Tor Development Roadmap: Wishlist for Nov 2006–Dec 2007

Roger Dingledine Nick Mathewson Shava Nerad

October 24, 2006

1 Introduction

Hi, Roger! Hi, Shava. This paragraph should get deleted soon. Right now, this document goes into about as much detail as I'd like to go into for a technical audience, since that's the audience I know best. It doesn't have time estimates everywhere. It isn't well prioritized, and it doesn't distinguish well between things that need lots of research and things that don't. The breakdowns don't all make sense. There are lots of things where I don't make it clear how they fit into larger goals, and lots of larger goals that don't break down into little things. It isn't all stuff we can do for sure, and it isn't even all stuff we can do for sure in 2007. The tmp{} macro indicates stuff I haven't said enough about. That said, here goes...

Tor (the software) and Tor (the overall software/network/support/document suite) are now experiencing all the crises of success. Over the next year, we're probably going to grow more in terms of users, developers, and funding than before. This gives us the opportunity to perform long-neglected maintenance tasks.

2 Code and design infrastructure

2.1 Protocol revision

To maintain backward compatibility, we've postponed major protocol changes and redesigns for a long time. Because of this, there are a number of sensible revisions we've been putting off until we could deploy several of them at once. To do each of these, we first need to discuss design alternatives with other cryptographers and outside collaborators to make sure that our choices are secure.

First of all, our protocol needs better **versioning support** so that we can make backward-incompatible changes to our core protocol. There are difficult anonymity issues here, since many naive designs would make it easy to tell clients apart (and then track them) based on their supported versions.

With protocol versioning support would come the ability to **future-proof our ciphersuites**. For example, not only our OR protocol, but also our directory protocol, is pretty firmly tied to the SHA-1 hash function, which though not yet known to be insecure for our purposes, has begun to show its age. We should remove assumptions thoughout our design based on the assumption that public keys, secret keys, or digests will remain any particular size indefinitely.

A new protocol could support **multiple cell sizes**. Right now, all data passes through the Tor network divided into 512-byte cells. This is efficient for high-bandwidth protocols, but inefficient for protocols like SSH or AIM that send information in small chunks. Of course, we need to investigate the extent to which multiple sizes could make it easier for an adversary to fingerprint a traffic pattern.

Our OR **authentication protocol**, though provably secure[**?**], relies more on particular aspects of RSA and our implementation thereof than we had initially believed. To future-proof against changes, we should replace it with a less delicate approach.

We might design a **stream migration** feature so that streams tunneled over Tor could be more resilient to dropped connections and changed IPs.

As a part of our design, we should investigate possible **cipher modes** other than counter mode. For example, a mode with built-in integrity checking, error propagation, and random access could simplify our protocol significantly. Sadly, many of these are patented and unavailable for us.

2.2 Scalability

2.2.1 Improved directory efficiency

Right now, clients download a statement of the **network status** made by each directory authority. We could reduce network bandwidth significantly by having the authorities jointly sign a statement reflecting their vote on the current network status. This would save clients up to 160K per hour, and make their view of the network more uniform. Of course, we'd need to make sure the voting process was secure and resilient to failures in the network.

We should **shorten router descriptors**, since the current format includes a great deal of information that's only of interest to the directory authorities, and not of interest to clients. We can do this by having each router upload a short-form and a long-form signed descriptor, and having clients download only the short form. Even a naive version of this would save about 40% of the bandwidth currently spent by clients downloading descriptors.

We should **have routers upload their descriptors even less often**, so that clients do not need to download replacements every 18 hours whether any information has changed or not. (As of Tor 0.1.2.3-alpha, clients tolerate routers that don't upload often, but routers still upload at least every 18 hours to support older clients.)

2.2.2 Non-clique topology

Our current network design achieves a certain amount of its anonymity by making clients act like each other through the simple expedient of making sure that all clients know all servers, and that any server can talk to any other server. But as the number of servers increases to serve an ever-greater number of clients, these assumptions become impractical.

At worst, if these scalability issues become troubling before a solution is found, we can design and build a solution to **split the network into multiple slices** until a better solution comes along. This is not ideal, since rather than looking like all other users from a point of view of path selection, users would "only" look like 200,000–300,000 other users.

We are in the process of designing **improved schemes for network scalability**. Some approaches focus on limiting what an adversary can know about what a user knows; others focus on reducing the extent to which an adversary can exploit this knowledge. These are currently in their infancy, and will probably not be needed in 2007, but they must be designed in 2007 if they are to be deployed in 2008.

2.2.3 Relay incentives

We need incentives to relay. [.....]

2.3 Portability

Our **Windows implementation**, though much improved, continues to lag behind Unix and Mac OS X, especially when running as a server. We hope to merge promising patches from Mike Chiussi to address this point, and bring Windows performance on par with other platforms.

