Frederick Douglas*, Rorshach, Weiyang Pan, and Matthew Caesar

Salmon: Robust Proxy Distribution for Censorship Circumvention

Abstract: Many governments block their citizens' access to much of the Internet. Simple workarounds are unreliable; censors quickly discover and patch them. Previously proposed robust approaches either have non-trivial obstacles to deployment, or rely on low-performance covert channels that cannot support typical Internet usage such as streaming video. We present Salmon, an incrementally deployable system designed to resist a censor with the resources of the "Great Firewall" of China. Salmon relies on a network of volunteers in uncensored countries to run proxy servers. Although any member of the public can become a user, Salmon protects the bulk of its servers from being discovered and blocked by the censor via an algorithm for quickly identifying malicious users. The algorithm entails identifying some users as especially trustworthy or suspicious, based on their actions. We impede Sybil attacks by requiring either an unobtrusive check of a social network account, or a referral from a trustworthy user.

Keywords: Censorship

DOI 10.1515/popets-2016-0026
Received 2016-02-29; revised 2016-06-02; accepted 2016-06-02.

1 Introduction

The Internet has proven to be an extremely powerful tool for enabling the free flow of information. Authoritarian governments have always been highly concerned with the control of information, and so they perceive the Internet in its natural form to be a grave threat. They seek to eliminate dissemination of inconvenient facts, to stifle free discussion of viewpoints different from their own, and especially to prevent the coordination of mass action. This last point has become a particular concern in the past few years, as social media platforms have proven excellent for coordinating protests.

As a result, many countries impose restrictions on their citizens' Internet use. These restrictions can take the form of dropping packets to/from particular IP addresses, disrupting resolution of DNS queries, and even inspecting packets headed to search engines to filter out undesirable queries. Any attempt to circumvent censorship must defeat all of these techniques. The simplest solution is to tunnel all traffic through an encrypted connection to a secret proxy server: the secure tunnel hides destination IP addresses, and guards against DNS interference and deep packet inspection. The vulnerability is then the server's identity: the server is viable for bypassing the censorship only so long as the censor does not know there is a proxy server at that IP address.

Salmon starts with the simple approach of assembling a collection of proxy servers, and fixes the vulnerability with an algorithm for distributing those proxy servers to the general public, while minimizing the number of servers a censor whose agents control some user identities can discover and block.

Salmon's algorithm has three core components. 1) We track the probability that each user is an agent of the censor (*suspicion*), and ban likely agents. 2) We track how much *trust* each user has earned, and distribute higher-quality servers accordingly. Trust also helps keep innocent users out of trouble, by isolating them from newer users, where the censor agents are more likely to be found. 3) Highly trusted users are allowed to *recommend* their friends to Salmon. We maintain a social graph of user recommendations, and assign members of the same connected component — a recommendation ancestry tree — to the same servers, whenever possible.

Other works [16, 22] have employed techniques with the same overall theme of gradually identifying and punishing suspicious users. Salmon is more robust than other approaches, thanks mainly to its trust levels. Our simulations show that the addition of our trust level logic significantly reduces how many users the censor can cut off from access to proxy servers. rBridge [22], the most robust previous server distribution technique,

***Corresponding Author: Frederick Douglas:** University of Illinois Urbana-Champaign, fed2@illinois.edu
Rorshach: rorshach.d@gmail.com
Weiyang Pan: University of Illinois Urbana-Champaign, pan30@illinois.edu
Matthew Caesar: University of Illinois Urbana-Champaign, caesar@cs.illinois.edu

allows over three times as many users to be cut off from the system as Salmon (§5.5).

Salmon's other main advantage over similar systems lies in striking the proper balance between making the system easily accessible to the general public, and making it difficult for the censor to insert many of its agents into the system. Previous proposals have gone a bit too far in one of those directions. In addition to recommendations, Salmon can optionally accept new users who can prove ownership of a well-established Facebook account. If Facebook is blocked, existing robust but low-bandwidth techniques are sufficient for getting the user through Salmon's registration process. The system can therefore be made open even to people who do not know any longtime Salmon users.

Salmon's name comes from its trust levels: just as salmon swimming upstream must hop up small waterfalls, and might briefly fall backwards, users need to advance up the trust levels. As a bonus, 'salmon' can be translated to Persian as 'free (as in freedom) fish'!

2 Salmon

2.1 Threat model

We assume that the censor is a branch of a national government, able to take full control of any router or server inside the country. We also assume that the censor is content to merely block access, rather than seek out and arrest those who attempt to access forbidden material. If the government did decide to carry out large-scale arrests against our users, they could easily do so by contributing many servers to our system as honeypots, and recording the users they receive. There is no good solution to this problem, short of manually verifying server volunteers. Even using onion routing will not prevent the censor from observing that specific citizens are evading censorship.

We assume the censor can employ large amounts of human labor. A task that requires tens or hundreds of full-time employees is within the capabilities of our censor. The Chinese government already has a much larger force monitoring posts on Chinese social media sites such as Weibo and Renren [1].

In summary, we assume:

– The censor has enough employees to perform labor-intensive tasks, such as examining a list of servers.
– Our users are not personally targeted by the censor.
– The censor cannot identify proxies via traffic fingerprinting. Anti-fingerprinting is important and has been studied [18, 21, 23], but is orthogonal to server distribution.
– The censor may try to block as many servers as it can immediately, or may be willing to simply discover servers, and not take action for months.

To reiterate, we are assuming that the censor cannot identify proxies via traffic fingerprinting *only* because proxy distribution is not an interesting question if the censor has this capability. Careful proxy distribution schemes are useful when the censor can block circumvention resources if and only if it receives the same information that a legitimate user needs to utilize them. If the censor can identify circumvention traffic, it has no reason to bother attacking the distribution mechanism. Both the fingerprinting problem and its solutions are generic to most circumvention systems.

2.2 Problem statement

Salmon is built to solve a single deceptively simple problem. We are in possession of some sensitive information that we wish to distribute to a large group of users, most of whom are strangers to us. Mixed in among those users are agents of an adversary — in our case, the censor. The adversary can use the secret information to damage us, and so tries to learn as much as possible. When the adversary inflicts that damage, the fact that they have learned the information is revealed to us. However, we do not know *how* they learned it.

In terms of censorship circumvention, we have:

– We want to give proxy server IP addresses to any interested user living under censorship.
– The censor's employees can pose as ordinary users.
– A censor can choose to block any proxy it discovers.
– Such a block reveals to us that the censor somehow learned there was a proxy server at that address.

This problem's difficulty comes from the requirement to keep the system open to all potential users, combined with censors' ability to behave identically to good users for as long as they wish. In fact, it's clear from these two facts that a perfect solution — one that keeps all servers from being blocked — is impossible: until the censor blocks a server, its agents can remain indistinguishable from good users.

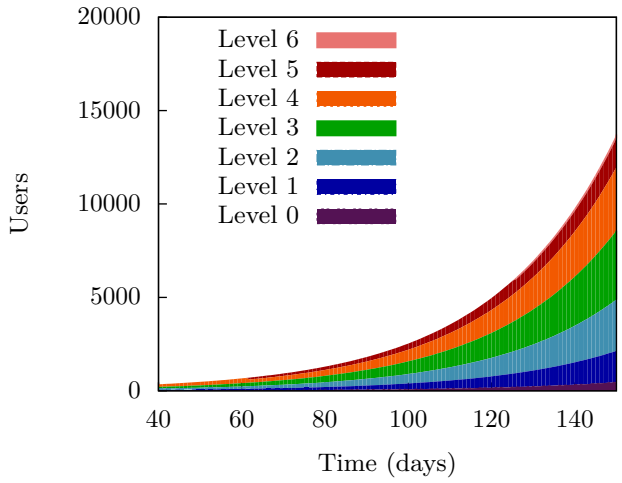

Fig. 1. Evolution over time of the proportions of users in different trust levels. Every day, for every 30 users, one new user enters the system at level 0.

