CS 798
Privacy in Computation and Communication

Module 7
Privacy in Communication: Censorship Resistance

Fall 2025

Internet surveillance and censorship

- As we mentioned earlier in the course, Internet surveillance and censorship happens all over the world
    - Even in countries you wouldn't immediately think of
- Surveillance typically does not *directly* interfere with Internet traffic
- Censorship does attempt to actively restrict information flows

Types of Internet censorship

Internet censorship can target two broad classes of online actions:

- What people can publish / post / say / chat about online
- What people can view / read online

How does Internet *surveillance* limit these classes of actions?

Who is the censor?

- Internet censorship is often done at a government or national backbone level
    - National firewall
- It can also be at an ISP level
    - Possibly direct or indirect instructions from governments
    - Different ISPs in a single country may implement different censorship policies
- Or at an app level
    - Individual apps may have *blocklists* in them of sites the app will not let you access, or words or phrases the app will not let you say

What can the censor see?

- App-based censor
  - Basically everything
  - How to talk about things the censor doesn't want you to talk about?
- Network-based censor
  - Packets you are sending and receiving
  - Packet headers: IP addresses, ports
  - Packet contents: may or may not be encrypted

What can the censor see?

Unencrypted protocols:

- Everything
- They can block selected traffic based on destination, metadata, content, or many other things

What can the censor see?

TLS

- The IP addresses of client and server
- As you saw in Assignment 1, typically the server hostname as well (SNI), but sometimes not
- What exactly the client is doing within the TLS connection is not *explicitly* revealed
    - e.g., what page on a web server is being fetched
- But packet sizes, directions, and inter-packet timings *are* revealed

What can the censor see?

VPNs and Tor

- The IP address of the client
- The IP address of the VPN server or Tor guard node
- *Not explicitly* the IP address or hostname of the destination server
- But still (usually) the packet sizes, directions, and inter-packet timings

What can the censor see?

What can the censor do with the packet sizes, directions, and inter-packet timings?

- In the TLS scenario, for example, the censor can tell what website the client is accessing
- It can crawl that website to find the sizes of each of the pages on it
  - And other things, like the number of embedded images
- It can match that to the pattern of traffic it sees from the client

Webpage fingerprinting

- This is called *webpage fingerprinting*
- It is a classification problem that you could approach using machine learning
    - But you don't always need such heavyweight tools
    - See Assignment 4, where you will do this manually
- This is typically a *closed-world* classification problem
- Webpage fingerprinting can be used to block individual pages on a TLS website
    - A censor might want to do this for youtube, github, etc.

Website fingerprinting

- In the VPN or Tor scenario, the destination server identity is not explicitly known to the censor
- The censor could still crawl websites they would like to block, and try to tell whether the client's traffic pattern matches one of them
  - But the client could also be accessing an allowed website
  - This is far more likely, in fact
  - This is typically an *open-world* classification problem

Webpage and website fingerprinting

- Webpage and website fingerprinting will typically have both false negatives and false positives
- But the *base rate* is also very low
- So even a very low false positive rate can be problematic for the censor

$\Rightarrow$ Base rate fallacy

- Suppose a base rate of 1 in 100,000 webpage visits is to a disallowed page
- If the fingerprinting classifier is 99.9% accurate (1 in 1000 false positive rate), what happens?

What do censors block?

