. When connections are short, there is a smaller chance that they will be interrupted. When they are interrupted, fixing a partially downloaded page only requires refreshing the page. Some browsers, such as Firefox and Chrome, automatically restart failed downloads making this seamless. Large web downloads and video work less well, because it is more likely that a download will be interrupted in the middle. Again, browsers that automatically restart failed downloads from the point of failure make this less of a problem.

The fact that clients must not be behind NAT is an impediment to usability. A NAT traversal mechanism that works within our threat model would be a great benefit. Typical NAT traversal technologies, such as STUN (Session Traversal Utilities for NAT) [21] and RTMFP (Real Time Media Flow Protocol) [22], rely on a stable third-party server to facilitate the connection, which is trivially defeated by the censor blocking the third party by IP address. (Also we believe it is better to avoid informing a third party of each flash proxy connection if it can be avoided.) Tricks involving low-level packet manipulation, for example pwnat [23], are not available to browser sockets. Ideally, any NAT traversal scheme will not require both the client and the proxy to know each other's IP address, so that facilitator registration can remain unidirectional. New technologies like WebRTC [24] may fill this need in the future, if they become sufficiently popular that flash proxies' use of them does not stand out as unusual.

Applications that inherently rely on long-lived connections, e.g. SSH, have a poor user experience. The session ends completely whenever a proxy goes offline, losing state and requiring a new login. For SSH, the most important property of a proxy is long lifetime, which flash proxies in the wild usually do not provide. On the other hand, protocols that use long connections but do not maintain much server-side state can still work tolerably well with enough application support. For example, an Internet Relay Chat (IRC) client that automatically reconnects after losing its proxy server will be usable with only brief interruptions. The flash proxy system can be extended with buffers in the transport plugins, to enable connection continuity across different flash proxy sessions.

5.3 Deployment Scenarios

Our proposal hinges on volunteer web pages hosting the flash badge. Recall that while the badge is running in the visitor's browser it has no impact on the visitor's experience at the site. Nevertheless, web site admins will likely want to configure the badge to best suit their objectives. For this reason the badge is highly configurable for different scenarios:

- **Opt-in vs. opt-out:** Sites that maintain user accounts (e.g., Facebook) can add a check-box to user profiles allowing users to specify whether they want to participate in this system or not. The Flash object will only be served to users who check the box. The default settings for the checkbox is a matter of policy. Moreover, the badge itself can be configured to only begin proxying after the visitor clicks on the badge.
- **Geographic limitations:** The badge can be configured to only serve clients in certain geographies.
- **Proxy targets:** If Tor is not used, the badge can be configured to only proxy to specific domains such as YouTube and Facebook.
- **Connectivity:** The badge is already configured to shut down when running on a mobile device so as not to use up the host's data plan. It can similarly be configured to shut down when it detects a low-bandwidth connection so as not to interfere with the host's browsing experience.

We envision two types of deployments. Commercial sites that are already blocked, such as YouTube, can deploy the badge on their home page so that their uncensored users can help their censored users reach the site. In these deployments, the badge will only forward traffic to YouTube, and possibly only from regions where YouTube is known to be censored.

Another type of deployment can come from people who are concerned about Internet filtering and choose to deploy the badge on their home page and blogs. Visitors to those pages will help censored users connect to Tor. People who choose to deploy the badge on their blogs can customize it as they wish, possibly serving clients only in certain geographies or forwarding traffic only to certain domains.

6 Rendezvous Protocols

The flash proxy system relies on a robust rendezvous mechanism that lets clients in the censored region register their IP address with the facilitator. If the censor could simply block the facilitator then the flash proxy system would break down.

The flash proxy rendezvous problem is related to, but somewhat different from the rendezvous problem in Tor. In the Tor system, rendezvous is used to communicate the address of an unblocked Tor bridge *into* the censored region. In the flash proxy system rendezvous is used to communicate the address of a client *out of* the censored region.

Our facilitator design is reasonably modular, so that the facilitator itself does not need to understand all of the potential rendezvous methods, or know in advance which will be used. The censored client runs a program implementing a certain rendezvous method; an uncensored recipient that understands the protocol then forwards the registration to the facilitator on the client's behalf. In this way, new rendezvous methods can be tested without redeploying the facilitator, and without requiring the cooperation of those who run the facilitator.

We experimented with two methods for communicating a small amount of information out of the censored region. The first uses cloud-based storage servers outside the blocked region. The second uses cooperating web sites. We note that many other systems, including Skype and Telex [3] can be used for rendezvous.

Storage servers. Cloud storage systems like S3 and Box.net provide a good opportunity for rendezvous. These services are difficult to block due to large collateral damage caused by blocking them. We experimented with a system that uses a variety of such cloud storage systems. As long as one system among many is unblocked, the information will get out.

