Performance Improvements on Tor
or,
Why Tor is slow and what we're going to do about it

Roger Dingledine Steven J. Murdoch

March 11, 2009

As Tor's user base has grown, the performance of the Tor network has suffered. This document describes our current understanding of why Tor is slow, and lays out our options for fixing it.

Over the past few years, our funding (and thus our development effort) has focused on usability and blocking-resistance. We've come up with a portable self-contained Windows bundle; deployed tools to handle the upcoming censorship arms race; further developed supporting applications like Vidalia, Torbutton, and Thandy; made it easier for users to be relays by adding better rate limiting and an easy graphical interface with uPnP support; developed an effective translation and localization team and infrastructure; and spread understanding of Tor in a safe word-of-mouth way that stayed mostly under the radar of censors.

In parallel to adding these features, we've also been laying the groundwork for performance improvements. We've been working with academics to write research papers on improving Tor's speed, funding some academic groups directly to come up with prototypes, and thinking hard about how to safely collect metrics about network performance. But it's becoming increasingly clear that we're not going to produce the perfect answers just by thinking hard. We need to roll out some attempts at solutions, and use the experience to get better intuition about how to really solve the problems.

We've identified six main reasons why the Tor network is slow. Problem #1 is that Tor's congestion control does not work well. We need to come up with ways to let "quiet" streams like web browsing co-exist better with "loud" streams like bulk transfer. Problem #2 is that some Tor users simply put too much traffic onto the network relative to the amount they contribute, so we need to work on ways to limit the effects of those users and/or provide priority to the other users. Problem #3 is that the Tor network simply doesn't have enough capacity to handle all the users that want privacy on the Internet. We need to develop strategies for increasing the overall community of relays, and consider introducing incentives to make the network more self-sustaining. Problem #4 is that Tor's current path selection algorithms don't actually distribute load correctly over the network, meaning some relays are overloaded and some are underloaded. We need to develop ways to more accurately estimate the properties of each relay, and also ways for clients to select paths more fairly. Problem #5 is that Tor clients aren't as good as they should be at handling high or variable latency and connection failures. We need better heuristics for clients to automatically shift away from bad circuits, and other tricks for them to dynamically adapt their behavior. Problem #6 is that low-bandwidth users spend too much of their network overhead downloading directory information. We've made a serious dent in this problem already, but more work remains here too.

We discuss each reason more in its own section below. For each section, we explain our current intuition for how to address the problem, how effective we think each fix would be, how much effort and risk is involved, and the recommended next steps, all with an eye to what can be accomplished in 2009.

While all six categories need to be resolved in order to make the Tor network fast enough to handle everyone who wants to use it, we've ordered the sections by precedence. That is, solving the earlier sections will be necessary before we can see benefits from solving the later sections.

Contents

1 Tor's congestion control does not work well

One of Tor's critical performance problems is in how it combines high-volume streams with low-volume streams. We need to come up with ways to let the "quiet" streams (like web browsing) co-exist better with the "loud" streams (like bulk transfer).

1.1 TCP backoff slows down every circuit at once

Tor combines all the circuits going between two Tor relays into a single TCP connection. This approach is a smart idea in terms of anonymity, since putting all circuits on the same connection prevents an observer from learning which packets correspond to which circuit. But over the past year, research has shown that it's a bad idea in terms of performance, since TCP's backoff mechanism only has one option when that connection is sending too many bytes: slow it down, and thus slow down all the circuits going across it.

We could fix this problem by switching to a design with one circuit per TCP connection. But that means that a relay with 1000 connections and 1000 circuits per connection would need a million sockets open. That number is a problem for even the well-designed operating systems and routers out there.

More generally, Tor currently uses two levels of congestion avoidance – TCP flow control per-link, and a simple windowing scheme per-circuit. It has been suggested that this approach is causing performance problems, because the two schemes interact badly.

Experiments show that moving congestion management to be fully end-to-end offers a significant improvement in performance.

