Circumventing Censorship on the Internet

Merlin Denker
RWTH Aachen University
merlin.denker@rwth-aachen.de

***Abstract*—In todays connected world the constant availability of information through the internet has become self-evident for most humans with access to a computer. Some parties therefore want to block or monitor the access of certain information. To circumvent these restrictions, different technical solutions have emerged to protect both content providers and their users. In this paper we will take a closer look at some solutions and evaluate if and how good they solve the problems they were designed for. In particular we will discuss the design goals of Tor, Infranet and Freenet, whether they effectively contribute to anonymity and which possible attack scenarios exist. In the conclusion we will summarize our findings and recommend which software is best suited for which case.**

I. Introduction

The widespread access to the internet allows a large portion of humanity to access the knowledge that has been accumulated by generations. Also it has become very easy to communicate across borders and share thoughts with people that one would otherwise never had talked to. People can also get a better view of how life in other regions is and compare it to their situation. These possibilities can be threatening to governments that suppress their populace, because people can see that there are systems that work out better than the one they live in. They can exchange thoughts about this and organize movements. Information about misbehavior of the government spreads easily and rapidly. Therefore a lot of governments actively try to censor data that is not in their favor and block communication among people they deem dangerous. Monitoring and filtering unprotected internet traffic is not really hard and various governments around the world already use such measures. It is worth noting that there might be other parties monitoring network traffic that a user wants to protect against, but we will focus on governments due to their unique power.

In regions where access to certain material is prohibited and internet traffic is monitored various solutions have come up to enable users to gain access to those information while keeping the risk of being detected low. A very popular technique is the use of virtual private networks or proxy servers. But since most of the commercial providers of such services are forced to keep logs of their users actions or might be compromised easily, these solutions no longer meet the criteria required by people who want to stay anonymous. Various more sophisticated approaches have come up and are actively used in todays internet.

Probably the most well known project is The Second-Generation Onion Router, also known as Tor. Tor uses a decentralized infrastructure to hide the communication paths between clients and servers. This allows clients to access data without the server knowing who requested it and also enables server owners to provide their services anonymously with the Tor location-hidden services feature.

Another approach that tackles the problems differently is the Infranet project. While Tor aims to hide who is communicating with whom, Infranets goal is to be as unsuspicious as possible. The use of Infranet is supposed to be completely hidden by embedding the Infranet requests in multiple unsuspicious HTTP-Requests to a so called Infranet Responder. This responder then fetches the requested resource from another server and transmits it back to the requester using techniques like steganography to hide the contents within images.

These two examples already show that the goals of software that aims to make anonymous use of the internet can vary. Key criteria that most of these projects tackle include

- hide who is requesting specific information,
- hide who is providing specific information,
- hide that there is a communication going on and
- hide the origin of data
- while still being able to verify that transmitted data has not been altered.

These goals often come at the cost of usability. While the biggest usability problem often is performance, such software could also require technical knowledge the average user does not have or secret knowledge such as entry points to a hidden network.

Throughout this paper we will assume that our adversary is a governmental censor, since these have high capacities and a more far reaching influence than private parties. Still governments also do not have full control over the internet, not even over all communication going on within their borders. They can monitor a large portion of traffic within their borders and it can be assumed that they can monitor near to all traffic that is passing their borders, if they wish to do so.

This paper will take a closer look at three projects, namely Tor, Infranet and Freenet (in that order). We will discuss their main goal and whether they are effective in reaching it. To do so, we will examine their design goals, evaluate whether those contribute to increased security and if they are fulfilled. At the end of this paper we will compare all three projects against the main criteria we have set up in this introduction and see that there is no perfect solution for all these problems.

II. The Second-Generation Onion Router

We now take a look at what problems Tor explicitly does not try to solve and what complications arise from those decisions.

After that we will check Tors actual goals and verify whether they have been met.

A. *Design Non-goals*

These are the design non-goals as listed in [1].

1) Not steganographic: The design goals of Tor explicitly state that Tor does not aim to conceal who is using the Tor network. Therefore, if a user lives in a region where access to the Tor network itself is prohibited, the sheer use of Tor might not be possible or even get the user into trouble.

2) Not peer-to-peer: A fully peer-to-peer oriented approach is more robust against attacks that try to shut it down or block it, since there is no single point of failure. But such networks have other unsolved problems [1] and the authors decided against this solution. As we have already seen, it would be easy to block the use of the Tor network anyway since it is not steganographic. The benefit of a fully peer-to-peer oriented solution is therefore not as big in Tors case as it might be somewhere else. Still, with a more centralized architecture the network is easier to compromise by an adversary which impacts the security. Tor tackles this problem with two approaches: Clients can decide the route through the Tor network themselves and can use only nodes they trust. Also the so-called leaky pipe topology allows clients to change the routes on the fly. An adversary would therefore need control over a large portion of the networks nodes to be able to successfully compromise its security.