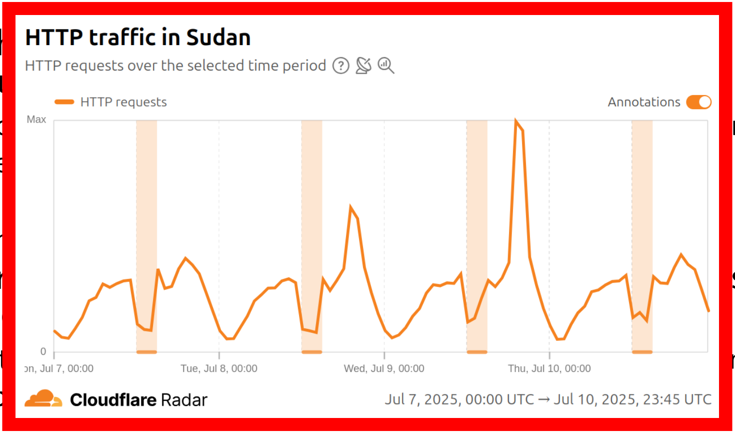
- Because of this, we don't typically see censors using webpage or website fingerprinting to block accesses in practice
  - But they may still be using them to mark a particular client as "suspicious" internally
- Censors *do* block traffic by other criteria:
  - IP addresses (possibly only traffic to particular ports)
  - Host names
  - Contents of traffic (if unencrypted)
  - Protocol being used
  - If it appears to be circumvention traffic

Censorship circumvention

- Tools that protect communication metadata (see Module 6) can be useful as *circumvention tools*
  - Tools that allow people to get around Internet censorship and access the free and open Internet
- Since the tools protect the metadata of who you're talking to, it is hard for the censor to block communications to some servers but not others
  - Note that the censor can always shut down the whole Internet (as happens in some countries for various reasons)

Censorship circumvention

- Tools th ule 6) can be usefu
  - Too nd access the free
- Since th king to, it is hard for servers but not oth
  - Not net (as hap



Censorship circumvention

- But as long as the "normal" way to access the Internet does not protect metadata, the censor will typically just try to block the use of such circumvention tools entirely
- We'll look at the example of censors trying to block Tor

Example: blocking Tor

- The simplest thing the censor could try is to block the places from which you might download the Tor Browser in the first place
- Mostly just block `torproject.org`
- On mobile, disallow Tor apps from the local app store

Example: blocking Tor

Circumvention responses:

- Mirror sites
  - EFF, Calyx
- GetTor service
  - Autoresponders using email or Telegram
  - Reply with a link to Tor Browser hosted on major sites less likely to be completely blocked, like Dropbox, github, Google Drive

Blocking Tor nodes

- Censor's next step: try to block the *use* of the Tor network
- Recall that using the Tor network involves connecting to Tor nodes
- Also, the list of Tor nodes is public
- So the censor just blocks the IP addresses (possibly only specific ports) used by Tor nodes

Bridges

- Circumvention response: have a bunch of Tor nodes that *aren't* in the public list
- People in regions that block access to Tor can use these nodes (called *bridges*) instead of the first node in their circuits
- Why only the first node? That is, why can a Tor user just use regular listed nodes for the second and third hops in their circuits?

Learning bridges

- A key question: if these bridges aren't public, how do Tor users in a censorial regime learn them?
- Note that if the *censor* learns about a particular bridge, it can block it, just as for regular Tor nodes
- This is the *bridge distribution* problem
    - More on this later, but also on Assignment 4

Blocking bridges

- As above, the censor could try to learn about bridges the same way that the users in their country would
- But the censor could also try to observe people *using* bridges it didn't already know about
- It can watch Internet traffic, looking for connections that look like they're using the Tor protocol

Blocking bridges

- Early versions of the Tor protocol were trivially recognizable
  - The TLS handshake (which happens *before* encryption starts) had a number of Tor-specific fields in it
- Modern versions are still quite recognizable from features of the traffic (packet sizes, for example)
- Censors do *active probing*: if they see a connection to a server that looks like it may be Tor traffic, the *censor* itself tries to connect to that server using Tor
  - If the server responds correctly, it is a bridge, and the censor blocks it
  - This isn't just for Tor; e.g., China also does active probing for Shadowsocks servers

Defending against active probing

- Can bridges tell when they are being connected to by the censor rather than by a "real" Tor user?
    - If they could, they could just refuse connections from the censor, or serve them some innocuous website, or something like that
- Figure out what IP addresses the censor is using?
    - They typically have the ability to (and often do) temporarily "hijack" an arbitrary IP address in the country and use that

