How the Great Firewall discovers hidden circumvention servers

Roya Ensafi
David Fifield
Philipp Winter
Nick Weaver
Nick Feamster
Vern Paxson
Much already known about GFW

● Numerous research papers and blog posts
  ○ Open access library: censorbib.nymity.ch

● We know...
  ○ What is blocked
  ○ How it is blocked
  ○ Where the GFW is, topologically

● Unfortunately, most studies are one-off
  ○ Continuous measurements challenging
Many domains are blocked

Client in China

DNS request for www.facebook.com

DNS resolver outside China
Many domains are blocked

Client in China

DNS request for www.facebook.com

DNS resolver outside China
Many domains are blocked

Client in China → DNS resolver outside China

DNS request for www.facebook.com

facebook is at 8.7.198.45
Many domains are blocked

Client in China

facebook: 8.7.198.45

DNS request for www.facebook.com

facebook is at 173.252.74.68

DNS resolver outside China
Many keywords are blocked
Many keywords are blocked

GET /www.facebook.com HTTP/1.1
Host: site.com
Many keywords are blocked

Client in China

GET /www.facebook.com HTTP/1.1
Host: site.com

Web server outside China

TCP reset
Encryption reduces blocking accuracy

Client in China

Server in Germany

Encrypted connection

HTTPS? VPN? Tor?
Encryption reduces blocking accuracy

Client in China

Server in Germany

Encrypted connection

HTTPS? VPN? Tor?

Port number? Type of encryption? Handshake parameters? Flow information?
Censors often test how far they can go

WHAT HAPPENED?

Update: On January 23, [https://github.com](https://github.com) was unblocked again.

On January 18, or possibly the day before (though our test data doesn’t cover this), the Great Firewall began to reset connections containing “github.com”. As a result, code-sharing projects hosted on a subdomain of Github, such as au.github.com, were blocked in China. The main Github website was mostly unaffected, for two reasons. Firstly, it’s hosted on github.com, without a subdomain. Secondly, it serves encrypted content only, thus preventing the Great Firewall from resetting connections based on keywords.

A day later, the block was extended through the inclusion of github.com, without subdomains, in the list of keywords causing connections to be reset. Chinese users could still access Github as long as they manually typed in [https://github.com](https://github.com) in their browser (notice the https). Strangely the [www.github.com](www.github.com) host was DNS poisoned, but not any other hosts. The www subdomain is not used by GitHub.

On January 21, DNS poisoning was extended to all github.com hosts including the root domain as well as all its subdomains. In effect, all of Github was blocked in China.

Interestingly, the blocking of Github has seemingly not been censored on social media. The keyword “github” has [not been blocked on Sina Weibo](http://weibo.com), and we have not detected any deleted posts containing “github” on Weibo.

For further information on how the blocking was introduced, including data references, see the Timeline at the end of this article.
Censors often test how far they can go

What happened?

Update: On January 23, https://github.com was unblocked again.

On January 18, or possibly the day before (though our test data doesn’t cover this), the Great Firewall began to reset connections containing “github.com”. As a result, code sharing projects hosted on a subdomain of Github, such as apps.github.com, were blocked in China. The main Github website was mostly unaffected, for two reasons. Firstly, it’s hosted on github.com, without a subdomain. Secondly, it serves encrypted content only, thus preventing the Great Firewall from resetting connections based on keywords.

A day later, the block was extended through the inclusion of github.com, without subdomains, in the list of keywords causing connections to be reset. Chinese users could still access Github as long as they manually typed in https://github.com in their browser (notice the https). Strangely the www.github.com host was DNS poisoned, but not any other hosts. The www subdomain is not used by Github.

On January 21, DNS poisoning was extended to all github.com hosts including the root domain as well as all its subdomains. In effect, all of Github was blocked in China.

Interestingly, the blocking of Github has seemingly not been censored on social media. The keyword “github” has not been blocked on Sina Weibo, and we have not detected any deleted posts containing “github” on FreeWeibo.

For further information on how the blocking was introduced, including data references, see the Timeline at the end of this article.
Active Probing
Assume an encrypted tunnel
1. GFW does DPI
1. GFW does DPI

Client in China → TLS connection → Server in Germany

Cipher list in TLS client hello looks like vanilla Tor!
2. GFW launches active probe
2. GFW launches active probe

Client in China \hspace{2cm} \text{TLS connection} \hspace{2cm} \text{Server in Germany}

Active prober
3. GFW blocks server

client in China: Active prober

Server in Germany: Block server

TLS connection

Yes, it was vanilla Tor!
Our “Shadow” dataset

- Clients in China repeatedly connected to bridges under our control

Clients in CERNET

EC2 Tor bridge

Tor, obfs2, obfs3

Clients in UNICOM

EC2 Tor bridge

Tor, obfs2, obfs3
Our “Sybil” dataset

- Redirected 600 ports to Tor port

Vanilla Tor bridge in France → Client in China
Our “Sybil” dataset

- Redirected 600 ports to Tor port

Holy moly, 600 bridges on a single machine!
Our “Log” dataset

- Web server logs dating back to Jan 2010
Where are the probes coming from?

- Collected **16,083** unique prober IP addresses
- **95%** of addresses seen only once
- Reverse DNS suggests **ISP pools**
  - adsl-pool.sx.cn
  - kd.ny.adsl
  - online.tj.cn
- Majority of probes come from **three** autonomous systems
  - ASN 4837, 4134, and 17622
Where are the probes coming from?