3) Not secure against end-to-end attacks: With control over enough exit nodes an adversary might be able to identify users via traffic shaping or timing attacks. Tor does not actively try to solve these problems but instead relies on the idea that a large enough number of exit nodes are too hard to control for a limited adversary. Also since users can choose their exit nodes and run their own entry nodes they have a lot of control over these two critical parts of the communication path. It is therefore highly unlikely that an adversary could get enough control over the network that a timing or traffic shaping attack would become a significant threat.

4) No protocol normalization: Tor does not do any protocol normalization, which means that applications running on top of it might easily expose the identity of the user. Therefore in most cases Tor has to be used in combination with special software in order to maintain anonymity of its users. For example, if a website ran a script in a clients browser that reads out the machines IP address locally and transmits it back over the secure channel, then the server will know who the client is. Also, if a user logs in to websites with an account that is associated with his real name through the Tor network, then that user cannot expect to remain unidentified. A widely used tool to combat these problems is the Tor browser which aims to make the client look as indistinguishable as possible by mimicking the most common browser features. Since all users of the Tor browser look the same to servers, it is not possible to tell them apart without further information. But the fact that they all show the same characteristics makes it easy to tell that the client is in fact using the Tor browser. It has been shown that the Tor browser can effectively be detected[2].

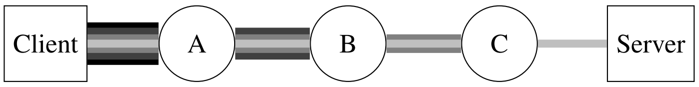

Fig. 1. Visualization of a layered communication channel. The inner communication channels are wrapped by and tunneled through the outer ones.

B. *Design goals*

We will now discuss the problems Tor wants to solve and verify whether these measures are effective.

1) Usability: Tors design goals explicitly state that usability is a security feature, since "anonymity systems hide users among users" [1]. Tor should therefore be easy to adapt and not require additional software or the manipulation of existing software. As we already discussed, Tor does not make use of protocol normalization and therefore does in fact need additional or modified software in most of its use-cases. It also adds a considerable delay to communication which prevents users from using Tor for their everyday normal traffic. This can easily be used as an argument by a censor to block all Tor-traffic, since the vast majority of innocent traffic does not use the network anyway. The usability goal can therefore not be considered fulfilled.

2) Prevent attackers from linking communication partners: Tors main goal is "to frustrate attackers from linking communication partners" [1]. The way to achieve this goal is to build a layered communication path in which every node only knows its predecessor and successor. The client opens a secure communication channel with the first node and can then order this node to expand the path to a specific successor node. The entry-node will then open a secure communication channel to it and from then on forward the requests of the client to this node. The client itself opens another secure communication channel with the new node through the two outer channels. This can be repeated as often as the client wishes, adding more layers of secure tunnels around the communication path. Finally, the client decides that there are enough layers to securely communicate and chooses the last node in the path as its exit-node. This exit-node will then forward traffic from and to the client via the secure path.

We will now look at who gets knowledge about which parts of the path. The client is only known to the entry-node. Each succeeding node will only know its predecessor and successor, therefore it can not know if its predecessor was the entry-node or if its successor is the exit-node. So we can conclude that the entry- and exit-node are only known to the client itself. The complete path in between these two nodes is also only known to the client, since every node only gains knowledge about a part of the path. It can not gain any knowledge about the length of the path and the nodes that are not directly adjacent to it.

We can conclude that without an active attacker the only one knowing the entry-node and the exit-node is the client itself. Entry- and exit-nodes do not know each other.

An adversary running a hostile node could however compromise the path. A possible attack scenario would be to loop the path through itself to impersonate multiple nodes in the path, possibly even the exit-node. If such a hostile could convince users to choose it as their entry-node the adversary would get complete knowledge of the full path and could read all data passed along it. Tor solves this problem by requiring nodes to authenticate themselves using asymmetric keys. An adversary would need access to multiple nodes private keys in order to impersonate them, which makes such an attack highly unlikely. A user running an own entry-node would also be less affected by this strategy, as long as the adversary does not know that this entry-node is linked to the specific user.

Another possible attack is to compromise a big number of nodes and observe the traffic going through them. If traffic analysis suggests that two endpoints are communicating with each other this could then be verified through timing attacks and traffic shaping. As we already discussed in chapter II-A3, solving these problems is not a design goal of Tor. Compromising enough nodes and convincing the directory servers that they are trustworthy and not linked to each other would be a very hard task. Also, the leaky-pipe-topology of Tor allows any node on the path to be the exit-node, therefore complete monitoring of all traffic on a path becomes even harder.