We should have **better support for portable devices**, including modes of operation that require less RAM, and that write to disk less frequently (to avoid wearing out flash RAM).

2.4 Performance: resource usage

We've been working on **using less RAM**, especially on servers. This has paid off a lot for directory caches in the 0.1.2, which in some cases are using 90% less memory than they used to require. But we can do better, especially in the area around our buffer management algorithms, by using an approach more like the BSD and Linux kernels use instead of our current ring buffer approach. (For OR connections, we can just use queues of cell-sized chunks produced with a specialized allocator.) This could potentially save around 25 to 50% of the memory currently allocated for network buffers, and make Tor a more attractive proposition for restricted-memory environments like old computers, mobile devices, and the like.

We should improve our **bandwidth limiting**. The current system has been crucial in making users willing to run servers: nobody is willing to run a server if it might use an unbounded amount of bandwidth, especially if they are charged for their usage. We can make our system better by letting users configure bandwidth limits independently for their own traffic and traffic relayed for others; and by adding write limits for users running directory servers.

On many hosts, sockets are still in short supply, and will be until we can migrate our protocol to UDP. We can **use fewer sockets** by making our self-to-self connections happen internally to the code rather than involving the operating system's socket implementation.

2.5 Performance: network usage

We know too little about how well our current path selection algorithms actually spread traffic around the network in practice. We should **research the efficacy of our traffic allocation** and either assure ourselves that it is close enough to optimal as to need no improvement (unlikely) or **identify ways to improve network usage**, and get more users' traffic delivered faster. Performing this research will require careful thought about anonymity implications.

We should also **examine the efficacy of our congestion control algorithm**, and see whether we can improve client performance in the presence of a congested network through dynamic 'sendme' window sizes or other means. This will have anonymity implications too if we aren't careful.

2.6 Performance scenario: one Tor client, many users

We should **improve Tor's performance when a single Tor handles many clients**. Many organizations want to manage a single Tor client on their firewall for many users, rather than having each user install a separate Tor client. We haven't optimized for this scenario, and it is likely that there are some code paths in the current implementation that become inefficient when a single Tor is servicing hundreds or thousands of client connections. (Additionally, it is likely that such clients have interesting anonymity requirements the we should investigate.) We should profile Tor under appropriate loads, identify bottlenecks, and fix them.

2.7 Tor servers on asymmetric bandwidth

Roger, please write? I don't know what to say here. [.....]

2.8 Running Tor as both client and server

many performance tradeoffs and balances that need more attention. Roger, please write. [.....]

2.9 Protocol redesign for UDP

Tor has relayed only TCP traffic since its first versions, and has used TLS-over-TCP to do so. This approach has proved reliable and flexible, but in the long term we will need to allow UDP traffic on the network, and switch some or all of the network to using a UDP transport. **Supporting UDP traffic** will make Tor more suitable for protocols that require UDP, such as many VOIP protocols. **Using a UDP transport** could greatly reduce resource limitations on servers, and make the network far less interruptable by lossy connections. Either of these protocol changes would require a great deal of design work, however. We hope to be able to enlist the aid of a few talented graduate students to assist with the initial design and specification, but the actual implementation will require significant testing of different reliable transport approaches.

3 Blocking resistance

3.1 Design for blocking resistance

We have written a design document explaining our general approach to blocking resistance. We should workshop it with other experts in the field to get their ideas about how we can improve Tor's efficacy as an anti-censorship tool.

3.2 Implementation: client-side and bridges-side

Our anticensorship design calls for some nodes to act as "bridges" that can circumvent a national firewall, and others inside the firewall to act as pure clients. This part of the design is quite clear-cut; we're probably ready to begin implementing it. To implement bridges, we need only to have servers publish themselves as limited-availability relays to a special bridge authority if they judge they'd make good servers. Clients need a flexible interface to learn about bridges and to act on knowledge of bridges.

Clients also need to **use the encrypted directory variant** added in Tor 0.1.2.3-alpha. This will let them retrieve directory information over Tor once they've got their initial bridges.

Bridges will want to be able to **listen on multiple addresses and ports** if they can, to give the adversary more ports to block.

Additionally, we should **resist content-based filters**. Though an adversary can't see what users are saying, some aspects of our protocol are easy to fingerprint *as* Tor. We should correct this where possible.

3.3 Implementation: bridge authorities

The design here is also reasonably clear-cut: we need to run some directory authorities with a slightly modified protocol that doesn't leak the entire list of bridges. Thus users can learn up-to-date information for bridges they already know about, but they can't learn about arbitrary new bridges.

3.4 Implementation: how users discover bridges

Our design anticipates an arms race between discovery methods and censors. We need to begin the infrastructure on our side quickly, preferably in a flexible language like Python, so we can adapt quickly to censorship.