Defending against active probing

- Better: hand out a secret along with the bridge information
    - You can only connect to the bridge if you know the secret
- This doesn't help against a censor that learned bridge information in the same way a Tor user would
    - But the censor could just block the bridge by IP address anyway in that case
- But it does help against censors looking for bridges by watching network traffic and then trying to confirm with active probing

Pluggable transports

- But there's still the problem of the censor recognizing the traffic as Tor traffic in the first place
- Circumvention response: *pluggable transports*
- Normally, the Tor protocol is transported from the client to the bridge using regular TCP
- Tor provides a number of alternative ways to transport this data

Pluggable transports

- The general idea is that you need a two-way (reliable) channel between the client and the bridge that you can use to send Tor protocol data
- The goal is to make this channel *not* look like the Tor protocol, and to make this channel unlikely to be blocked by the censor
- They are “pluggable” because they use a common API, so they are easy to swap in and out if the censor figures out how to block one of them

Censorship circumvention

- One observation that is commonly used to the circumventor's advantage is that the censor is simultaneously monitoring a *large* number of network connections
- In order to not fall behind, it has to process packets at basically the same rate they arrive (so on the order of microseconds per packet)
- The censor equipment therefore tends to take shortcuts
  - How robust is this observation?

Censorship circumvention

- For example, Geneva (Bock et al., 2019) observed that China's Great Firewall (GFW) does not validate the checksums on packets as they fly by

- One of the censorship circumvention strategies Geneva *automatically* (using genetic algorithms) came up with was to send a duplicate ACK packet when setting up the TCP connection
  - The second copy, however, had the reset (RST) flag set, but an *incorrect checksum*

- The bridge would ignore the RST packet because of the incorrect checksum, but the GFW didn't check the checksum, saw the RST, thought the connection was closed, and *ignored* the rest of the traffic on the connection

Censorship circumvention

- There are many of these network-level tricks to make the censor's equipment think it saw something different from what the bridge actually saw
    - And so mistakenly classify it as benign traffic instead of censorship circumvention traffic
- Challenge: such networking tricks usually need OS-level access on the client, and sometimes also on the bridge, so they're harder to deploy than just running an ordinary program
- What can we easily send from the pluggable transport?
    - TCP connections, UDP packets
    - With arbitrary content

Censorship circumvention

- So what kind of traffic should we send in order to circumvent censorship?
- It somewhat depends on the censor's approach to protocols
- Blocklist: block a specific list of protocols (e.g., the Tor protocol)
- Allowlist: block everything you don't recognize as a protocol you want to permit (e.g., HTTP, HTTPS, WebRTC)
- What are the tradeoffs between a blocklist and an allowlist (for the censor)?

Strategy: look like nothing

- One strategy is to transport the (Tor protocol) data over a *fully encrypted* protocol
- Most encrypted protocols (like TLS for example) begin with an unencrypted *handshake*, and may have other unencrypted portions (containing message types, lengths, or other metainformation)
- In a fully encrypted protocol, all of the TCP content is encrypted, from the very first byte
    - The *packet headers* are still not encrypted, but that's expected

Strategy: look like nothing

What is still revealed?

