Censorship-resistant Collaboration with a Hybrid DTN/P2P Network

Masterarbeit

von

Philipp Hagemeister

aus

Braunschweig

vorgelegt am

Lehrstuhl für Rechnernetze und Kommunikationssysteme

Prof. Dr. Martin Mauve

Heinrich-Heine-Universität Düsseldorf

März 2012

Acknowledgments

My thanks go to Marc Fontaine for asking stupid questions that turned out to be quite clever, and for pointing out that correctness is essential both in the real and the physical world.

I also thank Paul Baade for demanding impossible features which turned out to be the last piece in the puzzle.

Julius Römmler has notified me of orthographical, typographical, and (inadvertently) semantical errors. And told me to use fewer big words. Thanks!

I wish to thank Denis Lütke-Wiesmann for proofreading the thesis, and the footnotes.

Sven Hager found lots of overly short, overly long, and overly wrong statements. Thanks!

Thanks to Prof. Martin Mauve for coming up with the idea, shielding us from bureaucracy, asking for explanation and rationale at every step, and finding all the errors nobody else found.

Contents

List of Figures

Chapter 1

Motivation

Development of policies such as laws, political manifestos, examination regulations, articles, source code, and any other form of speech[1] can be greatly enhanced by computer supported cooperative work systems.

Unfortunately, speech – especially if political – faces attempts to censor or suppress it all over the world. The 2011 ”Freedom in the World” report of Freedomhouse[Pud11] rates 47 countries (with one third of the world population) as ”Not Free”. In these countries, people are denied basic civil liberties such as political participation. Similarly, the Amnesty International Report 2011[FIS11] mentions serious restrictions on freedom of speech and political participation in 48 countries (about 40% of the world's population).

Unsurprisingly, efforts have been made to censor computer-supported speech alongside more traditional censorship methods. Freedomhouse's Freedom On The Net Report 2011 [KCU11] rates 11 countries (with one quarter of the world population) as ”Not Free”, indicating that experts reported significant restrictions on access to and providers of controversial information. The OpenNet Initiative, which automatically measures availability of ”provocative” and ”objectionable” resources instead of relying on human expertise, confirms these assessments by finding significant censorship of these resources in 13 countries (with one quarter of the world population), and any censorship in 42 (60% of the population) in recent measurements[oni11].

Technology should resist censorship and allow free speech whenever possible. In fact, one could argue that free speech is needed the most in the face of censorship. Enabling censorship-resistant free speech has its downsides: The same technology that can be used to debate about democracy or draft an appeal for human rights can be used to foster racism or create a terrorist manifesto[2]. While

[1] In this thesis, speech is used in the legal sense, as an umbrella term for any distribution and development of potentially objectionable content.

[2] Although as scientists, we hope formal analyzers can find and point out fallacies in extremist thought.

these usage scenarios cannot be prevented without centralized control, I assume that the benefits of unfettered speech outweigh their downsides.

The goal of this thesis is to develop a framework which allows collaboration in the face of governmental censorship, and implement a prototype.

1.1 Distribution of Speech

Current systems for delivering speech include traditional media (e.g. television, newspapers) as well as internet-based services. Traditional media requires significant infrastructure and easily controllable delivery channels (relatively large parts of the radio spectrum and significant transportation vehicles/personnel, respectively) and is therefore owned or tightly controlled in non-democratic countries. In democratic countries, the huge barriers to entry can facilitate a concentration of media ownership, which, while not governmental, may impede or warp democratic consensus[Bak07].

In contrast, internet-based services such as facebook, twitter, and systems explicitly built for policy drafting (for example adhocracy[adh], echo[ech], or liquid feedback[liq]), pose lower barriers to entry and are therefore harder to control. While being able to reach every German via newspaper distribution or a TV channel is practically impossible for everyone but governmental or large commercial entities, virtually everyone can reach 2×10^9 internet users by setting up a website, a blog, or a twitter account.

1.2 Threat Model

Unfortunately, the desire for unfettered and accessible distribution of free speech is not shared by everyone. Therefore, various attackers may strive to impede internet-based services using a specific *protocol*. In this thesis, I assume an attacker whose goal is to disrupt or modify speech.

1.2.1 Nontechnical Attacks

The attack does not have to be technological in nature. For example, limited availability of technology, or the chilling effects of having to write under a real name (which have been effected in China[Lin11] recently) can suffice the goal of repressing free speech. Therefore, one goal of the software proposed

in this thesis is to allow contributors to stay **anonymous** or **pseudonymous**. The problem of availability of computers and associated tools and services is not addressed, although we can hope for the OLPC project[olp] and the power of Moore's law and free software, which should allow everyone to participate in a global computer-/internetbased discussion eventually. If a user's hardware is seized and she is accused of possessing illegal content, our software should encrypt its data and offer **plausible deniability**, so that she could plausibly claim to never have used the software, used it only for legal purposes, or used it for less serious legal infractions of local law.

1.2.2 Internet Access

Since our attacker is a governmental entity, it controls all centralized internet access mechanisms, in particular the internet service providers(ISPs). Naturally, the easiest way to circumvent censorship would be a decentralized non-censoring ISP.

It is unlikely that such an ISP could rely on landlines or other permanent, visible installations. While satellite and long-range terrestrial radio communications are inherently harder to control, neither approach has yielded decentralized communication networks so far. Due to the high cost of designing, launching and maintaining satellites, satellite internet tends to be expensive and low-bandwidth (and naturally high-latency). At the moment, (terrestrial) amateur radio requires expensive equipment and training, and is typically relegated to unfavorable low-bandwidth frequency bands.

On the other hand, IEEE 802.11 networks are cheap and widely deployed, but limited by their short range. While it is possible to construct an 802.11-based mesh network[AW09] on a country-wide scale, such a network has not been implemented yet. Mid-range GSM/CDMA2000/UMTS/LTE networks are widely deployed and available at little cost (and are reported to have been used to evade censorship in North Korea[Mac05]). However, operating such a network requires a relatively large radio spectrum allocation as well as expensive equipment. Therefore, the number of national networks tends to be fairly low even in technologically advanced countries – in Germany, there are just four GSM/UMTS/LTE operators.

Summarizingly, while it is possible that technological advances allow for a long-range high-bandwidth radio network with cheap small terminals (which would be virtually impossible to control, as evidenced by the reports of cell phones being available in North Korea, arguably the tightest-controlled nation)[3], no such solution exists yet. Therefore, we need to assume that the attacker can read, suppress, and modify the communication between users at different locations.

[3] At least from the perspective of computer scientists who are used to human ingenuity being the limiting factor in technological development, and do not care much about physical limitations like the Shannon–Hartley[Bel68] theorem.

1.2.3 Control over the User's Computer

Any discussion of control over communication links is moot if the attacker manages to control the user's computer. One avenue to that goal lies in limiting the (general-purpose) hardware to run only approved programs. Barring complicated hardware modification, a security exploit or the import of unrestricted hardware, this approach allows the attacker to forbid any censorship-busting software. Worryingly, such restrictions have already been considered and implemented in hardware one would normally consider to be general-purpose:

- UEFI (the upcoming BIOS replacement) includes a "Secure Boot" feature[uef11, chapter 27] which requires the Operating System code to be cryptographically signed. Since active "Secure Boot" will be required by Microsoft for the Windows 8 logo program[Mic11], a significant portion of desktop and notebook computers will be unable to run arbitrary code in the near future without explicit configuration.

- Apple iPad, iPhone, and iPod devices only run signed code. Furthermore, applications must also be signed. [Zov11]

- Various Android devices require the OS to be signed to boot as well.

However, in all of these instances, the restriction is commercial in nature; the certificate authority is then usually intent on preventing malicious – but sometimes also controversial – content, and not free speech. Therefore, this thesis assumes that the user can run arbitrary software on her devices.

Another way to gain control over the device instead of the communications link is government-sponsered malware (or any other malware intending to disrupt free speech), such as the German Staatstrojaner[Clu11]. Again, this thesis just assumes that the local device is not running malicious software.

1.2.4 Total Shutoff

Given these restrictions, the obvious course for the attacker is to turn off civilian Internet access, for example by mandating the ISPs to do so or by turning off major Internet exchange points. Fortunately, Internet access is vital for commercial activities as well as entertainment. Shutting it off is therefore a last resort, and likely to incite further uprising rather than quell it. This attack has been executed by the former Egyptian and Libyan governments[Cow11a][Cow11b] shortly before their respective displacements. In both cases, virtually all IP prefixes were withdrawn from the global BGP routes[DSA+11]. In North Korea, there is no generally available internet access, although the aforementioned limited

availability of technological devices in general may play a role in that situation. We conclude that **censorship-resistant software should be able to use alternative communication channels in the event of an internet outage**. As a positive side effect, this should improve the usefulness of the software in cases where the cause for internet outage is not an attacker, but a natural event or the lacking internet access in remote regions of the world.

If the attacker decides to not only shut down communications, but also electricity networks, a practical implementation must also consider alternate power sources. However, power outages can be bridged by batteries, solar cells, and chemical/kinetic power generators. Furthermore, shutting down electricity networks has even more drastic consequences on commerce and entertainment than turning off Internet access. Therefore, attacks on power networks are ignored in this thesis.

1.2.5 Physical Attacks

The next avenue of attack we have to consider is physically turning off some of the computers running the policy drafting software[4], notably those providing a centralized service. Fortunately, a number of well-connected countries are open to free speech (with small limitations). At the cost of additional latency, hosting critically imported systems in free countries can therefore prevent this attack. However, the shutoff of the German Pirate Party's Piratenpad service by the German police[Alt11] – which was motivated by some users posting illegal content on it – shows that physical attacks are possible even in countries which allow free speech. Load balancing solutions, which redirect traffic to backup servers may be used to alleviate the effects of physical attacks as well as outages caused by accidents or natural events.

1.2.6 IP Blocking

If physical attacks are not an option, a total shutoff of Internet access is not desired, and none of the other attacks mentioned above are possible and/or feasible, the attacker can still omit *some* packets. Internet routers can easily be configured not to forward some IP datagrams, or anounce bogus routes via BGP, or withdrawal of BGP routes. For example, Pakistan Telecom announced a bogus route to youtube's IP prefix via BGP[NCC08] in an effort to block it. In a controversy over videos critical of the king of Thailand, the country's ISPs blocked youtube in 2007[Ful07]. In 2011, Egypt and Lybia censored twitter and youtube by blocking traffic to their respective IP ranges[DSA+11]. IP-based blocking can be defated or hindered by using a large number of IPs with different prefixes, using DNS entries with low timeouts (*fast flux*)[HGRF08].

[4] or any other software that allows to transmit speech

1.2.7 DNS censorship

However, since the DNS resolver is usually set automatically via PPPOE or DHCP, and usually pointed to a server operated by the ISP, it is trivial for ISPs to block a DNS name by simply configuring their DNS server to answer the censored queries with a wrong reply (sometimes pointing to a server serving a censorship notice message via HTTP), NXDOMAIN reply (falsely indicating that the domain in question does not exist), or no reply at all. Many ISPs already configure their DNS servers to return false results for queries of nonexistant domains, which are then resolved to a server which serves advertisements for all HTTP requests[WKP].

False answers can be prevented by employing *DNSSec*[AAL+05], which provides cryptographic signatures in all DNS answers and methods for verifying them. However, DNSSec cannot prevent the DNS server from not answering at all. Furthermore, DNSSec is only deployed by a minority of domains as of 2012, and DNS stub resolvers used in most computers are not yet validating the responses anyways. Finally, simply not answering or answering with an invalid signature also fulfills the goal of an attacker, namely preventing the user from contacting the censored service by not providing the service's IP address.

Another alternative, commonly employed by malware[Ten09], is to (pseudo-)randomly generate, or distribute a list of a large number of domains. This approach requires the client to try to resolve (and verify, for example with DNSSec) all domains in the list until it finds one that has not been censored yet, and can therefore not be used with a generic client such as a web browser. DNS censorship is widely performed all over the world[opeb][AA08], and has even been considered in Germany[zug11] and the United States[Smi11][Lea11a].

Since DNS is a simple request/response protocol[5], it is also possible to intercept DNS requests to *all* DNS servers, including those that serve uncensored responses. Chinese censorship systems have been shown to implement this technique as early as 2002[Con02]. This attack prevents the client from simply querying an uncensored name server, but can still be defeated (or at least hindered) by using a large number of domain names.

1.2.8 Deep Packet Inspection

While all the previous attacks work fine, they do not enable the attacker to censor only parts of an internet service – the attacker can either employ one of the attacks to completely shut down the service, or let all traffic pass. Because of the aforementioned widespread use of the Internet, it is not feasible

[5] In most cases, a DNS request/response consists of a single UDP packet.

to totally block a popular service such as a search engine, wiki, or forum (and any other service that makes use of user-generated content) just to censor a few objectionable entries[6]. Instead, the attacker wants to just censor content that meets certain criteria (for example containing a word from a list, or being registered in a database). Such a censorship system requires three components:

- A gathering mechanism which understands the protocol (up to the transport, or even the application layer), and separates binary data and protocol headers from the transmitted content body. Such as mechanism is called *Deep Packet Inspection*, or *DPI* for short.

- A detection mechanism which decides whether the gathered content should be censored.

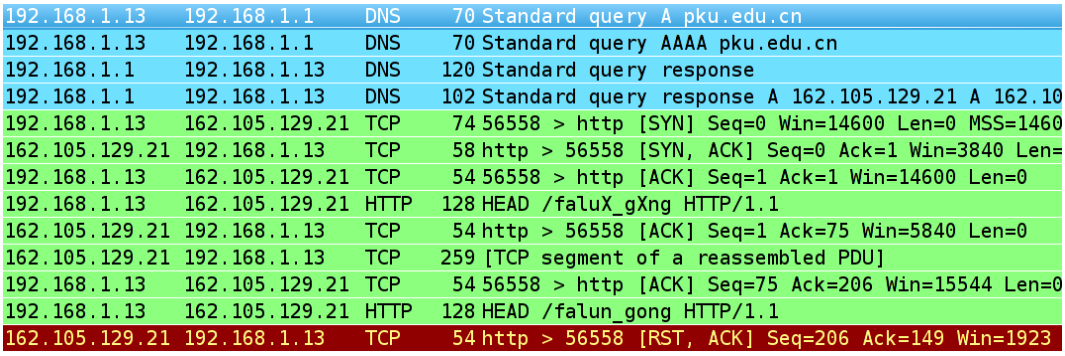
- A denial mechanism which performs the actual censorship, typically by configuring a temporary firewall rule that blocks all packets between the communicating hosts, or those that seem to belong to the same *flow* (the sequence of packets the objectionable content was gathered from, for example a TCP connection). Alternatively, the denial mechanism can inject a malicious packet. For example, a TCP connection can be shut down by injecting a packet with a set `RST` flag, which indicates an immediate abnormal connection termination.

Aside from using a secret protocol with custom encoding of content – which would provide *security by obscurity*, and therefore be futile unless public discussion of the protocol and applications using it is somehow prevented[7] – the only way to evade a censorship system that supports DPI is to **encrypt content**. As a stopgap measure, *obfuscation* – chosing a complex encoding, for instance by sending a random symmetric cryptographic key in and then encrypting all further communication with it – works as well. However, this approach relies on the attacker not having enough computing power to undo the obfuscation for all packets. In light of Moore's law and other likely advances in computing, obfuscation cannot be a permanent solution.

Today, DPI censorship is commonly available and deployed. DPI censorship is a central component of the "Great Firewall of China"[XMH11]. As shown in figure 1.1, requesting a HTTP resource containing the term `falun_gong` (*Falun Gong* is a religious movement banned in China) triggers the censorship mechanism.

[6] For instance, the youtube block in Thailand[Ful07] was motivated by just 20 videos, and not the youtube platform in general[Ros08].

[7] Which would be quite ironic for a protocol that strives to evade censorship



192.168.1.13	192.168.1.1	DNS	70	Standard query A pku.edu.cn
192.168.1.13	192.168.1.1	DNS	70	Standard query AAAA pku.edu.cn
192.168.1.1	192.168.1.13	DNS	120	Standard query response
192.168.1.1	192.168.1.13	DNS	102	Standard query response A 162.105.129.21 A 162.10
192.168.1.13	162.105.129.21	TCP	74	56558 > http [SYN] Seq=0 Win=14600 Len=0 MSS=1460
162.105.129.21	192.168.1.13	TCP	58	http > 56558 [SYN, ACK] Seq=0 Ack=1 Win=3840 Len=
192.168.1.13	162.105.129.21	TCP	54	56558 > http [ACK] Seq=1 Ack=1 Win=14600 Len=0
192.168.1.13	162.105.129.21	HTTP	128	HEAD /faluX_gXng HTTP/1.1
162.105.129.21	192.168.1.13	TCP	54	http > 56558 [ACK] Seq=1 Ack=75 Win=5840 Len=0
162.105.129.21	192.168.1.13	TCP	259	[TCP segment of a reassembled PDU]
192.168.1.13	162.105.129.21	TCP	54	56558 > http [ACK] Seq=75 Ack=206 Win=15544 Len=0
192.168.1.13	162.105.129.21	HTTP	128	HEAD /falun_gong HTTP/1.1
162.105.129.21	192.168.1.13	TCP	54	http > 56558 [RST, ACK] Seq=206 Ack=149 Win=1923

Figure 1.1: Screenshot of a packet dump of two HTTP requests to pku.edu.cn, the latter of which is terminated by an injected TCP RST packet (in red)

Since the attacker wants to block our protocol, but has to let other commonly used protocols pass, she may also want to detect specific *protocols* with DPI. For example, Iran blocked the Tor anonymization network[8][ira11]. Tor tries to emulate an HTTPS handshake, but did use SSL certificates with shorter expiration times than regular SSL certificates. This discrepancy was used to detect Tor connections, and block them.

We conclude that the designed protocol must be **indistinguishable from commonly used protocols** such as HTTPS.

The attacker can also block all encrypted protocols including SSL/TLS-based ones such as HTTPS. However, this has massive side effects on commerce and entertainment, as no banking site and many popular websites will stop working. Iran did block all SSL/TLS connections temporarily in 2012[ira12]. However, the block was quickly circumvented by tunnelling the encrypted connections through plain-text HTTP, as Tor's obfsproxy[JA12] demonstrated in the Iranian case. In the end, this becomes a cat-and-mouse game which prevents virtually all connections from regular users, and henceforth becomes indistinguishable from a total shutoff.

1.2.9 Active Attacks

Correctly implemented encryption makes gathering of transmitted content impossible. Naturally, the simplest attack against encryption would be outlawing any encryption or blocking encrypted text. Fortunately, neither attack is feasible: Unencrypted communication can be intercepted easily by anyone, and would make attacks *by third-party attackers* (for instance pranksters or regular criminals) trivial. Additionally, *steganographic* techniques can be used to hide encrypted content in seemingly

[8] For more information on Tor, refer to chapter 2.4.3

innocuous information, for example by transmitting information over the least significant bits of the color values in an image.