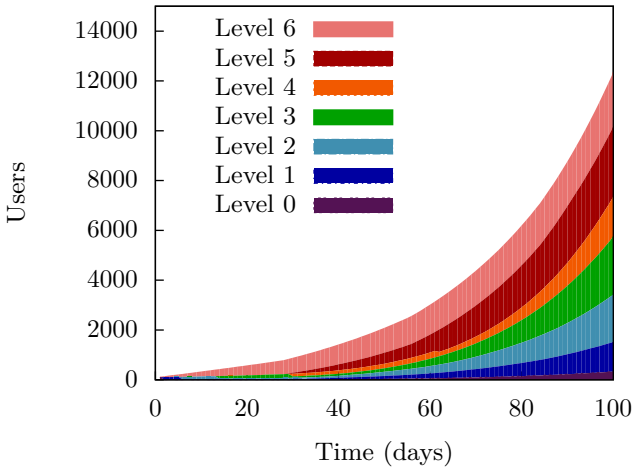

Fig. 2. Similar to fig. 1, but the system launches with 20 special users (friends of the administrators), who are considered to be above the trust hierarchy, and can recommend one level 6 user per day. Level 6 users recommend one level 5 user every four weeks.

2.3 Algorithm

So long as we open our system to the general public, a perfect solution is impossible. Perfection is not required, however: a censor that can only block 10 out of 1,000 servers has certainly not accomplished much. Our goal is to limit the censor's ability to block servers. Blocking servers is simply the tool the censor uses to work towards its actual goal: denying our users free Internet access. Therefore, we perform all evaluations in terms of how many users remain able to access one of our servers in the wake of the censor's attack. We now describe the algorithm.

Salmon's algorithm has three core components. 1) We track the probability that each user is an agent of the censor (*suspicion*) and ban likely agents. 2) We track how much *trust* each user has earned, and distribute higher-quality servers accordingly. 3) We maintain a social graph of user *recommendations*; when assigning servers, we group together members of the same connected component.

Suspicion: The censor's weapon is its ability to hide among real users while damaging the system. When three users are the suspects for a blocked server, we have no choice but to say they each are equally likely to be the culprit. That is, in a block event on a server with n users, we consider each user to be innocent with probability $\frac{n-1}{n}$. Note that we are defining "users of the server" to include every user we have given the server's address to; no matter when they last connected, a user never "leaves" a server.

Salmon's algorithm for weeding out agents builds this information into something less uncertain: the probability that a user is not an agent is the product of its probabilities of innocence from every block event it has been involved with. Throughout the paper we use the more concise term "suspicion" — the complement of the probability of innocence. We permanently ban any user whose suspicion exceeds the threshold of suspicion, T.

Trust levels: In a system like Salmon, some users can reasonably be considered more trustworthy than others: most obviously, friends and friends-of-friends of the people running the system. Additionally, a user who has known a proxy address for months without the address getting blocked seems less likely to be an agent. We base this heuristic on the censor's optimal strategy: as we will demonstrate, the censor does not benefit (in terms of maximizing users without access to Salmon) from waiting long periods of time before beginning to block the servers its agents have collected.

Salmon quantifies the concept of trust with discrete *trust levels*. All entities in the system — both users and servers — live in a single specific trust level at any given time. When the system assigns a server to a user, it will only choose a server of the user's trust level. Users can move up and down over time. Servers only move up.

Level 0 is the entry trust level. Users not recommended by a trusted user start here. There are negative levels to accommodate level 0 groups who experience a server block, extending as far down as a user could possibly fall before being banned: users are banned solely

on high suspicion, not low trust. The levels reach up to a maximum of L, a parameter to be decided upon.

A user loses a trust level every time a server whose address it knows is blocked. A user gains a trust level if it goes for a period of time without its assigned server being blocked. That period doubles every level: promotion from level n to $n+1$ takes 2^{n+1} days, *e.g.* promotion to level 6 takes over two months. (Levels ≤ 0 all take one day). We chose the exponential durations to strike a balance between quickly lifting good users out of the less desirable bottom levels, and protecting the highest levels. As we describe in the next subsection, forcing the censor to endure hefty delays is a very effective defense, especially given that we allow users to join via recommendation. We chose maximum level $L = 6$ as a good compromise in the same tradeoff.

A server's trust level is the minimum trust level of its assigned users. For instance, if a new server is assigned to a level 2 user, the server has been chosen to start at trust level 2. If another level 2 user is assigned to the server, and then the first user attains level 3, the server remains at level 2. Then, if the second user attains level 3 before any other users are assigned to the server, the server rises to level 3. Of course, a server never loses trust levels, because its users would only lose a trust level if it were blocked!

The trust levels benefit higher-trust users in two ways. First, servers are not equal. Although we require a minimum bandwidth (100KB/s) from our volunteers, more generous volunteers might provide more. For instance, our own servers each offer about 1.5MB/s. Salmon assigns faster servers to more trusted users.

Second, users at higher trust levels are better isolated from censor attacks. Only a censor willing to leave servers unblocked for months at a time, or able to obtain a recommendation from a trusted user, will get access to servers at high trust levels. Our innocent high trust users are therefore less likely to experience server block events, and to be banned.

Recommendation graph: To help the trust level logic give high quality servers to as many deserving users as possible, Salmon incorporates a recommendation system: a user at maximum trust level L may recommend friends to Salmon; these friends join at level $L-1$. Users whom Salmon axiomatically assumes trustworthy (*i.e.*, friends of the directory server admins) are sent to a special level outside of the hierarchy, which is attainable only in this manner. They can recommend other users to level L, and are allowed to recommend after a much shorter delays than level L users: once per D_* days vs. once per D_L days. We have chosen $D_* = 1, D_L = 30$, as we describe in §3.6.

We must be careful about allowing such recommendations in the system we have described thus far: recommendations give a patient censor the ability to exponentially grow a collection of agents at high trust levels. The censor could reach practically any agent/user ratio it desired. There are two ways to deal with the censor recommendation issue.

First, and most simply, we can delay the censor. Getting significant exponential growth from the recommendation system takes several months: the first wave of users must wait over four months to recommend the second wave; the second and all subsequent waves must wait over two months to begin recommending. We expect that our volunteer servers can change IP addresses somewhat easily — easily enough that many would be willing to do so once every several months, but certainly not as frequently as *e.g.* once a week. We discuss this defense further in §3.5.

In addition to recovering from a recommendation-based mass block, we can take prophylactic action. Users want to avoid being grouped with agents, so they should naturally want to be grouped with their recommend{er,ee}s: real-world friends whom they trust not to be agents. We use the following logic: when assigning a new server to a user u, we first look for one that has already been given to a member of its connected component — always a tree, so call it $T(u)$ — in the recommendation graph. It is natural to group users in this way, as friends could easily share proxy login credentials among themselves regardless of our logic. Indeed, a design discussed later (Proximax [16]) is built entirely upon this fact. Because this sharing of servers among friends is so natural, this logic takes precedence over the trust levels: a user may be placed on a higher-level server if it includes a recommendation friend.

If there are no such servers with room left, we instead choose a server with enough free slots for all of $T(u)$. Enough slots on the chosen server are reserved to ensure that the rest of the users in the tree can join later. Let M be the maximum users allowed in a group. If $|T(u)| \geq M$, only a fresh, unassigned server will be used. Helpfully, this means that a group of M or more friends who build an agent-free recommendation tree receive servers guaranteed to be agent-free.

Just as the trust levels keep innocent, highly trusted users isolated from impatient censor agents, the recommendation graph tends to keep agents who were created by recommendation in groups with themselves. Both mechanisms share a fundamental purpose: keeping cen-

sor agents as tightly grouped together as possible. The more evenly the agents can spread throughout the system, the more servers they will discover.

It is possible that some users might wind up in the same recommendation tree as an agent, *e.g.* if the censor hands out free recommendation codes in an Internet forum. Users should be careful about the source of their recommendations. This scenario does not spell guaranteed banning for the user u, however; it simply makes them more likely to be grouped with an agent, depending on what fraction of users in $T(u)$ are agents.

Summary: Users are given one server at a time. The server is shared among a group of users. If a user's server gets blocked, we *trust* the user less, and *suspect* them more. Suspicion is used to decide on bans for users. Trust is used to reward reliable users with better servers, and to keep them safe from the collateral damage of an impatient censor. Users gain trust if the servers they have been given remain unblocked for long periods of time. Highly trusted users can recommend a limited amount of friends to become highly trusted. In the undirected graph of recommendations, members of a connected component are given the same servers.

3 Implementation

The previous section presented the theoretical design of Salmon: the algorithm for careful proxy distribution. This section deals with the real-world details of our implementation: the architecture of the system's software components, restricting account creation with Facebook, our choice of values for the tunable parameters present in our design, and a possible attack stemming from a practical issue not included in the threat model.