- Packet sizes
- Packet timings
- The fact that the protocol is fully encrypted

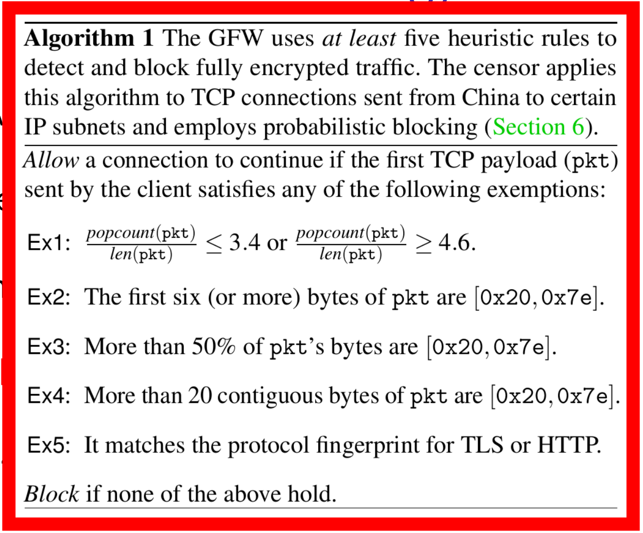
Examples: obfs4, Shadowsocks

Strategy: look like nothing

What is still rev

- Packet size
- Packet tim
- The fact t

Examples: obfs



Algorithm 1 The GFW uses *at least* five heuristic rules to detect and block fully encrypted traffic. The censor applies this algorithm to TCP connections sent from China to certain IP subnets and employs probabilistic blocking (Section 6).

Allow a connection to continue if the first TCP payload (`pkt`) sent by the client satisfies any of the following exemptions:

Ex1: $\frac{popcount(\texttt{pkt})}{len(\texttt{pkt})} \leq 3.4$ or $\frac{popcount(\texttt{pkt})}{len(\texttt{pkt})} \geq 4.6$.

Ex2: The first six (or more) bytes of `pkt` are $[\texttt{0x20}, \texttt{0x7e}]$.

Ex3: More than 50% of `pkt`'s bytes are $[\texttt{0x20}, \texttt{0x7e}]$.

Ex4: More than 20 contiguous bytes of `pkt` are $[\texttt{0x20}, \texttt{0x7e}]$.

Ex5: It matches the protocol fingerprint for TLS or HTTP.

Block if none of the above hold.

Strategy: look like allowed traffic

- A second strategy is to make your circumvention traffic look like allowed traffic
    - An allowed protocol, destined for a server that's not blocked
    - The protocol you intend the censor to see is called the *overt* protocol
    - The traffic (for example, the Tor protocol traffic) you are hiding within that protocol is the *covert* data
- This is actually quite hard to do really well
    - There are almost always ways to distinguish real traffic of a given protocol from traffic pretending to be that protocol
- But what property of the censor can we use here to our advantage?

Strategy: look like allowed traffic

- We might only have to make traffic that looks “close enough” to the real protocol
- But: the censor’s technological capabilities may improve
- And then a “close but not quite” protocol may stick out even more
- Examples: SkypeMorph, Marionette
    - You will implement a simple version of this strategy on Assignment 4

Strategy: be allowed traffic

- In order to defend against a censor being able to eventually detect the differences between a real use of a protocol and a pluggable transport pretending to use the protocol, we can alternately have the pluggable transport *actually use* the allowed protocol

- This often isn't as easy as it sounds

- You still have to protect the packet sizes and timings from the underlying Tor protocol from affecting the packet sizes and timings from the overt protocol
  - At least not too much

Strategy: be allowed traffic

Videoconferencing has some nice properties as an overt protocol

- It is extremely common, and so blocking it would be problematic for the censor
- It is relatively high bandwidth
- It is bidirectional

Examples: DeltaShaper, Snowflake

Robustness vs. throughput

When hiding covert data inside an overt protocol, there are two properties of interest:

- Robustness: how likely is it that the hiding remains effective, even as the censor's capabilities improve?
- Throughput: how much covert data (per unit time) can you hide in each direction?
  - Relatedly latency: how long does it take for your covert data to get through?

These two properties are often in conflict!

Robustness vs. throughput

- It is much easier to hide a small amount of data in any given protocol than a large amount of data
- For this reason, many pluggable transports have two phases:
- A *bootstrapping* (also called *signalling*) phase, where the client learns a small amount of information over a channel that is very hard to block
    - The client can learn bridge addresses through similar email or Telegram channels as mentioned earlier for GetTor
    - These channels are not secret, but they are hard to block

Robustness vs. throughput

- In the second phase, the client does most of its data transfer through pluggable transports, using bridge IP addresses (and other information) learned in the bootstrapping phase

- These channels *are* secret: the censor should not be able to tell which uses of the overt protocol are real uses, and which are being used to carry covert traffic

Where do the bridges run?