There have been two proposals to resolve this problem, but their underlying principle is the same: use an unreliable protocol for links between Tor relays, and perform error recovery and congestion management between the client and exit relay. Tor partially funded Joel Reardon's thesis [13] under Ian Goldberg. His thesis proposed using DTLS [14] (a UDP variant of TLS) as the link protocol and a cut-down version of TCP to give reliability and congestion avoidance, but largely using the existing Tor cell protocol. Csaba Kiraly *et al.* [3] proposed using IPSec [1] to replace the entire Tor cell and link protocol.

Each approach has its own strengths and weaknesses. DTLS is relatively immature, and Reardon noted deficiencies in the OpenSSL implementation of the protocol. However, the largest missing piece from this proposal is a high-quality, privacy preserving TCP stack, under a compatible license. Prior work has shown that there is a substantial privacy leak from TCP stack and clockskew fingerprinting [4, 8]. Therefore to adopt this proposal, Tor would need to incorporate a TCP stack, modified to operate in user-mode and to not leak identity information.

Reardon built a prototype around the TCP-Daytona stack [12], developed at IBM Labs, and based on the Linux kernel TCP stack. This implementation is not publicly available and its license is unclear, so it is unlikely to be suitable for use in Tor. Writing a TCP stack from scratch is a substantial undertaking, and therefore other attempts have been to move different operating system stacks into user-space. While there have been some prototypes, the maturity of these systems have yet to be shown.

Kiraly *et al.* rely on the operating system IPsec stack, and a modification to the IKE key exchange protocol to support onion routing. As with the proposal from Reardon, there is a risk of operating system and machine fingerprinting from exposing the client TCP stack to the exit relay. This could be resolved in a similar way, by implementing a user-mode IPsec stack, but this would be a substantial effort, and would lose some of the advantages of making use of existing building blocks.

Prof. Goldberg has a new student named Chris Alexander picking up where Joel left off. He's currently working on fixing bugs in OpenSSL's implementation of DTLS along with other core libraries that we'd need to use if we go this direction.

Impact: High.

Effort: High effort to get all the pieces in place.

Risk: High risk that it would need further work to get right.

Plan: We should keep working with them (and help fund Chris) to get this project closer to something we can deploy. The next step on our side is to deploy a separate testing Tor network that uses datagram protocols, based on patches from Joel and others, and get more intuition from that. We could optimistically have this testbed network deployed in late 2009.

1.2 We chose Tor's congestion control window sizes wrong

Tor maintains a per-circuit maximum of unacknowledged cells (`CIRCWINDOW`). If this value is exceeded, it is assumed that the circuit has become congested, and so the originator stops sending. Kiraly proposed [2, 3] that reducing this window size would substantially decrease latency (although not to the same extent as moving to an unreliable link protocol), while not affecting throughput.

Specifically, right now the circuit window size is 512KB and the per-stream window size is 256KB. These numbers mean that a user downloading a large file receives it (in the ideal case) in chunks of 256KB, sending back acknowledgements for each chunk. In practice, though, the network has too many of these chunks moving around at once, so they spend most of their time waiting in buffers at relays.

Reducing the size of these chunks has several effects. First, we reduce memory usage at the relays, because there are fewer chunks waiting and because they're smaller. Second, because there are fewer bytes vying to get onto the network at each hop, users should see lower latency.

More investigation is needed on precisely what should be the new value for the circuit window, and whether it should vary. Out of 100KB, 512KB (current value in Tor) and 2560KB, they found the optimum was 100KB for all levels of packet loss. However this was only evaluated for a fixed network latency and relay bandwidth, where all users had the same `CIRCWINDOW` value. Therefore, a different optimum may exist for networks with different characteristics, and during the transition of the network to the new value.

Impact: Medium. It seems pretty clear that in the steady-state this patch is a good idea; but it's still up in the air whether the transition period will show immediate improvement or if there will be a period where traffic from people who upgrade get clobbered by traffic from people who haven't upgraded yet.