Overall Tor is pretty effective in conceiling the communications partners. Even attackers with a lot of technological expertise and influence on a significant portion of the internet like the NSA can not effectively compromise the Tor network[3].

C. Protecting service providers

Tor does not only provide protection for end-users, but also for service providers. We will now first take a look at how this works and then discuss its effectiveness and possible weaknesses.

The concept Tor uses is called location-hidden services. A provider of a hidden-service, lets call him Bob, can easily setup such a service by registering it in the so called lookup service. First of all, the provider has to generate an asymmetric key pair to authenticate his service and to change its setup later. He then registers his service at the Tor lookup service, announcing several onion routers as his introduction points. As the last step he builds circuits to his introduction points and tells them to wait for connections. A client, we will call her Alice, who wants to connect to the hidden service can connect to the lookup service to receive the details of the hidden service. She then chooses an onion router as a so called rendezvous point, connects to one of the introduction-points and leaves a message there for the service provider that tells him about the rendezvous point she chose. Along with this information, she passes a rendezvous cookie that Bob can use to authenticate himself at the rendezvous point and the start of a Diffie-Hellman handshake. All those information are signed with the public key of Bobs hidden service. Alice then opens a Tor connection to her rendezvous point and waits for Bob to connect. Bob receives Alices message from the introduction point and opens a Tor connection to the rendezvous point on his own. He transmits Alices rendezvous cookie and the second half of the Diffie-Hellman handshake. The Rendezvous point then connects Alices and Bobs circuit. It is worth noting that the rendezvous point knows neither Alice nor Bob and cannot decrypt any of their communication since the Diffie-Hellman handshake is already complete when the circuits are connected. Alice can now open a TCP connection to Bobs hidden service via this secure connection.

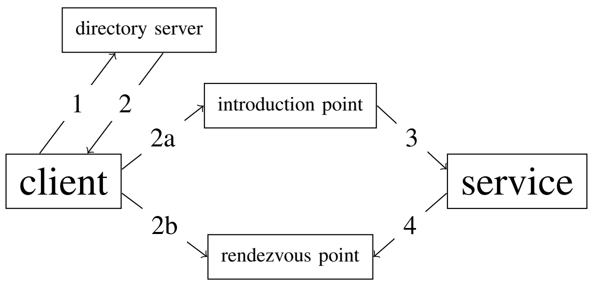

Fig. 2. Example of the establishment of a connection to a hidden service. All connections are tunneled through Tor.

1) Denial of Service attacks: As with all online services, hidden services are a subject to Denial of Service (DoS) attacks by flooding the service with connections or requests. For the hidden services themselves, this is not a big issue with Tor, since every client that wants to connect will have to do this via an introduction point and provide the first half of a Diffie Hellman handshake and a rendezvous cookie encrypted with the services public key. That means that opening a connection is a costly operation and the service can just deny connections that do not provide correct information. If the service is experiencing heavy load the server can also prioritize certain clients that may have higher priority to it and just deny all other connections.

There is however another point that can be attacked with a DoS attack: The introduction points. These nodes do not verify the information clients use to announce themselves to the hidden service and can therefore be flooded with relatively low-cost connection requests. It would however require significantly more clients to perform such an attack instead of blocking just one host, since multiple nodes have to be blocked. Even if an attacker would succeed in this the owner of the hidden service could easily update the information on the lookup service and use other introduction points. It is also possible that the owner of the service hands out lists to trusted clients out-of-band that tell them which introduction points will be used in the future[1]. Another option for the service provider is to use introduction points that are not published via the lookup service and provide this information out-of-band.

The service could also be denied if the service provider chose a compromised node as its introduction point. But since the service provider and the clients all connect via secure Tor circuits to it, the attacker could not gain any knowledge about them this way. He could however just block all connection attempts and therefore deny connections via this introduction point. The service provider would probably notice this pretty soon, update its service information on the lookup service and not use this node anymore.

A Denial of Service attack is therefore not to be considered a real threat for a location-hidden service in Tor.

2) Compromise Rendezvous points: There are two possible scenarios of an adversary using a compromised rendezvous point. The first one is to just have a compromised node in the Tor network and wait for clients to use it as a rendezvous point. As we already discussed before in II-B2, the rendezvous point can not gain any knowledge about the two communication partners or the data they exchange. Also it is very unlikely that a specific node is chosen by a service or a client that the adversary tries to monitor, so he would have to compromise a lot of nodes. This is not really practical and would not yield any benefit anyway.