- When a client learns a bridge IP address in the bootstrapping phase, where does the IP address come from?
- Traditionally, a volunteer runs a server somewhere, and tells its IP address to the bootstrapping server
  - Watch out for the *censor* enrolling its own machines as bridges!
- Problem: most people have limited ability to run a server somewhere, particularly with a static IP address

IP agility

- As censors learn the IP addresses of bridges (and block them), it is helpful if more bridges could be created at new IP addresses
- They could be run by the same people as the old (now blocked) bridges, or new people
- Ideally, the circumvention tool automatically notices when its bridge is blocked, and obtains a new IP address
- Even better if it doesn't interrupt what the user is doing but they seamlessly transition from using one bridge to another

Strategies to get more bridge IP addresses

- We'll look at some strategies used to get more IP addresses that can serve bridge traffic
- Nowadays, the most common way to run a server on the Internet is in "the cloud"
  - i.e., other people's computers
- Long ago, Tor (for example) made available public cloud images volunteers could run that would start a bridge

Cloud-based bridges

- Pro: once the bridge was started, it had a stable IP address
- Pro: Anyone with a cloud account could quite easily run a bridge
- Pro: If an IP address got blocked, you could usually just shut that bridge down and start a new one on a different IP address
- Con: Quite expensive, particularly for volunteers

Cloud-based bridges

- Recent idea (SpotProxy, Kon et al., 2024): as pluggable transports gain increased IP agility, transient *Spot VMs* in clouds can be effectively used as bridges
- Spot VMs are cloud-based virtual machines that are made available at a steeply discounted price (savings of up to 90%), but which can just disappear at no notice if a higher-paying customer arrives
  - Makes costs to volunteer bridge operators much more reasonable
  - Heavily relies on IP agility

Serverless bridges

- IP agility also admits another avenue: serverless bridges
- People run bridges without running servers
- For example, they run a bridge *in their web browser*
- Snowflake is such a style of bridge

Snowflake bridges

- You can run a snowflake bridge just by opening a tab in your web browser at https://embed-snowflake.torproject.org/
- As long as the tab is open, your bridge is running
- You can also install a browser extension that effectively just visits that page in the background all the time
- (Snowflake is also available as a server you can run on your own machine or a cloud machine, if you prefer that style of bridge)

Snowflake bridges

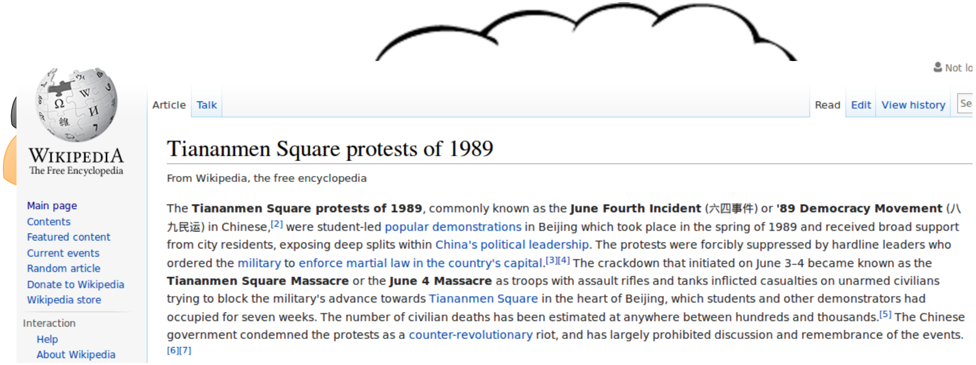
- Tricky bit: users are often behind Network Address Translation (NAT), which can limit the ability to receive incoming connections from the Internet
- The Snowflake pluggable transport uses WebRTC (the standard web-based audio/video communication protocol) as the overt protocol, which already has support for getting around the NAT problem