The idea is that the facilitator signs up for an account on all cloud storage servers that the system will use. It sets permissions so that anyone may write to the storage server, but only the facilitator may read from it. Clients in the censored region who want service write their IP address to all the storage servers over HTTPS and the facilitator retrieves them by periodically polling the servers.

Our experiments showed that this method has high latency. For example, it can take several seconds for data written to S3 in one geographic location to become available for reading in another location. Hence, while this channel is insufficient for general network access, it is fairly well suited for rendezvous.

Web sites. Our second approach, which is more speculative, is to embed rendezvous messages in standard HTTP requests.

A client wishing to register with the facilitator would send an HTTP request to a participating web server where one of the HTTP headers (e.g., a cookie header) contains a crafted random string. The web server would be configured with a secret key that lets it recognize the pattern in the HTTP request and forward the request to the facilitator. The censor, who does not have the secret key, cannot recognize that the request encodes a flash proxy registration request. If many web servers participate in this scheme then the censor's only hope for blocking these messages (besides blocking all web sites) is to attempt to compile a list of cooperating sites to block, by crawling the web and trying to use each site to rendezvous with the facilitator. If many web sites participate in this scheme, then blocking all of them will cause considerable collateral damage.

To experiment with this approach, we modified the Apache web server to accept rendezvous requests for the flash proxy system. This modification could be packaged as an Apache module for ease of deployment. Users might discover participating web sites through word of mouth or social media; or if there are enough of them, a browser plugin could discover them automatically during the course of normal web browsing. When the client wishes to register with the facilitator it chooses a random web site that is participating in this rendezvous mechanism. Next, it encrypts the message ($0^{32} \| \text{IPaddr}$) with that server's public key and embeds the resulting ciphertext in an HTTP request sent to the web server. The ciphertext is embedded in a session cookie header that is normally used when interacting with this web site. The censor cannot tell the difference between a real session cookie and a rendezvous request, since both appear to be random strings. The client uses this header in a request for a page that does not exist (which can be generated by choosing a random string or a random English word). When the web server receives an HTTP request with a session cookie that results in a 404 response, the server tries to decrypt the contents of that session cookie. If it detects the 0^{32} pattern, it forwards the encoded IP address to the facilitator. The server sends a 200 response so that the client knows that the rendezvous request was successful. Note that the public-key decryption step is only used on HTTP requests that result in a 404 response.

In our prototype, we used identity-based encryption (IBE) [25], in which a server's public key is its domain name. With IBE, clients do not need to use some other mechanism to learn a server's public key before using it for rendezvous.

7 Related Work

Blocking resistance is an ongoing area of research. It appears that there is no single, simple solution, but rather many different problems must be solved together for effective blocking resistance.

Simple proxy systems suffer from the problem that they typically reside on static IP addresses and are expected to be relatively long-lived, making them easy to block. It will always be a race between users striving to find new usable proxy addresses and the censor to block them. Additionally, single-hop proxies cannot provide the same security properties that Tor can; for example they may be susceptible to subversion of the proxy server itself.

Infranet [26] is a design to conceal traffic that would otherwise be blocked within seemingly normal HTTP traffic. A user's ordinary innocuous browsing provides cover for messages sent and received to some other blocked server. Infranet requires the cooperation of unblocked web servers. We note that Infranet's covert channel could be used as a method of rendezvous.

In some cases, the mere presence of statistical anomalies in traffic, such as an encrypted stream, could be enough to cause a censor to block access to a resource, merely on suspicion that the opaque stream may be used for circumvention. This illustrates the need for protocol camouflage, making circumvention traffic look like "normal" traffic, at least from the point of a censoring firewall. The Tor Project made a proposal for pluggable transports [13], which allow using a variety of different camouflage or circumvention techniques, depending on what is mutually supported by the client and entry node.

Recently Wustrow et al. introduced Telex [3], a system that allows cryptographically "tagging" normal TLS streams so that an ISP-level router may redirect it to a blocked destination. Telex is different from other proposals in that it involves action by entities in the middle of the network path, not only at the edges. Unlike with Infranet, unblocked web sites do not need to know about or participate in the circumvention.

The Tor Project has enhanced blocking resistance by introducing non-public bridges in addition to public relays [5]. Bridge addresses go not into the main relay database, but into a special bridge database. The set of bridges is partitioned and each partition is distributed over a different channel, for example email, HTTPS, or non-electronic means. The bridge database hands out only a few bridge addresses at a time, with some additional restrictions making it hard for one user to learn many or all of them. The goal is to prevent a scarce and precious resource (bridge addresses) from being completely enumerated, while still allowing anyone to learn the few addresses they need to get connected.

8 Conclusions

We have introduced flash proxies, a new method of producing many short-lived proxies that for the purpose of censorship circumvention. Rather than attempting to hide the addresses of proxies, we aim to create so many that it is not feasible to block them all. Performance experiments are promising and the system is ready to be deployed. The flash proxy badge is installed on the project home page, and visitors are already acting as flash proxies. There are already sufficiently many active flash proxies to provide intermittent service. The project code is open source and available from `https://crypto.stanford.edu/flashproxy/`.