Therefore, the attacker must either break the encryption or use an application-level attack (see below). While it is possible that the attacker can find flaws in the algorithm or its implementation, commonly used encryption schemes have to bear scrutiny from security professionals all over the world, and are therefore unlikely to be vulnerable (in a way that can be exploited by the attacker) in the first place. But even if they are vulnerable, both algorithms and implementations can be exchanged relatively quickly, if not by software updates then by modifying encryption preferences so that another algorithm is used.

A significantly easier task for the attacker is to exploit a flaw in the trust model (see chapter 2.3) and perform a *man-in-the-middle attack* by presenting *his* key to both parties, and then freely relay, see, censor, and modify the communication between them. In 2011, this attack was implemented in Iran, where gmail users were presented a valid SSL certificate signed by the compromised diginotar certificate authority[Adk11][Lea11b]. Unlike all previous attacks, man-in-the-middle attacks are *active*, i.e. require the attacker to send additional packets. Unlike passive attacks such as blocking packets to certain IPs, active attacks are detectable and distinguishable from mere network failures.

A resourceful attacker may even participate in the network he attempts to censor, and generate valid messages to find IP addresses to block. In 2011, the ”Great Firewall of China” was found to be identifying and subsequently blocking Tor bridges (i.e. publicly offering entry points in the otherwise blocked Tor network) by connecting to them and sending Tor-specific commands to distinguish them from regular HTTPS hosts[Wil12].

Lastly, the usefulness of an application can be diminished even when its communication works. If the attacker cannot prevent an application from working, she can still try to imitate one or more users and post useless messages, or encourage commercial spammers to do so.

1.2.10 Conclusions

The modeled attacker tries to inhibit certain services using certain protocols. He has governmental privileges, and may perform non-technical censorship unless the service provides anonymity or pseudonymity. The attacker can trivially block IP addresses, DNS names, protocols, and objectionable keywords.

However, it is generally in the attacker's interest to not interfere with other, non-objectionable services[9]. Therefore, a well-designed protocol that tries to allow free speech should imitate other widely used protocols.

1.3 Decentralization

As laid out in the previous chapter, the attacker's capabilities allow him to disrupt centralized services (such as regular web applications) with ease. Unfortunately, *all the services mentioned in chapter 1.1 are centralized*. We are therefore in need of a decentralized system for speech distribution. While this system should evade any censorship of communication, it also **must work when traditional communication networks are unavailable**, as is the case when the attacker performs a total shutoff ($\rightarrow$ 1.2.4) attack.

Decentralization has desireable side effects as well: Any outage caused by an intentional attack could also be caused by accidental misconfiguration, accidents, or natural disasters. No matter the nature of the outage, a decentralized system will be more robust than a centralized one. In particular, the ability to use the system off-line or on connections with very large delays increases the usability in cases where such a condition is present naturally[10].

As a further advantage, a free[11] decentralized system allows basically everyone to run an instance, and does not require any trust in a central operator. In contrast, there is no significant incentive to allow simple deployments by others in a traditional centralized system, and the operator must be trusted to not to keep detailed logs of plain-text passwords, real names or voting behavior.

1.4 Collaboration

While uncensored communication is great, effective *collaboration* requires more than simply being able to communicate with others. In particular, the development of policies of any nature (be it manifestos, laws, or source code) can be greatly enhanced by **version control** which allows synchronization, tracking and merging of changes as well as branches of alternatives. In a distributed system with potentially hostile participants, it is critical to be able to deny and organize changes to the collaboratively drafted policy.

[9] In plain text: Don't mess with kitten pictures.
[10] for example in space; see chapter 2.2 for details
[11] as in free speech

1.5 Structure of this Thesis

This chapter has explained the basic premise of this thesis, namely how to allow uncensorable distribution of speech. The remainder of the thesis explains solutions to the problem spaced posed here. It is structured as follows:

In chapter 2, we examine the basic building blocks necessary to construct the desired system.

The system's structure is then mapped out in chapter 3.

Finally, chapter 4 describes the implementation of a simple prototype for the designed system.

Chapter 2

Components

Before designing and building a censorship-resistant speech delivery system, we need to discuss its basic building blocks, namely decentralized systems. We examine two existing classes of decentralized systems:

- Peer-to-peer networks (chapter 2.1) allow computers on a network such as the Internet to provide a distributed service.

- Delay tolerant networks (chapter 2.2) are used in cases where traditional communication networks are not available and are required to address the attacks described in chapter 1.2.4.

Additionally, a fully realized system (especially services such as voting) will require a security model (chapter 2.3).

Afterwards, we discuss existing anonymization networks (chapter 2.4) and ways to realize a peer-to-peer system on top of them, or make use of their built-in peer-to-peer functionalities.

For each of these components, we discuss both existing work as well as extensions thereof to address potential attacks on and other design considerations of the component.

Finally, cooperative work($\rightarrow$ 1.4) needs to be tracked and managed with revision control systems (chapter 2.5).

The challenges in embedding each component into the designed system are summarized in the respective *Integration Notes* subchapters.

2.1 Peer-To-Peer Networks

In a client/server system, there is a significant asymmetry between the nodes; clients only contact servers. This fosters a relatively small number of servers and is therefore prone to censorship. For instance, while almost every person in a developed country uses at least one HTTP client, only a fraction operate HTTP servers. The vast majority must rely on third parties to publish their content.

In contrast, all nodes in a peer-to-peer (**P2P**) network can and do talk to each other. Therefore, P2P networks tend to be significantly more resilient; shutting down a single node or a centralized service (such as DNS) does not kill the network. Additionally, open P2P networks do not only have more nodes, but also more node *operators*; most servers are operated by just one organization, whereas the nodes in P2P networks are regularly controlled by thousands of people.

Virtually all P2P networks run on the Internet; advanced P2P networks typically have network-wide addressing and routing schemes which are *overlaid* over IP. P2P networks can be classified as unstructured and structured.

In an **unstructured** network like Gnutella[Rip01], nodes connect to each other at random, and only use some heuristics (for example the number of current connections) to select their partners. Therefore, unstructured networks are prone to falling apart into two or more independent partitions – sets of nodes that are connected to each other, but cannot reach a substantial portion of the network.

Structured networks like Chord[SMK+01] and Kademlia[MM02] assign each node a (typically random) network address. The node then uses a network-specific algorithm to determine which nodes to connect to – typically many in its proximity (by address), and some nodes far away. Structured networks allow efficient **Distributed Hash Tables** (**DHT**s) by assigning each node the address space in its proximity. If any node wants to store or look up a value in the DHT, it calculates the hash value (→ 2.3) of the key, finds the nodes which handle that address, and advises them to store, or asks them for the values associated with that key. Structured networks are designed to avoid breaking up into multiple partitions. However, the order of the network also makes it easier to knock out for an active attacker, who can introduce a large number of nodes in order to be assigned the authority over a large portion of the address space.

2.1.1 Bootstrapping

To join a P2P network, a new node must somehow connect to any node already in the network. This process is called *bootstrapping*. Once the new node finds a gateway node to the network, it can find additional nodes over the P2P overlay network.

Bootstrapping is a critical step in the face of a censoring attacker, since it often has to rely on centralized services if scanning or multicast are not options. On the other hand, *registration* of P2P node addresses at the bootstrap provider is not a problem as the bootstrap provider can simply join the P2P network himself, and discover nodes and/or accept incoming registrations.

Every bootstrap entry consists of the application protocol, the (IP) address as well as the (TCP/UDP) port. The following bootstrapping options are possible and feasible:

Static Addresses

The simplest form to find addresses of other peers is to store the addresses in the code or include them in software updates. Since empirical studies have shown that the past availability is a good predictor for uptime in the future[SR06], long-lived nodes (for example those explicitly maintained by organizations in data centers) are good candidates for inclusion in the software distribution.

Alternatively and additionally, the program can store its last peer list on a permanent medium before exiting, to speed up and ensure the start from the second execution time on.

The user can also retrieve initial addresses from a secondary channel (e.g. a phone call or text message) and manually input them into the program.

HTTP(S)

HTTP and HTTPS traffic is extremely common, up to the point that the protocols are used synonymously with *internet* in the popular conception. Therefore, it is unlikely to be blocked completely. On the other hand, the pervasiveness also means that virtually any censorship system can block specific HTTP hosts, requests, and answers. HTTPS can be used to avoid these, but requires a domain name. While the HTTP URL can specify an IP address instead of a domain name to avoid reliance on DNS, these addresses are likely to be blocked by Layer 3 censorship.

DNS

DNS is probably the only protocol which is available and unblocked in more networks than HTTP is. However, if the user does not manually configure a DNS server of her choosing, it is also the simplest protocol to censor. DNSSec can be used to validate the answers (for a detailed discussion,

see chapter 1.2.7). In order to include not only IP addresses, but also protocol type and port numbers in the answers, the DNS response must be encoded into multiple `TXT`, `AAAA` or `A` records.

Existing Infrastructure

Existing centralized, but unblocked infrastructure can be used to disseminate bootstrap information. Any web service that allows (if possible unregistered) content to be uploaded works fine, be it forums, social networks, pastebins, filehosting services, online office suites, or webmail.

Other public services such as IRC also qualify for bootstrapping. IRC, in particular, is widely used by botnets for initiation of communication.[BY07] Non-web email is also sufficiently widespread to serve as a bootstrap method. For example, Tor provides an email interface finding its bootstrap-equivalent bridge nodes.[torc] On cell phones, SMS text messages or even data transmission via phone calls are also an option.

Other P2P Networks

Our system should also be able to piggyback onto existing P2P networks, in particular anonymization networks. If these networks have better bootstrapping methods, or are specifically unblocked for some reason, our network should use them to contact a central bootstrapping server (for example over HTTPS) or retrieve the information stored by the P2P network.

The decentralized currency system bitcoin[Nak09] provides an interesting alternative. Bitcoin's design mandates that every transaction must be carried on by all bitcoin nodes forever (and transactions carry a de-facto minimum fee of 0.0005 bitcoins, approximately 0.0020 Euro at the time of writing). Since the parameters of a transaction can contain freely chosen bytes, it is possible to store bootstrap information in a decentralized system that will never delete it. Nevertheless, the currently lacking spread and cost make this at most a theoretical proposition.

Multicast

IP multicast and protocols that build on it (like mdns[Che11]) are prime candidates for bootstrapping and do not require any centralized infrastructure. IP multicast or scanning (see below) are the only bootstrap options that work in a local (or campus-area) network without Internet connection. Unfortunately, IP multicast is generally **not** available on the public Internet.

Scanning

In some cases, it is feasible just to scan the entire network, or even Internet. If 1000 nodes are randomly placed over the global IPv4 address space, and the node scans 1000 addresses per second, it can expect to find a node in just over an hour. Intelligent choosing of addresses to scan can significantly reduce that number[DG08], as can exponentially growing network speeds.

Scanning only works if the density of peers to scanned addresses is high. Therefore, it is not possible in the global IPv6 internet with its $3 * 10^{38}$ addresses.

Decoy Routing

Decoy routing provides an virtually uncensorable way to communicate with the outside world. However, it requires an ISP that supports it. The application picks *any* IP address routed through the ISP and sends a cryptographic *sentinel* bytestring. Once the ISP detects that specific bytestring, it handles out encryption and redirects the traffic to an uncensored proxy server.[WWGH11]

Currently, there is only one experimental implementation of decoy routing, *Telex*. Decoy routing also requires the cooperation of an ISP that is willing to hijack certain connections to his customers, which is notoriously difficult to attain.

2.1.2 NAT Traversal

Although IP has been designed for end-to-end connectivity, connection requests to many machines are blocked by **firewalls** and **Network Address Translation** (NAT). Iff two peers are both blocked, it is necessary to trick both firewalls/NATs into assuming that their respective node is initiating the connection. This is achieved by coordinating both peers to connect to each other with the help of an unblocked arbitration node. Since the additional node is needed, and the process can fail in the presence of certain NATs, nodes behind firewalls/NATs should not be included in the set of bootstrap nodes.

Fortunately, the NAT traversal approach **STUN** has been standardized in RFC 5389[RMMW08], and should be sufficient to traverse the most common NATs and firewalls.

2.1.3 Broadcasting

In some cases, for instance when new data becomes available, the P2P network should allow any note to notify all other interested nodes in the network. Unstructured P2P networks only offer a crude broadcast mechanism – since peers do not know much about the structure of their surrounding network, they have to re-broadcast the message to all their peers. In contrast, there are multiple different proposals[EAABH03][PWC03][VYF06] for structured broadcast networks.

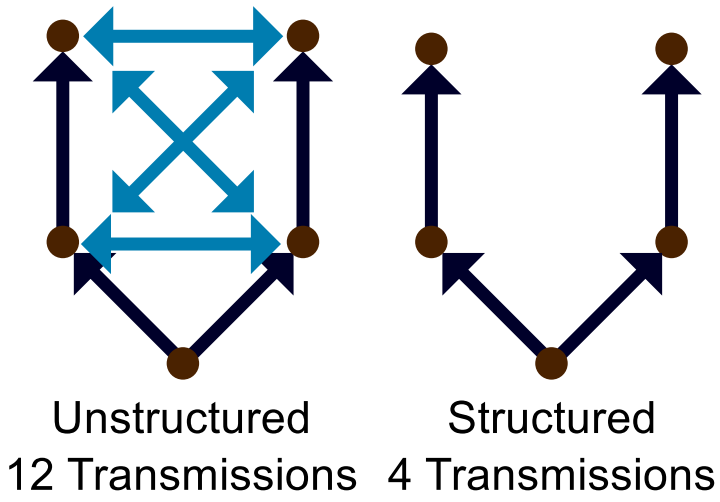


Figure 2.1: Crude broadcasting in an unstructured vs optimal broadcasting in a structured P2P network

These broadcast algorithms construct a connected digraph with a small average vertex degree. Some algorithms take unforeseen node failures into account and include redundant transmission routes in order to bridge failures before or immediately after they occur.

Of course, there are some limitations of broadcast messages both on unstructured as well as structured networks[1]. First of all, nodes will generally try to remember incoming broadcasts for a short time, and ignore broadcasts they have already seen.

Additionally, a **Time-To-Live** (**TTL**) is a field in the broadcast message that gets set by the sender to the maximum number of hops he wants the message to go, and is decremented by each peer. If the TTL reaches 0, the message is ignored. The counterpart of the TTL is the **Hop Limit**. It starts at 0 and gets *incremented* by each peer the message traverses. If the Hop Count reaches an implementation-defined value, the message is ignored as well. While the TTL allows the original sender to continue the maximum spread of the network, the hop count allows the *network* to do the same. Using both values ensures maximum flexibility and security.

[1] In a structured network, these limitations are not necessary if the broadcast algorithm does not send redundant messages, and if the network is stale

2.1.4 Integration Notes

P2P networks are essential to allow every user to disseminate information without having to rely on third parties. They are inherently censorship-resistant due to the large number of nodes and the implementation of virtual overlay routing and addressing schemes.

There are a number of bootstrapping methods, nearly all of which can be censored somehow. Nevertheless, the sheer richness and numerous variations of bootstrapping schemes allow us to evade all but the most sophisticated censorship systems.

Due to the high number of nodes behind a NAT or restrictive firewall, NAT traversal (typically with STUN) is essential for the finished system if we want it to run anywhere. Similarly, the broadcasting algorithm should be fine-tuned so that messages reach most peers even in the event of (accidental or intentional) failure of a node, but not waste bandwidth with superfluous transmissions.

2.2 Delay-Tolerant Networks

Many current network protocols require timely interaction between the communicating nodes. For example, TCP's three-way handshake means that any communication over TCP will take at least three times the (unidirectional) delay between the nodes. This is not an issue in a local network where the delay is less than a millisecond, but becomes apparent in global connections: Since the delay is bound by the speed of light in current communication technologies, and may be physically bound so, the round-trip delay between Düsseldorf and Mountain View is at least $2\frac{9000km}{3*10^8 m/s} = 6*10^{-2}s = 60ms$. While this effect can be minimized in some circumstances by physically positioning nodes in close proximity, **delay-tolerant network (DTN)** applications are explicitly designed to avoid "chatty" communication in the firstplace, and therefore do work even in high-delay environments.

In particular, the delay in **interplanetary commmunication** is on the scale of minutes; while a 60ms round-trip delay between Düsseldorf and Mountain View is tolerable, the 10 min delay between Düsseldorf and Mars makes chatty protocols insufferable.[BHT+03] DTNs can also help to provide Internet access to rural regions without network connectivity. Small computers attached to busses or donkeys that visit remote villages can allow people in these remote regions to interact with modern internet-based services with a delay of days or weeks.[SKZ+06] Communications in mobile ad-hoc, naval, and sensor networks may be delayed not because of distance, but due to temporary outages.[OKD06][RSB+08][WWT08]

Another avenue of communication are human-carried[2] storage devices. Since modern storage devices such as microSD cards, SD cards, thumb drives, and cell phones are easy to smuggle even over restricted borders, DTNs can enable "pocket-switched" networks[HCS+05]. Because of the low availability, high costs, and censorship of internet connections, crude "manual" versions of such DTNs are currently used in Cuba[Fer12].

RFC 4838[CBH+07] lays out the basic architecture and challenges in a DTN. RFC 5050[SB07] and RFC 5325 [BRF08] define two Layer 3/4 protocols for DTN applications. DTN research is ongoing, particularily in the following areas:

- **Routing** profits not only from accurate custody and replication specifications, but also accurate models to predict the movement of human or satellite nodes [LDS03][HCY08][LZC10][pro11].
- **Security** is challenging because the access to central authorities is limited. Additionally, the nature of DTNs necessitates additional privacy and confidentiality considerations.[KZH07][FRB08][SFWL11]
- **Simulation** applications allow the validation of routing techniques and tests of applications.[KOK09]

[2] or pigeon-carried, as a more efficient usage of ornithological resources than RFC 1149[Wai90]

- **Applications** that run on DTNs need to be developed.[OK06][HKL+07][WH11] This thesis develops one.

2.2.1 Integration Notes

In this thesis, we assume that the DTN is only one-hop, and that its mediums are mass storage devices. Additionally, our application creates complete replications of the data in a project in order to be certain that *any* remote node – no matter what its current known information is – gets all the information of the project. The primary challenge lies in designing the protocol and application so that they still work. These assumptions are justified by the limited real-world application of our system, which consists primarily in allowing communications via smuggled miniature mass storage devices.

Notably, multi-hop networks can still be implemented just by copying the information from one device to another. Similarly, because we do not place any restrictions on delay in the first place, the information stored on a thumb drive can be transferred over a ”real” multi-hop DTN.

In future work, we not only expect to extend the application to support multi-hop networks, but also existing DTN protcols such as Bundle and Licklider. Conversely, the architecture allows for multiple DTN implementations, each of which provides its own implementation of the common *transport* interface (→ 3.3).

2.3 Security

Although a collaboration system strives to facilitate the flow of information, there are reasons why not all users should be given full access to all information. In particular, a collaboration system should allow commercial or personal discussions to remain private. Additionally, even public collaboration should be restricted from unlimited write access; most importantly to deny attackers' attempts to vandalize it.