Router-based censorship circumvention

- Another source of IP address for bridges is to give censored users IP addresses to connect to *that don't actually have bridges running at them*
- Then deploy *routers in the network* that can detect censorship circumvention traffic destined for those IP addresses, and send that traffic to an actual bridge
- Examples: Slitheen (Bocovich & Goldberg, 2016), Conjure (Frolov et al., 2019), NetShuffle (Kon et al., 2024)

Router-based censorship circumvention

Router-based censorship circumvention

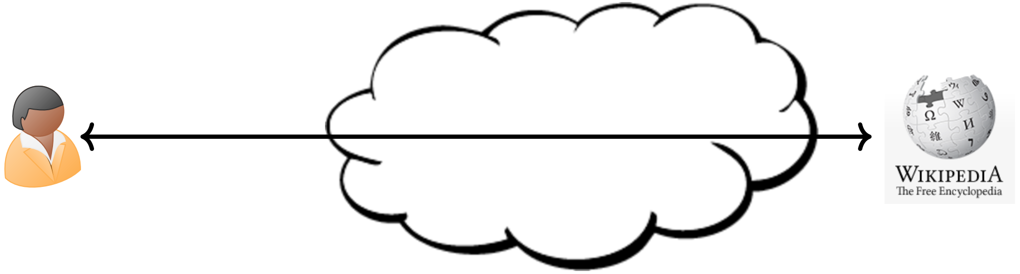


Tiananmen Square protests of 1989

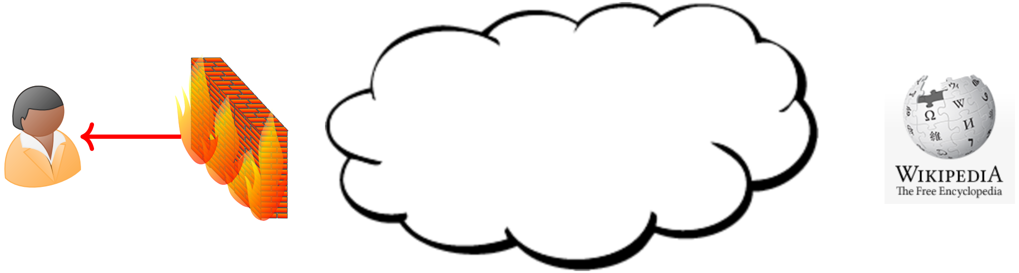
From Wikipedia, the free encyclopedia

The **Tiananmen Square protests of 1989**, commonly known as the **June Fourth Incident** (六四事件) or **'89 Democracy Movement** (八九民运) in Chinese,[2] were student-led popular demonstrations in Beijing which took place in the spring of 1989 and received broad support from city residents, exposing deep splits within China's political leadership. The protests were forcibly suppressed by hardline leaders who ordered the military to enforce martial law in the country's capital.[3][4] The crackdown that initiated on June 3–4 became known as the **Tiananmen Square Massacre** or the **June 4 Massacre** as troops with assault rifles and tanks inflicted casualties on unarmed civilians trying to block the military's advance towards Tiananmen Square in the heart of Beijing, which students and other demonstrators had occupied for seven weeks. The number of civilian deaths has been estimated at anywhere between hundreds and thousands.[5] The Chinese government condemned the protests as a counter-revolutionary riot, and has largely prohibited discussion and remembrance of the events. [6][7]