3.1 Components

Our implementation of Salmon comprises three components: a Windows client program for the users ($\sim$ 4,300 C++ SLOC, as counted by the CLOC tool, excluding GUI code); a server daemon for the volunteers (Linux and Windows; $\sim$ 1,400 C SLOC for the Linux version); and a central directory server to keep track of servers, and distribute their IP addresses to users ($\sim$ 2,700 D and 1,000 C++ SLOC, using MongoDB for storage). All of the components have been released under the GPLv3, and are available at https://github.com/SalmonProject.

The entirety of the algorithm described in this paper is carried out by the directory server. Since the censor can easily block the directory server, the client program communicates with it via any commonly used encrypted email service that the censor does not block (or control). The email address used to communicate with the directory server serves as a Salmon account's identifier.

Underneath our own client and server programs, we run SoftEther VPN. SoftEther is mature, and comes tested for censorship circumvention by VPN Gate [19]. Throughout the paper, we discuss distributing "proxy server addresses." In reality, the directory server also provides the Salmon client with the server's X.509 certificate, and with login credentials, which it must also instruct the server in question to accept.

Our algorithm is vulnerable if the censor can create an arbitrary number of user identities. After such a censor begins attacking, any server that has not already received a full load of innocent users will be blocked. Therefore, we must be very careful about how we allow new users to join the system. We accept new user registrations in two ways, which we now describe.

3.2 Facebook registration

A user can create a new Salmon account by demonstrating ownership of a valid Facebook account. In our initial deployment, a "valid" Facebook account is simply one that existed before 2015. Any enhancements to Facebook's screening of fake accounts would make Salmon more robust, by weeding out any accounts the censor created earlier, *e.g.* for social engineering purposes [2]. Cat-and-mouse games could be played with account requirements, such as post frequency or patterns. For a more sophisticated defense, the system could attempt to shut out accounts with fake profile pictures. For instance, it could require profile pictures to show a face, check via reverse image search that those pictures do not exist outside of Facebook, and reject any Facebook account whose face matches other accounts that already created a Salmon identity.

Of course, Facebook is itself a prime censorship target. Low bandwidth covert channels, such as through Skype [15] or email [26] are helpful here. While their performance is not quite what users would want for general Internet usage, they are sufficient for Salmon's one-time registration process, with its single Facebook post.

To prevent double registrations, the directory server must remember which Facebook profiles have created Salmon accounts. Since our logic must be able to test

a Facebook profile for membership in the set of already used profiles, an intruder who gains control over the directory server will be able to do the same. To prevent the intruder from going beyond membership tests, and obtaining a full list of Salmon Facebook profiles, we store the links hashed.

Our use of Facebook is not guaranteed to limit the censor to one Salmon account per employee. Iran, in particular, is known to already be using realistic fake profiles to try to make connections to Iranian Facebook users, who are typically much more careful with privacy settings than the typical Facebook user. In the case of China, "one account per employee" might actually be overwhelming for a Salmon deployment with hundreds or thousands of servers, given the size of its censorship operation. (That said, those employees would be vastly more likely to have active, well established accounts on Renren than on Facebook). In such a case, we could disable Facebook registration, so that new users could only join via recommendation. The closest design would then be rBridge, relative to which Salmon would still provide an easier and more reliable path for new users to join, even without non-recommended account creation.

Therefore, the Facebook registration mechanism can be removed without dismantling Salmon; it is an additional feature meant to provide openness equal to VPN Gate, while being significantly harder to block. Since VPN Gate is currently successfully serving users (as of May 2016), this is a good goal. A recommendation-only version of Salmon would still serve users, and until that becomes necessary, the system is open to the public.

Even if the censor suddenly adds many agents and discovers many servers before we realize we need to disable Facebook registration, we can have servers reset IP addresses once Facebook registration is disabled. In the time before we are forced to disable Facebook, we grow much more rapidly, and with a more widely seeded social graph, than we could with only recommendations.

3.3 Recommended registration

A user can create an account at the recommendation of an existing user who is sufficiently trusted by the system. This process is simple: the directory server generates a short alphanumeric code for the recommender to give to the recommendee, and when the recommendee registers, they provide the code rather than proving ownership of a Facebook profile. The recommendee must not begin its life immediately able to recommend, or else a censor with a single recommender account could instantly create arbitrarily many identities. The details of when recommendation is possible, and what the result is for the recommendee, are described in §2.3.

Also described in §2.3 is the fact that part of the algorithm involves tracking who recommended whom. That is, we are storing some subset of a social graph of our users, perhaps making the machine storing it a target. However, an attacker gaining access, especially a national censor, would in fact not find much new information. The graph stored is a tiny shadow of a true social graph. The graph is a forest of trees, with each user having at most one incoming recommendation edge. With $n-1$ edges for n users, a recommendation forest would have fewer than 1% of the edges of the corresponding set of Facebook users.

Additionally, national censors may have other options for collecting this sort of data. Of the two countries we are most interested in, China and Iran, only Iran might need to resort to breaking into servers to obtain social network data about its citizens. Renren, the Chinese Facebook clone, is as ubiquitous in China as Facebook is in free countries. The Chinese government certainly has access to all of Renren's data; there is not much more they could want.

Finally, we are not directly storing edges between Facebook IDs, but rather between email addresses, which we suggest to be a fresh account just for Salmon, in case a censor distributes malicious copies of Salmon to steal passwords. An email address is linked to a hash of a Facebook ID (if the user registered with Facebook). While these measures do not make meaningful social connection data irretrievable, they do mean that an intruder will not directly find a subset of Facebook's graph. In particular, the hashing means that if the intruder does not already know that a certain Facebook account exists, they will not know what they are looking at when they see it in the graph.

3.4 Detecting blocked servers

In addition to being blocked by the censor, servers can also simply go offline, as many volunteer servers will be individuals' personal computers. It is crucial to distinguish between these cases, or else churn (to say nothing of a censor providing intentionally short-lived servers) would cause many users to be needlessly banned.

We want to view a server as "blocked" if it is still functioning perfectly fine, and connections to it only fail due to a censor's interference. The censor only controls

traffic within its own country, and so we can conclude a server is blocked if and only if a host outside the censoring country can reach the server at the same time that the client cannot. The check is done by the directory server.

Experience has led us to cover some corner cases, such as a server coming online just as the client reports it to be down, or a client with intermittent internet access. Unfortunately, putting this issue to rest on the theoretical side requires more than patching a few bugs. If there is an ongoing non-zero probability that a user's server will be erroneously classified as blocked, then eventually that user is going to be banned. We hope our implementation is robust enough that these erroneous bans would take years, but without formal verification of the entire system, we cannot be sure. Ideally, some more sophisticated approach, which would take these observations as input, and prevent or reduce false positives at the cost of increasing false negatives, should be used. The development of such an approach would be an interesting avenue of research.

3.5 Server churn

Salmon prevents churn from causing erratic availability for users by adding new servers to a group — without penalizing the group's users — when all of the group's servers are offline, but not blocked. Users stay on the same server until it goes offline, at which point they are entitled to receive a new server. They also retain access to the previous server, and can choose to switch back to it whenever it comes back online. When told to connect, the Salmon client attempts connections to its collected servers in order of a rough heuristic score of advertised bandwidth and observed RTT. Servers observed to be offline within the last week are tried after all others.

While the assignment of multiple servers to a single group adds significant complexity to the implementation, it does not change the algorithmic concepts. A group of servers with at least one online at any given time provides essentially the same service as a single server that is always online. All clients in the group are entitled to learn about all of the group's servers, and no clients outside the group are entitled to learn any. Therefore, for purposes of analysis, we assume that each group of users receives a single server, which is always usable until blocked by the censor. We refer to users being "grouped together" or "assigned to a server"; these terms are synonymous.

Additionally, some servers, such as those run on home Internet connections, may periodically change IP addresses. Our implementation allows the client to stay with such a server. The server keeps the directory informed of its address, and when the client tells the directory that the server is offline, the directory simply informs the client of the new address.

This IP address churn can in fact be extremely useful to the system, if it can be intentionally induced. A blocked server can "return to life" by simply changing its IP address, and reconnecting to the directory server (which will of course not assign it to the exact same group of users that got it blocked). A volunteer can provide an email address at which to be notified if their server becomes blocked. The email includes instructions for forcing a typical home Internet connection to change IP addresses.

3.6 System parameters

Our implementation assigns up to 10 users to a single group, and asks servers to provide a minimum of 100KB/s. We hope that many of our volunteers will provide more than the minimum; the servers we are ourselves hosting each provide around 1.5MB/s.