Fortunately, there are cryptographic primitives we can use:

- **Cryptographic hash functions** map any input to a number of fixed size, in a way that makes it infeasible to perform the reverse mapping. Moreover, given any output of a cryptographic hash function (a *hash sum*, or just a *hash*), it is infeasible to find *any* input to the hash function which yields the same hash (a so-called *collision*). Of course, collisions are unavoidable when mapping an infinite input set to a fixed output set according to the pigeonhole principle. However, cryptographic hash functions are designed so that collisions cannot be found except by exhaustive search, which is believed to be impossible with current hardware for an output size. For instance, a 256 bit output means that more than 2^{128} hashes have to be calculated in order to find a collision with probability $\frac{1}{2}$, even when accounting for the birthday paradox. Therefore, the rest of this thesis justifiably assumes that collisions are impossible.

- **Symmetric encryption** schemes can prevent anyone who does not have the shared key to read the message's contents. By adding a cryptographic hash to the message (a so called Message Authentication Code (*MAC*)), it can also be assured that the message has not been modified by anyone except those having the key.

- With an **asymmetric encryption** scheme, encryption and decryption uses a pair of keys instead of a single key: Everyone can encrypt content with the *public key*, but only those privy to the *private key* can decrypt it. Just like symmetric encryption, asymmetric encryption can be proofed against tampering with a MAC. Since asymmetric encryption tends to be significantly slower than symmetric encryption, we may want to asymmetrically encrypt a key which is then used in conjunction with a symmetric algorithm.

- **Digital signatures** can be used to sign a content with a private key, and can be tested with the corresponding public key. To speed up the signature generation and verification, a cryptographic hash of the content is often signed instead of the content itself.

As discussed in chapter 1.2.3, we assume that the user's computer is not compromised, and therefore neither is the private key. The problem of key distribution, however, remains – we need to ensure the

key is actually generated by the user we expect it to be, and not by an attacker.

2.3.1 Trust Models

To establish trust in a key, most trust schemes pick a number of *trust anchors* – keys they already trust – and trust only those keys that provide a cryptographic signature of the identity and the public key (a *certificate*) by one or multiple trust anchors.

Trust models are necessary because it is infeasible to exchange large numbers of public keys. For example, it would be impossible to verify the identidy of an arbitrary person, as doing so would mean the maintenance of 7×10^8 identities and associated keys. Instead, it is only necessary to check that an identity card has been issued by one of the 2×10^2 countries.

TLS/SSL / Multiple root certificates

TLS[DR08] and its predecessor SSL are widely used on today's internet in almost all applications. In SSL's trust model, each application has a set of trust anchors, the so-called *root certificates*, all of which are trusted. These root certificates are usually selected by the application or operating system vendor. The certificate for a given domain name is then signed with the root certificate, or another intermediate certificate, which is itself signed by the root certificate.

The fundamental vulnerability of this system is the large number of root certificates. An attacker who can compromise *any* root certificate can sign other certificates for *all* domains.

Unsurprisingly, the track record of SSL security is not very good. In 2011, hackers managed to to compromise diginotar, one of the trust anchors preconfigured in common web browsers and operating systems. The thusly obtained certificates were used to intercept secured communication (a so-called *man-in-the-middle attack*, because the attacker relays the traffic between two systems, and presents himself as the remote peer to both)[Adk11]. Also, in some cases root certificates could be simply bought[geo][tru].

Due to its simplicity and wide availability, SSL can still be used for some tasks such as bootstrapping. However, due to the cost of obtaining a certificate, the need to have a domain name, and the questionable trust model, we will not consider the SSL trust model for collaborative applications.

Centralized Trust Anchor and Recursive Subdivision

While a single root of trust seems to be a significant flaw on first sight, it also means that it is sufficient to secure one trust anchor instead of multiple ones. This model is used for identity cards: A country (trust anchor) hands out certificates to its citizens (entities).

In the technical realm, DNSSec[AAL+05] is used to secure DNS information. It has only a single trust anchor for the root domain. The keys for the top level domains (like `.com.`, `.de.`) are signed by this root trust anchor, which in turn sign the public key associated with a domain. Since the top level domain does not matter except for public relation purposes, this model allows us to pick a trusted top level domain outside of the realm of the attacker. Therefore, DNSSec provides an adequate defense against our attacker. Unfortunately, using DNSSec requires all users to possess a domain name. Once widely deployed, DNSSec seems to be an excellent alternative to SSL when it comes to securing central servers.

Web of Trust and Sybil Attacks

Instead of relying on certificates by a central authority, PGP's trust model[AR97] relies on the trust the user generates by signing other user's certificates: If Alice trusts Bob and Carol, (or, more precisely, has verified their keys), and Bob and Carol trust Dave, Alice also trusts Dave. Users can also indicate their level of trust; if Bob and Carol place only minimal trust into Dave, Alice may require additional certificates in order to trust Dave.

The web of trust model closely matches an intuitive understanding of trust, and is very flexible. It must be guarded against *Sybil attacks*, where the attacker pretends to be multiple personalities that all trust each other. Notably, centralized systems can be seen as a special case of the web of trust; one in which all users trust the central authority, and the central authority trusts certain users.

Since it is the only decentralized model, a web of trust seems to be by far the best system to implement in a decentralized collaboration software.

External Trust

Instead of only including certificates, a user may also assign trust based on other criteria. For example, *proof of work*[3] schemes can be used to impede Sybil attacks; the attacker must then amass

[3] a cryptographic puzzle that requires a certain amount of computing power to solve, but can be easily verified

significant amounts of computing power (and possibly outdo other attackers) in order to present a significant number of identities. For instance, the cryptocurrency Bitcoin[Nak09] generates trust in the log of transactions by requiring them and the last proof-of-work problem to be included as inputs to a proof of work problem. While an attacker could try to present a different history of transactions, he would have to solve the computational problems faster than the rest of the network, and faster than all competing attackers.

Similarily, in some applications it might be useful to prove that an action has been authorized by a human. Various forms of Turing tests are widely used to prevent attackers from automatically creating multiple accounts, for example.

Additionally, it might be useful to consult certificate metadata; a certificate that has been seen multiple times over years is more trustworthy than a new one. Also, comparing the certificate with certificates acquire from other locations prevents attackers in one locality (for example the user's ISP) to forge certificates. convergence.io[con] is an experimental framework that allows these kinds of checks in a web browser.

Unlimited Trust

In some cases, it is feasible to trust everyone, including attackers. For example, the source code of open-source software can be studied by attackers when the security of open-source software does not depend on the obscurity of the implemented algorithms. Most Wikipedia entries can be edited by everyone.

Unlimited trust works not only when attackers lack sophistication and perseverance, but also when it is easy to undo malicious changes after the fact. In the distributed collaboration system laid out in this thesis, a version control system ($\rightarrow$ 2.5) allows exactly that.

2.3.2 Integration Notes

In the bootstrapping process, it is useful to rely on conventional centralized trust models. The widely-used SSL has been shown to be susceptible to attacks, but can be supplanted or replaced by DNSSec or convergence. To ensure additional confidentiality between P2P nodes (in addition to the application-level encryption described in chapter 3.2.8), we can use opportunistic asymmetric encryption[Lan09] as well as steganographic[4] techniques such as obfsproxy[JA12] in order to avoid detection of the

[4] Steganography is the science of hiding encrypted content among unobjectionable normal content.

protocol in the first place. For DTN media, a complete system should implement plausibly deniable[5] symmetric and asymmetric encryption of the stored data as well as steganographic techniques.

The integration of the cryptographic primitives and trust models into the overall architecture is discussed in detail in chapter 3.2 – this section only discusses the underlying technology for the trust model.

For the user's keys, the most important question is where to store them. For a web application, there are the following options:

- The simplest way is to store the keys on a **server**, and perform all cryptographic operations with them on the server. The server still needs to authenticate the user, for example with a password, client-side SSL certificate, or a keycard such as the eID function[Küg10] in modern German identification cards. Using a password or client-side SSL certificate means that no hardware or software but a capable web browser is required for the encryption; since all the encryption happens on the server, the user is only concerned with authentication, and not with encryption and signatures. However, this approach means that the resulting system is essentially centralized because the user's keys are only available on a single server, or a tightly administrated group of servers.

- Instead of generating the key and storing it somewhere, we can also recalculate it at runtime from the user's identity (email address, real name, or similar) and password. Iff the password is sufficiently random, this method allows a user-friendly way to store the key; in the user's head. However, the client- or server-side code that performs the regeneration of the key must still be trusted. If the attacker manages to inject code once, he immediately knows the private key.

- The only solutions that do not require the user to trust the server he uses are **native client-side** encryption, signature generation and signature verification systems outside of the control of client-side JavaScript code. To increase usability, it is prudent to allow client-side code to ask for either operation, or at least allow the user to sign arbitrary text in an input field with a single click instead of having to copy and paste the text to and from the encryption application. Currently, this requires the installation of plugins. Experimental plugins that embed OpenPGP into a web browser exist[Gol12], and the eSign function of the German identity card and similar hardware solutions allow native client-side security at the cost of the need for specialized hardware and software.

[5] Plausibly deniable encryption allows the user to store encrypted content among *other* "alternative" encrypted content in such a way that it cannot be proven whether the stored data is random or encrypted information.

Voting

If the key is also intended for voting or it its owner should be granted special access, it is also necessary to be able to not only verify the *user* from the system's perspective, but also the real-world **identity**. While web-of-trust systems can be used to establish an identity, their vulnerability to Sybil attacks regularly requires personal verification[6] by a centralized authority. Alternatively, the authority can also rely on another trusted authentication system such as eID, eSign or extended OpenID[Gol12].

A totally different approach would be to not restrict users, but the nodes in the network. If the network only consists of trustworthy nodes that the attacker cannot gain control of, and registration at these nodes is centrally managed, secret voting is possible, and user credentials can be shared between servers. However, the secrecy and trust in the outcome of the vote only hold as long as all servers remain trustworthy. As described in chapter 1.2.5 and incorporated in the design, we explicitly assume that the attacker is able to compromise parts of the network.

Nevertheless, centralized voting (or distributed voting facilitated by a centralized trusted host) is the only option if votes should remain secret. Fully distributed secret voting is impossible.[BS05]

[6] or the use of external verification methods such as PostIdent

2.4 Anonymization Networks

An anonymization network is a system that prevents attackers from associating messages with their senders and/or receivers, even if the attacker controls the links between local sender and the anonymization network as well as between anonymization network and the remote receiver[7], and sometimes even if the attacker controls some of the nodes that form the anonymization network. Figure 2.2 shows this setup.

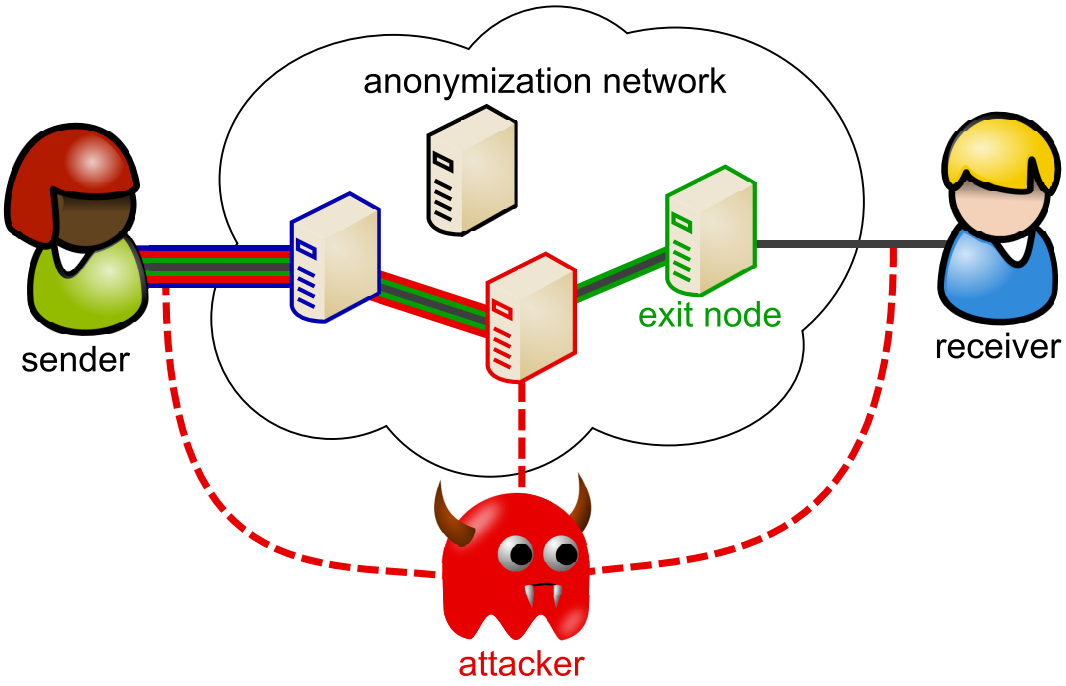


Figure 2.2: Model of an anonymization network. The attacker can intercept an encrypted version of the traffic between sender and first node, encrypted traffic that goes to one of the nodes in the network as well as plain traffic between exit node and receiver.

In the context of this thesis, the anonymization network anonymizes *machines*. Of course, that is not necessarily sufficient to protect the identities of *people*: If the user transmits confidential/identifying data in the clear, an attacker listening on the unencrypted link between anonymization network and receiver can read it just fine. In 2007, Dan Egerstad publicly demonstrated this attack by setting up an exit node and then intercepting passwords of embassy workers that were sent in plain. Additionally, although a service being contacted cannot correlate multiple messages by the user from the message headers, it may be able to do so based on the messages' contents. For instance, a web application can advise the user's web browser to store *cookies*, which the web browser then includes in all further

[7] Assuming the receiver is not part of the anonymization network. In these cases, we call the anonymization network node that establishes the actual connection an *exit node*.

requests unless configured not to do so. Furthermore, the specific setup (for example installed fonts and plugins) of a web browser may allow surprisingly specific identification[Eck10].

Encryption is a necessary condition to prevent correlating traffic into and out of the anonymization network. Also, the anonymization network is typically set up to an unfiltered internet connection in order to avoid local censorship. Therefore, anonymization networks face very similar attackers as speech distribution software, most of the attack model and defenses against individual attacks described in 1.2 apply for anonymization networks as well. In turn, dedicated anonymization networks can be used to avoid most of the considered attacks, with the exception of total internet shutoff.

Anonymization networks can anonymize traffic at various protocol levels:

An anonymization network could work like a VPN[8], and be implemented with virtual **data link (L2)** or **network(L3) layer** network interfaces[9]. However, doing so has quite a few disadvantages:

- Every time a new interface is set up, there needs to be lengthy communication about the precise setup. For example, an L2 anonymization network would typically necessitate a DHCP exchange before it can be used. An L3 anonymization network would require a similar initialization phase. To avoid the initialization phase, a static configuration method has to be implemented.

- Since IPv4 addresses are fairly limited, most computers have just one public IPv4 address assigned to them. Therefore, exit nodes would be required to perform Network Address Translation, and therefore hinder P2P applications ($\rightarrow$ 2.1.2).

- Due to different codebases, the actual implementation of standardized protocols such as TCP or IP varies across operating systems. These differences can be detected and thereby allow associating of a communication channel with the operating system the user uses[Tal04].

- Chatty protocols designed for local networks (such multicast DNS or NetBIOS) would endanger anonymization and needlessly waste limited traffic, and would therefore need to be filtered on the virtual network interface without compromising the functionality of legitimate applications.

- Bidirectional channels would be required.

- Applications may not expect the local IP address to constantly change.

[8] Virtual Private Network

[9] To reduce confusion and enhance readability, we will refer to the network layers of the OSI model as L1-L7 in this discussion[Nor].

- Since adding virtual network interfaces (rightfully) requires elevated privileges on most operating systems, an anonymization software would need to run as an elevated user, and be adapted to every operating system.

Anonymization networks can provide **transport layer (L4)** service, for example by offering a SOCKS[LGL+96] proxy. This does away with all of the issues with L2 or L3 service except the need for bidirectional communication channels and problems for P2P application. However, it requires the application to either support the specific anonymization network, or support generic (e.g. SOCKS) proxies. Fortunately, virtually all web browsers support SOCKS proxies.

To allow not only the sender of a message (or the *client* in client-server applications), but also the receiver (the *server*) to remain anonymous – and therefore allow P2P networks without elaborate NAT traversal tricks – the application needs to be aware of an anonymity-preserving addressing scheme. Since these **session layer (L5)** anonymization networks are all but required for constructing an anonymous P2P network, we discuss them in detail in chapter 2.4.2.

In theory, an anonymization network could also serve as an **application layer (L7) proxy**. This allows the anonymization network to strip potentially identifying information such as application version numbers from the exchanged messages. However, since this approach prevents encrypted connections between sender and receiver, it is not suitable for an anonymization network with potentially untrusted members. Instead, it can be used *in conjunction* with a real anonymization network. Since a censorship-resistant collaboration software will strive to not include anonymity-defeating information such as detailed version numbers or history in the first place, L7 anonymization networks are not useful for our purpose.

Instead of anonymizing network communication, anonymization networks can also offer **high-level services** such as data storage. Since real-time collaboration requires low-latency communication, and version control implies a potentially large number of stored resources (which may need to be regularly refreshed in the distributed file system). Therefore, high-level anonymization networks are left out of the following discussion.

2.4.1 Mix networks

To understand the challenges of P2P networks built on top of anonymization networks, it is necessary to understand the basic architecture of modern anonymization networks. Such an anonymization network consists of a set of core nodes, the so-called *mixes*[Cha81]. In some network designs, each node serves as a mix, whereas in others, the set of mixes is restricted to centrally-approved or even

-maintained nodes. In any case, each mix relays traffic to other mixes. If the network provides an L2, L3 or L4 service, a subset of mixes also serves as exit nodes.

It is vital that all packets sent through an anonymization network look alike (otherwise, an attacker could correlate series of lengths or other characteristics of packets). Therefore, anonymization networks usually enforce a fixed length for all packets by padding them if necessary. Additionally, mixes may want to randomly delay some messages to prevent the attacker from simply correlating the incoming messages to the outgoing ones by temporal order.

To ensure that an attacker who gains control of a mix cannot undo anonymity, the user picks a series of mixes as his *tunnel*. This pick is random, but constrained both by technical criteria (for example, the final node must be an exit node, and high-bandwidth and low-latency nodes may be preferred) as well as (partially conflicting) privacy criteria (the mixes should be geographically diverse and operated by different entities, and chosen from a large set). The user can vary the number of nodes to balance speed and privacy. A series of just one mix does not offer privacy if that mix happens to be controlled by an attacker, whereas a series of all available mixes will offer maximum privacy at the cost of extremely high latency.

To prevent an evil mix in the series of picked mixes to compromise privacy, the user encrypts the traffic multiple times with keys only known to the respective mixes, in reverse order. If the user Alice picks the mixes A, B, C with the public keys K_A, K_B, K_C (and private keys K_A^+, K_B^+, K_C^+) as well as an encryption scheme E, this process goes as follows in a circuit-switched network:

1. Alice creates a local tunnel to A by sending $E(K_A, \text{"}Create\ Tunnel\text{"})$.

2. A creates the tunnel to Alice.

3. Alice extends the tunnel by sending $E(K_A, E(K_B, \text{"}Create\ Tunnel\text{"}))$.

4. A decrypts the message and calculates $E^{-1}(K_A^+, E(K_A, E(K_B, \text{"}Create\ Tunnel\text{"})))$
   $= E(K_B, \text{"}Create\ Tunnel\text{"})$, and sends this message to B.