3.5 Resisting censorship of the Tor website, docs, and mirrors

We should take some effort to consider **initial distribution of Tor and related information** in countries where the Tor website and mirrors are censored. (Right now, most countries that block access to Tor block only the main website and leave mirrors and the network itself untouched.) Falling back on word-of-mouth is always a good last resort, but we should also take steps to make sure it's relatively easy for users to get ahold of a copy.

4 Security

4.1 Security research projects

We should investigate approaches with some promise to help Tor resist end-to-end traffic correlation attacks. It's an open research question whether (and to what extent) **mixed-latency** networks, **low-volume long-distance padding**, or other approaches can resist these attacks, which are currently some of the most effective against careful Tor users. We should research these questions and perform simulations to identify opportunities for strengthening our design without dropping performance to unacceptable levels.

We've got some preliminary results suggesting that **a topology-aware routing algorithm** [**?**] could reduce Tor users' vulnerability against local or ISP-level adversaries, by ensuring that they are never in a position to watch both ends of a connection. We need to examine the effects of this approach in more detail and consider side-effects on anonymity against other kinds of adversaries. If the approach still looks promising, we should investigate ways for clients to implement it (or an approximation of it) without having to download routing tables for the whole internet.

We should research the efficacy of **website fingperprinting** attacks, wherein an adversary tries to match the distinctive traffic and timing pattern of the resources constituting a given website to the traffic pattern of a user's client. These attacks work great in simulations, but in practice we hear they don't work nearly as well. We should get some actual numbers to investigte the issue, and figure out what's going on. If we resist these attacks, or can improve our design to resist them, we should.

4.2 Implementation security

Right now, each Tor node stores its keys unencrypted. We should **encrypt more Tor keys** so that Tor authorities can require a startup password. We should look into adding intermediary medium-term "signing keys" between identity keys and onion keys, so that a password could be required to replace a signing key, but not to start Tor. This would improve Tor's long-term security, especially in its directory authority infrastructure.

We should also **mark RAM that holds key material as non-swappable** so that there is no risk of recovering key material from a hard disk compromise. This would require submitting patches upstream to OpenSSL, where support for marking memory as sensitive is currently in a very preliminary state.

There are numerous tools for identifying trouble spots in code (such as Coverity or even VS2005's code analysis tool) and we should convince somebody to run some of them against the Tor codebase. Ideally, we could figure out a way to get our code checked periodically rather than just once.

We should try **protocol fuzzing** to identify errors in our implementation.

Our guard nodes help prevent an attacker from being able to become a chosen client's entry point by having each client choose a few favorite entry points as "guards" and stick to them. We should implement a **directory guards** feature to keep adversaries from enumerating Tor users by acting as a directory cache.

4.3 Detect corrupt exits and other servers

With the success of our network, we've attracted servers in many locations, operated by many kinds of people. Unfortunately, some of these locations have compromised or defective networks, and some of these people are untrustworthy or incompetent. Our current design relies on authority administrators to identify bad nodes and mark them as nonfunctioning. We should **automate the process of identifying malfunctioning nodes** as follows:

We should create a generic **feedback mechanism for add-on tools** like Mike Perry's "Snakes on a Tor" to report failing nodes to authorities.

We should write tools to **detect more kinds of innocent node failure**, such as nodes whose network providers intercept SSL, nodes whose network providers censor popular websites, and so on. We should also try to detect **routers that snoop traffic**; we could do this by launching connections to throwaway accounts, and seeing which accounts get used.

We should add **an efficient way for authorities to mark a set of servers as probably collaborating** though not necessarily otherwise dishonest. This happens when an administrator starts multiple routers, but doesn't mark them as belonging to the same family.

To avoid attacks where an adversary claims good performance in order to attract traffic, we should **have authorities measure node performance** (including stability and bandwidth) themselves, and not simply believe what they're told. Measuring bandwidth can be tricky, since it's hard to distinguish

between a server with low capacity, and a high-capacity server with most of its capacity in use.

Operating a directory authority should be easier. We rely on authority operators to keep the network running well, but right now their job involves too much busywork and administrative overhead. A better interface for them to use could free their time to work on exception cases rather than on adding named nodes to the network.

4.4 Protocol security

In addition to other protocol changes discussed above, we should add **hooks for denial-of-service resistance**; we have some prelimiary designs, but we shouldn't postpone them until we realy need them. If somebody tries a DDoS attack against the Tor network, we won't want to wait for all the servers and clients to upgrade to a new version.

5 Development infrastructure

5.1 Build farm

We've begun to deploy a cross-platform distributed build farm of hosts that build and test the Tor source every time it changes in our development repository.