Router-based censorship circumvention

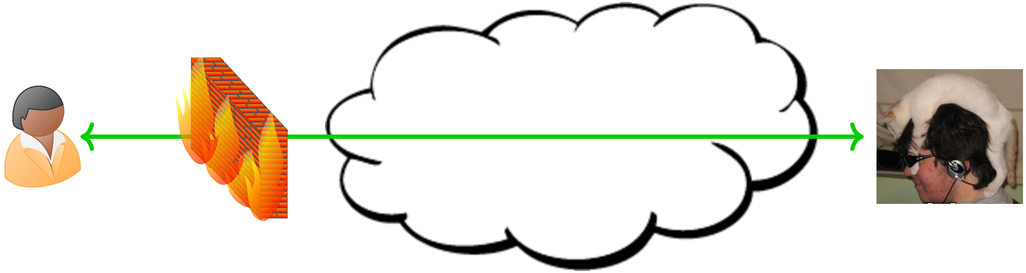


Router-based censorship circumvention

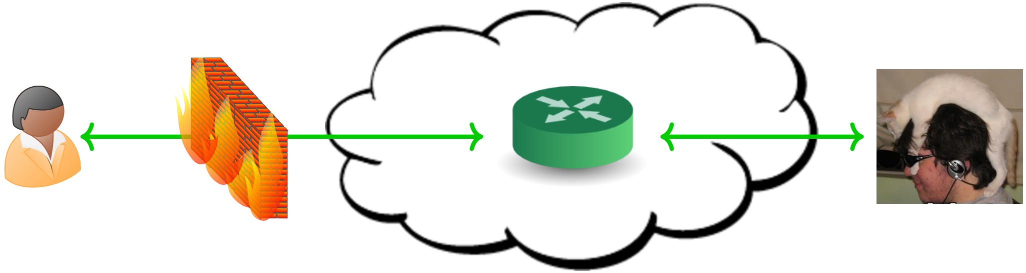


Router-based censorship circumvention

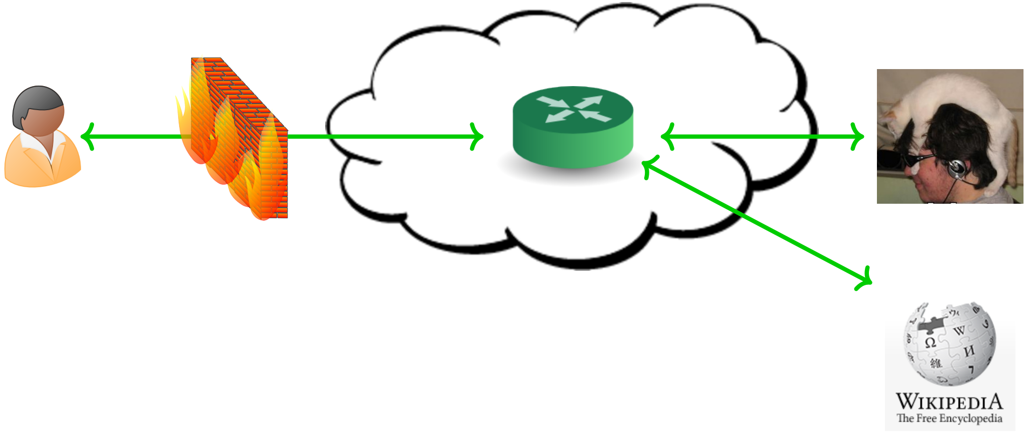
Router-based censorship circumvention



Router-based censorship circumvention



Router-based censorship circumvention



Bridge distribution

- We mentioned earlier the *bridge distribution problem*:

- How do we give bridge information to censored users while preventing *the censor* from learning them?
  - If the censor learns the bridge in the same way as the intended users do, it can block the bridge
  - Recall this is different from the censor trying to learn bridges by watching network connections, or active probing

- This problem is in some sense impossible to solve perfectly

Bridge distribution

- One technique is *rate limiting*
- Limit the rate at which new bridges are handed out
- Either overall, or to individual requesters
  - But if the latter, then you need to be able to limit the ability for the censor to pretend to be many different requesters
  - Sometimes you “farm this out” to another service that already tries to limit the number of accounts one person can have, like gmail
- The rate will depend on how many bridges you have available, compared to how quickly you see new users

Bridge distribution

- But it would still be helpful to be able to distinguish the censor from “real” bridge users
- Key observation: we can leverage the *social graph*
    - Who is friends with whom?
    - In real life, not social network “friends”
- The observation is that it’s hopefully rare that a real bridge user (unknowingly) has a censor as a friend

Bridge distribution

- Simplest version: people give bridge information to their friends (and not to the censor)
- Problem: How do you start the process?
- Problem: What happens if someone *is* unknowingly friends with a censor?
- Problem: What if you don't have any bridge-using friends?