5. B decrypts the message and creates a new tunnel, connected to A.

6. Alice extends the tunnel once again by sending $E(K_A, E(K_B, E(K_C, \text{"}Create\ Tunnel\text{"})))$.

7. A decrypts the message and calculates $E^{-1}(K_A^+, E(K_A, E(K_B, E(K_C, \text{"}Create\ Tunnel\text{"}))))$
   $= E(K_B, E(K_C, \text{"}Create\ Tunnel\text{"}))$, and sends this message to B.

8. B decrypts the message and calculates $E^{-1}(K_B^+, E(K_B, E(K_C, \text{"}Create\ Tunnel\text{"}))) = E(K_C, \text{"}Create\ Tunnel\text{"})$, and sends this message to C.

9. C decrypts the message and creates a new tunnel, connected to B.

10. From now on, Alice can send data/setup connections by sending $E(K_A, E(K_B, E(K_C, data)))$ to A.

11. A decrypts the message and sends $E(K_B, E(K_C, data))$ to B.

12. B decrypts the message and sends $E(K_C, data)$ to C.

13. C decrypts the message and handles the data. If C serves as an exit node to the general internet, the data may actually be instructions on what host to connect to, or what data to send to which host.

Since the decryption process resembles peeling the layers of an onion, this concept is also known as *Onion Routing*[RSG98]. In practice, it can be sped up by arranging symmetric keys during the tunnel creation phase, and using fast symmetric instead of slow asymmetric encryption from then on.

In a packet-switched anonymization network, the tunnel setup process is not applicable. Instead, each layer contains an identifier of the next mix, and the sent data contains an (encrypted) return path; the data being sent to A is then $E(K_A, \text{"}A\text{"} + E(K_B, \text{"}B\text{"} + E(K_C, E(K_{Receiver}, \text{"}Sender\text{"} + data))))$. This requires the final receiver to be aware of the protocol (and the sender of the message), but also opens new possibilities to conceal messages: With *Garlic Routing*, the mixes A, B, and C are free to join other packets to the same next mix in a single message.

Like any other distributed system, decentralized anonymization networks require some kind of bootstrapping to get the directory of all available mixes (which includes their public keys). Since a blocked bootstrap prevents the system from working at all, and a bootstrap message sent by the attacker can be used to mislead the user to use only mixes under the attacker's control, both censorship evasion and trust model are vital for the bootstrapping process and the whole system.

2.4.2 Hidden services

In the preceding descriptions, mix networks have shown to be able to ensure privacy for senders of messages. However, we also want the *receivers* to stay anonymous – especially since every peer in a P2P network is a potential sender and receiver.

Mix networks can be extended to enable privacy of receivers by letting some mixes serve as *introduction points*. If Bob wants to set up an anonymous service, he picks out a series of mixes ending with an introduction point mix, and sets up the route as usual by iteratively extending it. However, instead of sending data to the final mix, he instructs it to offer an introduction point to a specified name. Bob is free to choose as many introduction point mixes as he wants.

To prevent an attacker from registering any name, the name is typically a representation of a public half of a asymmetric keypair Bob generates, and the introduction mix requires a digital signature by that key to set up the service. Bob then publishes the public key and the introduction point's name at some kind of database. From now on, Bob is reachable by the public key.

If Alice wants to connect to him, she first acquires the public key (for example over the conventional bootstrapping mechanism in a P2P network, or by communicating with a peer Bob has communicated with). Then, he looks up all active introduction points for the key, and establishes a tunnel to one of them. He then simply sends the data he wants to send to Bob over that route. Conveniently, Alice already has Bob's public key, and can encrypt all traffic to make sure she isn't communicating with an attacker who just impersonates Bob.

2.4.3 Common implementations

Fortunately, anonymization networks are no merely theoretical concepts; many independent implementations are freely available, and are subject to research on specific details which are ignored here, such as resistance against advanced correlation attacks and congestion avoidance protocols. This chapter describes some of the most popular and most examined services with special respect to applicability for P2P collaboration applications.

One-Hop Anonymization

The simplest form of anonymization networks is an L2/L3 VPN. Routing all traffic through a virtual network interface to a remote destination allows local censorship evasion without having to modify the application. However, since IPv4 addresses are scarce, VPNs are often combined with NAT, leading to a situation where NAT traversal techniques ($\rightarrow$ 2.1.2) must be implemented in order to get a working P2P network. However, this solution requires the use to acquire a trusted server in a censorship-free country. While there is a multitude of such commercial providers, the required technical knowledge and the high cost of setting up a VPN is likely to discourage many users. Additionally, VPNs are – unlike dedicated anonymization networks – usually not designed to evade censorship.

Like VPNs, simple SOCKS and HTTP proxies are (commercially as well as free of charge) available, but require dedicated application set up and share all other disadvantages of VPNs.

If VPN and SOCKS protocols are blocked via DPI censorship (1.2.8), the connection can still be tunneled through HTTP[opea] or DNS[NNR09]. However, both methods again require a specific trusted server in a censorship-free country as well as technical proficiency, and are therefore unlikely to be adapted by large numbers of users.

Tor

Tor[DMS04](The Onion Router) is probably the anonymization network with the most users and most research focus. The Tor network is widely deployed; it consists of 3780 publicly listed mixes at time of writing and constantly transfers about 1 GiB/s.[torb] Even more impressively, Tor has been actively blocked by Iranian[ira11] and Chinese[Wil12][tora] firewalls. Tor supports hidden services as well as anonymization of TCP/IPv4 connections over an L4 SOCKS proxy.

Tor is a circuit-switched network, and only provides bidirectional connections with reliable transmission and congestion control[10]. To increase performance, Tor uses asymmetric cryptography only in the initialization stages to arrange keys for (faster) symmetric cryptography.

Tor's trust model and bootstrapping is rather simple; central *directory servers* list all mixes and their public keys, and are simply queried via HTTPS. To prevent IP blocking of mixes, the Tor project also maintains a list of *bridge relays* – initial mixes which are not published in the Tor directory servers. Addresses of bridges can be requested via HTTPS or email. To prevent listing all available bridges, each IP and email address (limited to large email providers) is always shown the same three bridges.[torc]

Tor offers hidden services as described above. Introduction points for a given service name are registered and queried from the regular directory servers. Tor uses a further level of indirection to allow for symmetric cryptography on the final connection between the sender Alice and the receiver Bob: Alice picks a random secret and a *rendezvous point* mix of her choosing, sends the secret (encrypted with Bob's public key) to Bob via the introduction point, and then both Alice and Bob connect to the rendezvous point and use Alice's secret to establish a symmetrically encrypted connection.

Tor is written in C, but does not include a library for applications that wish to offer hidden services. Instead, hidden services can be statically configured in Tor's configuration files. Tor then proxies HTTP traffic to the hidden service to a (typically local) TCP/IPv4 service.

[10] i.e. TCP-like connections

I2P

In many aspects, I2P[int] is the opposite of Tor. It has no native exit node/IP tunneling functionality, and supports only hidden services. Although it can be used as a stand-alone application, it is primarily a Java library. Furthermore, I2P is a fully distributed P2P network where every node functions as a mix. At the time of writing, I2P consists of at least 12000 mixes.[i2pa]

I2P is a packet-switched network; every tunnel is unidirectional. This allows bundling of multiple packets from a mix to another (*garlic routing*, in the style of onion routing) to further complicate correlation of packets to a mix to those sent to the next mix. I2P comes with the reliable connection-oriented protocol *SSU*[i2pb].

The I2P network is based on a DHT; every mix picks a random value and occupies that position in the DHT. A global *network database* of mixes as well as hidden services is stored and synchronized by a subset of nodes, in a manner so that every node knows at least one of the notes holding the network database. To hamper Sybil attacks ($\rightarrow$ 2.3.1), each node must solve computationally intensive proof-of-work puzzles to get into the network. I2P employs a simple HTTP bootstrap mechanism. To avoid censorship, it randomly selects URLs out of a list of bootstrap URLs.

I2P can be integrated into an application as a Java library, but also offers two protocols (SAM and BOB) that applications can implement to communicate with a running I2P instance.

Other implementations

Phantom[Bra11] is a highly experimental anonymization network. Like I2P, it is decentralized. However, it uses a block of unassigned IPv6 addresses as network addresses. Therefore, all IPv6-capable applications immediately work with Phantom without the need for modifications.

Freenet[CSWH01] and **GNUNet**[BGH+02] are examples of data-storage networks. Both FreeNet (written in Java) and GNUNet (written in C) can be used by other applications to store data. Unfortunately, this requires all members of a collaboration suite to regularly poll the data-storage network for new information.

2.4.4 Integration Notes

Many of the problems posed in the threat model($\rightarrow$ 1.2) – in particular anonymity and DNS/IP/DPI censorship – are already solved by anonymization networks. Therefore, we can safely build our col-

laboration software on top of existing anonymization networks. Almost all anonymization networks offer interfaces for applications that build on top of them. Hidden services, identified by a public key, allow anonymous P2P networks on top of anonymization networks.

However, while mix networks are extraordinarily safe, the additional bandwidth and latency costs as well as the additional overhead by encryption and anti-correlation measures severely limit the available bandwidth and complicate debugging. Widespread anonymization networks may also be hampered by targeted efforts of attackers to censor them. Therefore, anonymization networks should be an option, but not the only possible way of communication in a P2P collaboration network.

Also, every considered anonymization network requires internet access. Aside from experimental network designs that are intended to run in parallel to the internet, such as cjdns[cjd] or mesh networks like BATMAN[JNA08], collaboration over DTN is required in the case of total censorship.

2.5 Revision Control

A revision control system manages a set of files over time (a *repository*). A single state of the tracked files is called a *revision*[11]. Each revision is identified by a globally unique identifier.

A *commit* establishes a new revision. Commits can be annotated with metadata such as author name, timestamp, cryptographic signatures, and a comment.

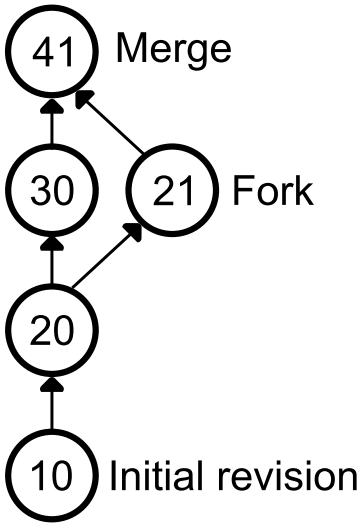


Figure 2.3: Terms in a Revision Graph. Note that the revision identifiers are for illustrative purposes only.

Usually, revisions are annotated with their predeceeding revisions, forming a **DAG**[12]. The first revision in a repository has no predecessors. Most revisions have exactly one predecessor. When Bob commits a new revision (`21` in the example DAG in figure 2.5, based on one of Alice's revisions (`20`), he creates a *fork* or *branch* - a different line of causal changes. Later on, when Alice gets Bob's changes, she creates a new revision(`41`) that unifies hers and Bob's changes, a *merge*.

Typically, two succeeding revisions build onto each other. The difference between them (the *change*, *delta* or *patch* of the commit) tends to be small relative to the size of all files. The revision control system should be able to not only reproduce deltas, but arbitrary differences(**diff**s) between any two revisions.

[11] In the context of revision control, the term *version* is often used as a synonym of *tag* – a human-readable annotation of a revision. Since tags are irrelevant in the context of this paper, and *version* is easily confused with *revision*, we will not use the former term.

[12] directed acyclic graph

2.5.1 Centralized Revision Control

The simplest revision control systems are centralized. In such a system, a central server establishes an order of revisions by serializing all change requests, allowing consecutive revision numbers - every node in the revision DAG has at most one predecessors and one sucessor; the whole DAG is linear. Each user uses her client to pull changes from the server and push changes to it. Users can create branches by copying all tracked files into an alternative directory. The inverse operation, a merge of the branch into the main tree, tries to integrate both versions of the merged fileset by merging their content, if possible with respect to the common ancestor. Examples of centralized version control systems include subversion(svn)[sub], CVS and SCCS.

Centralized revision control systems do not allow offline commits. Also, the central server is a single point of failure and can easily be blocked by network operators. Compromising the server tends to have disastrous effects on data integrity, since historic information is usually not stored on the clients. Centralized revision control is naturally unfit for P2P systems.

The logical adaption of centralized systems in a P2P network is a dynamically chosen primary copy (or group of primary copies). It is not applicable in DTNs since even if the high delays would allow reaching an agreement over which node should fulfill the role of the central server, the number of nodes is potentially unbounded.

2.5.2 Graph-based Distributed Revision Control

In contrast to a centralized system, every peer in a distributed revision control system such as Bazaar[baz], git[git], and mercurial[mer][13] maintains a local repository he can commit to. Synchronization between peers is done manually by fetching the contents of the remote repository, and integrating them into the local one.[14] In general, the revision tree in these systems is a DAG, where the root nodes represent branches.

If two users commit new changes independently, there will be multiple branches with the same name. For example, figure 2.5.2 shows a situation where another user based the commit `110` on the original commit `100` while the local user added an independent commit `200`. This leads to a situation where there is no clear state of the *master* branch in the repository; multiple commits can be said to represent the newest state of the branch. To resolve this conflict, a user must merge the two commits to create

[13] In the following discussion, we concentrate on git because it has the simplest data model. bazaar and git use more complex abstractions which can be converted into git's model.

[14] The opposite operation of pushing is equivalent to a pull to the remote repository. Since it requires the pushing collaborator to have full access to the remote repository, it is not suitable to collaboration, but instead used to publish the contents of the local repository to a publicly served one.

a new root commit of the branch. In figure 2.5.2, the local user has merged `200` and `110` to the new root commit of the master branch, `310`.

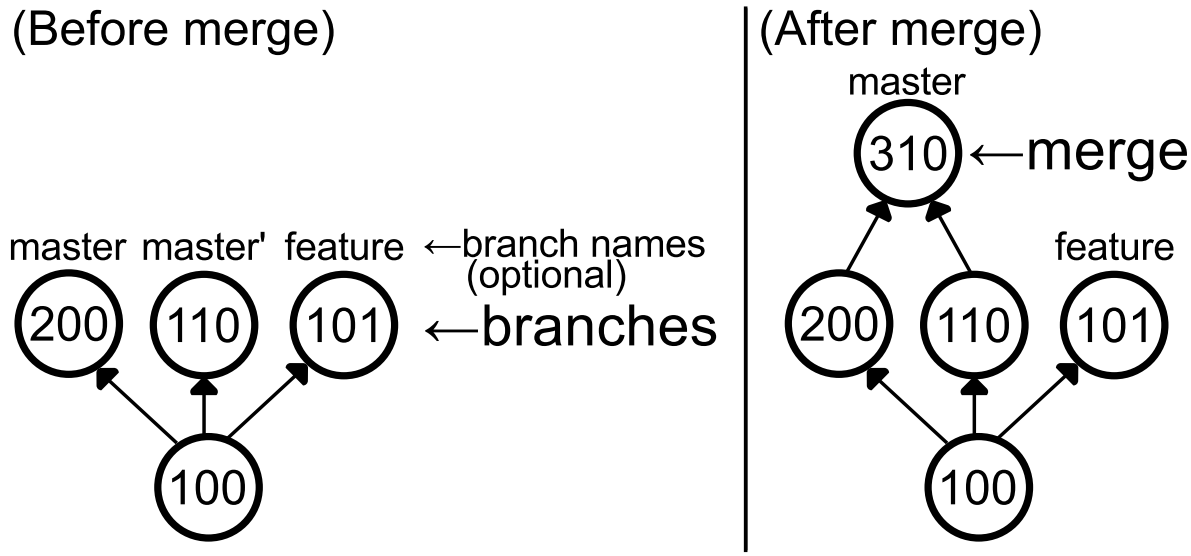


Figure 2.4: Merging different versions of a branch in a graph-based distributed revision control system.

Since we do not want to burden users with merging if at all possible, most merges (for example if two commits change two different files) can be done automatically, without user interaction. This system works reasonably well if every user is using the same automatic merging algorithm, **and** the same exact algorithm to get the revision ID of the merge commit.

In particular, the number of the commits (and corresponding identifiers) generated by automatic merging should be small. While a simple solution would be to not merge automatically at all unless either a new commit is made or a conflict which requires manual intervention happens, this would mean that the newest state of the repository is not determined by one revision, but needs to be constantly recalculated from numerous revisions. Unfortunately, the merge commit metadata is usually prone to variance across system.

For example, git records the username, date as well as the parent commits in order. Even worse: If multiple peers attempt to merge in parallel, the resulting commits have to be merged again. In the worst case, when multiple peers are active at the same time, and merge before getting the opportunity to communicate – unfortunately *precisely* the situation we face in a P2P or delay-tolerant network – automatically generated commits are added to the repository ad infinitum. Figure 2.5.2 shows the revision graph of an attempt of automatic merging. Since both merging peers continued to create merge commits which unified the two latest commits in each iteration, there is never a single root commit.

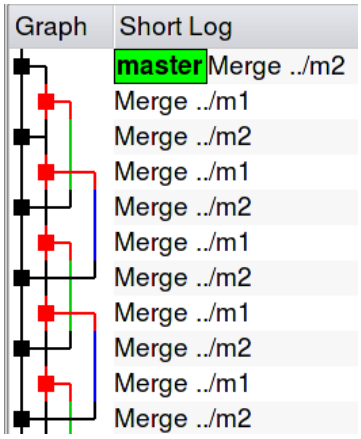


Figure 2.5: Screenshot of the revision graph after simplistic automated merging with git.

These properties must all be set to consistent values to ensure that the result of the merge commit is reproducable. Alternative approaches, such as rewriting the history, are only applicable if the commits have not been broadcasted to other peers before the merging is attempted. Therefore, consistent merging seems the most practiable option for graph-based revision control systems.[Haa11]

Fortunately, no special effort is required to ensure that the revision identifiers of multiple merge commits are equal. This is because git uses a SHA1 hash over the content of the merge (and every other) commit as an identifier. Since the content of each commit also includes the identifiers of the revision(s) it is based on, each commit content is unique. Therefore, each hash over the commit content is also unique[15] and thereby fulfills the primary requirement (global uniqueness) of a revision *identifier*.

2.5.3 Excursus: Content-Addressable Storage

To store not only the commits, but also the content associated with them, git is based on a content-addressable storage (CAS). A CAS just stores immutable blocks of content (i.e. bytestrings). The only way to access these is to query for a hash of the content[16]. While this interface seems extremely limited at first sight, git stores commits, directory trees, and file contents in it, as shown in figure 2.6.

File contents are just stored as blocks, with a short prefix to make sure their contents never match the contents of a repository. Each directory tree contains a plain-text listing of filenames and the hashes of the corresponding file contents. Commits contain a commit message and other commit metadata, a list of the hashes of the preceeding commits in the version graph – none for the first commit, one

[15] As defined in chapter 2.3, we disregard the possibility of hash collisions here.

[16] In practice, the CAS also takes various hints, and allows to delete a block when given its hash. These operations are irrelevant for the following discussion, and can be understood as mere optimizations.