Ultimately, the question of group size comes down to a fundamental tradeoff: serving more users with a given number of servers on one hand, and providing more bandwidth and limiting the censor's ability to block servers on the other. We wanted to push the group size as high as possible while both staying safe from the censor, and providing users with good throughput. Given our minimum bandwidth requirement of 100KB/s for the servers, if all clients in a group of 10 attempt to transfer large amounts of data simultaneously over TCP, they will each get around 10KB/s. We did not want to design into the system the potential for users to see single digit KB/s bandwidth: dial-up, which any modern VPN-based anti-censorship technique ought to outpace, is 7KB/s. Therefore, on the implementation side, we decided on 10 as the upper bound. On the algorithmic side, our simulations using size 10 groups were sufficiently robust.

Our simulations led us to set the suspicion threshold $T = \frac{1}{3}$. For our maximum group size $M = 10$, users are banned after witnessing 4 server blocks in full groups.

We chose highest level $L = 6$ as a compromise between allowing users to become highly trusted in a reasonable timeframe, and keeping the censor recommending agents at a slow rate. Similarly, we chose $D_* = 1$

day and $D_L = 30$ days for the wait between recommendations for special users and level L users, respectively.

3.7 Volunteer safety

A Salmon server's logs are an appealing target for the censor. The censor could gather Salmon logs with much less effort by running honeypot servers, but volunteers should still be aware of this point. More immediately, as a side effect to providing users with unfiltered Internet access, Salmon causes users' actions to appear to come from the server they are using. This obviously has the potential to cause trouble for the server volunteers, which must be prevented.

We do not know how various legal systems would ultimately parse this issue, but logically speaking, a Tor exit node and a volunteer VPN server ought to be equivalent in terms of liability. A Tor exit node breaking laws is indistinguishable from its users breaking laws. A VPN volunteer who breaks laws and fabricates logs showing their users doing it is similarly indistinguishable. Because Tor has managed to function without anything terrible happening to its exit node providers, we expect that VPN server volunteers should be equally safe.

BitTorrent deserves a special mention, as the most likely source of abuse. We block BitTorrent by only allowing outgoing connections to common ports, such as DNS, HTTP(S), ssh, etc. Skype is content with access to 80&443, and at least in Iran, torrents are not censored, so this policy is not restrictive.

3.8 Privacy

Some circumvention approaches build on top of Tor, or otherwise attempt to provide users with privacy / anonymity. As a one-hop VPN-based approach, which furthermore relies on unvetted volunteers to provide servers, Salmon users should not feel private. We clearly convey this fact, including the example of potential government honeypots, which should get users' attention (and is easily the greatest threat). Privacy would be ideal (and the lack of privacy disqualifies Salmon for truly dangerous activities), but given the prevalence of paid VPN services as a circumvention method, Salmon is not any worse than the status quo.

Fig. 3. The interface of a logged in Salmon client. The advanced options include recommendations, and sending server credentials to mobile devices.

3.9 Usability

We want Salmon to be easily available to all censored Internet users, not just technically adept ones. Our initial client implementation is for Windows, and is compatible back to XP. A SoftEther client is available for Linux; if there is sufficient demand, the Salmon client can easily be modified to run under Wine, as its Windows-specific code is almost entirely simple GUI components.

The client runs on top of SoftEther, and takes care of all configuration tasks. Even when the client has amassed a list of servers, the user is presented with a single simple "Connect" button (figure 3), and the Salmon client finds the best available one to connect to. The email-based communication with the directory server can be handled automatically by the client, using the vmime library. There is also a manual email option as a fallback. The client is available in English, Chinese, and Persian.

The client can export its list of server credentials to iOS (in convenient one-tap `.mobileconfig` files) and Android devices, as SoftEther supports L2TP IPSEC.

3.10 The zig-zag attack

Proxy-based techniques without careful distribution, such as VPN Gate or the direct distribution of Tor [5] hidden bridges, face a problem known as the zig-zag attack [3]. In this attack, the censor somehow discovers a few servers, and watches their citizens' traffic for connections to those servers. Citizens communicating with the servers are assumed to be using the same circumvention system, and so any other IP addresses they communicate with are worth close inspection, since there are likely to be some new proxy servers among them.

This exact attack does not affect careful distribution schemes such as Salmon, because clients are not free to try any server whenever they wish. However, a variant, which we will call the *active zig-zag attack*, does apply. The difference is simple: the censor now blocks a server once it has finished observing its users, *forcing* the users

onto new servers. In addition to discovering servers, the attack would damage the reputations of the innocent users who had been placed on those servers.

Defending against the zig-zag attack entails making it difficult for the censor to confirm the presence of a proxy server. Such a defense requires both authentication and camouflage. If the server cannot verify that a given connection is coming from a user who is "supposed" to be talking to them, it has no choice but to act as a functioning proxy, fully exposing itself to a zig-zagging censor. The server's behavior after deciding to reject a connection is just as important: if it doesn't look like a legitimate, innocuous server, the censor might still block it. The situation then becomes a steganographic arms race.

Each Salmon user has unique login credentials for each of their servers. Connecting to a server without valid credentials yields a page from what appears to be an ordinary HTTPS server (since credentials are requested via HTTP authentication). Of course, the issue of camouflage goes beyond authentication behavior, most notably to traffic fingerprinting. However, counter-fingerprinting techniques apply to just about any circumvention system, including those based on Tor.

The censor could also take another, more limited approach to zig-zagging, in which the discovery of one server in a group can reveal the rest. Suppose the censor is secretly in control of a popular site that it censors, and has the site give cookies to visitors. A Salmon user (with one server known to the censor) who repeatedly visits this site via different VPN servers (due to churn), presenting the same cookie with each visit, will prove to the censor that the IP addresses seen presenting the cookie are VPN servers. Note that not all of these VPN servers, even the originally discovered one, need to be Salmon servers for this attack to be effective. To mitigate this threat, users should use incognito/private browsing while on Salmon, and should not leave a browser session open for days at a time.

3.11 Deployment

We have a small initial deployment of Salmon running, with a few servers and about a dozen users in Iran and China. After working through some bugs, such as unstable Internet connections causing the client program to wrongly report a server block, the system is working for these users. Traffic fingerprinting therefore does not appear to be in use.

Our most interesting experience involved the WiFi at one user's university. The WiFi is public but with very limited bandwidth, except for connections to the school's VPN. However, connecting to another VPN prevents the use of SoftEther in its default configuration, short of using a VM. There is no particular indication that the purpose of this setup was to interfere with circumvention; one of the authors attended a school in a free country with the same setup. This problem can be solved by having the client's traffic enter the tunnel at a higher layer, such as with SOCKS. The SoftEther client has a SOCKS mode, although SOCKS entails additional configuration, and should not be our default.

4 Related work

Some anti-censorship systems force a censor to choose between entirely blocking a legitimate service their citizens find useful, and accepting that their censorship can be evaded. Ideally, such an approach should force the censor to do no less than blocking the entirety of the Internet outside the country's borders. In theory, that extreme option is always available, as demonstrated by North Korea. In practice, however, a country whose citizens have already experienced the Internet is unlikely to be able to turn back the clock.

Other circumvention techniques rely on helpers that can be blocked by the censor with no collateral damage, and so are effective only as long as those helpers stay hidden from the censor. Any such approach faces the same fundamental problem that Salmon addresses: users learn enough about the helpers to block them, and it is difficult to prevent the censor from posing as an ordinary user. Members of this class are the closest relatives to Salmon; we compare against them.

To summarize, the outside helpers that circumvention uses fall into one of two groupings. Their communication with users is either easily blocked, so the helper must be kept hidden; or too hard/painful for the censor to block, so the helper need not be hidden.

4.1 Easily blocked helpers

VPN Gate: We assume the censor can assign many full-time human employees to the task of shutting down censorship circumvention systems. If the censor cannot afford to keep humans watching our system for new servers to block, then a careful distribution algorithm

might not be necessary: if automated large-scale blocking can be stopped, this sort of censor would be stymied. VPN Gate [19] takes this approach.

VPN Gate publishes (in chunks of 100 for each query) a complete list of the servers in its system. By mixing in fake IP addresses, and addresses of real web sites the censor does not want to block, VPN Gate makes it impossible for the censor to directly block the entire list. To stymie the obvious solution of testing all the addresses for the presence of a functioning VPN server with an automated scan, the proxy servers coordinate to detect and ignore patterns of connection attempts that appear to come from an automated scanning process.