The other possible attack scenario is the attacker connecting to the service on its own, using a compromised rendezvous point. Again, there is no way of gaining knowledge about who the service provider is, neither for the client nor the rendezvous point. This attack scenario does not provide any advantage above the one described previously.

All in all we can conclude that compromising rendezvous points is not a reasonable attack scenario for Tors hidden services.

3) Evaluation of location-hidden services design goals: The first design goal for location-hidden services listed in [1] is access-control. A service provider should be able to filter which incoming connections he wants to accept. As we already discussed in II-C1, this goal is fulfilled.

Another goal is robustness, which is described as three distinct features:

- being able to maintain anonymity even in case of router failure
- not being tied to a single onion router
- ability to migrate the service to other onion routers

The first feature is fulfilled since the service provider is only authenticated via his private key and the only nodes that verify his identity are accessed via a secure Tor connection. So even the nodes that verify him do not know who he really is. His entry-node on the other hand does not know that he is providing a hidden-service in the first place. The other two features are also given, since a service can and should use multiple introduction points and is able to update the list of introduction point as he wishes.

Smear-resistance is also a design goal. This means that nobody else should be able to impersonate the hidden service and trick clients into connecting with a fake server. This is achieved by the clients providing a rendezvous-cookie at the introduction-point, which is encrypted with the services public key. Only the legitimate service provider can decrypt this and send it back to the client at the rendezvous-point.

The last design goal for location-hidden services specified in [1] is application-transparency, which means that users and developers should not have to manipulate their existing software to use Tor hidden services. If one is using a Tor proxy already, every standard TCP application can use Tor connections to send its data on a secure channel. The proxy software is responsible for building the connection to the hidden-service or the client. All other standard software is not affected in its use and does not have to be changed in order to work with Tors hidden services. However, as discussed in II-B1, this software might need to be altered in order to not divulge its location or identity data. This problem is the same for hidden services as it is for all other Tor connections.

III. Infranet

Infranet is a project that aims to protect its users from censorship by disguising the access of censored data as innocent web traffic which is not blocked or prohibited. This is achieved by building a communication channel in which the client, called Infranet requester, communicates which resource it actually wants to request via a series of standard HTTP requests. The URLs that are available for communicating on the Infranet channel are chosen by the server. After a series of HTTP requests, that form a valid Infranet requests, have been made, the server, called Infranet Responder, fetches the requested data on behalf of the client and embeds it in another resource using steganographic methods. The resource in which the requested data is hidden is then transmitted back to the Infranet requester.

For this to work the responder obviously has to have Infranet software running and should also serve normal HTTP content as well, or it would be easi to tell that something suspicious is going on. The responder should also be in a different location than the requester, where access to the requested data is not blocked, otherwise the responder could not catch the resources for the requester in the first place.

While the requester needs to contact a webserver of which he knows that it is an infranet responder, such knowledge must not be made public. If this information was publicly available, an adversary could just block access to this server and therefore render the infranet responder useless. This means that the information which servers are infranet responders has to be communicated out-of-band, for example via physical devices. In addition to that, a client has to have access to the infranet requester software, which might be hard to get in regions with censored internet.

A. Design goals

We will now take a look at the design goals of infranet as mentioned in [4] and discuss whether those goals are met.

1) Deniability for any Infranet requester: The main goal of infranet is of course that no attacker should be able to tell whether any person is requesting data via Infranet. Infranet responders have to be run alongside usual web servers, otherwise it would be obvious that somebody is using infranet. In addition to that, the URLs used to communicate with the responder must lead to real resources that are available on the webserver. Given these premises and the fact that Infranet uses otherwise unmodified HTTP requests leads to the conclusion that without comprimising the requester or the responder it is not possible to tell for sure that there is an Infranet communication going on.

2) Statistical deniability for the requester: While it might not be possible to tell the usage of Infranet with certainty, a censor might monitor the traffic of a client and notice statistical anomalies. One possibility would be a client that accesses certain URLs very often, that are otherwise used rarely by normal web users. To tackle this problem, the operators of Infranet responders are encouraged to use real access logs of their web servers so they can figure out which URLs are used how often in what patterns. This allows the responder to provide URLs to the requester that mimic standard browsing behaviour. Another possible anomaly a censor could notice is that requesting a certain resource leads to different responses every time. This will happen, because the responder embeds the requested data in another resource, for example an image. Images usually do not change their content every time they are accessed, so this would be suspicious. Operators of Infranet responders should make sure to use changing resources like webcam images when hiding data. Those images for example change very frequently and it is therefore not possible to tell that there is information hidden in them.