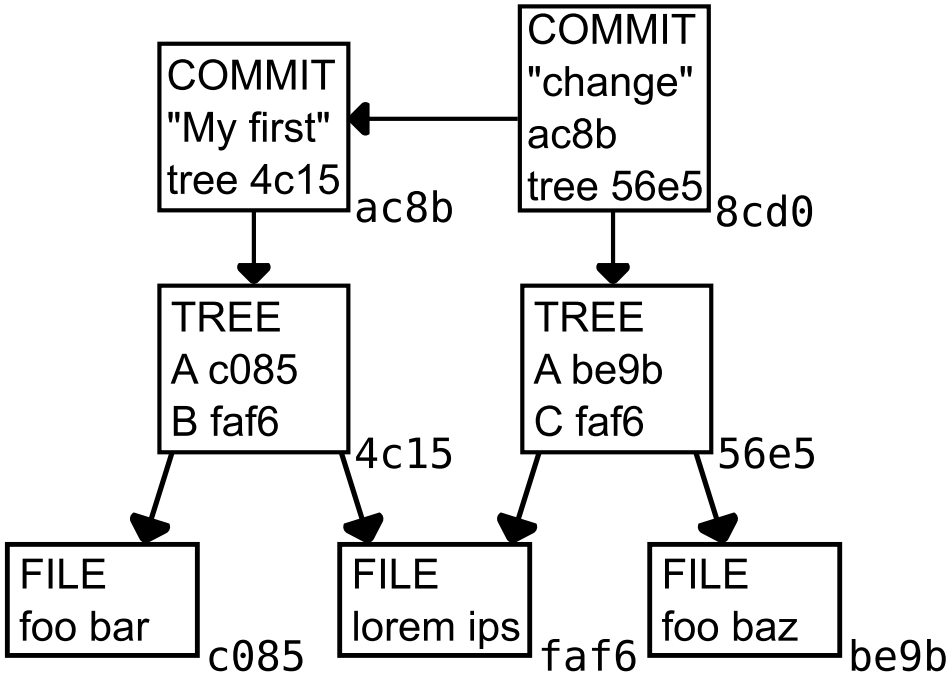


Figure 2.6: git's usage of content-addressable storage. The content of the blocks is shown inside the rectangle, and the hash of the content at the lower right of each rectangle. The arrows visualize the relations between the blocks, but are not explicitly stored by the CAS, but a function of the content.

for normal commits, and more for merge commits – and the hash of the corresponding tree. As a consequence, renaming and copying of files is very simple, all that's needed is a change of filename in the directory tree object. git is therefore said to track *content*, not files.

Extra storage apart from the CAS is just needed to store mutable content. With git's design, the only mutable content are the references of branches to their current commit. The CAS model also allows the CAS to perform transparent optimizations in order to preserve storage. For example, the CAS is not only free to compress single blocks, but can also store similar blocks (for example the contents of two revisions of a file that differ only slightly) with *delta compression*, i.e. as one base block and a list of change operations. Then, if git asks for the content of a block, the CAS extracts the base block and re-applies the change operations.

2.5.4 P2P Revision Control

Although bazaar, git, and mercurial are distributed, they still impose a client-server model; the client (the developer's machine) pushes changes to and pulls changes from a git/ssh/HTTP server.[17] These systems are distributed only in the sense that the remote server is not *required*, and multiple remote servers can be used. Attempts have been made to use version control in a true P2P network (where data is automatically distributed to every peer, instead of only to the configured server(s)).

The first P2P revision systems like Pastwatch[YCM06] merely provided a distributed storage space where users can backup their repositories; if users wanted to incorporate each others changes, they need to explicitly pull changes from other user's repositories. Although technically using a P2P network, the workflow does not differ significantly from a conventional (client-server) distributed revision control system like git.

PlatinVC[Mur10] is a more advanced P2P revision control system. It extends mercurial with a virtual remote which is in fact a distributed P2P network. Users can push and pull to the P2P remote just like they can to any other server. However, PlatinVC keeps mercurial's way of repository sharing; if a user pushes revisions that are not based on the latest commit in the network, she "is informed that an unnamed branch was created, and is asked to to solve it"([Mur10], §7.2.3).

Naturally, this approach only works in low-latency and connected P2P networks where there is a single commonly managed status of the network at a time. This approach fails if the user cannot get virtually instant feedback from the network. Unfortunately, *instant feedback is not possible in DTNs*. Also, the additional user intervention needed to generate the new "latest" revision may be acceptable when the user is a developer who is actively in consciously pushing data to the P2P network. In the case of our general CSCW system, we have to expect nontechnical users which may not be aware, and *should* not be aware that they're interacting with a distributed system, it is not.

2.5.5 Patch-based Distributed Revision Control

darcs[dar] and the related *Camp* project take a different approach to revision control than graph-based systems: Instead of storing the complete state of the repository at the time of a revision, darcs expresses each revision as a *patch* of the previous revision. A patch is a set of instructions to modify the files. These types of supported actions include the addition, deletion, and renaming of a file as well as line-based changes like "replace the word `bar` with `baz` in the first line of file A".

[17] It is also possible to push and pull from/to arbitrary filesystem location. However, the developer has to set up a virtual filesystem – which typically relies on a client-server protocol such as FTP,NFS,SFTP,SMB,or WebDAV – if she wants to push to another machine.

Although some patches depend on others (for example, a patch which changes the content of a file depends on the patch which adds said file), in many cases the order of patches is reversible. Users are also generally free to cherry-pick some patches and ignore others; darcs does not require a single root per branch like graph-based revision control systems do. As such, darcs eliminates the problem with automatic merging in graph-based revision control systems discussed above. Its model also matches the intuitive semantics of changing a line better than git's, where the actual change is only visible in the delta compression optimization step.

However, the patch-based approach also has some downsides: For once, a minimal implementation of a patch-based system is quite complex; it must know the different change types, and be able to order changes in a consistent manner. Conversely, the theoretical underpinning of patched-systems (*patch algebra*) is complex[Jac09] as well, and still being developed and formally proven. The darcs implementation is also experimental in some aspects, with known corner cases where the runtime of applying updates becomes exponential in order of the number of patches. Additionally, the lack of multiple independent implementations effectively makes integration with darcs (which is written in Haskell) mandatory.

Similar to git, darcs is also based on a CAS, although to a much lesser degree, as numerous indices are kept apart from the CAS.

2.5.6 Document-oriented Database Systems

Instead of tracking the state of whole repository, the replication mechanisms of **document-oriented databases** like CouchDB only manage the state of single documents[cou]. We should therefore consider such *document/revisions* data models in addition to the *revision/files* revision control systems presented above.

While document database systems sacrifice the abilities to restore the whole repository to a certain revision, and to group changes, both of these abilities are not necessarily required in informal collaboration work. In fact, the existing cooperative policy drafting systems such as adhocracy do not include any features that would allow users to always group two changes, for example two different comments, in a single revision. Both graph-based as well as patch-based revision control systems can simply be applied to each single file, or even single paragraph, in order to eliminate the need for automatic merges.

Interestingly, CouchDB uses neither in its equivalent of automatic merging. Instead, it picks one of the revisions by an arbitrary, but consistent algorithm (it simply picks the candidate with the lowest hash value). Both versions are stored in the database, but it is up to the application to query for the

discarded one, and merge it with the latest version of the document. Adopting this simplistic scheme into a collaboration system atop a DTN would mean that the system drops some versions at random (from a user's point of view). It also opens the door for attackers who could generate an "alternative history" of the document by artificially generating entries with a low hash value, and then posing these into contention with early versions of the real document, thus ensuring that the attacker's branch gets always picked as the "latest" one.

Nevertheless, revisioning only single files instead of the whole repository seems like an extremely simple way to minimize the need for automatic merging, and will therefore be considered in the final system design.

2.5.7 Integration Notes

CAS is an extremely powerful but also extremely simple concept that can be used to not only store graph-based, but also patch-based, and versioned-files revision control systems, and is the foundation of the block concept used for network communication. For details, refer to chapter 3.2.3.

If a graph-based revision control system is to be used in a distributed environment, automatic merging is desirable, but poses significant problems and requires modification of the revision control system with a carefully designed consistent merge function. The current solutions for using graph-based revision control systems in P2P networks do not offer automatic merging, and are not applicable in DTNs.

Patch-based revision control systems pose numerous advantages, but also introduce complexity and will require further work in order to remove corner cases and simplify integration into CSCW systems aimed at end users.

Document database systems minimize the need for automatic merging. Since a "file" will ultimately represent a (rarely-changed) single proposal or comment, the loose consistency guarantees may be sufficient for some CSCW applications.

Additionally, we also must consider the algorithms needed to **merge** actual **conflicts**; if two users modify the same document, paragraph, or word, the system should not only offer a user-friendly interface to reconciliate the conflict, but also powerful algorithms that assist the user or can automatically merge content, for example by comparing the differences of both versions to the common ancestor or using patch algebra.

Chapter 3

Architecture

We want the collaboration software to be universally accessible from virtually every device and operating system. We also want users to be able to use it within seconds, without requiring a complicated installation. Therefore, our collaboration software must be a web application. As an added bonus, almost all efforts to centralize control over the software installed on a user's computer (→ 1.2.3) only restrict native applications; and not web applications.

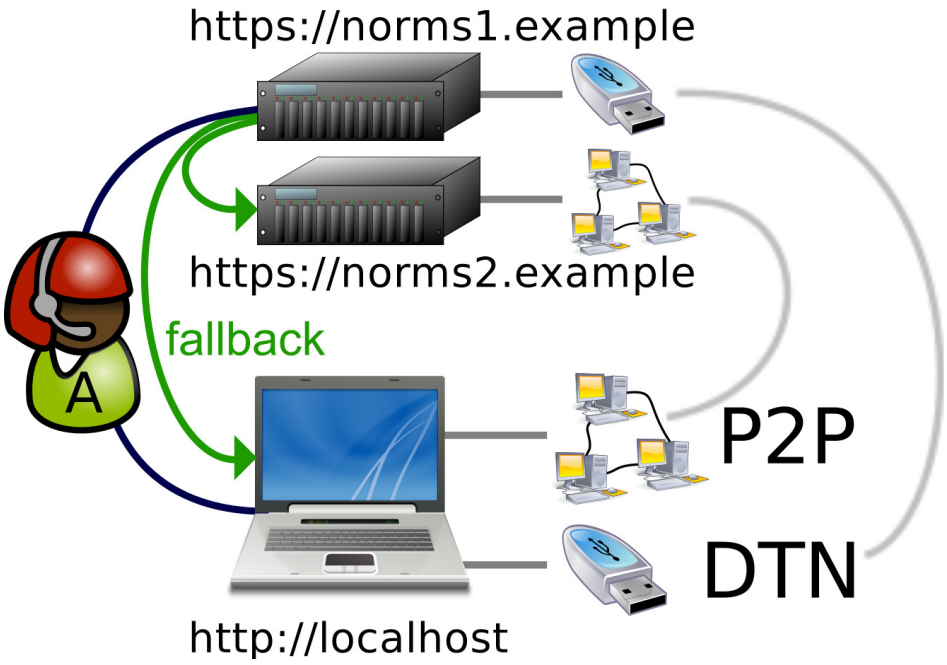


Figure 3.1: High-level overview of the architecture

A simple web application has two major drawbacks: It cannot implement P2P/DTN connections, and it is easily censorable, for example by blocking its IP address. To mitigate DNS and IP censorship, the web application should be offered by multiple IPs, and should – if possible – transparently fall back to these. In the case that all of these are blocked, or total internet shutoff, the collaboration software

can be installed on the local machine, and serve the web application from there. With modern web browsers, it is also possible to shift most of or even the entire application to the web browser.

The various peers – whether serving thousands of users from a data center or just a notebook – then distribute the contributions of their users to each other via DTN and P2P networks. Optionally, the content can be synchronized with external, third-party collaboration applications.

3.1 Implementation Architecture

The architecture of the implementation stresses flexibility over anything else. Therefore, we not only allow for multiple user interfaces, but also for multiple applications with different semantics. For instance, an automated test application to verify network connectivity does not need any storage components, but does require some kind of timely event or callback functionality to register successful transmission. The implementation is split into a number of components:

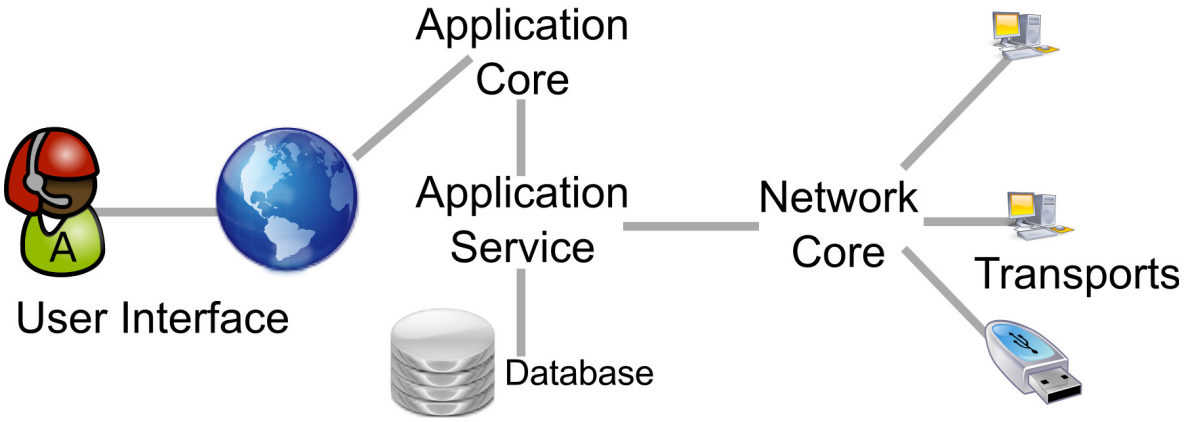


Figure 3.2: Overview of the Implementation Architecture

- The **User Interface** is a web application. It mainly consists of application-specific logic, but also allows administrators and users to configure the server and see its current state. For example, the user interface allows the user to start using attached USB thumb drives for data transmission. The user interface is discussed in detail in chapter 3.4.

- **Applications** form the central component of the whole system. The application is implemented in two parts: The **application service** provides an abstract system that the **application core** builds upon. For example, the first implemented application service provides a filesystem-like interface. The first implemented application core provides a simple cooperative normsetting platform. Whereas the application service layer only deals with abstract files, directories and

their revisions, the application core understands the format of these files to represent proposals and comments. Messages to and from the application service are JSON[1]-compatible dictionaries. The application service provides the following interface:

`handleMessage(msg)`	Handles an incoming (broadcasted) message. The message is already parsed (i.e. comes in as a JSON-compatible dictionary). This method is mostly used for short, time-sensitive messages, such as notifications about new content. The implementation also passes a hint of the endpoints (→ 3.3) that sent the message alongside; this hint can then be passed along to the `getBlock` function of the network core (see below).
`listRoot(selector)`	Called by a remote peer. Lists a number of root entries. The exact semantics of this method depend on the revision control model in use.
`getBlock(blockId)`	Called by a remote peer. Downloads a piece of content.
`handleEndpoint(ep)`	Called when a new endpoint (→ 3.3) becomes available. Typically, the application may want to ask for the current state of content on the remote end.

The application is also responsible for storing the managed content on disk, typically in a local database.

- The **network core** mediates between application and transports. It provides the following interface:

[1] JSON[jso] (JavaScript Object Notation) is a simple data interchange format that can only represent dictionaries (also known as hash tables or maps), arrays (also known as lists), characater strings, integers, floating point numbers and booleans. Complex structures can be represented in JSON by nesting dictionaries and arrays. Nevertheless, the small and simple set of data types allows for efficient and portable serialization.

`broadcast(projectID, msg)`	Encodes and broadcasts a message to all endpoints. Typically, the message is an application's notification of new content.
`listRoot(ep, projectID, selector)`	Called by a local application to list the root contents of a remote endpoint, typically when the application is notified with `handleEndpoint`.
`getBlock(ep, projectID, blockId)`	Called by a local application to get the specified block.

The network core makes sure to always maintain a connection to the network for each project. The current implementation multiplexes the data of multiple applications/projects over a single endpoint per peer. This allows a shared bootstrapping mechanism for multiple projects.

The network core also broadcasts all outgoing messages to all remote endpoints that are interested in the specified project. A typical pattern is that a remote peer announces the availability of a new datum to an application, which then checks whether it already knows of the datum. If it does not, the application advises the core to relay the message to all other connected peers. A hop count and a TTL entry are used to ensure that all broadcasts time out (→ 2.1.3).

- The **transports** implement the actual endpoints. They are also responsible for bootstrapping. For details, refer to chapter 3.3. The transport API is extremely simple:

`endpoint.send(bytes)`	Send a bytestring to the current endpoint.

The transport also offers callbacks to notify the network core about the following events:

`onRecv(ep, bytes)`	Bytestring received.
`onNewEndpoint(ep)`	A new endpoint is available.
`onEndpointError(ep)`	An endpoint is no longer available.

This design allows us to replace the implementation of virtually any element. For example, connecting via an anonymization network such as Tor or I2P only requires the implementation of a transport that tunnels messages between the network cores. Transports that are only capable of data storage (like DTN transports or data-storage anonymization networks(→ 2.4.3)) just implement the network core's API.

3.1.1 Finding Project Nodes

Every application core must always run the `RunningProjectsApplication` identified by the static project identifier `f1a9647882add21d02b8d372a6da940140c1ba556896825331c23c56-aeb99d57`. This application has no backing data storage. Its `listRoot` method lists the IDs of all the applications/projects (see below) that are currently running on the local machine. If this set changes, the application broadcasts a corresponding change event to all connected peers. This application also relays seeks for any nodes that run a certain project.

3.2 Applications and Projects

The system should be able to handle multiple independent **projects**; drafting of examination rules should not interfere with activity in a pro-democracy manifest in any way. Each project is also associated with various meta information such as a name, security/access control information, and an application type. If a more detailed control is required, the project can simply be divided in individual subprojects and a meta project which lists them. Projects loosely correspond to the notion of *instances* in the centralized *adhocracy* system. Each user is interested in a subset of projects, and each peer is interested in the union of all the projects its users are interested in.

3.2.1 Project Structure

We can use the cryptographic primitives from chapter 2.3 to securely globally uniquely identify projects with a *project ID* as follows:

1. The implementation selects a set of cryptographic algorithms and the corresponding **algorithm ID**. Algorithms may change in response to advances in cryptography. Since it is critical that all nodes support the selected algorithms, the algorithm is typically chosen statically in code, and not by a user.

2. When creating a new project, the creator generates an asymmetric key pair for the project. Possession of the **private key** equals unrestricted administrator rights over the project. The **public key** will be used to verify signatures over the project's metadata and data.

3. The creator also selects the project's **type**.

Algorithm ID	Immutable, go into project ID
Public key	
Project type	
Human-readable name	Signed with the project's key, mutable for administrators
Human-readable description (optional)	
Tags/categories (optional)	
Security model ID	
Revision ID	
Signature	

Figure 3.3: Project header

4. From now on, the creator can calculate the project ID as

$$projectID = hash(algorithm\ ID, public\ key, project\ type)$$

This design ensures that each project ID is of a managable size (typically 256 bits), and not as large as the public key. Nevertheless, the project ID is unique because it is impossible to find hash collisions. Since public key and project type are immutable, an attacker cannot publish wrong information about the project (such as an identical project with an unsupported project type, or an identical project with a public key of the attacker's choosing). If the attacker modifies one of these properties, he automatically creates a new, unrelated project.