A system that relies on the ability to identify automated probes is not robust. For one thing, other spaces, such as click fraud, have seen sophisticated techniques developed for making automated behavior look human [17]. More fundamentally, even without such techniques, a censor with enough resources has the option to make its scanning processes fully human.

A simple harvesting script running for 9 days on a single free Amazon EC2 instance was able to verify that 3,101 IP addresses are working VPN Gate servers. The harvester script tested 44,039 IP addresses. We evaded the collective defense mechanism while harvesting from just a single IP address by exploiting VPN Gate's *raison d'être*: after verifying a VPN server by successfully connecting to it, the scraper uses that connection to make its next query for new servers.

With the list we compiled, a censor could, with no collateral damage, essentially completely shut down VPN Gate. Only a handful of unreliable servers would remain available, hidden among many blocked servers.

Tor: Tor's [5] goal of Internet *privacy* is orthogonal to our goal of Internet *access*. However, the Tor Project has also attempted to provide access to censored users by distributing hidden "bridge" relays, but without a rigorous method for pinpointing infiltration by a censor. Designs using Tor do so only to gain Tor's privacy; their ability to provide access would not be reduced by switching to one-hop proxies. Similarly, Salmon's techniques could be used to distribute Tor bridges, rather than proxy servers.

Careful proxy distribution: Previous proposals have taken approaches similar to Salmon. Like Salmon, these techniques aim to prevent a censor acting exactly like an ordinary user from discovering too many servers. These proposals include recent work such as Proximax [16] and rBridge [22], earlier work such as keyspace hopping [7], and an analysis of Tor blocking [4] containing a sketch of a proto-Salmon or -rBridge. These techniques differ from Salmon, and from each other, both in the algorithms they use to identify censor agents, and in how they attempt to limit the number of agent users the censor can register. Keyspace hopping, the earliest work in this space, relies entirely on computational puzzles to rate-limit the censor. The later works consider distribution strategies.

Proximax [16] is purely invitation-based, with an approach even more open than Salmon's rate-limited recommendations. In fact, "invitation" is not quite the right term. The vast majority of users are not official Proximax users; friends simply share any and all servers they discover. The system compensates for this openness by being much more restrictive with distribution of new servers. Only the very small group of officially registered users, who are roughly equivalent to Salmon's special above-the-trust-system users, are given fresh servers. Proximax tracks which servers given to which registered users become blocked, and prefers to give servers to the registered users whose friends get the fewest servers blocked.

rBridge, discussed next, was demonstrated to resist a censor's attack somewhat more effectively than Proximax. Proximax's less accurate discernment makes intuitive sense: it operates at a coarser resolution. Proximax tracks behavior at the granularity of entire social graph clusters, whereas Salmon and rBridge examine individual users.

rBridge: Like Salmon, rBridge [22] seeks to distribute proxy servers (Tor bridges, specifically) to the public while limiting a censor's ability to block them. As its core feature, rBridge prevents the central directory server from learning which users received which servers, in order to protect users from an attack in the context of Tor. Users receive a stream of credits while their servers remain unblocked, and can redeem credits to receive more servers. Users can only join via recommendation, and existing users are only eligible to obtain recommendation tickets if they keep their credit balance above a threshold. Salmon offers major improvements over rBridge in a few regards: robustness, accessibility for new users, and efficient use of server resources.

First and most simply, Salmon is far more robust to an attacking censor than rBridge. Salmon's trust levels, combined with permanent banning, make it over three times as robust to the sort of attack that rBridge considers, as we show in our evaluation section (§5.5).

Second, rBridge does not offer a particularly friendly, or even predictable, path to becoming a user. rBridge only accepts new users who have gotten an in-

vitation ticket from a friend, which the directory server distributes to random users who have kept their credit balance above a threshold. We seek to provide access to all censored users, not just friends of trusted users. rBridge users have no way to know when (if ever) a ticket might be issued to them. They have no control over the process, beyond requesting few enough bridges to remain eligible to receive a ticket.

Third, by designing specifically for Tor, rBridge accepts significantly degraded performance for any given amount of server resources. Tor's multi-hop design results in a third of the aggregate bandwidth and twice the RTT (for Tor's default of 3 hops) of a one-hop system with equivalent server resources. VPN Gate cites performance as a reason for providing VPN servers rather than Tor bridges. Because our first concern is providing high performance circumvention, we also prefer the one-hop design.

In theory, Salmon, rBridge, and any other careful distribution approach can distribute either Tor bridges or VPN servers. The choice is not tied to the details of the distribution strategy, but rather to implementation goals. That said, rBridge's tradeoffs and distinguishing features would be wasted outside the context of Tor.

Flash proxies [8] are an especially creative attack on the censor's ability to block proxies. Rather than hiding the servers, this approach aims to constantly add new ones to the system, so quickly that the censor simply cannot keep up. Participating websites embed scripts that turn visiting web browsers into Tor bridges for the duration of their visit to the page.

4.2 Hard to block helpers

Decoy routing: Radically different from proxy server approaches, decoy routing seeks to embed censorship circumvention into the fabric of the Internet. This concept was independently proposed as decoy routing [14], Cirripede [11] and Telex [25]. Tapdance [24] is an evolution of Telex, which is less disruptive to ISPs' networks. Rebound [6] makes no changes to how packets are routed, to help remain undetected, in exchange for diminished bandwidth.

Clients use specially chosen values for randomized fields (TCP ISNs for Cirripede, TLS hello nonces for Telex) to complete a key exchange and indicate their desire to access censored sites. Routers throughout the Internet are modified to monitor for, and divert such flows to servers that forward the traffic to the desired destination. The entire exchange is invisible to the censor: the client initiates the connection to a site that is allowed by the censor, and known to use TLS. Until the traffic reaches the decoy router, it is indistinguishable from traffic truly meant for the cover site.

Although decoy routing can withstand [13] the best proposed attacks [20], there is a deployability problem. For instance, to our knowledge, four years after the release of Telex only the University of Michigan network is running Telex decoys. This type of system cannot be deployed just anywhere: it must live inside ISP-grade routers in the most heavily transited portions of the Internet, so that all censored users will have at least one path to an innocuous site that passes through a decoy router. Large ISPs are for-profit companies with no special interest in free speech, who might even seek good business relations with censoring governments.

Domain fronting: [9] In addition to decoy routing's strategy of redirecting traffic at collaborating nodes in the network layer of the Internet, it is possible to do similar redirections further up the stack. CDNs can function by having browsers address HTTP(S) requests to the true logical destination (the domain name in the "Host" field of the HTTP header), but physically send them to an IP address controlled by the CDN. If that CDN server does not have the requested item, it can retrieve it from the true server, and then finish the client's request. If the HTTP Host field is hidden inside a TLS session, and the visible portion of the request (such as DNS resolution) pretends to be aimed at an innocuous site also hosted on that CDN, the censor cannot distinguish requests to blocked sites from requests to innocuous sites. If the blocked site we redirect to is a proxy server, then general circumvention is accomplished.

The question at the heart of this technique is whether the censor is willing to block many foreign sites that its citizens use, and which it ideally would like to allow, in order to keep its censorship in place. There are currently working implementations of domain fronting; just as with VPN Gate, for the time being, this technique is effective. However: Google, with its highly popular assorted services, ought to be one of the best examples of unpalatable collateral damage, and yet China has now completely blocked it. With domestic versions of all web services popular among Chinese users, we should not assume that there is any major web company that the Chinese government is not willing to block.

Piggybacking: Anything that sends encrypted and authenticated messages to any recipient in the world, and is allowed by the censor, can be used to break censorship. Email is a good candidate: SMTPS and IMAPS are now standard, and email is too deeply embedded

into the modern world for a censor to block entirely. Individual providers such as Gmail may be blocked, but many providers are available. In fact, due to these qualities, Salmon clients use email as a low bandwidth (and high latency, relative to the Internet) uncensorable tunnel to the central directory server. SWEET [26] takes this communication a step further: it uses email to carry the user's actual Internet traffic.

Other proposals tunnel IP packets through secure VoIP calls [12], or video calls [15]. Tunneling only through VoIP does not meet our bandwidth goals, as it is essentially dial-up (or worse, given the aggressive psychoacoustic compression algorithms used in modern voice communication). Worse, because systems such as Skype have no reason to recover dropped packets, the censor can attack covert tunnelled flows with tolerable levels of disruption to Skype calls [10]. Furthermore, secure VoIP and video chat do not enjoy the status of being painful for a censor to block. In fact, these protocols are often targets themselves, for the same reasons that censors block social media platforms.