3) Responder covertness: As we already discussed it is critical that responders are not known to the censor. Since the information about which server is a responder is communicated out-of-band we can consider this goal to be met by design. If the statistical deniability for the requester is fulfilled then there is also no way to be certain that a server is a responder.

4) Communication robustness: This design goal is described as being able to continue covered Infranet communication in the presence of an adversary actively trying to block it. If a censor believes to have identified an Infranet responder, the only way for him block the Infranet communication should be to block all normal traffic to that server as well. In [4] it is considered to use SSL as a communication channel since a lot of innocent content is provided via SSL and blocking SSL would block most normal traffic as well.

Yet one problem emerges: If the censor has control over all communication going on and uses some kind of cache he might transmit a cached resource requested by a client instead of forwarding the request to the Infranet responder. The responder would never get the command to fetch a certain blocked resource and even if it did it could not transmit a modified resource with hidden information in it, since the censor will just serve this resource from its cache. A website that displays webcam images will fundamentally not work if such a cache is in place, so it would be reasonable for the censor to not serve this specific resource from cache. However, there is no reason not to serve the URLs, used to communicate what resource is requested, from cache. Therefore the requests will ultimately never be delivered to the Infranet responder. The only way to tackle this would be to use unique URLs that are not cached yet, but that would be suspicious and harm deniability for the requester and the responder.

5) Performance: While [4] explicitly states performance as a design goal, it is not really clear why that is important. Good performance is always nice to have, but in this particular case it will not increase security for its users. Tor in comparison needs performance for its usability goal to hide users among users, but Infranet is hiding the use of Infranet itself so there is no need to hide users among others. If such a usability goal was relevant for Infranet it would fail anyway since Infranet responders are communicated out-of-band which poses a huge hurdle to start using Infranet. Since this paper only focuses on security concerns, the performance of Infranet is therefore not really relevant to us.

IV. Freenet

Freenet is an anonymous data storage and retrieval system. Users provide a part of their local hard drive as storage for the network and can anonymously publish and retrieve files that are identified through a unique key derived from a descriptive string via a hash function. To retrieve a file, a user must have knowledge of its descriptive text string and calculate the corresponding key. He then sends a request to a Freenet node he knows. Knowledge of such nodes is gained out-of-band. That node will check whether it has this specific file cached in its local storage and sends it back to the user if this is the case. If the file is not present in the local storage it will check which of its known nodes is most likely to have this file, based on its key. All nodes keep tables of the nodes they know and which files they are thought to hold. If a node does not know who has this specific file it will just choose a node of which it knows that it has files with a similar key. If that node fails to retrieve the file, the next node will be tried. If all other nodes also fail, a failure is passed upstream to the preceding node. If one of these nodes can in fact return the file, the node will keep a copy of the file itself and pass it to the preceding node. Information about the node that originally found the file in its local storage is passed along with the file. This information can be altered however to protect storage holders and publishers. To prevent circles, each request is given a pseudo-unique identification number. If a node receives a request with an ID it already knows it will instantly reply with a failure so the other node can query the next node in its list. Also, each request is given a hops-to-live limit to reduce the load on the network. Nodes can alter excessively large hops-to-live values.

A user who wants to publish a file in Freenet must calculate the file key using the chosen descriptive string. He then sends the file to his own node along with a hops-to-live counter. This will determine how many nodes will store the file initially. The first node then issues an insert request that behaves just like a retrieval request in terms of how the nodes decide which node to contact next. Each node on the path checks if a file with the same key already exists in its storage and if yes responds with this file. The user then knows that a collision has occured and has to choose another key. If the hops-to-live counter, which is decremented each hop, reaches zero, the last node will respond with an all-clear message, indicating that the file can be inserted. The file is then passed along this path and

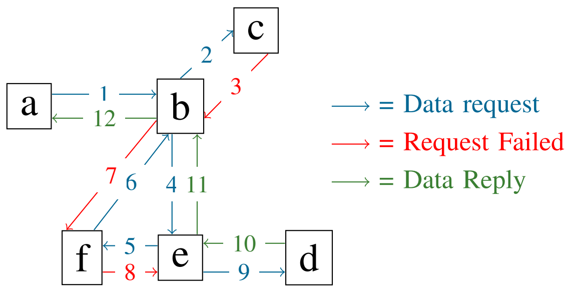

Fig. 3. Sample Freenet data request sequence, adapted from [5]. Request 7 fails because b already knows this request.

each node stores a copy of the file. They note the first node as the origin of the file. However, just like with usual requests, each node can alter this information and list itself or any other node as the origin in order to protect privacy of the author. This altered information is not only stored in the local storage but is also passed along for the next node that has to store the file. So no node on the path can tell for sure which node really is the origin.