The project also has mutable properties, in particular presentational properties such as a name, a description, tags, and categories. The project's root security information (as laid out in the following sections 3.2.7 and 3.2.8) are also included. To allow further updates, the list of mutable properties also contains a revision ID which is incremented on each update. Newer revisions of projects displace older ones permanently. Since the project properties are rather stable and tend not to be incompatible with prior changes, this simplistic versioning can be expected to suffice. All these properties are then signed with the project's private key. Therefore, only an administrator can change important project properties.

If the project is not public, its content (messages and blocks) is symmetrically encrypted as detailed in chapter 3.2.7. The corresponding blocks of encrypted symmetric keys are stored by the project itself; each encrypted key is stored in an (unencrypted) block of its own and managed like other project content. `listRoot("symkeys")` is also special-cased by the implementing application to return an unencrypted list of these special blocks.

Other intermediary certificates are attached in every case which needs them. If the security model only allows selected users to create content, every message and every block must be signed and include the signing key as well as its certificate chain up to a certificate from the project's key. Voting systems are implemented in the same fashion; every vote must be signed by a key that must be signed by the voting authority, which must be certified (as a voting authority) by the project's key.

Apart from the project header and the aforementioned cryptographic functionalities, the structure of blocks and messages is left up to the application.

3.2.2 The ProjectListApplication

The ProjectListApplication serves as a distributed, fully replicated database of all projects available in the system. While it is not technically necessary to run it– after all, in a closed network the project's participants already know the project's type – virtually any peer runs this application. The main purposes of the ProjectListApplication are allowing users to search for projects and updating project metadata.

Naturally, the metadata of the ProjectListApplication cannot be distributed over itself. Therefore, it is simply known as the project with the ID `ec43065d660f5788240ee6203f692bed8c1ac026-b822e9abea309b72dcb8c0b1`. It uses a special security model in which everyone can write anything, but only valid blocks are accepted. The remaining properties of a project are undefined for the ProjectListApplication.

Every project is stored as a block whose ID is the project ID, and whose contents is a representation (encoded as JSON) of the project header. `getBlock` can be used to get the information about a project whose ID is known. `listRoot` returns a list of all project IDs. Locally, the implementation will typically have a way to get subsets of the whole list and display them in the user interface, for example in a "Search for projects" dialog. If a new project is added, or an existing project is modified, the new project header block is encoded in the message and broadcasted. Upon receiving a new block – whether as a result of an explicit request or a broadcast –, this application checks the cryptographic signature of the project properties. If it is correct, the information is stored in-memory. Otherwise, the information is discarded.

The ProjectListApplication is typically not backed up by a database; its content is ephemeral. On application start, the application creates in-memory entries for all locally available projects and starts synchronizing with other peers. The current implementation does not contain a way to ever delete projects from the global database. However, it is possible that an additional unsigned header which contains the wall-clock time of the last update will added in future revisions.

3.2.3 Application Services

All implemented (versioned-)filesystem-like services share a number of similar patterns. In particular, blocks are used as content-addressable storage (CAS; their ID is a cryptographic hash over its content). Similar to the project ID, this behavior guarantees globally unique block IDs. Additionally, the synchronization of two CAS can never fail – at most, the CASs can contain superfluous blocks after synchronization, and even that is only possible if we ever delete blocks. Since blocks are never deleted in our system, it is always possible to restore any old version of the filesystem-like structure. Blocks are also fully replicated as any other design would impede functionality in the event of a total internet shutoff.

When a new block is inserted into the system, or becomes known to a node, the node broadcasts a message `newBlock` with the block ID. While currently not implemented, the introduction of a `newBlock`**`s`** message for multiple blocks is anticipated. Upon receiving a `newBlock` message, the application checks whether it has the block in store. If it does, it simply ignores the message. Otherwise, it requests the block, typically from the node that sent it.

3.2.4 Revision Control Application Services

Graph-based revision control systems (such as git) can natively run on the abstract filesystem-like service; they use `listRoot` to transmit the list of branches and the associated latest commit hashes, and only require the CAS (`getBlock`) apart from that.

Patch-based systems can also use the CAS to store the patches, but may require adaption in order to be able to serve the list of all patches via the `listRoot` method. One such solution may be the *bundling* of patches where `listRoot` only returns the hash of an entry in the CAS which lists a set of patches.[2] In order to avoid flooding the CAS with a large number of slightly different bundles, care must be taken so that the bundles are identical across systems. For example, it might be prudent to bundle a year's worth of patches only after another year has passed, assuming all patches are distributed to all active nodes within that timeframe.

In a **document database** system, we need a way to get the set of all revisions of the entries. Since this set will be huge, exchanging the whole set on every communication (even if both nodes already have a very similar set of blocks in their locale storages, or are fully synchronized) in a `listRoot` call is not scalable. To reduce the size of the exchanged information, we could bundle revisions entries similar to the bundles in patch-based systems. However, this solution only works for old entries that

[2] Bundles are already implemented in darcs, although they are not stored in the CAS.

we can assume to have been distributed to every node of the network, and does nothing to address the large number of entries that can accumulate in a high-traffic policy drafting forum even in small periods of time.

Instead, `listRoot` can leave out old entries. To make sure that differences in older content are not missed, the hashes of the older content are themselves hashed and included in the answer. For example, assume the current time is 2012 and a peer with the following database receives a `listRoot` query given the database 3.4.

Block Content	Block ID (hash)
`time=1 A`	cf3d
`time=500 B`	7a1c
`time=999 C`	4f92
`time=1000 D`	d0d6
`time=1234 E`	a98d
`time=2001 F`	5dea
`time=2002 G`	6dd4
`time=2010 H`	de32
`time=2011 I`	5e16

Figure 3.4: Example block database state. The content of the blocks has been simplified; in practice, each block content contains the file name, revision id, and the content of the file at that revision.

The naive answer would look simply list all hashes, and therefore return a voluminous answer[3]. Instead, `listRoot` can return not only a list of hashes, but also qualifiers (akin to branch names in git), as shown in figure 3.5.

Record type	Record data	Hash	*Computed as* (not transmitted)
timeframe	0-999	867d	hash((cf3d, 7a1c, 4f92))
timeframe	1000-1999	ab18	hash((d0d6, a98d))
timeframe	2000-2009	9943	hash((5dea, 6dd4))
block		de32	hash(`time=2010 H`)
block		5e16	hash(`time=2011 I`)

Figure 3.5: Example optimized `listRoot` answer

[3]Note: hashes are shortened in this example. In practice, hashes are typically at least 32 Bytes long.

If the receiving node already knows all blocks but `5dea` and `5e16`, its `2000-2009` hash will not match the answer. It now requests `listRoot(1000-1999)` and gets the list of blocks in that timeframe, optionally again simplified to multiple (shorter) timeframes. Eventually, the peer determines that it misses precisely the blocks `5dea` and `5e16`, and requests both via the CAS function `getBlock`.

If the nodes use previously agreed-upon timeframes (in the example steps of thousand and ten), both nodes can precalculate and cache the hash of all block hashes in each timeframe.

3.2.5 Interaction with the Application Core

Since blocks are immutable, the application service can easily store them in a way that is more accessible to the application core. For example, graph- and patch-based version control systems usually extract the latest revision of the managed files into the filesystem. Of course, this can be skipped if the host does not actually run the application, as is the case for DTN media such as USB thumb drives that do not have their own processor or user interface, and conversely have no use for data intended for the application core.

Instead of maintaining two databases[4] – one for the blocks and one for the application core's view of the data – it is also possible to only store the application core's view of the data and include the block ID (i.e. the hash of the block content) in this database. The only restriction on the single-database model is that it must be possible to extract the original content from the application's data given a block ID, i.e. there must be a deterministic bijective serialization/deserialization algorithm.

3.2.6 Policy Drafting Application

In the prototype, we assume the following structure of a simple policy drafting system:

- Each project has a set of **proposals**, which can be edited. Every proposal has a title and a longer textual description. In a minimal system, the textual description is just arbitrary plain text. In a slightly more advanced system, the textual content is subdivided in paragraphs, and is not plain text anymore, but formatted text, stored in a portable markup language. To increase usability, the user interface may allow the user to use a WYSIWIG editor to change the markup. For now, we assume that proposals are independent of each other. Eventually, the system must support

[4] The term database is used loosely here for any way to store data, including files on a filesystem and in-memory data structures.

voting for proposals. In the final system, we also want to be able to restrict users from editing proposals, and to show the (verified) user name of the proposal author.

- Each proposal has a set of **comments**, each of which is versioned and consists of textual content and can be eventually voted upon, much like a proposal. Additionally, comments are planned to be able to refer to either another comment, or a paragraph/line on the proposal.

- Eventually, the production-ready policy drafting system should also be able to store **binary files** (such as images, PDF files, etc.), and offer markup constructs to link to or embed them into proposals and comments.

While we can simply export all block contents into a **relational database**, such a database only offers reduced flexibility. In particular, multiple implementations (or versions of the same implementation) may support different attributes. Additionally, since many items have to be revisioned, the database design must explicitly include revision numbers.

In contrast, a **document-oriented database** allows us to simply submit all blocks with little exporting, and make use of the database's indices and native revision model in order to store various revisions of proposals, comments, and files. Additionally, documented-oriented databases generally do not require an explicit schema change to accomodate new or different attributes of an entry.

It is also possible to base the policy drafting application on a versioned filesystem as offered by graph- and patch- based revision control systems, using a data structure similar to the import/export format in our version of adhocracy. In that format, each proposal is primarily a directory. It contains various files (such as `title`, `description`) or a JSON file `index.json` which contains the proposal's properties. A subfolder `comments` which is created on proposal creation contains all the comments in a similar format.

3.2.7 Write Authorization

In many projects, we do not want everyone to be able to change everything. Instead, we want to implement one of the trust models described above. Since the project structure (→ 3.2) associates a public key and a security model ID with each project, we can restrict write authorization by either setting the security model ID to the name of the used model. From now on, every restricted message or block(hereafter referred to as *content*) must include a digital signature, a cryptographic hash of the public key and an identifier of the key and its certificate chain. To validate the content, the node then retrieves the chain of certificates and public keys. If intermediate certificates are allowed by the trust

model and are in use, it iterates up the certificate chain. The final certificate must be signed by the project's public key. If the content fails to meet any of those steps, it is ignored by the node.

To allow revocation of certificates, the node must also query for the block with the ID `hash("revocation", public key)` for each public key in the certificate chain. If the result is a revocation certificate signed by a valid authority, the content is ignored. Note that this model of certificate revocation only works if the node can communicate through a channel (including DTNs) which is not controlled by the attacker – otherwise, the attacker can simply withhold the revocation certificate. Therefore, in order to include reliable certificate revocation, every piece of signed content must also include the ID of another content, which must contain a list of IDs of all certificate revocation certificates. With this extension, an attacker who controls all communication channels can still transmit old content and added own content, but must withhold new content (which would inevitably come with the updated certificate revocations) from other users.

In any case, public keys of authorized users, certificates, revocation certificates, and revocation certificate lists are stored in the CAS and updated just like any other content the project manages.

Some applications may also wish to grant additional privileges for some action. For example, if a block contains the public key of its author or a hash thereof, the application can additionally allow any revisions of this document as long as they are signed by the public key[5]. In a similar fashion, a policy drafting application may allow comments from everyone, but restrict the creation of proposals to a closed user group.

Nodes are also free to use their own local trust model in addition or instead of the trust model suggested by the project. They are free to verify content with their web of trust or other mechanisms. However, this is only practicable if the underlying application service supports it; graph-based and inter-document approaches tend to work better here since they can express or tolerate permanent differences in the set of accepted data between the nodes. Therefore, if a web-of-trust security model is chosen, special attention should be paid to the application service.

3.2.8 Read Authorization

Not all content is public; closed groups like the professors at a university or employees of a company will want to prevent outsiders from reading their drafts and comments. While we can simply deny read access from unprivileged users in a centralized system, a distributed system requires some kind of encryption.

[5] The security model ID specifies which of these overrides are allowed.

Since we want the encryption to be fast, and do not want to re-encrypt the whole repository content every time access is granted to a new user, we encrypt all content and messages with a symmetric encryption scheme and a randomly generated key. The symmetric key is then encrypted with an asymmetric encryption algorithm and the public key of each user an administrator wants to grant access to. (Naturally, messages and content that make up these asymmetrically encrypted symmetric keys are excempt from symmetric encryption.) The user requests the encrypted key and uses her asymmetric private key to decrypt it.

Since we do not encrypt the content for each user, the global symmetric key needs to be exchanged if it is compromised. Notably, this also means that while we assume an administrator is delegated with enabling access, every user who can read the content herself can allow any other user to read access. This is not a security vulnerability since the user can always share her private key or the decrypted content with the other user.

3.3 Transports

Transports actually connect our system with an *endpoint*, some kind of external system. They come in two forms:

- **Dumb** transports just relay messages to specified endpoints, where another instance of the system is running. The network cores at either side encode and decode the messages. The basic P2P transport is a typical dumb transport.

- **Smart** transports connect to endpoints that cannot implement the application themselves. These are typically DTN endpoints. Smart endpoints are equipped with one or more *virtual application services* which offer the same basic interface that regular application services do, i.e. a CAS with some extra metadata. The virtual application services do not need to offer a database to any application core; they just need to store the data. In the prototype implementation, we reuse large parts of the application service for the virtual application service.

Figure 3.6 shows the interaction in detail. We note that the UI can also trigger administrative actions, and receives notifications from the transports directly. The communication to the UI is largely an artifact of the current state of the implementation and used to configure plugged-in thumb drives and the like. This communication is expected to be reduced as much as possible; for example by using a fault-tolerant filesystem and file format instead of requiring the user to eject a DTN medium.

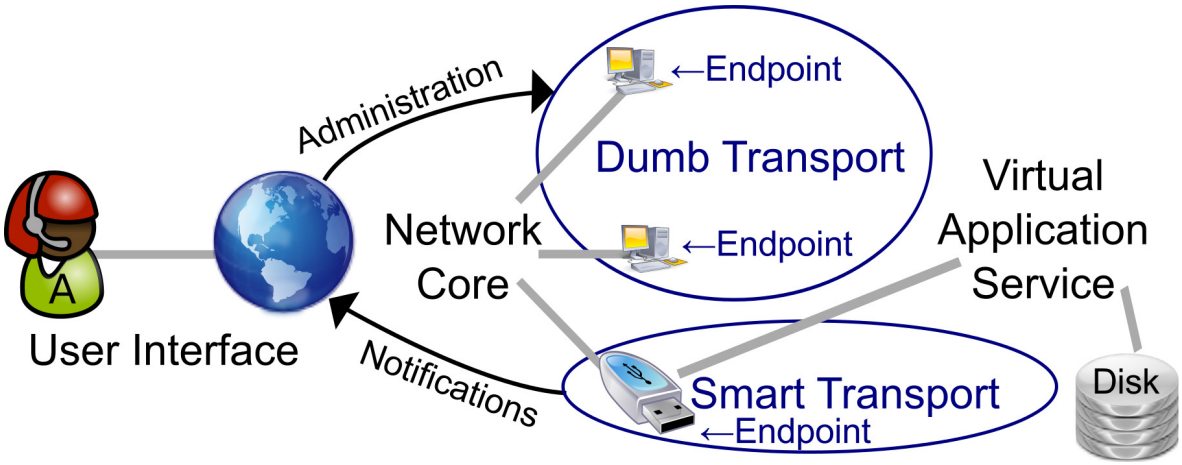


Figure 3.6: Detail view of the transports in the implementation architecture

At the current stage, the only implemented transports are a simple P2P and a simple DTN (over USB thumb drives) one. Eventually, these transports should be refined and extended with cryptographic, steganographic and performance features.

The envisioned transports include bridges to other policy drafting applications. For example, an *adhocracy* transport could automatically synchronize between various adhocracy installations (endpoints) and the local system. Similarly, a transport can be used to automatically export policies, say, into OpenDocument files.

Another possible transport would be similar to the P2P one, but actually tunnel through an anonymization network (→ 2.4).

Yet another transport could mirror the data to some kind of shared storage, for example facebook or flickr. Steganographic techniques can be used to hide the presence of policy drafting information in the social network service.[BFV10]

3.4 Web Application

The biggest strength of a web application is that it can be used from virtually every device just by entering a URL. Web applications and the HTTP protocol they're based on are common and well-understood when it comes to aspects such as security, scalability, and development tools. On the flipside, this also means that virtually any censorship system can understand and block HTTP/web applications.

Web applications are commonly secured with HTTPS[6]. The HTTPS protocol[Res00] is simple: Regular HTTP requests and responses are simply transmitted over an SSL/TLS-encrypted connection. Under the assumption that SSL/TLS prevents the attacker from reading the transferred data, the attacker can then only block *all encrypted* communication, which eventually leads to blocking *all* communication (→ 1.2.8). In chapter 2.3, we also saw that the trust model of SSL/TLS can be vulnerable. However, we also examined alternative models such as *convergence*[con] which correct these flaws. We therefore assume that HTTPS is sufficient to secure web applications against DPI attacks.

3.4.1 Server Fallback

However, HTTPS cannot protect against DNS and IP blocks. To circumvent those, we employ two kinds of *fallback* hosts:

Alternative **servers** that are reachable under different domain names and IP addresses. These servers are well-connected, set up in free countries, and have valid SSL/TLS certificates. While it is in theory possible to use a large number of domains and IP addresses – akin to domain generation algorithms[KHKK10] and *fast-flux* rapid switching of IP addresses in DNS records[NH08] in the context of malware – it is probably not feasible to set up and distribute IP addresses and DNS entries (as well as the needed certificates) faster than the attacker can block them. Therefore, alternative servers are only meant to defend against ”casual” censorship and network outages.

This design is more flexible than simply associating a domain name with multiple servers or associating multiple servers with the same IP address since it allows for multiple *providers* of the same service. For example, our university project could not easily setup and maintain servers outside of Europe, whereas a Korean University project might be able to provide multiple servers across asia. While we could set up a domain name that resolves both to the German and the Korean server, this would mean the SSL/TLS certificate would have to be shared amongst both universities (which would rightfully never be allowed for security reasons).

If the attacker is resourceful enough to block all central servers, we need a different, **local** fallback host. This also allows fallback in the case of total internet shutoff or an unanticipated successful censorship action. Figure 3.7 shows the UI in the prototype; the web application examines which of the preconfigured fallback hosts it can switch to. The user can also configure the web application to automatically switch to a suitable fallback server[7].

[6] The alternative SHTTP standard[FGM+99] remains virtually unused.
[7] The automatic switching is primarily intended for cases where the original server is regularly unreachable/censored.