Effort: Low effort to deploy – it's a several line patch!

Risk: Medium risk that we haven't thought things through well enough and we'd need to back it out or change parts of it.

Plan: Once we start on Tor 0.2.2.x (in the next few months), we should put the patch in and see how it fares. We should go for maximum effect, and choose the lowest possible window setting of 100 cells (50KB).

2 Some users add way too much load

Section 1 described mechanisms to let low-volume streams have a chance at competing with high-volume streams. Without those mechanisms, normal web browsing users will always get squeezed out by people pulling down larger content and tolerating high latency. But the next problem is that some users simply add more load than the network can handle. Just making sure that all the load gets handled fairly isn't enough if there's too much load in the first place.

When we originally designed Tor, we aimed for high throughput. We figured that providing high throughput would mean we get good latency properties for free. However, now that it's clear we have several user profiles trying to use the Tor network at once, we need to consider changing some of those design choices. Some of those changes would aim for better latency and worse throughput.

2.1 Squeeze over-active circuits

The Tor 0.2.0.30 release included this change:

```
- Change the way that Tor buffers data that it is waiting to write.
  Instead of queueing data cells in an enormous ring buffer for each
```

```
client->relay or relay->relay connection, we now queue cells on a
separate queue for each circuit. This lets us use less slack memory,
and will eventually let us be smarter about prioritizing different
kinds of traffic.
```

Currently when we're picking cells to write onto the network, we choose round-robin from each circuit that wants to write. We could instead remember which circuits have written many cells recently, and give priority to the ones that haven't.

Technically speaking, we're reinventing more of TCP here, and we'd be better served by a general switch to DTLS+UDP. But there are two reasons to still consider this separate approach.

The first is rapid deployment. We could get this change into the Tor 0.2.2.x development release in mid 2009, and as relays upgrade, the change would gradually phase in. This timeframe is way earlier than the practical timeframe for switching to DTLS+UDP.

The second reason is the flexibility this approach provides. We could give priorities based on recent activity ("if you've sent much more than the average in the past 10 seconds, then you get slowed down"), or we could base it on the total number of bytes sent on the circuit so far, or some combination. Even once we switch to DTLS+UDP, we may still want to be able to enforce some per-circuit quality-of-service properties.

This meddling is tricky though: we could encounter feedback effects if we don't perfectly anticipate the results of our changes. For example, we might end up squeezing certain classes of circuits too far, causing those clients to build too many new circuits in response. Or we might simply squeeze all circuits too much, ruining the network for everybody.

Also, Bittorrent is designed to resist attacks like this – it periodically drops its lowest-performing connection and replaces it with a new one. So we would want to make sure we're not going to accidentally increase the number of circuit creation requests and thus just shift the load problem.

Impact: High, if we get it right.

Effort: Medium effort to deploy – we need to go look at the code to figure out where to change, how to efficiently keep stats on which circuits are active, etc.

Risk: High risk that we'd get it wrong the first few times. Also, it will be hard to measure whether we've gotten it right or wrong.

Plan: Step one is to evaluate the complexity of changing the current code. We should do that for Tor 0.2.2.x in mid 2009. Then we should write some proposals for various meddling we could do, and try to find the right balance between simplicity (easy to code, easy to analyze) and projected effect.

2.2 Throttle certain protocols at exits

If we're right that Bittorrent traffic is a main reason for Tor's load, we could bundle a protocol analyzer with the exit relays. When they detect that a given outgoing stream is a protocol associated with bulk transfer, they could set a low rate limit on that stream. (Tor already supports per-stream rate limiting, though we've never found a need for it.)

This is a slippery slope in many respects though. First is the wiretapping question: is an application that automatically looks at traffic content wiretapping? It depends which lawyer you ask. Second is the network neutrality question: remember Comcast's famous "we're just delaying the traffic" quote. Third is the liability concern: once we add this feature in, what other requests are we going to get for throttling or blocking certain content? And does the capability to throttle certain content change the liability situation for the relay operator?