5 Evaluation

To demonstrate that Salmon can withstand an attack by a realistic censor, we simulated users and censor agents requesting proxy servers. We measured how many servers the censor can block, and how many censor agents and innocent users are banned. Ultimately, though, the censor cares most about how many users are left without access when it has done all the damage it can. This quantity (as a fraction of all users) is used for the vertical axis of the charts.

To ensure that the simulations are faithful to the real system, the simulation environment was built around our implementation of the central directory server: the component of the system where all of the algorithm logic lives. Specifically, functions that would be called upon receipt of an email from a client, or a message from a server sent through TLS, are instead called directly from the simulation framework.

The simulation iterates through one day at a time, having each user check whether they have access to a server, and requesting a new one if not. Agents behave similarly, and additionally have the opportunity to block any server they have been given, whenever their strategy dictates.

All simulations have 10,000 users, including the censor's agents, which comprise $\{1\%, 2\%, 5\%, 10\%\}$ of all users. Each of these percentages represent a serious attack by the censor: even at the lowest level, the censor has assembled 100 real Facebook accounts, and gone through the Salmon registration process for each one. The users join the system in the pattern depicted in Figure 2, representing exponential word-of-mouth growth, as well as a steady flow of recommendations from the small number of "special" trust level users (the Salmon administrators' personal friends).

We vary the number of servers available to the system, from 1,000 to 2,000. The charts' horizontal axes represent a related quantity: users per server, varying from 5 to the system's maximum of 10. In our implementation, a user who registers when all servers are full is informed of the lack, and encouraged to ask any friends they have in free countries to consider becoming Salmon volunteers. As a fallback, they are given 16 confirmed (by the process in §4.1) VPN Gate servers.

Allowing newly registered users to go without Salmon servers when no empty servers are available has a major benefit for the older users. If a Salmon system can grow for a time without the censor inserting agents, an increasing number of users actually become *invincible* to (having their servers discovered by) the censor. Suppose the system gains a significant initial userbase by the time the censor takes notice and begins inserting agents. Salmon fills servers one by one, rather than evenly distributing users among all available servers, and server groups have a hard cap on membership. Therefore, any user whose server group is filled before agents begin joining is invincible to being banned.

While the system's logic — whether to give a user a server, which server to give, when to ban a user, recommendations — is deterministic, the simulations are a random process. The order in which users (with agents mixed in) act is randomized for each run of the simulation, leading to different groupings of users, and therefore different effects from the blocking of servers. We ran all of our simulations multiple times; the points on our charts are means, and the bars are 95% confidence intervals. 10 replications were enough to shrink most of the 95% confidence interval bars smaller than the charts' point icons.

5.1 Adversary strategy

The most important aspect of this sort of simulation is the behavior of the adversary. Our simulated adversary should follow the strategy that best achieves the real-world adversary's goal. Against Salmon's distribu-

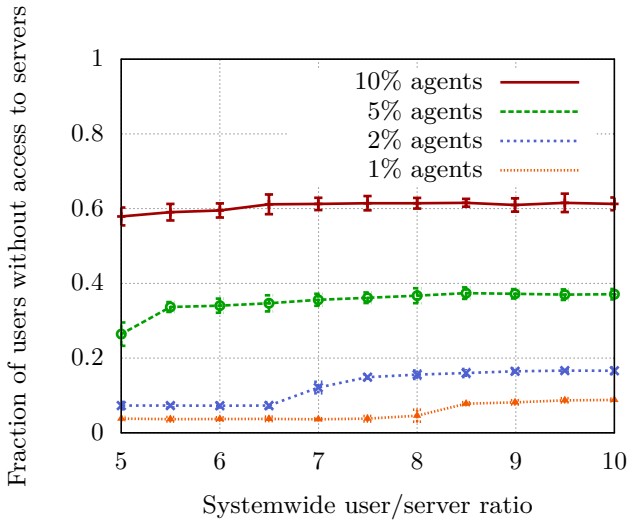

Fig. 4. This simulation omits the trust level mechanism, benefitting the censor. Vertical axis represents users without access to servers after the censor has blocked all the servers it can.

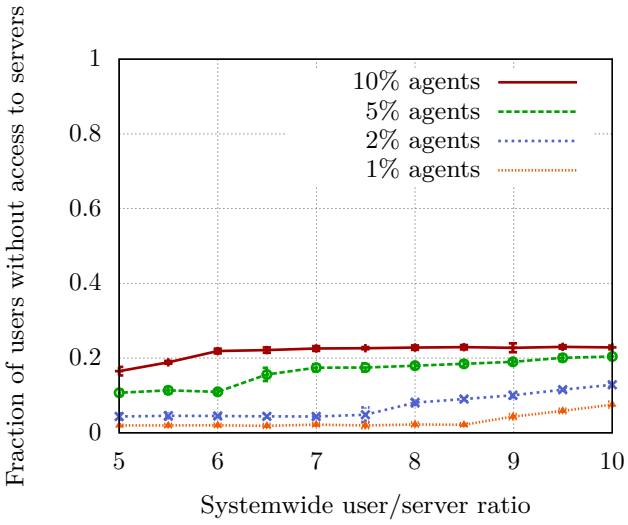

Fig. 5. Same as Figure 4, but with trust level logic enabled. The users join the system in the same pattern as in Figure 2.

tion strategy, a censor can essentially never block every single citizen from using Salmon; because server groups fill up, a single full group without agents is safe forever. The adversary therefore has the quantitative goal of maximizing the fraction of users whom it can cut off from the Salmon system.

Due to the design of the system, one agent represents four opportunities to block a server (assuming the agent only blocks when its group is full). Therefore, a censor with A agents has the potential to block up to $4A$ servers. A server needs only one agent assigned to it for the censor to be able to block it. Blocking a server costs the censor one blocking opportunity for every agent who had been assigned to the server, and therefore every agent beyond the first in a given group is a waste. Then the optimal course of events for the censor is for all agents to always be alone in their groups.

Once an agent is in the system and has requested a server, the only action it can perform is to block its current server. As long as the agent's server is not blocked, nothing besides trust will change for the agent and its group. The agent will never see any other servers, and no other users will join (once the group has filled) or leave the group. The censor can only control when its agents will be in the market for a new server, and therefore has two levers to build a strategy with: when agents first join the system, and when they block servers.

If all of the agents joined at once, all but the first and last few would be placed in all-agent groups. In this situation, a quarter of the censor's blocking opportunities have already been expended in the worst way possible: not only have a minimum of servers been discovered, but there has been no collateral damage to legitimate users. On the other hand, the censor must be sure not to insert agents too slowly relative to the rate at which real users join, or multiple groups will fill entirely with real users, becoming invincible (within our distribution system). Of course, if there are fewer agents than groups, this is inevitable, but lagging behind the rate of joining users exacerbates the problem.

In the real world, learning this rate should be difficult for the censor, as we do not publish user statistics. It is overwhelmingly unlikely that the censor would be able to land its agents exactly in every tenth spot. For evaluation purposes, the most natural compromise between this perfect censor, and one that has all agents join at once, is the only other simple configuration: a uniformly random permutation of the order in which users (including agents) join, while agents are joining.

The other question of censor strategy is the timing of server blocks. It turns out that having all agents block simultaneously does not clump them in the same way that having them all join simultaneously would. At the time of the mass block, there is a certain distribution of users and agents, based on when the agents joined, across levels and groups. A mass block event simply reshuffles the group component of that distribution.

5.2 Benefit of trust levels

To show the effect of Salmon's trust level logic, we ran two sets of simulations: one with the standard trust level

logic (Figure 5), and one without (Figure 4). In both cases, we assume the censor is not patient enough to wait over four months for its agents to be able recommend more into the system. Such a censor is considered in §5.4. Users enter the system in the pattern described in Figure 2, and the censor begins attacking when the userbase (sans agents) reaches 10,000: day 94.

Because users leave level 0 after just two days, its membership is consistently quite small. The censor's agents are more effective the more evenly distributed throughout the system they are. Therefore, having all agents join at once, or initiating blocking while most agents are at the same level, would accomplish very little. We therefore assume the censor is at least somewhat patient: it is willing to wait long enough to distribute its agents into levels 0, 1, and 2. (Recall that we fight more patient adversaries with the servers' ability to occasionally change IP addresses.)