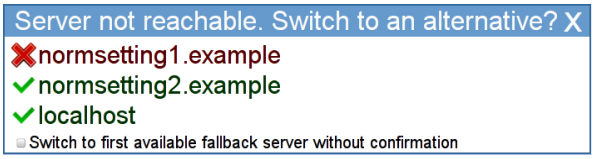


Figure 3.7: User interface for server fallback

The implementation of server fallback makes use of modern web technologies, in particular WebSockets[Hic09], for security as well as performance reasons. Conventional techniques to communicate with other servers than the one hosting the application[8], such as JSONP[9][Ipp05] place total trust in the remote host. This allows an attacker who gains control of one of the fallback servers[10] to inject code that publishes the user's identity, data, and prevents the user from further access to the original service.

Since WebSockets allow TCP-like connections from the (client-side) web application to a remote host, they not only prevent these kinds of code injections attacks and assuage performance considerations over the number of HTTP requests, but also allow additional security checks. For example, the web application can ask the fallback server for additional (on top of SSL/TLS) certificates.

3.4.2 Preventing Malicious Fallback Servers

In particular, the additional verification opportunities can be used as shown in figure 3.8 to ensure that even when the attacker manages to capture a legitimate fallback server (for example by seizing the servers), the server can be invalidated in short order. The original server (assumed to be trustworthy) picks a trustworthy certificate authority (CA) or serves as one and automatically certifies potential fallback hosts for a short period of time, for example a day. The original server includes this certificate in the code he sends to the client.

When the connection between original server and client is severed, the client can still verify that the fallback server is still trusted by the CA. Notably, this does not require any communication between the client and the server or the CA, both of which are supposedly censored.

Unfortunately, the added verification of the fallback host above and beyond SSL/TLS comes at a price: It is now possible to prevent fallback entirely by compromising the CA or preventing communication

[8] To prevent web resources from retrieving resources at other domains with the user's credentials, the *Same-Origin Policy* limits interaction between web applications from different domains (*origins*) in all modern browsers. [Zal10]

[9] Since the same-origin policy is not applied when client-side JavaScript code is loaded from another domain, JSONP requests work by requesting *and then executing* code from a remote domain in the context of the current origin. For the purposes of this discussion, it is necessary to note that this approach works, but is obviously horribly insecure if the remote host can't be trusted.

[10] Refer to chapters 1.2.5, 1.2.9, and 1.2.3 for the corresponding attacks

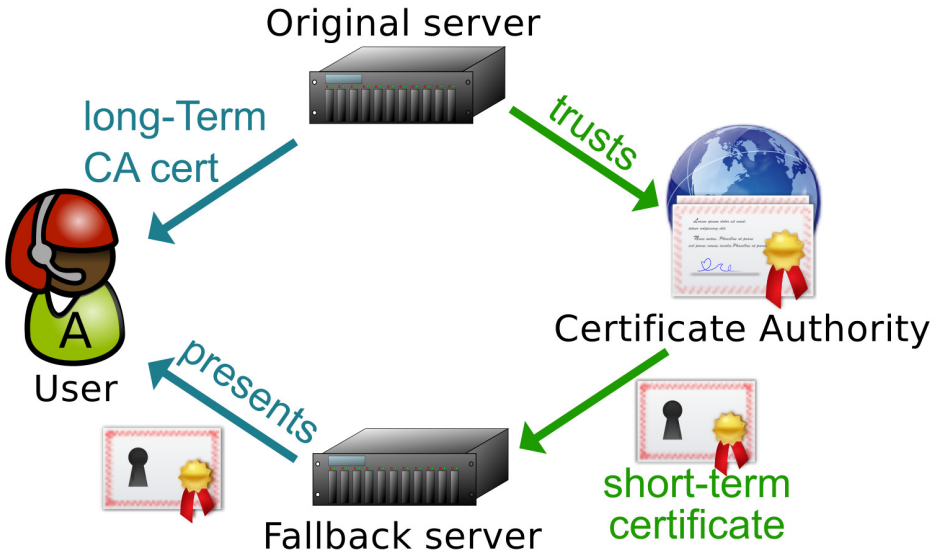


Figure 3.8: Fallback verification model

between CA and fallback servers. Compromise of the original server is not made any worse; an attacker who controls the original server can simply send malicious code (or just a blank page) instead of the client-side fallback code. The protocol also increases the general complexity of the system without offering absolute security: If the user switches to the fallback server while the certificate is still valid, and the attacker manages to set up a malicious server within the certificate duration or before the CA notices the compromise, the user is still lured to a malicious fallback server.

For now, we choose to error on the side of simplicity, and rely on careful vetting and SSL/TLS to secure fallback servers.

3.4.3 Offline Web Applications

In a traditional web application which is downloaded from the original server on each visit, fallback is only possible as long as the application runs in the user's browser while the server is being blocked. After the user closes the application, any further visits to the application's URL result in an error message or a site under the attacker's control. Therefore, we need to advise the web browser to permanently store the web application while the original server is still uncensored.[11]

The *Offline Web Applications* feature of HTML5 ([HH10], 5.6) allows exactly that. Upon loading the web application, the server refers to a *manifest* URL. A web browser which supports Offline Web

[11] In practice, this means that a short visit to the original server while evading censorship, for example while travelling or while using a slow anonymization network, is enough to permanently store the application in the web browser's configuration. This makes the application available until the user switches to another web browser or machine.

Applications downloads the manifest and extracts a list of all URLs of the resources of the application (i.e. markup, style sheets, JavaScript code files, images, etc.). The web browser stores all these resources in its configuration.

If the web application's URL is revisited later on, the web browser loads the application from its configuration immediately, without the need for any network interaction (although the application can itself initiate network interaction, for instance set up WebSocket connections). In parallel, the web browser redownloads the manifest file. If the manifest file has changed, the web browser re-downloads all resources and stores them for the next time the application is used.[12] If the server returns a `404 Not Found` or `410 Gone` status code, the web browser deletes the application cache. In all other cases – notably when the server is not reachable because of censorship or network outage, and when SSL/TLS fails to establish a trusted connection[13] – the browser does nothing but continue running the stored application.

Offline Web Applications therefore allow the original server to install a web application in the user's web browser that cannot be deleted unless the attacker manages to compromise HTTPS security. In conjunction with the fallback mechanism, this allows our application to switch between the central site, multiple fallback servers, as well as a locally installed server.

3.4.4 Client-Side Web Applications

When the fallback servers are not available, we require the installation of a local software. By including the necessary installation and documentation files in the manifest file, we can make sure the user can install the software even after the censorship/network outage is in effect. However, while locally installed software may be unavoidable when access to USB thumb drives or other DTN as well as P2P connections is required, many users do not wish to or are not able to install native software. Instead, the application should remain usable even if no fallback is available or selected.

This can be accomplished by storing the content in the web browser's cache (for example using Web Storage[Hic11] or the FileSystem API[Uhr11]) ahead of time and then providing an **offline version** of the web application that runs entirely client-side. An advanced offline version could even allow offline changes (such as comments or proposals) which are synchronized as soon as the connection to the server is restored. In the end, the offline web application becomes just another peer, changes are distributed and integrated just like they are with another node in the DTN/P2P network.

[12] The prototype web application developed in this thesis recognizes this case and switches to the new version of the application.

[13] In all popular web browsers, SSL/TLS trust can **not** be overriden by the user when requesting a manifest via HTTPS. This solves the security problem of users who "just click through" all HTTPS errors. Neat!

This thesis does not include an implementation of a client-side offline web application, although the prototype strives to be portable to the client-side. In particular, the data format and templating language used in the server-side web application are language-independent and should therefore be easily portable to a future client-side application.

In the future, it might even be possible to also run the P2P and maybe even DTN components on the client. The highly experimental WebRTC[web12] API could allow offline applications to directly communicate without requiring a central server (after the connection setup).

Chapter 4

Implementation

The implementation (codenamed `d2p`) is a prototype through and through; it is geared towards simplicity and interchangeability. Unsurprisingly, the implementation closely follows the architecture laid out in chapter 3.1.

The system is implemented in Python 3, which was chosen for its fast development speed and interfaces to high-level web as well as low-level socket and event programming. Python also allows us to move parts of the application into native C code if the need should ever arise. Python is also available on a large number of platforms, including Android[and] and iOS[pyo]. Tornado[tord] serves both as a web server and asynchronous platform in general.

As a templating language, we use the platform-independent mustache[mus]. This allows us to reuse the templates should we ever switch the implementation language, but more importantly, allows future client-side applications ($\rightarrow$ 3.4.4) to reuse the templating code. The rest of the client-side code is written in JavaScript, relying on the jQuery[jqu] and underscore.js[und] libraries.

The only available application is a policy drafting solution, which is based on an extremely simple document database, which in turn just uses a simple filesystem-backed CAS. The policy-drafting system supports proposals under revision control (where revision numbers are simply the wall-clock time of creation) that can be commented upon. Figure 4.1 and 4.2 show some screenshots. Apart from the main application and associated configuration options, the prototype also implements a simple web fallback ($\rightarrow$ 3.4.1).

The source code of the implementation is available on a CD which accompagnies the written thesis. The source code will also be published onto http://github.com/phihag/d2p and other online repositories.

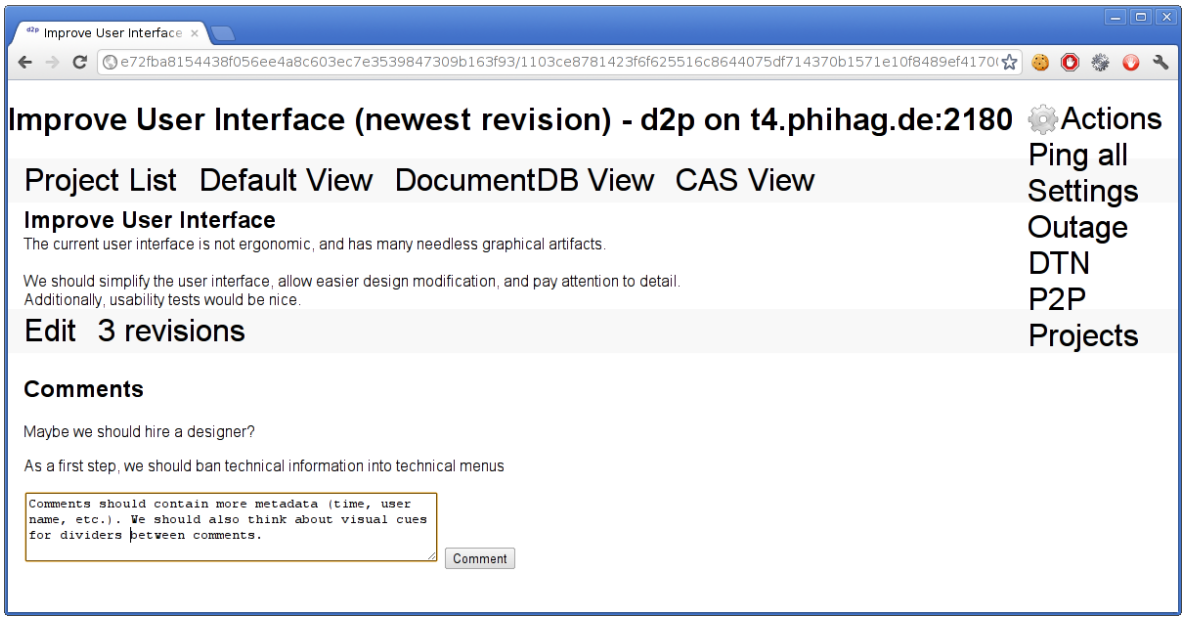


Figure 4.1: Screenshot of the policy drafting application.

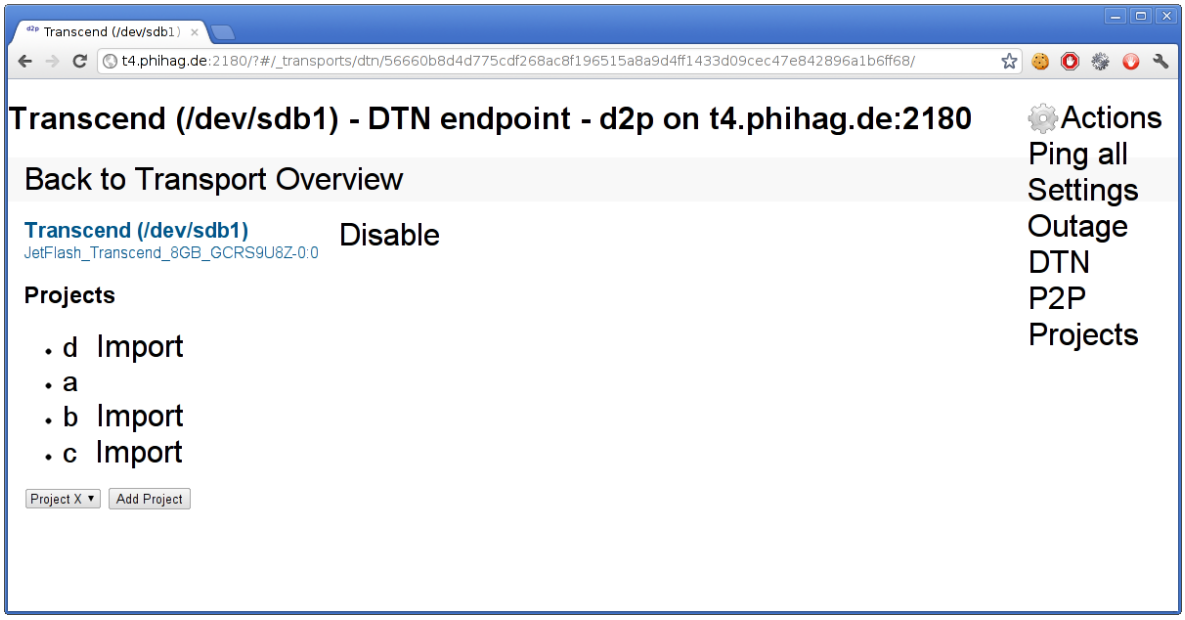


Figure 4.2: Screenshot of the configuration of a DTN endpoint.

Chapter 5

Conclusion

This thesis describes the framework for a distributed censorship-resistant policy drafting system. By relying on a DTN as well as a P2P network, the system can work even without internet access.

These and other attacks are described and evaluated in a detailed **threat model** which includes non-technical attacks. The threat model also contains outlines of defense against the various attacks as well as an historical overview.

The various components of the proposed system are discussed in detail. Apart from the basics and current implementations of **DTN** and **P2P** networks, this includes a discussion of censorship-resistant **anonymization networks**, particularly hidden services that allow P2P services over these anonymization networks. Additionally, the challenges of **revision control** and **security** in distributed delay-tolerant networks need to be researched.

Finally, the **architecture** assembles these components into one policy drafting system. It allows applications to interact over P2P, DTN, anonymization networks via an extremely simple protocol. The architecture also accounts for the integration of third-party centralized systems. Furthermore, the chapter presents concrete solutions for various security and performance problems. The web-based nature of the system means that it will be extremely portable. Nevertheless, modern web standards are used to allow for automatical fallback to alternative servers from the web application.

The thesis comes with a prototype implementation of the designed system, including the web fallback.

5.1 Future Work

The prototype application is extremely basic. Its implementation, and partially its design should be extended in a number of areas, in particular:

5.1.1 General

The threat model may need to be updated as new attacks evolve and technological realities (for example the proliferation of cheap small computers) change. Additionally, it would be useful to have a formal attack tree in order to simulate an attacker's response.

Also, the security would be greatly enhanced if we could test the application in simulators, and develop attack toolkits and packet sniffer decoding modules for it.

The implementation should automatically reload itself if it is in developer mode and a change in one of the source code files is detected.

The `ProjectListService` should be implemented; we should try to generate a useful list of projects even in the presence of numerous projects, spammers, and/or attackers.

5.1.2 P2P

The current P2P network is unstructured; it should be supplemented by structured network implementation. Additionally, further research is necessary to harden the network against sybil attacks and improve its broadcast throughput and reliability. Opportunistic asymmetric encryption and SSL support for servers should both be considered.

Also, additional bootstrap mechanisms should be conceived and implemented in order to aggravate blocking of the vulnerable bootstrapping process. In particular, an IP multicast bootstrap would enable usage of the system in a LAN without Internet connection[1].

The integration of obsproxy[JA12] or a similar solution may be a good idea in order to avoid broad blocks of encryption content.

Finally, NAT traversal and other related anti-firewall techniques like STUN are vital to allow everyone to connect to the P2P network, and should therefore be implemented.

[1] The technical name of the Covert Café Constellation.

5.1.3 DTN

The DTN storage format assumes a mounted mass storage device. The integration should be improved to the point that the user just needs to click once, and then drag-and-drop projects to a device icon. Additionally, (deniable) encryption and steganography should be integrated in a user-friendly way.

Multi-hop DTNs should be considered, and if need be implemented. The system should also run on existing DTN protocols, in particular Bundle and Licklider.

5.1.4 Security

The implementation, and to a lesser degree the design, currently assumes single-user nodes. Multiple users should be supported and be able to authenticate. Special attention should be paid so that the web-based fallback works.

The public key infrastructure integration needs to be developed both in design as well as implementation. It should integrate with commonly used protocols such as OpenPGP[2], and support the German identification card as well as other authentication tokens.

When the public key infrastructure is in place, the authorization restriction features (→ 3.2.7, 3.2.8) should be implemented and tested for vulnerabilities. The same goes for voting and rating features.

5.1.5 Version Control

The current implementation contains a very simple document database and CAS. Both should be extended as described in chapter 2.5.3 and 3.2.4, in particular with delta encoding and efficient synchronization protocols.

We should also evaluate the other options of using revision control systems described in 2.5.

5.1.6 Anonymization Networks & Transports

The anonymization networks described in chapter 2.4 should be integrated as transports.

[2] and make use of Evgeni Golov's work there

Import/exports and/or full-fledged transports should be designed and written for other policydrafting platforms such as adhocracy.

The steganographic secure storage on twitter, flickr, facebook described in [BFV10] should be integrated and implemented.

5.1.7 Web Application

The web server fallback should be extended and improved. We should consider client-side authentification instead of SSL when connecting to a fallback host.

The URL design of the web application should be improved in order to facilitate caching. However, it must not interfere with the Offline Web Applications feature.

The offline fallback mode, which stores the state of projects on the client, and can synchronize these changes once the connection is restored, should be designed and implemented.[3]

The ”futuristic” offline client-side application should be considered. With this application, the Python server just relays P2P broadcasts, whereas the client-side application actually implements application and network logic. Browser-to-Browser P2P connections with WebRTC should be evaluated, and if possible implemented.

The web application should be supported and tested on a wide array of platforms, including mobile and limited ones.

5.1.8 User Interface

The current design and usability reflects the prototype status. Both should be significantly improved before presenting the application to ”real” users.

The UI also needs various features, including comments which refer to a singular line or paragraph [4], a WYSIWIG and image editor, notifications on changes, import/export from/to office documents, and private messages.

[3] Currently being designed and written by Tim van Cleef.
[4] Currently being designed and developed by Julius Römmler.

Bibliography

[AA08] Y. Akdeniz and K. Altiparmak. Internet: Restricted access a critical assessment of internet content regulation and censorship in turkey, 2008.

[AAL+05] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose. DNS Security Introduction and Requirements. RFC 4033 (Proposed Standard), March 2005.