Impact: Medium-high.

Effort: Medium effort to deploy: need to find the right protocol recognition tools and sort out how to bundle them.

Risk: This isn't really an arms race we want to play. The "encrypted bittorrent" community already has a leg up since they've been fighting this battle with the telco's already. Plus the other downsides.

Plan: Not a good move.

2.3 Throttle certain protocols at the client side

While throttling certain protocols at the exit side introduces wiretapping and liability problems, detecting them at the client side is more straightforward. We could teach Tor clients to detect protocols as they come in on the socks port, and automatically treat them differently – and even pop up an explanation box if we like.

This approach opens a new can of worms though: clients could disable the "feature" and resume overloading the network.

Impact: Medium-high.

Effort: Medium effort to deploy: need to find the right protocol recognition tools and sort out how to bundle them.

Risk: This isn't really an arms race we want to play either. Users who want to file-share over Tor will find a way. Encouraging people to fork a new "fast" version of Tor is not a good way to keep all sides happy.

Plan: Not a good move.

2.4 Throttle all streams at the client side

While we shouldn't try to identify particular protocols as evil, we could set stricter rate limiting on client streams by default. If we set a low steady-state rate with a high bucket size (e.g. allow spikes up to 250KB but enforce a long-term rate for all streams of 5KB/s), we would probably provide similar performance to what clients get now, and it's possible we could alleviate quite a bit of the congestion and then get even better and more consistent performance.

Plus, we could make the defaults higher if you sign up as a relay and pass your reachability test.

The first problem is: how should we choose the numbers? So far we have avoided picking absolute speed numbers for this sort of situation, because we won't be able to predict a number now which will still be the correct number in the future.

The second problem is the same as in the previous subsection – users could modify their clients to disable these checks. So we would want to do this step only if we also put in throttling at the exits or intermediate relays, a la Section 2.1. And if that throttling works, changing clients (and hoping they don't revert the changes) may be unnecessary.

Impact: Low at first, but medium-high later.

Effort: Low effort to deploy.

Risk: If we pick high numbers, we'll never see much of an impact. If we pick low numbers, we could accidentally choke users too much.

Plan: It's not crazy, but may be redundant. We should consider in Tor 0.2.2.x whether to do it, in conjunction with throttling at other points in the circuit.

2.5 Default exit policy of 80,443

We hear periodically from relay operators who had problems with DMCA takedown attempts, switched to an exit policy of "permit only ports 80 and 443", and no longer hear DMCA complaints.

Does that mean that most file-sharing attempts go over some other port? If only a few exit relays permitted ports other than 80 and 443, we would effectively squeeze the high-volume flows onto those few exit relays, reducing the total amount of load on the network.

First, there's a clear downside: we lose out on other protocols. Part of the point of Tor is to be application-neutral. Also, it's not clear that it would work long-term, since corporate firewalls are continuing to push more and more of the Internet onto port 80.

To be clearer, we have more options here than the two extremes. We could switch the default exit policy from allow-all-but-these-20-ports to accept-only-these-20-ports. We could even get more complex, for example by applying per-stream rate limiting at the exit relays to streams destined for certain ports.

Impact: Low? Medium? High?

Effort: Low effort to deploy.

Risk: The Tor network becomes less useful, roughly in proportion to the amount of speedup we get.

Plan: I think we should take some of these steps in the Tor 0.2.2.x timeframe. The big challenge here is that we don't have much intuition about how effective the changes should be, so we don't know how far to go.

2.6 Better user education

We still run across users who think any anonymity system out there must have been designed with file-sharing in mind. If we make it clearer in the FAQ and our webpage that Tor isn't for high-volume streams, that might combine well with the other approaches above.

Overall, the challenge of users who want to overload the system will continue. Tor is not the only system that faces this challenge.

3 The Tor network doesn't have enough capacity

Section 1 aims to let web browsing connections work better in the face of high-volume streams, and Section 2 aims to reduce the overall load on the network. The third reason why Tor is slow is that we simply don't have enough capacity in the network to handle all the users who want to use Tor.