- Collected **16,083** unique prober IP addresses
- **95%** of addresses seen only once
- Reverse DNS suggests **ISP pools**
  - adsl-pool.sx.cn
  - kd.ny.adsl
  - online.tj.cn
- Majority of probes come from **three** autonomous systems
  - ASN 4837, 4134, and 17622
Are probes hijacking IP addresses?

● While probe is active, **no other communication** with probe possible
  ○ Traceroutes **time out** several hops before destination
  ○ Port scans say all ports are **filtered**

● What do probes have in common?
  ○ IP TTL
  ○ IP ID
  ○ TCP ISN
  ○ TCP TSval
  ○ TLS client hello
  ○ Pcaps online: [nymity.ch/active-probing/](https://nymity.ch/active-probing/)
What do probes have in common?

● All probes...
  ○ Have narrow IP TTL distribution
  ○ Use source ports in entire 16-bit port range
  ○ Exhibit patterns in TCP TSval
● Does not seem like off-the-shelf networking stack
● User space TCP stack?
TCP's initial sequence numbers

- TCP uses 32-bit initial sequence numbers (ISNs)
- Protects against off-path attackers
- Attacker must guess correct ISN range to inject segments
- Every SYN segment should have random ISN
What we expected to see
What we did see
What we did see
TLS fingerprint

- Probes all share uncommon TLS client hello
- Not running original Tor client
  - No randomly-generated SNI
  - Unique (?) cipher suite
- Measured on a busy Tor guard relay:
  - Observed 236,101 client hellos over 24 hours
  - Only 67 (0.02%) had identical setup
  - Recorded only client hellos, no IP addresses
Tor probing

Active prober → Tor bridge

Establish TCP and TLS connection

VERSIONS

VERSIONS

NETINFO

Close TCP and TLS connection
Physical infrastructure

- **State leakage** shows that probes are controlled by **centralised entity**
- Not clear **how** central entity controls probes
- **Proxy network?**
  - Geographically distributed set of proxy machines
- **Off-path device in ISP’s data centre?**
  - Machines connected to switch mirror ports
Blocking is reliable, but fails predictably
In 2012, probes were batch-processed
Today, probes are invoked in real-time

- Median arrival time of only **500 ms**
- Odd, linearly-decreasing **outliers**
Blocked protocols
Protocols that are probed or blocked

- **SSH**
  - In 2011, not anymore?

- **VPN**
  - OpenVPN occasionally
  - SoftEther

- **Tor**
  - Vanilla Tor
  - obfs2 and obfs3

- **AppSpot**
  - To find GoAgent?

- **TLS**

- **Anything else?**
Oddities in obfs2 and obfs3 probing

- Tor probes don’t use reference implementations
  - obfs3 padding sent in one segment instead of two
- Probes sometimes send duplicate payload
  - State leakage?

```
2014-08-29 15:44:01  60.216.143.31  obfs2  eef890766636...
2014-08-29 15:44:01  14.135.253.56  obfs2  eef890766636...
2014-08-29 15:44:02  14.135.253.56  tls    160301
```
Probe type and frequency since 2013
Find your own probes

- SoftEther: POST /vpnsvc/connect.cgi
- AppSpot: GET /twitter.com
- tcpdump ‘host 202.108.181.70’
- More instructions on nymity.ch/active-probing
Trolling the GFW
Block list exhaustion

```bash
for ip_addr in "$ip_addrs"; do
    for port in $(seq 1 65535); do
        timeout 5 tor --usebridges 1 --bridge "${ip_addr}:${port}"
    done
done
```

One /24 network can add **16 million** blocklist entries
File descriptor exhaustion

- Processes have OS-enforced **file descriptor limit**
  - Often 1,024, but configurable
  - Every new, open socket brings us closer to limit

- What's the **limit** for active probes?

- Attract many probes and **don't ACK data**, **don't close socket**

- Will GFW be **unable** to scan new bridges?
Make GFW block arbitrary addresses

● See VPN Gate’s “innocent IP mixing”
  ○ See censorbib.nymity.ch/#Nobori2014a

● For a while, GFW blindly fetched and blocked IP addresses

● Add critical IP addresses to server list
  ○ Windows update servers
  ○ DNS root servers
  ○ Google infrastructure

● GFW operators soon started verifying addresses
Circumvention
Problems in the GFW's DPI engine

- DPI engine must **reassemble stream** before pattern matching
- TCP stream often not reassembled
  - Server-side manipulation of TCP window size can "hide" signature
  - Exploited in brdgrd: [gitweb.torproject.org/brdgrd.git/](http://gitweb.torproject.org/brdgrd.git/)
- Ambiguities in **TCP/IP parsing**
  - See [censorbib.nymity.ch/#Khattak2013a](http://censorbib.nymity.ch/#Khattak2013a)
- TCP/IP-based circumvention **difficult to deploy**
  - "Hey, how about you run this kernel module for me?"
Pluggable transports to the rescue

- **SOCKS interface** on client
- Turn Tor into something else
  - Payload
  - Flow
- Several APIs
  - Python
  - Go
  - C
Pluggable transports that work in China

- **ScrambleSuit**
  - Flow shape *polymorphic*
  - Clients must prove knowledge of *shared secret*

- **obfs4**
  - Extends ScrambleSuit
  - Uses Elligator elliptic curve key agreement

- **meek**
  - Tunnels traffic over **CDNs** (Amazon, Azure, Google)

- **FTE**
  - Shapes ciphertext based on *regular expressions*

- *More is in the making!*
  - WebRTC-based transport
Q&A

Roya Ensafi — rensafi@cs.princeton.edu — @_royaen_

David Fifield — david@bamsoftware.com

Philipp Winter — phw@nymity.ch — @phw

Code, data, and paper: nymity.ch/active-probing/