[adh] adhocracy. `http://trac.adhocracy.de/`.

[Adk11] Heather Adkins. An update on attempted man-in-the-middle attacks. `http://googleonlinesecurity.blogspot.com/2011/08/update-on-attempted-man-in-middle.html`, 2011.

[Alt11] Florian Altherr. #servergate - polizei beschlagnahmt piratenpartei-server, 2011.

[and] android-scripting. `http://code.google.com/p/android-scripting/`.

[AR97] A. Abdul-Rahman. The pgp trust model. In *EDI-Forum: the Journal of Electronic Commerce*, volume 10, pages 27–31, 1997.

[AW09] I.F. Akyildiz and X. Wang. *Wireless mesh networks*, volume 1. John Wiley & Sons Inc, 2009.

[Bak07] C.E. Baker. *Media concentration and democracy: Why ownership matters*. Cambridge Univ Pr, 2007.

[baz] Bazaar. `http://bazaar.canonical.com/en/`.

[Bel68] D.A. Bell. *Information theory and its engineering applications*. Pitman Publishing, 1968.

[BFV10] S. Burnett, N. Feamster, and S. Vempala. Chipping away at censorship firewalls with user-generated content. In *Proc. 19th USENIX Security Symposium, Washington, DC*, 2010.

[BGH+02] K. Bennett, C. Grothoff, T. Horozov, I. Patrascu, and T. Stef. Gnunet-a truly anonymous networking infrastructure. In *In: Proc. Privacy Enhancing Technologies Workshop (PET*. Citeseer, 2002.

[BHT+03] S. Burleigh, A. Hooke, L. Torgerson, K. Fall, V. Cerf, B. Durst, K. Scott, and H. Weiss. Delay-tolerant networking: an approach to interplanetary internet. *Communications Magazine, IEEE*, 41(6):128–136, 2003.

[Bra11] Magnus Brading. Generic, decentralized, unstoppable anonymity: The phantom protocol. 2011.

[BRF08] S. Burleigh, M. Ramadas, and S. Farrell. Licklider Transmission Protocol - Motivation. RFC 5325 (Informational), September 2008.

[BS05] F. Brandt and T. Sandholm. Decentralized voting with unconditional privacy. In *Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*, pages 357–364. ACM, 2005.

[BY07] P. Barford and V. Yegneswaran. An inside look at botnets. *Malware Detection*, pages 171–191, 2007.

[CBH+07] V. Cerf, S. Burleigh, A. Hooke, L. Torgerson, R. Durst, K. Scott, K. Fall, and H. Weiss. Delay-Tolerant Networking Architecture. RFC 4838 (Informational), April 2007.

[Cha81] David L. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Commun. ACM*, 24(2):84–90, February 1981.

[Che11] S. Cheshire. Multicast dns. `http://files.multicastdns.org/draft-cheshire-dnsext-multicastdns.txt`, 2011.

[cjd] cjdns whitepaper. `https://raw.github.com/cjdelisle/cjdns/master/rfcs/Whitepaper.md`.

[Clu11] Chaos Computer Club. Chaos computer club analyzes government malware, 2011.

[con] Convergence. `http://convergence.io/details.html`.

[Con02] Global Internet Freedom Consortium. The great firewall revealed. `http://www.internetfreedom.org/files/WhitePaper/ChinaGreatFirewallRevealed.pdf`, 2002.

[cou] Apache couchdb: Technical overview. `http://couchdb.apache.org/docs/overview.html`.

[Cow11a] James Cowie. Egypt returns to the internet, 2011.

[Cow11b] James Cowie. Libyan disconnect, 2011.

[CSWH01] I. Clarke, O. Sandberg, B. Wiley, and T. Hong. Freenet: A distributed anonymous information storage and retrieval system. In *Designing Privacy Enhancing Technologies*, pages 46–66. Springer, 2001.

[dar] darcs. `http://darcs.net/`.

[DG08] C.G. Dickey and C. Grothoff. Bootstrapping of peer-to-peer networks. In *Applications and the Internet, 2008. SAINT 2008. International Symposium on*, pages 205–208. IEEE, 2008.

[DMS04] R. Dingledine, N. Mathewson, and P. Syverson. Tor: The second-generation onion router. In *Proceedings of the 13th conference on USENIX Security Symposium-Volume 13*, pages 21–21. USENIX Association, 2004.

[DR08] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246 (Proposed Standard), August 2008.

[DSA+11] A. Dainotti, C. Squarcella, E. Aben, K.C. Claffy, M. Chiesa, M. Russo, and A. Pescapé. Analysis of country-wide internet outages caused by censorship. In *Proc. of IMC*, 2011.

[EAABH03] S. El-Ansary, L. Alima, P. Brand, and S. Haridi. Efficient broadcast in structured p2p networks. *Peer-to-Peer Systems II*, pages 304–314, 2003.

[ech] echo - living democracy. `http://echo.to/en/`.

[Eck10] P. Eckersley. How unique is your web browser? In *Privacy Enhancing Technologies*, pages 1–18. Springer, 2010.

[Fer12] Alfredo Fernandez. Flash drives are cuba's internet. 2012.

[FGM+99] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. RFC 2616 (Draft Standard), June 1999. Updated by RFC 2817.

[FIS11] F. FISCHER. *AMNESTY INTERNATIONAL REPORT 2011*. FISCHER VERLAG, 2011.

[FRB08] S. Farrell, M. Ramadas, and S. Burleigh. Licklider Transmission Protocol - Security Extensions. RFC 5327 (Experimental), September 2008.

[Ful07] T. Fuller. Thailand blocks users' access to youtube. *The New York Times*, 12, 2007.

[geo] Ssl für unternehmen: Georoot. `http://www.geotrust.com/de/enterprise-ssl-certificates/georoot/`.

[git] git. `http://git-scm.com/`.

[Gol12] Evgeni Golov. Pki-basiertes identitätsmanagement für adhocracy, 2012.

[Haa11] Janine Haas. Versionskontrolle in verzögerungstoleranten netzwerken, 2011.

[HCS+05] P. Hui, A. Chaintreau, J. Scott, R. Gass, J. Crowcroft, and C. Diot. Pocket switched networks and human mobility in conference environments. In *Proceedings of the 2005 ACM SIGCOMM workshop on Delay-tolerant networking*, pages 244–251. ACM, 2005.

[HCY08] P. Hui, J. Crowcroft, and E. Yoneki. Bubble rap: social-based forwarding in delay tolerant networks. In *Proceedings of the 9th ACM international symposium on Mobile ad hoc networking and computing*, pages 241–250. ACM, 2008.

[HGRF08] T. Holz, C. Gorecki, K. Rieck, and F.C. Freiling. Measuring and detecting fast-flux service networks. In *Proceedings of the Network & Distributed System Security Symposium*, 2008.

[HH10] D. Hyatt and I. Hickson. Html5. *World Wide Web Consortium WD WD-html5-20100304, March*, 2010.

[Hic09] Ian Hickson. The websockets api. `http://www.w3.org/TR/2009/WD-websockets-20091029/`, 2009.

[Hic11] Ian Hickson. Webstorage. `http://www.w3.org/TR/2011/WD-webstorage-20110208/`, 2011.

[HKL+07] T. Hyyryläinen, T. Kärkkäinen, C. Luo, V. Jaspertas, J. Karvo, and J. Ott. Opportunistic email distribution and access in challenged heterogeneous environments. In *Proceedings of the second ACM workshop on Challenged networks*, pages 97–100. ACM, 2007.

[i2pa] I2p netdb statistics. `http://stats.i2p/cgi-bin/total_routers_year.cgi`.

[i2pb] I2p: Ssu. `http://www.i2p2.de/udp.html`.

[int] Introducing i2p. `http://www.i2p2.de/techintro.html`.

[Ipp05] B. Ippolito. Remote json-jsonp, 2005.

[ira11] Iran blocks tor; tor releases same-day fix. `https://blog.torproject.org/blog/iran-blocks-tor-tor-releases-same-day-fix`, 2011.

[ira12] Iran partially blocks encrypted network traffic. https://blog.torproject.org/blog/iran-partially-blocks-encrypted-network-traffic, 2012.

[JA12] Nick Mathewson Jacob Appelbaum. Tor project: Pluggable transports for circumvention. `https://gitweb.torproject.org/torspec.git/blob/HEAD:/proposals/180-pluggable-transport.txt`, 2012.

[Jac09] J. Jacobson. A formalization of darcs patch theory using inverse semigroups. Technical report, Technical Report CAM report 09-83, UCLA, 2009.

[JNA08] D. Johnson, N. Ntlatlapa, and C. Aichele. Simple pragmatic approach to mesh routing using batman. 2008.

[jqu] jquery. `http://jquery.com/`.

[jso] Json. `http://json.org`.

[KCU11] S. Kelly, S. Cook, and Freedom House (U.S.). *Freedom on the net 2011: a global assessment of internet and digital media*. Freedom House, 2011.

[KHKK10] C. Kolbitsch, T. Holz, C. Kruegel, and E. Kirda. Inspector gadget: Automated extraction of proprietary gadgets from malware binaries. In *Security and Privacy (SP), 2010 IEEE Symposium on*, pages 29–44. IEEE, 2010.

[KOK09] A. Keränen, J. Ott, and T. Kärkkäinen. The one simulator for dtn protocol evaluation. In *Proceedings of the 2nd International Conference on Simulation Tools and Techniques*, page 55. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2009.

[Küg10] D. Kügler. Advanced security mechanisms for machine readable travel documents. Technical report, Technical report, Federal office for information security (BSI), Versjon 2.05, 2010.

[KZH07] A. Kate, G.M. Zaverucha, and U. Hengartner. Anonymity and security in delay tolerant networks. In *Security and Privacy in Communications Networks and the Workshops, 2007. SecureComm 2007. Third International Conference on*, pages 504–513. IEEE, 2007.

[Lan09] A. Langley. Opportunistic encryption everywhere. In *In W2SP*, 2009.

[LDS03] A. Lindgren, A. Doria, and O. Schelén. Probabilistic routing in intermittently connected networks. *ACM SIGMOBILE Mobile Computing and Communications Review*, 7(3):19–20, 2003.

[Lea11a] Patrick Leahy. Preventing real online 5 threats to economic creativity and theft of intellectual 6 property act of 2011. `http://leahy.senate.gov/imo/media/doc/BillText-PROTECTIPAct.pdf`, 2011.

[Lea11b] N. Leavitt. Internet security under attack: The undermining of digital certificates. *Computer*, pages 17–20, 2011.

[LGL+96] M. Leech, M. Ganis, Y. Lee, R. Kuris, D. Koblas, and L. Jones. SOCKS Protocol Version 5. RFC 1928 (Proposed Standard), March 1996.

[Lin11] Yang Lina. Beijing requires real names in microblog registration, december 2011.

[liq] Liquid feedback - project. `http://liquidfeedback.org/project/`.

[LZC10] Q. Li, S. Zhu, and G. Cao. Routing in socially selfish delay tolerant networks. In *INFOCOM, 2010 Proceedings IEEE*, pages 1–9. IEEE, 2010.

[Mac05] R. MacKinnon. Chinese cell phone breaches north korean hermit kingdom. *YaleGlobal Online*, 17, 2005.

[mer] Mercurial scm. `http://mercurial.selenic.com/`.

[Mic11] Microsoft. Windows 8 system requirements. `http://msdn.microsoft.com/library/windows/hardware/hh748188`, 2011.

[MM02] P. Maymounkov and D. Mazieres. Kademlia: A peer-to-peer information system based on the xor metric. *Peer-to-Peer Systems*, pages 53–65, 2002.

[Mur10] Patrick Murherjee. A fully decentralized, peer-to-peer based version control system, 2010.

[mus] mustache. `http://mustache.github.com/`.

[Nak09] S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system. 2009.

[NCC08] RIPE NCC. Youtube hijacking: A ripe ncc ris case study, 2008.

[NH08] J. Nazario and T. Holz. As the net churns: Fast-flux botnet observations. In *Malicious and Unwanted Software, 2008. MALWARE 2008. 3rd International Conference on*, pages 24–31. Ieee, 2008.

[NNR09] L. Nussbaum, P. Neyron, and O. Richard. On robust covert channels inside dns. *Emerging Challenges for Security, Privacy and Trust*, pages 51–62, 2009.

[Nor] ISO Norm. Iec 7498-1: 1994 (e). *Information technology–Open Systems Interconnection–Basic Reference Model: The Basic Model*.

[OK06] J. Ott and D. Kutscher. Bundling the web: Http over dtn. *Proceedings of WNEPT*, 2006.

[OKD06] J. Ott, D. Kutscher, and C. Dwertmann. Integrating dtn and manet routing. In *Proceedings of the 2006 SIGCOMM workshop on Challenged networks*, pages 221–228. ACM, 2006.

[olp] One laptop per child project. `http://one.laptop.org/`.

[oni11] Opennet filtering data, 2011.

[opea] Connecting to an openvpn server via an http proxy. `http://openvpn.net/index.php/open-source/documentation/howto.html#http`.

[opeb] Opennet: Dns tampering articles. `http://opennet.net/filtering-types/dns-tampering`.

[pro11] Probabilistic routing protocol for intermittently connected networks. `http://tools.ietf.org/html/draft-irtf-dtnrg-prophet-09`, 2011.

[Pud11] A. Puddington. *Freedom in the World 2011: The Annual Survey of Political Rights & Civil Liberties*. Freedom in the World. ROWMAN & LITTLEFIELD, 2011.

[PWC03] V.N. Padmanabhan, H.J. Wang, and P.A. Chou. Resilient peer-to-peer streaming. In *Network Protocols, 2003. Proceedings. 11th IEEE International Conference on*, pages 16–27. IEEE, 2003.

[pyo] Pyobjc. `http://pyobjc.sourceforge.net/`.

[Res00] E. Rescorla. HTTP Over TLS. RFC 2818 (Informational), May 2000.

[Rip01] M. Ripeanu. Peer-to-peer architecture case study: Gnutella network. In *Peer-to-Peer Computing, 2001. Proceedings. First International Conference on*, pages 99–100. IEEE, 2001.

[RMMW08] J. Rosenberg, R. Mahy, P. Matthews, and D. Wing. Session Traversal Utilities for NAT (STUN). RFC 5389 (Proposed Standard), October 2008.

[Ros08] J. Rosen. Google's gatekeepers. *New York Times Magazine*, 28, 2008.

[RSB+08] C. Rigano, K. Scott, J. Bush, R. Edell, S. Parikh, R. Wade, and B. Adamson. Mitigating naval network instabilities with disruption toler. In *Military Communications Conference, 2008. MILCOM 2008. IEEE*, pages 1–7. IEEE, 2008.

[RSG98] M.G. Reed, P.F. Syverson, and D.M. Goldschlag. Anonymous connections and onion routing. *Selected Areas in Communications, IEEE Journal on*, 16(4):482–494, 1998.

[SB07] K. Scott and S. Burleigh. Bundle Protocol Specification. RFC 5050 (Experimental), November 2007.

[SFWL11] S. Symington, S. Farrell, H. Weiss, and P. Lovell. Bundle Security Protocol Specification. RFC 6257 (Experimental), May 2011.

[SKZ+06] A. Seth, D. Kroeker, M. Zaharia, S. Guo, and S. Keshav. Low-cost communication for rural internet kiosks using mechanical backhaul. In *Proceedings of the 12th annual international conference on Mobile computing and networking*, pages 334–345. ACM, 2006.

[Smi11] Lamar Smith. Stop online piracy act. `http://thomas.loc.gov/cgi-bin/bdquery/z?d112:h.r.03261:`, 2011.

[SMK+01] I. Stoica, R. Morris, D. Karger, M.F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. *ACM SIGCOMM Computer Communication Review*, 31(4):149–160, 2001.

[SR06] D. Stutzbach and R. Rejaie. Understanding churn in peer-to-peer networks. In *Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*, pages 189–202. ACM, 2006.

[sub] Apache subversion. `http://subversion.apache.org/`.

[Tal04] G. Taleck. Synscan: Towards complete tcp/ip fingerprinting. *CanSecWest, Vancouver BC, Canada*, 2004.

[Ten09] G. Tenebro. W32. waledac threat analysis. *Symantec Whitepaper*, 2009.

[tora] China blocking tor: Round two. `https://blog.torproject.org/blog/china-blocking-tor-round-two`.

[torb] Tor metrics portal: Network. `http://metrics.torproject.org/network.html`.

[torc] Tor project: Bridges. `https://www.torproject.org/docs/bridges`.

[tord] Tornado web server. `http://www.tornadoweb.org/`.

[tru] Clarifying the trustwave ca policy update. `http://blog.spiderlabs.com/2012/02/clarifying-the-trustwave-ca-policy-update.html`.

[uef11] Uefi specification 2.3.1, september 2011.

[Uhr11] Eric Uhrhane. File api: Directories and system. `http://www.w3.org/TR/file-system-api`, 2011.

[und] underscore.js. `https://github.com/documentcloud/underscore/`.

[VYF06] V. Venkataraman, K. Yoshida, and P. Francis. Chunkyspread: Heterogeneous unstructured tree-based peer-to-peer multicast. In *Network Protocols, 2006. ICNP'06. Proceedings of the 2006 14th IEEE International Conference on*, pages 2–11. IEEE, 2006.

[Wai90] D. Waitzman. Standard for the transmission of IP datagrams on avian carriers. RFC 1149 (Experimental), April 1990. Updated by RFC 2549.

[web12] Webrtc 1.0: Real-time communication between browsers. 2012.

[WH11] L. Wood and P. Holliday. Using http for delivery in delay/disruption-tolerant networks. 2011.

[Wil12] Tim Wilde. Knock knock knockin' on bridges' doors. `https://blog.torproject.org/blog/knock-knock-knockin-bridges-doors`, 2012.

[WKP] N. Weaver, C. Kreibich, and V. Paxson. Redirecting dns for ads and profit.

[WWGH11] E. Wustrow, S. Wolchok, I. Goldberg, and J.A. Halderman. Telex: Anticensorship in the network infrastructure. In *proceedings of the 20th USENIX Security Symposium*, 2011.

[WWT08] Y. Wang, H. Wu, and N.F. Tzeng. Cross-layer protocol design and optimization for delay/fault-tolerant mobile sensor networks (dft-msn's). *Selected Areas in Communications, IEEE Journal on*, 26(5):809–819, 2008.

[XMH11] X. Xu, Z. Mao, and J. Halderman. Internet censorship in china: where does the filtering occur? In *Passive and Active Measurement*, pages 133–142. Springer, 2011.

[YCM06] A. Yip, B. Chen, and R. Morris. Pastwatch: A distributed version control system. In *Proceedings of the 3rd conference on Networked Systems Design & Implementation*, volume 3, 2006.

[Zal10] M. Zalewski. Browser security handbook. *Google Code*, 2010.

[Zov11] Dino A. Dai Zovi. Apple ios 4 security evaluation, 2011.

[zug11] Bundesgesetzblatt jahrgang 2010 teil i nr. 6, 2011.

Ehrenwörtliche Erklärung

Hiermit versichere ich, die vorliegende Masterarbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt zu haben. Alle Stellen, die aus den Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Düsseldorf, 12. März 2012 Philipp Hagemeister