Why do we call this the third problem rather than the number one problem? Just adding more capacity to the network isn't going to solve the performance problem. If we add more capacity without solving the issues with high-volume streams, then those high-volume streams will expand to use up whatever new capacity we add.

Economics tells us to expect that improving performance in the Tor network (i.e. increasing supply) means that more users will arrive to fill the void. So in either case we shouldn't be under the illusion that Tor will magically just become faster once we implement these improvements. We place the first two sections higher in priority because their goals are to limit the ability of the high-volume users to become even higher-volume users, thus allowing the new capacity to be more useful to the other users. We discuss the supply-vs-demand question more in Section 7.1.

3.1 Tor server advocacy

Encouraging more volunteers to run Tor servers, and existing volunteers to keep their servers running, will increase network capacity and hence performance.

Impact: High, assuming we work on the plans from Section 1 and Section 2 also.

Effort: Medium to high, depending on how much we put in.

Risk: Low.

Plan: A clear win. We should do as many advocacy aspects as we can fit in.

3.1.1 Talks and trainings

One of the best ways we've found for getting new relays is to go to conferences and talk to people in person. There are many thousands of people out there with spare fast network connections and a willingness to help save the world. Our experience is that visiting them in person produces much better results, long-term, than Slashdot articles.

Roger and Jake have been working on this angle, and Jake will be ramping up even more on it in 2009.

Advocacy and education is especially important in the context of new and quickly-changing government policies. In particular, the data retention question in Germany is causing instability in the overall set of volunteers willing to run relays. Karsten's latest metrics[1] show that while the number of relays in other countries held steady or went up during 2008, the numbers in Germany went down over the course of 2008. On the other hand, the total amount of bandwidth provided by German relays held steady during 2008 – so while other operators picked up the slack, we still lost overall diversity of relays. These results tell us where to focus our efforts.

3.1.2 Better support for relay operators

Getting somebody to set up a relay is one thing; getting them to keep it up is another thing entirely. We lose relays when the operator reboots and forgets to set up the relay to start on boot. We lose relays when the operator looks through the website and doesn't find the answer to a question.

We've been working on a new service for relay operators called Tor Weather[2]. The idea is that once you've set up your relay, you can subscribe to get an email whenever it goes down. We need to work on the interface more, for example to let people subscribe to various levels of notification, but the basic idea seems like a very useful one.

With Tor Weather you can also subscribe to watch somebody *else*'s relay; so this service should tie in well for the people doing advocacy, to let them focus their follow-ups when a relay they helped set up disappears.

We are also considering setting up a mailing list exclusively for relay operators, to give them a better sense of community, to answer questions and concerns more quickly, etc.

We should also consider offering paid or subsidized support options so relay operators have a place to go for help. Corporations and universities running relays could get direct phone, email, or IM support options.

3.1.3 A Facebook app to show off your relay

We're currently developing a Facebook application that will allow relay operators to link their Tor relays to their Facebook profile. Volunteers who desire can therefore publicly get credit for their contribution to the Tor network. This would raise awareness for Tor, and encourage others to operate relays.

Opportunities for expansion include allowing relay operators to form "teams", and for these teams to be ranked on the contribution to the network. (Real world examples here include the SETI screensaver and the MD5 hash crack challenges.) This competition may give more encouragement for team members to increase their contribution to the network. Also, when one of the team members has their relay fail, other team members may notice and provide assistance on fixing the problem.

3.1.4 Look for new ways to get people to run relays

We are not primarily social engineers, and the people that we are good at convincing to set up relays are not a very huge group.

We need to keep an eye out for more creative ways to encourage a broader class of users to realize that helping out by operating a relay will ultimately be something they want to do.

3.2 Funding more relays directly

Another option is to directly pay hosting fees for fast relays (or to directly sponsor others to run them).