To prevent collisions, users can generate private namespaces that are signed with an asymmetric key pair. After that they are the only one who can publish files in this namespace.

If a node receives a file, either through a request that it forwarded to another node or by a newly published file, it will save this file in its local storage. If the storage capacity is not big enough, the least recently used file will be deleted. This means that data-storage in Freenet is not permanent. Old and rarely used files might be completely deleted while files that are requested often can potentially remain in the network forever.

A. Design goals

Just as with Tor and Infranet, in the original paper describing Freenet [5] several goals have been set for the development. We will take a look at these goals and discuss their impact on Freenets security.

1) Anonymity for both producers and consumers of information: For producers of information it is critical that the file is widely spread in the network in order to conceil its origin. While inserting the file, every node on the path could consider its predecessor as the potential origin, or at least suspect it to know the real origin. If a substantial amount of nodes can be compromised and the potential origins of files by the same author are monitored regularly an attacker might be able to narrow down options of potential origins to just a few or only one node. However, for this to be effective, the adversary would need to have control of multiple nodes that are close to the publisher. Close in this context means only a few hops away. Since each node only communicates with those nodes it knows and trusts, an adversary would have to gain the trust of the publisher or the nodes he trusts. The further away the adversary gains trust, the smaller its chances of being able to monitor the publishing of new files. Also a publisher could just use different keys to sign his file, making it hard to detect files that are published by the same author. This is however not applicable if the author needs to sign his files in order to gain some kind of authenticity. All in all the risk of this attack being successfully performed is drastically reduced if nodes have a higher chance of altering origin information of files. This information is only needed to retrieve files if they are deleted from the local storage or to guess good routes, so one could argue that a trade-off between security and performance has to be made.

Consumers on the other hand also might have reasons to remain anonymous. The main measure that protects their privacy is that on the path through the Freenet each node only knows its predecessor and its successor. It is also not known whether a predecessor was the original requester of a file or if he just forwarded a request. A critical thing here is again how many compromised nodes an adversary can place close to the requester. If he is close enough then a lot of the users requests will pass through the compromised nodes, allowing possible timing attacks. But gaining trust as a compromised node is arguably very hard.

If a users traffic is being completely monitored by an adversary, he could gain knowledge about which file keys the user requests or publishes. Freenet tackles this with two measures: On the one hand all communication is encrypted with SSL, so it is more or less impossible to read the users Freenet requests and therefore gain knowledge about the files he requests or publishes. Even if this was possible, the attacker would not directly gain knowledge about the contents of those files since all files in Freenet are stored in an encrypted manner. Only those who know the descriptive text string (which they also need to search the file in the first place) can generate the key required to decrypt the file.

2) Deniability for storers of information: It is reasonable that nobody wants to store files with illegal material for other people on their hard drive. To allow storers to deny the knowledge of a files contents, all files in Freenet are encrypted. To decrypt it, the descriptive text string is required. The storer has the public hash of the files descriptive string, so it is theoretically possible to launch a dictionary attack on this string and gain knowledge of the descriptive string. But since a storer has no intent to decrypt the files stored on his machine as he would lose deniability, this does not matter.

3) Resistance to attempts by third parties to deny access to information: An adversary might try to deny access to certain files or the Freenet network as a whole. Since information about Freenet Nodes are passed out-of-band, it is not possible to know every Freenet node and block all traffic to it. Also Freenet traffic is encrypted via SSL and can therefore not be blocked as a whole easily without affecting wide parts of the internet.

A censor could also connect to Freenet and keep requesting a file that he wants removed. Each request will contain information about which node provided the file. That node could be shut down and the process can be repeated until no nodes with the file are left. The problem that occurs with this strategy is that every node on the path can alter the origin information and therefore it is highly likely that the wrong node will be shut down. Also, requesting a file will result in the file getting distributed even more, since every node on the path will save a copy of the file in its local storage. So in conclusion, this attack would probably lead to spreading the file instead of removing it, since blocking all nodes is arguably not possible.

4) Efficient dynamic storage and routing of information: With the same argument as we already used for Infranet, performance can not be seen as a security concern for Freenet. But the way Freenet increases its performance actually raises the security as a side effect. A key component that increases Freenets performance is the fact that retrieved files are distributed among all nodes on the path. This means that all nodes on this path will gain knowledge about the origin of this file, which in return means they can make a profound guess the next time a file with a similar key is requested. A route to such a file can therefore be found faster.

