Tor Development Roadmap: Wishlist for Nov 2006–Dec 2007

Roger Dingledine Nick Mathewson Shava Nerad

October 22, 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 very 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. 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

In order 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 cryptographers and other outside collaborators in order 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 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 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 infinitely.

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 very 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.

2.2 Scalability

2.2.1 Improved directory performance

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 on 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

Use less RAM when we have very little. Make buffer code smarter [.....]

Use less RAM when we have very little. Make buffer code smarter [.....]

2.5 Performance: network usage

2.6 Blue-sky: UDP

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. The design here 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.

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

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.

4 Security

4.1 Security research projects

Mixed-latency [.....]

long-distance padding [.....]

router-zones [.....]

defenses against end-to-end correlation [.....]
We don't expect any to work right now, but it would be very useful to learn that one did. Alternatively, proving that one didn't would free up researchers in the field to go work on other things.

4.2 Implementation security

Encrypt more keys [.....]

Talk Coverity or somebody with a copy of vs2005 into running tools on our code [.....]

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.

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 very 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.

6 User experience

6.1 All-in-one bundle

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

6.2 LiveCD Tor

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

6.3 Interface improvements

6.4 Firewall-level deployment

6.5 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 Documentation

7.1 Unified documentation scheme

Keep track of all the docs we've got [.....]

Unify the docs into a single book-like thing [.....]
This will also help us identify what sections of the "book" are missing.