Acknowledgments

We are grateful for many helpful conversations on this topic with Drew Dean, Pat Lincoln, Ian Schuler, and Vinod Yegneswaran; and to Steve Beaty for help in testing.

The work is supported by the Defense Advanced Research Project Agency (DARPA) and Space and Naval Warfare Systems Center Pacific under Contract No. N66001-11-C-4022. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Defense Advanced Research Project Agency and Space and Naval Warfare Systems Center Pacific. Distribution Statement "A:" Approved for Public Release, Distribution Unlimited.

References

1. The OpenNet Initiative: OpenNet Initiative Internet censorship data. `http://opennet.net/research/data` (November 2011)
2. Dingledine, R., Mathewson, N., Syverson, P.: Tor: The second-generation onion router. In: Proceedings of the 13th USENIX Security Symposium. (August 2004)
3. Wustrow, E., Wolchok, S., Goldberg, I., Halderman, J.A.: Telex: Anticensorship in the network infrastructure. In: Proc. 20th USENIX Security Symposium. (2011)
4. Ultrareach Internet Corp.: Ultrasurf proxy. `http://www.ultrasurf.us/`
5. Dingledine, R., Mathewson, N.: Design of a blocking-resistant anonymity system. Technical Report 2006-1, The Tor Project (November 2006)

6. Kadianakis, G., Mathewson, N.: Obfsproxy architecture. `https://www.torproject.org/projects/obfsproxy` (December 2011)
7. Weinberg, Z., Wang, J., Yegneswaran, V., Briesemeister, L., Boneh, D., Wang, F.: StegoTorus: A camouflage proxy for the Tor anonymity system
8. Tor Metrics Portal: Users. `https://metrics.torproject.org/users.html` (2011)
9. Lewman, A.: Tor partially blocked in China. `https://blog.torproject.org/blog/tor-partially-blocked-china` (September 2009)
10. McLachlan, J., Hopper, N.: On the risks of serving whenever you surf: Vulnerabilities in Tor's blocking resistance design. In: Proceedings of the Workshop on Privacy in the Electronic Society (WPES 2009), ACM (November 2009)
11. Wilde, T.: Knock knock knockin' on bridges' doors. `https://blog.torproject.org/blog/knock-knock-knockin-bridges-doors` (January 2012)
12. Winter, P., Lindskog, S.: How China is blocking Tor. Technical report, Karlstad University (April 2012)
13. Appelbaum, J., Mathewson, N.: Pluggable transports for circumvention. `https://gitweb.torproject.org/torspec.git/blob/HEAD:/proposals/180-pluggable-transport.txt` (October 2010)
14. W3C: Cross-origin resource sharing. `http://www.w3.org/TR/cors/` (April 2012)
15. Adobe Systems Incorporated: Adobe Flash Player 9 security. `http://wwwimages.adobe.com/www.adobe.com/content/dam/Adobe/en/devnet/flashplayer/pdfs/flash_player_9_security.pdf` (July 2008)
16. Tor Metrics Portal: Time in seconds to complete 5 MiB request. `https://metrics.torproject.org/performance.html` (2012)
17. Loesing, K.: torperf measurements-HOWTO. `https://gitweb.torproject.org/torperf.git/blob_plain/HEAD:/measurements-HOWTO` (2011)
18. Arlitt, M., Williamson, C.: Internet web servers: workload characterization and performance implications. IEEE/ACM Transactions on Networking **5**(5) (Oct 1997)
19. Paxson, V., Floyd, S.: Wide area traffic: the failure of Poisson modeling. IEEE/ACM Trans. Netw. **3** (June 1995) 226–244
20. Little, J.D.C.: A proof of the queuing formula $L = \lambda W$. (1960)
21. Rosenberg, J., Mahy, R., Matthews, P., Wing, D.: Session Traversal Utilities for NAT (STUN). RFC 5389 (Proposed Standard) (October 2008)
22. Adobe Systems Incorporated: Real Time Media Flow Protocol. `http://labs.adobe.com/technologies/cirrus/` (October 2008)
23. Müller, A., Evans, N., Grothoff, C., Kamkar, S.: Autonomous NAT traversal. In: 10th IEEE International Conference on Peer-to-Peer Computing (P2P). (2010)
24. W3C: WebRTC 1.0: Real-time communication between browsers. `http://dev.w3.org/2011/webrtc/editor/webrtc.html` (January 2012)
25. Lynn, B.: PBC library. `http://crypto.stanford.edu/pbc/`
26. Feamster, N., Balazinska, M., Harfst, G., Balakrishnan, H., Karger, D.: Infranet: Circumventing web censorship and surveillance. In: Proceedings of the 11th USENIX Security Symposium. (2002)