Another effect that emerges from this behaviour is that a lot of requests for similar keys will be directed to this node which is very likely to have them stored locally. If that is true, it can return them quickly. If not, it will forward the request and gain a copy of the file when the forwarded request returns. In the future, other clients are very likely to request the same file from this origin, since it has a key similar to those they know the node has. When the next client requests this file it will already be stored locally and can be returned quickly. This leads to nodes becoming specialized for a specific set of keys, since they will be queried for those keys pretty often and are therefore very unlikely to delete them. Paths that found this specified node will be used over and over again, without the nodes along the path having to try out different options that may lead to a lot of failures.

Newly published files with similar keys also tend to propagate towards already specialized nodes, because the inserting mechanism uses the same routing algorithms as the search. This increases the chance of new files being present at or close to a specialized node when they are requested.

This self-reinforcing effects do not only lead to an increased performance of Freenet, but also have a value for security and anonymity. If specialized nodes emerge, they will draw most of the requests for specific files to them, which in return reduces load on the nodes close to the node that originally published the file. This makes it harder to track where the files originated from. Of course this only has an effect if the specialized node is not the original publisher, but even if the publishing node turns out to become the specialized node it can plausibly deny that it is in fact the origin of the data since any node can possibly become specialized for certain file keys.

5) Decentralization of all network functions: In Freenet, as opposed to Tors network structure, there are no central instances that control the network in any way. Everything works on a peer to peer basis and there is no way to look up any routes or addresses. Every node just forwards packets it receives in directions where it sees the highest chance of fulfilling the request. With no central infrastructure there is also no single point of failure. But a fully decentralized architecture also poses some challenges and risks. The first obvious problem is that one has to gain knowledge of other Freenet nodes out-of-band via a trusted source. A user that does not personally know someone who runs a node has to contact others that do, but those people will probably not disclose who they are if they also do not know the new user. Otherwise an attacker could just pretend to be someone willing to setup a node, ask for other nodes and then use this knowledge to compromise these nodes or shut them down. Even if a user, that is not an attacker, asked for other nodes, an attacker would probably be able to see this request. If the sheer running of a Freenet node was illegal in that area, this might already be enough to get the user into trouble. Also the adversary could monitor the users traffic if they can identify him and maybe gain knowledge about other Freenet nodes. Another possibility for the attacker would be to tell the user about a Freenet node that is run by the attacker. The user would then have to trust this node in order to gain access to the Freenet network and the attacker could completely monitor and manipulate all his requests.

Publicly asking for other nodes is therefore not an option for someone who relies on anonymous access to Freenet. This means that users that do not have any trusted source of information about Freenet nodes can practically not join Freenet securely. In regions where running Freenet might be illegal, people will probably be very careful about who they tell about the fact that they are running a Freenet node. So even if one knows someone that does run a node, it is not very likely that this person will share information. This makes it even harder to join the Freenet network for those who need it the most.

Another problem that emerges from the decentralized structure of Freenet is that collisions can occur. If two persons insert new files with the same key nearly simultaneously, or a second file with the same key is inserted at a region where the first file can not be found on the initial insert path, both files will coexist within the network. This could be used by an attacker to replace a genuine file version or at least make it harder for requesters to get the correct version. Also it could be hard to tell which file is the correct version, depending on the contents of the file. This attack is only feasible against files that are signed with their descriptive text string, there also exist other methods of signing that are more secure. Users can create a personal namespace and add their files to it, using a private key only they have. The attacker would then have to gain knowledge of the key or forge a hash collision. Both of these scenarios are highly unlikely and probably exceed the capabilities of a limited adversary.

V. Conclusion

As we have seen there are vastly different problems to solve in varying situations and various solutions have come up to solve them. There clearly is no single perfect solution to all

	Tor	Infranet	Freenet
hide requesters	✓	✓	✓
hide providers	✓	✓	✓
hide communication	✗	✓	✗
hide origin	✗*	✗*	✓
verify data	✗*	✗	✓

✓= fulfilled, ✗= not fulfilled
✗* = not fulfilled, but solved elsewhere

Fig. 4. Evaluation of fulfillment of criteria by different projects

problems, because trade-offs have to be made between certain criteria.

In our introduction we formulated five key criteria that are most important to the majority of anonymizing software users. We have discussed different aspects of Tor, Infranet and Freenet and can now evaluate these three projects under these criteria.