The main problem with this approach is that the efficiency is low: at even cheap hosting rates, the cost of a significant number of new relays grows quickly. For example, if we can find 100 non-exit relays providing

[1] https://www.torproject.org/projects/metrics
[2] https://weather.torproject.org/

1MB/s for as low as $100/mo (and at that price it'd be renting space on a shared server, with all the resource sharing hassles that comes with), that's $120k per year. Figure some more for maintenance and coordination, the overhead to find 100 locations that are on sufficiently different networks and administrative zones, etc.

The amount of work involved in running them as exit relays might be a few times this cost, due to higher hosting fees, more effort involved in establishing and maintaining the right relationships, having lawyers nearby, etc.

Plus the costs just keep coming, month after month.

Overall, it seems more sustainable to invest in better code, and community outreach and education.

Impact: Medium.

Effort: High.

Risk: Low.

Plan: If we end up with extra funding, sure. Otherwise, I think our time and effort are better spent on design and coding that will have long-term impact rather than be recurring costs.

3.3 Handling fast Tor relays on Windows

Advocating that users set up relays is all well and good, but if most users are on Windows, and Tor doesn't support fast relays on Windows well, then we're in a bad position.

Nick has been adapting libevent so it can handle a buffer-based abstraction rather than the traditional Unix-style socket-based abstraction. Then we will modify Tor to use this new abstraction. Nick's blog post[3] provides more detail.

Impact: Medium.

Effort: High, but we're already halfway through.

Risk: Low.

Plan: Keep at it. We're on schedule to get a test version (one that works for Nick) out in September 2009. Then iterate until it works for everybody.

3.4 Relay scanning to find overloaded relays or broken exits

Part of the reason that Tor is slow is because some of the relays are advertising more bandwidth than they can realistically handle. These anomalies might be due to bad load balancing on the part of the Tor designers, bad rate limiting or flaky network connectivity on the part of the relay operator, or malicious intent. Similarly, some exit relays might fail to give back the 'real' content, requiring users to repeat their connection attempts.

Mike has been working on tools to identify these relays: SpeedRacer[4] and SoaT[5]. Once the tools are further refined, we should be able to figure out if there are general classes of problems (load balancing, common usability problems, etc) that mean we should modify our design to compensate. The end goal is to get our tools to the point where they can automatically tell the directory authorities to leave out certain misbehaving relays in the network status consensus, and/or adjust the bandwidths they advertise for each relay.

Impact: Low.

Effort: Medium.

Risk: Low.

Plan: Keep at it. We're on schedule to get a test version (that works for Mike) out in mid 2009. Then iterate until it works for everybody.

[3] https://blog.torproject.org/blog/some-notes-progress-iocp-and-libevent

[4] https://svn.torproject.org/svn/torflow/trunk/README.PerfMeasurements

[5] https://svn.torproject.org/svn/torflow/trunk/NetworkScanners/README.ExitScanning

3.5 Getting dynamic-IP relays back into the relay list quickly

Currently there is a delay of 2-5 hours between when a relay changes its IP address and when that relay gets used again by clients. This delay causes two problems: relays on dynamic IP addresses will be underutilized (contributing less to the total network capacity than they could), and clients waste time connecting to relay IP addresses that are no longer listening.

There are several approaches that can mitigate this problem by notifying clients sooner about IP address changes. The first approach is to continue on our path of simplifying directory information (see Section 6.1): if we can put out "diffs" of the network status more often than once an hour, clients can get updated quicker. A second approach is for each relay to estimate how volatile its IP address is, and advertise this in its descriptor. Clients then ignore relays with volatile IP addresses and old descriptor. Similarly, directory authorities could prioritise the distribution of updated IP addresses for freshly changed relays.

As a last note here, we currently have some bugs that are causing relays with dynamic IP addresses to fall out of the network entirely. If a third to half of the relays are running on dynamic IP addresses, that's really bad.

Impact: Low-medium.

Effort: Low-medium.

Risk: Low.