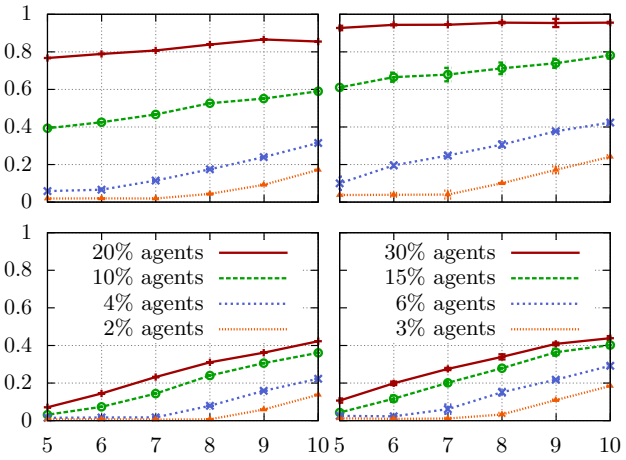

Fig. 6. Agents and users all start at trust level 6. Agents each recommend {1,2} new agents (left, right). Users {are not, are} grouped to servers by recommendation tree (top, bottom). Agent percentages refer to final fraction after recommendations.

5.3 Recommendation attack

In the algorithm design section (§2.3), we discussed the possibility of a patient censor using the recommendation system to grow a huge network of agents. We have one solution in place: forcing this process to cost at least several months of inactivity on the part of the censor, and then having volunteers change their IP addresses after the mass block. However, we should also attempt to minimize the number of servers that would be affected by such a mass block in the first place. We therefore checked whether our strategy of grouping users by recommendation helps in this regard.

With our chosen delays for trust level promotion and recommendations, it would take an extremely patient censor to achieve true exponential growth. The initial batch of agents must wait over 4 months to be able to recommend, having started from level 0, and the agents they recommended must wait over 2 months to make recommendations of their own. The censor must therefore wait for well over 6 months before the growth even begins to be exponential. If the censor must wait this long, we have already won. The censor has allowed an effective censorship circumvention system to be freely available to its citizens for over 6 months. Furthermore, even when the censor does mass-block servers in this manner, this will be a very infrequent event, so it will not be much of a burden for our server volunteers to acquire new IP addresses, as described earlier.

Therefore, we consider an only somewhat patient censor, who is only willing to wait long enough to recommend one or two additional agents per original agent (requiring a wait of 4 or 5 months respectively). Even if agents make just one recommendation, their ranks are doubled; a censor may judge the wait to be acceptable in exchange for that benefit. Figure 6 shows the benefit of our logic of grouping users from the same recommendation tree together: larger initial batches of agents can cut off from Salmon the majority of users, even up to over 95%, if this logic is not in place. The system's weakness when not enforcing the recommendation grouping is due to lack of assistance from the trust levels: against the somewhat patient adversary, the agents and users will all have reached the highest trust level together by the time the blocking starts.

5.4 Patient censor

Of course, a censor may choose to have its agents lie dormant, appearing to be innocent users, and thereby save the ability to block many servers simultaneously for an emergency. Although we can rely on our IP address change defense for eventual recovery in such a situation, the bulk of the blocked servers would likely remain blocked for a day or two, while volunteers got around to changing their IP addresses. In a fast moving situation, the censor might be satisfied with just one or two days of disruption.

This is a hard problem, and unfortunately one that can potentially affect any circumvention system: if the censor develops some effective countermeasure, it can wait to deploy it until it will have high impact. For instance, VPN Gate is described as having evolved

through a cat and mouse game with the Chinese censor. According to its statistics page, it is currently successful in serving users in China. Given how easily we were able to automatically enumerate the real VPN Gate servers (§4.1), it seems unlikely that the censor was permanently stumped. The censor may have decided to stop putting effort into the cat and mouse game, saving its next move for when it badly needed to shut down VPN Gate for a specific reason.

This is an important problem, and a solution would make anti-censorship systems considerably more robust. A good solution would have to mutate a circumvention system without outright replacing it: if the fallback is something that could run alongside the normal system, rather than in place of it, then choosing to hide it from the censor is simply a tradeoff between robustness and capacity. A system such as Salmon could make that tradeoff by keeping a reserve of unused servers.

Salmon does not fully solve this problem, but it can at least mitigate it somewhat. Users whose server groups are filled before the censor starts infiltrating agents into the system are guaranteed to stay safe, with their servers remaining unblocked. In this way, at least some core of trusted users would retain their access during the censor's emergency blocking.

5.5 Comparison with rBridge

We compared Salmon to rBridge, the most robust similar proxy distribution proposal. We adapted the simulations of Salmon depicted by figure 5 to the conditions of rBridge's original evaluations: censor agents join by being recommended by innocent users, rather than via Facebook. In Salmon's case, this means they join at level 5, rather than 0, and have tied themselves to an unfortunate innocent user with the recommendation logic. We consider the same 5% agent case as in the original rBridge evaluations.

We changed rBridge's group size from the original choice of 40 to Salmon's choice of 10 (the group size considerations are the same for both Salmon and rBridge). We assume the censor will wait just long enough for its agents to earn enough rBridge credits to request 2 additional servers. We assume that the innocent rBridge users have accumulated infinite rBridge credits: they always have the right to request a new server.

rBridge gives all users 3 servers for robustness to churn, which Salmon accomplishes by giving free replacement servers for down (but not blocked) servers. rBridge cannot have the robustness to churn extracted from its algorithm logic: due to the requirement to keep user-server mappings secret from the directory server, users cannot report an offline-or-blocked server to the directory for it to confirm which is the case. Querying on a specific server requires anonymity, unlike receiving a random server, which can be accomplished with non-anonymous OT. If the user's only Tor bridge is unavailable, it cannot have an anonymous conversation with the directory. Therefore, rBridge must treat server churn the same as the censor's blocking, incurring the damage to users of a "blocked" server when Salmon would incur none. We therefore evaluate rBridge with its suggested parameter of 3 servers per user.

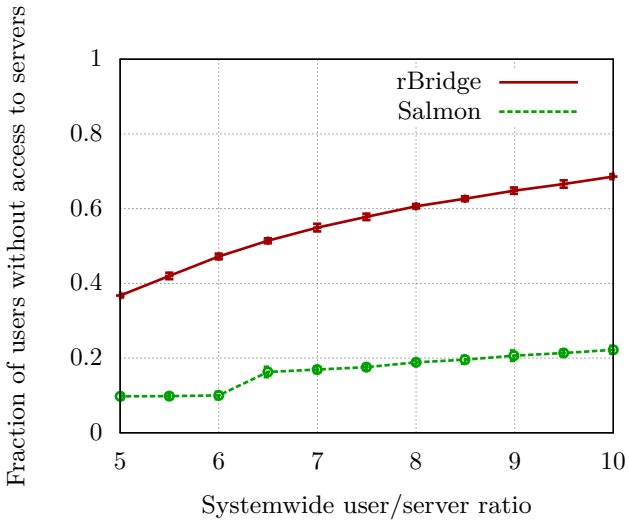

Fig. 7. Comparison of Salmon and rBridge. The censor attacks as in the rBridge paper: by being recommended by innocent users. 5% of 10,000 users are agents.

Figure 7 shows the results of our comparison. Salmon is much more able to weather the censor's attack than rBridge, losing fewer than a third as many users as rBridge: 22% vs 69% at the highest system load. Salmon's trust levels and recommendation grouping keep the agents corralled among a small minority of the users. This limits the damage they can do, especially at user/server ratio 10, where *no* blocked server can be replaced, and being grouped with an agent guarantees that a user will be shut out of the system. rBridge's agents are spread evenly throughout the system, maximizing the innocent users' exposure to them.

5.6 Steady state

The previous simulations all consider a censor who over time amasses a large number of agents in the system, and then tries to shut it down in one fell swoop. Also

of interest is the case of a censor who is content to be a nuisance, using a constant trickle of agents to block a constant trickle of servers (and perhaps with some collateral damage).

Consider the end result of the previous analyses: the censor is out of agents. Therefore, any user who has not been banned, and is in a full group, is invulnerable. In fact, they become invulnerable as soon as they join their agent-free group, even before the last agent has been banned from the system. This indicates that these simulations might give useful information even without looking at the end results with an assumption that the censor will never return.