Bridge distribution

- Add a notion of “reputation”
- Bridge users gain reputation if their bridges remain unblocked, lose reputation if their bridges get blocked
    - Higher reputation makes it easier to get and share bridges
- Censor-friend has a conundrum: block the bridge it learned (and lower both its own and its friend’s reputation score), or leave the bridge unblocked and keep a high reputation to try to learn more bridges in the future

Bridge distribution

- Who keeps track of users, friend connections, and reputations?
- The simple solution (e.g., Salmon (Douglas et al., 2016)) is the *bridge authority*
- The bridge authority knows everything about all bridges in the system, all users in the system, which user knows about which bridge, which users are friends with which other users, all users' reputations
- Problems with this approach?

Bridge distribution

- In Lox (Tulloch & Goldberg, 2023), the bridge authority knows:
  - The set of all bridges in the system, and whether they have been blocked or not
- It does *not* know:
  - The set of users
  - Who is friends with whom
  - Which users know about which bridges
  - The reputation of each user
- Who *does* know these things?
  - The users themselves!

Bridge distribution

- In Lox, users can interact (anonymously) with the bridge authority to:
    - Invite friends
    - Join Lox with an invitation from a friend
    - Join Lox without an invitation
    - Raise their reputation score
    - Get new bridges if theirs are blocked (but lower their reputation score)
- All using *zero-knowledge proofs* so that the bridge authority cannot learn any sensitive information
- The Tor Project is currently implementing Lox

Internet shutdowns

- As briefly mentioned before, a country's regime sometimes creates a complete Internet shutdown
    - e.g., just in 2024, India, Ethiopia, Iran, Myanmar, Sudan, Bangladesh, . . .
- The challenge to the censorship circumventor is *much* larger
- Possible goals:
    - Full interactive Internet access
    - Just get messages out to the world (one way)
    - Communicate locally ($\sim$ city-wide), but not globally

Internet access with Internet shutdowns

- You'll need to find a channel that isn't blocked
- Satellite? If the satellite company allows people in your country to use it
- Possibly cellular voice? India's 2021 Internet shutdown, for example, still allowed voice calls
  - ⇒ Dolphin (Kumar Sharma et al., 2023) implemented an old-style audio modem, but updated to work with the voice compression used on cellular voice channels, to communicate with its bridge via an audio telephone call

Getting messages out

- Some smartphones (iPhone 14 and later) have built-in satellite communication
- But just for sending a small amount of data
  - The intended use is to send your location if you're lost in the mountains or something like that
- It turns out you can sneak a small amount of arbitrary data out through this channel (Heinrich, 2024)
  - Even in some countries where this satellite feature isn't explicitly supported
  - Only sending, no receiving

Local communication

- Even if all communication channels are blocked, you may still want to communicate *locally*
    - To organize a protest, for example
- *Mesh networking* apps use direct phone-to-phone communication (Bluetooth, Wi-Fi Direct) to exchange messages, which can be sent directly or relayed through a "mesh" of phones to spread the range
- Apps like Firechat, Bridgefy, Briar have been used in the past
    - None of those protects local communication metadata

Local communication

- More recent systems protect both the contents of private messages and metadata
- Moby (Pradeep et al., 2022)
    - One-to-one communication (messaging functionality)
- Anix (Kamali and Barradas, 2025)
    - One-to-one, group, and microblogging (anonymous broadcast) functionality

Wrapping up

- In this course, we looked at many ways to achieve privacy in computation and communication
- Doing so keeps sensitive information out of the hands of attackers who might compromise servers, of companies that might exploit the information, and sometimes even of governments that might not be working in your interest
- When you go on to design and build systems that perform computation and communication, please keep these lessons and techniques in mind, to avoid causing the next big data breach, and to protect privacy, freedom, and autonomy online!