Plan: Track down and fix bugs for Tor 0.2.2.x. Continue simplifying directory information so we can get new info to clients quicker.

3.6 Incentives to relay

Our blog post on this topic[6] explains our work to-date on this topic. The current situation is that we have two designs to consider: one that's quite simple but has a serious anonymity problem, and one that's quite complex.

I think we should move forward with the first (simple but flawed) design. There are several pieces to moving it forward. The first phase is changing Tor's queueing mechanisms to be able to give some circuits priority over others. This step also ties into the other development items in this document regarding cell-, circuit-, and connection-priorities. The second phase is then redesigning the "gold star" mechanism so the priority earned by relays lasts long enough that there's a sufficient anonymity set for them. We'll need to look at current and projected network metrics to discover a good upper bound on relay churn. The question to answer is: "What period of time, taken as a rolling snapshot of which relays are present in the network, guarantees a sufficiently large anonymity set for high-priority relays?" Hopefully the answer is something like 7 or 14 days. There are other missing pieces in there, like "what do we mean by sufficiently?", that we'll just have to guess about. The third phase is to actually sort out how to construct and distribute gold-star cryptographic certificates that entry relays can verify.

Notice that with the new certificates approach, we can reward users who contribute to the network in other ways than running a fast public relay – examples might include top sponsors, users who run stable bridge relays, translators, people who fix bugs, etc.

Impact: Medium-high.

Effort: Medium-high.

Risk: Medium-high: if we screw up the balance of our community-oriented infrastructure, we might end up hurting more than we help.

Plan: Accomplishing the three phases above will put us in a much better position to decide whether to deploy this idea. At the same time, the more complex options might become more within reach as other research teams investigate and refine them, so we should keep an eye on them too.

[6] https://blog.torproject.org/blog/two-incentive-designs-tor

3.7 Reachable clients become relays automatically

Even if we don't add in an incentive scheme, simply making suitable users into relays by default should do a lot for our capacity problems.

We've made many steps toward this goal already, with automated reachability testing, bandwidth estimation, UPnP support built in to Vidalia, and so on.

There are a few risks here though. First, relaying traffic could introduce anonymity vulnerabilities, and we need to learn more about that first. (That's on the roadmap for 2009.) Second, making clients into relays by default could make some users upset. Third, this approach could change how sysadmins view Tor. By putting ourselves into the same category as Skype, we would scale up the "blocking Tor connections" arms race by a level that's hard to predict. Also, we need to finish deployment of Section 3.3 before we can roll this out, or we'll just make a bunch of Windows machines crash.

We had originally been avoiding the "everybody a relay" design until we had a better plan for scaling the directory to be able to distribute tens of thousands of relay addresses. I think these two plans are not as related as we first thought, though. For example, a very simple answer for what to do if we get more relays than our current directory scheme can handle is to publish only the best relays, for some metric of best that considers capacity, expected uptime, etc. That should be a perfectly adequate stopgap measure. The step after that would be to consider splintering the network into two networkstatus documents, and clients flip a coin to decide which they use. Ultimately, if we are so lucky as to get into the position of having too many relays, we'll want to look at the distribution and properties of the relays we have when deciding what algorithms would best make use of them.

Impact: High.

Effort: Medium, now that we've done a lot of hard work already.

Risk: Medium.

Plan: Wrap up our investigations into the anonymity implications of being a relay, at the same time as working on a plan for exactly how the Tor client should decide if it's suitable for elevation to relay status. This is going to happen, it's just a matter of how soon.

4 Tor clients choose paths imperfectly

Even when we sort out the congestion control issues, the problem of users abusing the network with too much traffic, and the question of overall capacity, we still face a fourth problem. Users need to choose their paths in such a way that everybody is using the network efficiently.

Right now, Tor relays estimate their capacity by observing the largest traffic burst they've seen themselves do in the past day. They advertise that bandwidth capacity in the directory information, and clients weight their path selection by the bandwidth of each relay. For example, a relay that advertises 100KB/s peak bandwidth will be chosen twice as often as a relay that advertises 50KB/s peak bandwidth.