We need to **get more participants**, so that we can test a larger variety of platforms. (Previously, we've only found out when our code had broken on obscure platforms when somebody got around to building it.)

We need also to **add our dependencies** to the build farm, so that we can ensure that libraries we need (especially libevent) do not stop working on any important platform between one release and the next.

5.2 Improved testing harness

Currently, our **unit tests** cover only about XX% of the code base. This is uncomfortably low; we should write more and switch to a more flexible testing framework.

We should also write flexible **automated single-host deployment tests** so we can more easily verify that the current codebase works with the network.

We should build automated **stress testing** frameworks so we can see which realistic loads cause Tor to perform badly, and regularly profile Tor against these loads. This would give us *in vitro* performance values to supplement our deployment experience.

5.3 Centralized build system

We currently rely on a separate packager to maintain the packaging system and to build Tor on each platform for which we distribute binaries. Separate package maintainers is sensible, but separate package builders has meant long

turnaround times between source releases and package releases. We should create the necessary infrastructure for us to produce binaries for all major packages within an hour or so of source release.

5.4 Improved metrics

We'd like to know how the network is doing. [.....]

We'd like to know where users are in an even less intrusive way. [.....]

We'd like to know how much of the network is getting used. [.....]

5.5 Controller library

We've done lots of design and development on our controller interface, which allows UI applications and other tools to interact with Tor. We could encourage the development of more such tools by releasing a **general-purpose controller library**, ideally with API support for several popular programming languages.

6 User experience

6.1 Get blocked less, get blocked less broadly

Right now, some services block connections from the Tor network because they don't have a better way to keep vandals from abusing them than blocking IP addresses associated with vandalism. Our approach so far has been to educate them about better solutions that currently exist, but we should also **create better solutions for limiting vandalism by anonymous users** like credential and blind-signature based implementations, and encourage their use. Other promising starting points including writing a patch and explanation for Wikipedia, and helping Freenode to document, maintain, and expand its current Tor-friendly position.

Those who do block Tor users also block overbroadly, sometimes blacklisting operators of Tor servers that do not permit exit to their services. We could obviate innocent reasons for doing so by designing a **narrowly-targeted Tor RBL service** so that those who wanted to overblock Tor clould no longer plead incompetence.

6.2 All-in-one bundle

a.k.a "Torpedo", but rename this. [.....]

6.3 LiveCD Tor

a.k.a anonym.os done right [.....]

6.4 A Tor client in a VM

a.k.a JanusVM [.....]
which is quite related to the firewall-level deployment section below

6.5 Interface improvements

Allow controllers to manipulate server status. [.....]

6.6 Firewall-level deployment

Another useful deployment mode for some users is using **Tor in a firewall configuration**, and directing all their traffic through Tor. This can be a little tricky to set up currently, but it's an effective way to make sure no traffic leaves the host un-anonymized. To achieve this, we need to **improve and port our new TransPort** feature which allows Tor to be used without SOCKS support; to **add an anonymizing DNS proxy** feature to Tor; and to **construct a recommended set of firewall configurations** to redirect traffic to Tor.

This is an area where **deployment via a livecd**, or an installation targetted at specialized home routing hardware, could be useful.

6.7 Assess software and configurations for anonymity risks

which firefox extensions to use, and which to avoid. best practices for how to torify each class of application. [.....]

clean up our own bundled software: E.g. Merge the good features of Foxtor into Torbutton [.....]

6.8 Localization

Right now, most of our user-facing code is internationalized. We need to internationalize the last few hold-outs (like the Tor installer), and get more translations for the parts that are already internationalized.

Also, we should look into a **unified translator's solution**. Currently, since different tools have been internationalized using the framework-appropriate method, different tools require translators to localize them via different interfaces. Inasmuch as possible, we should make translators only need to use a single tool to translate the whole Tor suite.

7 Support

would be nice to set up some actual user support infrastructure, especially focusing on server operators and on coordinating volunteers. [.....]

8 Documentation

8.1 Unified documentation scheme

We need to **inventory our documentation.** Our documentation so far has been mostly produced on an *ad hoc* basis, in response to particular needs and requests. We should figure out what documentation we have, which of it (if any) should get priority, and whether we can't put it all into a single format.

We could **unify the docs** into a single book-like thing. This will also help us identify what sections of the "book" are missing.

8.2 Missing technical documentation

We should **revise our design paper** to reflect the new decisions and research we've made since it was published in 2004. This will help other researchers evaluate and suggest improvements to Tor's current design.

Other projects sometimes implement the client side of our prototocol. We encourage this, but we should write **a document about how to avoid excessive resource use**, so we don't need to worry that they will do so without regard to the effect of their choices on server resources.

8.3 Missing user documentation

Discoursive and comprehensive docs [.....]