We will define an attack wave to be the agents and users who join the system during any contiguous period of time. For those who are only ever grouped with other members of their wave, the environment is identical to the previous "static" simulations, *i.e.*, the probability of being banned is equivalent to the fraction of banned users that the simulations compute. Most groups can be packed full, unless recommendation component sizes are nearly all between six and nine. Therefore, the fraction of users banned in static simulations ought to approximate the rate at which users are banned, given the current probability that new users are agents, and (relative to the rate of new users) the current rate at which servers are joining. However, we leave a fully rigorous analysis for future work. Additionally, we have not formally proven that the censor strategy we evaluate against is optimal; a formal proof of the censor's strategy's optimality would also make for a good component of a future analysis of Salmon's algorithm.

6 Conclusion

Anti-censorship via volunteer proxy servers is an attractive approach: it is easy to deploy, it provides performance sufficient for all typical uses of the Internet, and censored users are already familiar with the concept of proxy/VPN servers. However, proxy servers are vulnerable to being blocked. Previous approaches have been either too loose, or too strict: *e.g.*, VPN Gate [19] is open to the world but defends only against somewhat lazy censors, while rBridge [22] users have an (unpredictably) hard time inviting friends.

Salmon strikes a good balance: registering a new Salmon account requires either a Facebook account that is older than Salmon, or a recommendation from a highly trusted Salmon user, with the ability to recommend limited to once per month. These are substantial obstacles to a censor trying to quickly inject enough of its agents to cause significant damage; at the same time, any ordinary Facebook user can easily join.

By banning suspicious users, and partitioning users based on trust, Salmon limits the damage a censor can inflict. The censor's agents are concentrated into a smaller set of users; banning them brings less collateral damage to innocent users. These techniques provide significantly better defense than previous approaches.

6.1 Theory and Implementation

This paper describes both an algorithm for weeding out agents of a censor trying to discover VPN servers, and an actual implementation of such a system. There are many real-world details that do not fit meaningfully into the theoretical analysis. There are some reasonable but not axiomatic assumptions that must be made to keep these details out of the analysis, and if these assumptions are violated, the system's algorithmic performance could be worse than expected.

Most prominent is the blocked-server detection described in §3.4. Although our detection mechanism has been iterated through some edge cases, we actually need a guarantee of zero false positives, or else all legitimate users will be banned given enough time. In practice, a not-quite-perfect system might be close enough to perfect that a user can survive for years or decades, but there is no way to know ahead of time. This issue could also interfere with an aspect of the analysis: the invulnerability that users gain when they are in a full group without censor agents. A user who is hit with a false positive must join a new group; a previously invulnerable user could join a group with an agent.

In addition to server churn issues, the behavior of real world users will certainly also add complications. In the theoretical analysis, our goal is to preserve good users' access to the system. In the real world, a user might find another circumvention system, leave the country, etc. Such an abandoned account cannot be counted in Salmon's favor. If many such accounts piled up, we should consider optimizations, such as merging groups with inactive users (with inactive users immediately banned if the merged group sees a block).

References

[1] What is internet censorship? Amnesty Intl., March 2008.

[2] Iran hackers use fake Facebook profiles to spy on US and Britain. The Telegraph, May 2014.

[3] Dingledine, R. https://blog.torproject.org/blog/research-problems-ten-ways-discover-tor-bridges, 2011.

[4] Dingledine, R., and Mathewson, N. Design of a blocking-resistant anonymity system.

[5] Dingledine, R., Mathewson, N., and Syverson, P. Tor: The second-generation onion router. In *Proceedings of the 13th Conference on USENIX Security Symposium - Volume 13* (Berkeley, CA, USA, 2004), SSYM'04, USENIX Association.

[6] Ellard, D., Jones, C., Manfredi, V., Strayer, W. T., Thapa, B., Van Welie, M., and Jackson, A. Rebound: Decoy routing on asymmetric routes via error messages. In *IEEE 40th Conference on Local Computer Networks (LCN)* (2015), pp. 91–99.

[7] Feamster, N., Balazinska, M., Wang, W., Balakrishnan, H., and Karger, D. Thwarting web censorship with untrusted messenger discovery. In *Privacy Enhancing Technologies 2003* (Dresden, Germany, March 2003).

[8] Fifield, D., Hardison, N., Ellithorpe, J., Stark, E., Boneh, D., Dingledine, R., and Porras, P. Evading censorship with browser-based proxies. In *Privacy Enhancing Technologies* (2012), Springer, pp. 239–258.

[9] Fifield, D., Lan, C., Hynes, R., Wegmann, P., and Paxson, V. Blocking-resistant communication through domain fronting. *Proceedings on Privacy Enhancing Technologies 2015*, 2 (2015), 1–19.

[10] Geddes, J., Schuchard, M., and Hopper, N. Cover your acks: Pitfalls of covert channel censorship circumvention. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security* (2013), pp. 361–372.

[11] Houmansadr, A., Nguyen, G. T. K., Caesar, M., and Borisov, N. Cirripede: circumvention infrastructure using router redirection with plausible deniability. In *Proceedings of CCS* (2011).

[12] Houmansadr, A., Riedl, T. J., Borisov, N., and Singer, A. C. IP over Voice-over-IP for censorship circumvention. *CoRR abs/1207.2683* (2012).

[13] Houmansadr, A., Wong, E. L., and Shmatikov, V. No direction home: The true cost of routing around decoys. In *Proceedings of the 2014 Network and Distributed System Security (NDSS) Symposium* (2014).

[14] Karlin, J., Ellard, D., Jackson, A. W., Jones, C. E., Lauer, G., Mankins, D. P., and Strayer, W. T. Decoy routing: Toward unblockable internet communication.

[15] Li, S., Schliep, M., and Hopper, N. Facet: Streaming over videoconferencing for censorship circumvention. In *Proceedings of the 13th Workshop on Privacy in the Electronic Society* (2014), ACM, pp. 163–172.

[16] McCoy, D., Morales, J. A., and Levchenko, K. Proximax: Fighting censorship with an adaptive system for distribution of open proxies. In *Proceedings of the International Conference on Financial Cryptography and Data Security* (St Lucia, February 2011).

[17] Miller, B., Pearce, P., Grier, C., Kreibich, C., and Paxson, V. What's clicking what? techniques and innovations of today's clickbots. In *Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, 2011, pp. 164–183.

[18] Mohajeri Moghaddam, H., Li, B., Derakhshani, M., and Goldberg, I. Skypemorph: Protocol obfuscation for tor bridges. In *Proceedings of the 2012 ACM conference on Computer and communications security* (2012), pp. 97–108.

[19] Nobori, D., and Shinjo, Y. VPN Gate: A volunteer-organized public VPN relay system with blocking resistance for bypassing government censorship firewalls. In *Proceedings of the 11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)* (Seattle, WA, 2014), USENIX, pp. 229–241.

[20] Schuchard, M., Geddes, J., Thompson, C., and Hopper, N. Routing around decoys. In *Proceedings of the 2012 ACM conference on Computer and communications security* (2012), pp. 85–96.

[21] Wang, Q., Gong, X., Nguyen, G. T., Houmansadr, A., and Borisov, N. Censorspoofer: asymmetric communication using ip spoofing for censorship-resistant web browsing. In *Proceedings of the 2012 ACM conference on Computer and communications security* (2012), pp. 121–132.

[22] Wang, Q., Lin, Z., Borisov, N., and Hopper, N. rBridge: User reputation based tor bridge distribution with privacy preservation. In *NDSS* (2013).

[23] Weinberg, Z., Wang, J., Yegneswaran, V., Briesemeister, L., Cheung, S., Wang, F., and Boneh, D. Stegotorus: a camouflage proxy for the tor anonymity system. In *Proceedings of the 2012 ACM conference on computer and communications security* (2012), pp. 109–120.

[24] Wustrow, E., Swanson, C. M., and Halderman, J. A. Tapdance: End-to-middle anticensorship without flow blocking. In *23rd USENIX Security Symposium (USENIX Security 14)* (2014), pp. 159–174.

[25] Wustrow, E., Wolchok, S., Goldberg, I., and Halderman, J. A. Telex: Anticensorship in the network infrastructure. In *Proceedings of the 20th USENIX Security Symposium* (August 2011).

[26] Zhou, W., Houmansadr, A., Caesar, M., and Borisov, N. SWEET: Serving the web by exploiting email tunnels. *Privacy Enhancing Technologies Symposium* (2013).