Tor does an especially good job in hiding who is communicating with whom, therefore the first two criteria, hiding who is requesting information and who is providing it, can be considered fulfilled. However, detecting that someone is using the Tor network is pretty easy. Therefore Tor does not fulfill the second criterion which is hiding that there is a communication going on. For normal Tor communication, hiding the origin of data is beyond of what Tor tries to achieve and it does not care about this problem, therefore not fulfilling this criterion. On the other hand there are the Tor location-hidden services, which do such a good job in hiding the origin of their data that even the NSA can not trace them. So we can argue that Tor does in fact hide the origin of data where it is necessary and does not care about this information for other data. The last criterion we formulated is that it should be able to verify that transmitted data has not been altered. Since Tor uses SSL Encryption on all its layers, altering data within a Tor connection should be practically impossible without it being noticed. However if the protocol that is tunneled through Tor is not encrypted or otherwise protected a manipulation of data on the path outside the Tor network is indeed possible. Again, this criterion is not within Tors goals and Tor actually does not try to solve it, but it is doing a good job at those parts of the connection where it can influence the outcome.

Infranets main goal is clearly to hide that someone is requesting information. If requests are not detected at all then it is also not possible to obtain knowledge about who is requesting data. There are a lot of recommendations by the developers how to run a responder so that traffic is disguised as normal web browsing. If one follows these recommendations it is pretty unlikely that a communication is detected and therefore the first and third criterion are well fulfilled. Hiding who is providing information is also solved, since Infranets design actually relies on the fact that Infranet responders are hidden and knowledge of these can only be obtained through out-of-band means. Hiding the origin of data however is not within Infranets goals, since it tries to allow access to publicly available material that has been blocked or restricted. A censor already knows that this material exists and where it originated from, otherwise he would not be able to block it in the first place and the use of Infranet would not be necessary. So the fourth criterion is not really a matter to Infranet. Verifying that data transmitted via Infranet has not been altered is basically not possible. Every Infranet responder is in fact a proxy that can act as a man-in-the-middle. Even if one tried to obtain hashes or signatures for resources requested via Infranet the responder could manipulate those. A verification can only be done via out-of-band means, so the fifth criterion is clearly not fulfilled.

The last software we discussed is Freenet. Requesting and publishing files in Freenet is a bit like Tors architecture, where every node only knows its predecessor and successor. So no node can know for sure who is requesting data and who is providing it. Also, every user runs its own node, so there is nothing like an entry-node that could be compromised. All communication is encrypted with SSL, so even a local eavesdropper can not gain knowledge about the contents of ongoing communication. Requesters of data are therefore hidden pretty well. The same argument applies for providers (in Freenets case the storagers of files), even though it might be possible to trace parts of a path back through origin data that has been saved by nodes along the path. But this information may be changed randomly, so there is never a way to tell for sure where data came from. Since Freenet is peer-to-peer based there will always be various nodes involved in a request that will detect ongoing communication. These nodes can possibly be set up by an attacker and therefore the third criterion can not be considered fulfilled by Freenet. Freenets strength clearly lies within the fourth criterion which is hiding the origin of data. Everyone can anonymously publish files within the network, which are pretty much impossible to be traced back to their origin. Also, since all files are signed, it is very easy to detect whether they have been altered. We discussed a possible attack on this by inserting files that use the same key, but by using personal namespaces this attack is rendered impractical.

Overall we can come to the conclusion that there is not a single perfect solution to all privacy related problems. Which software to use depends heavily on the personal situation, the abilities of ones adversary and the data that a user is trying to access or publish. Tor is a recommendation in most use-cases, but it depends on using other anonymizing software like the Tor Browser on top of it and the usage of Tor can easily be blocked. Infranet on the other hand reaches its goals very well, but relies on web server owners setting up secret Infranet responders that have to remain secret to adversaries. Knowledge about them has to be passed out-of-band, which might be hard in some situations. If one manages to acquire knowledge about an Infranet responder, communication can be considered pretty much secure. Freenet is doing a really good job in hiding the origin of data, but it also has the problem with getting into it through out-of-band means. It is also only applicable to data that are within the Freenet network and cannot be used to access usual websites like it is possible with Tor and Infranet. But if one wanted to publish or spread information anonymously, Freenet is a good choice.

References

[1] R. Dingledine, N. Mathewson, and P. Syverson, “Tor: The second-generation onion router,” Naval Research Lab Washington DC, Tech. Rep., 2004.

[2] R. Broenink, “Using browser properties for fingerprinting purposes,” in *16th biennial Twente Student Conference on IT, Enschede, Holanda*, 2012.

[3] “Tor stinks,” https://edwardsnowden.com/docs/doc/tor-stinks-presentation.pdf, accessed: 2017-11-16.

[4] N. Feamster, M. Balazinska, G. Harfst, H. Balakrishnan, and D. R. Karger, “Infranet: Circumventing web censorship and surveillance.” in *USENIX Security Symposium*, 2002, pp. 247–262.

[5] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong, “Freenet: A distributed anonymous information storage and retrieval system,” in *Designing privacy enhancing technologies*. Springer, 2001, pp. 46–66.