There are several problems with our current algorithm that are worth fixing.

4.1 We don't balance traffic over our bandwidth numbers correctly

Selecting relays with a probability proportional to their bandwidth contribution to the network may not be the optimal algorithm. Murdoch and Watson [10] investigated the performance impact of different relay selection algorithms, and came up with a model to describe the optimal path selection strategies based on how loaded the network is.

Tor's current selection strategy is optimal when the network is fully loaded. That is, if every single byte is going to be used, then weighting by capacity is the right way to do it. But if the network is not fully loaded, then the fast relays end up with less load than the slow relays. To compensate, clients should pick faster relays with higher probability.

In particular, we can estimate the network load because all Tor relays publish both their capacity and usage in their relay descriptor (but see Section 4.2 for problems that crop up there). The Tor network is currently loaded at around 50%. This level is much higher than most reasonable networks, indicating that our plan in Section 3 to get more overall capacity is a good one. But 50% is quite far from 100% when it becomes to optimal load balancing.

To find the optimum relay selection probabilities the model, Steven used a hill-climbing algorithm to minimize network latency, with a Tor directory snapshot as input. The results (shown in Figure 1 and Figure 2) depend on the network load relative to overall capacity. As load approaches capacity, the optimum selection probabilities converge to the one currently used by Tor: relay bandwidth proportional to network capacity. However, as load drops, the optimized selection algorithm favors slow relays less and faster relays more; many relays are not used at all.

Anecdotal evidence supports the theory that the fast relays in the Tor network have more spare capacity than they should. Several users have posted that they get much better Tor performance if they hard-code their paths to only use the fastest ten relays (and ignore the huge anonymity implications, of course).

The relay selection probabilities in these graphs are tuned to a particular level of network load. Figure 3 shows how average network latency is affected by relay selection probabilities, for different levels of network load. For all load levels examined, the optimized selection probabilities offer lower latency when compared to Tor's current selection algorithm. However, there's a downside to tailoring for a particular load level: if we see a much heavier load in practice than the one we had in mind when we tuned our selection biases, then we end up overbalancing the network in the other direction.

Specifically, each probability distribution has a cut-off point at which (according to the model) at least one relay will have a higher load than its capacity, at which its queue length, and hence latency, will become infinite. For the optimized selection probability distributions, this cut-off point is a few percent above the level they were designed to operate at. For Tor's current selection algorithm, it is when the overall network capacity equals the overall network load.

In this respect the Tor selection algorithm reaches the theoretical optimum, as no network can operate at greater than 100% utilization while maintaining finite latency. However, in a real instantiation of any of these alternative probability distributions, the network latency would not become infinite; instead a connection would time out and a different circuit would be selected. So in practice, if the wrong probability distribution was selected, the network would converge at a different one. Unfortunately the standard queuing theory models cannot handle this case; we need to move to a simulation rather than using equations and assumptions, to estimate the real effect.

Impact: Low-medium.

Effort: Medium, since we still need to get a better sense of the correct network load to expect, and we need to experiment to see if the model actually matches reality.

Risk: Low, since we can always back out the changes.

Plan: It seems clear that some adjustments should be done in terms of biasing selection toward the faster relays. The exact load level to anticipate remains an open question though. Fortunately, in our new networkstatus algorithm, the directory authorities declare the bandwidths for each relay. So we can just reweight them on the fly and clients will use the new numbers. That means once enough clients have upgraded to using the bandwidths specified in the networkstatus, we can start to experiment with shifting the biases and see what results we get.

4.2 The bandwidth estimates we have aren't very accurate

Weighting relay selection by bandwidth only works if we can accurately estimate the bandwidth for each relay.

Snader and Borisov [15] examined three strategies for estimating the bandwidth for each relay. The first strategy was Tor's current approach of looking for peaks in the actual bytes it's handled in the past day. The second strategy was active probing by the directory authorities. For their third strategy, they proposed that

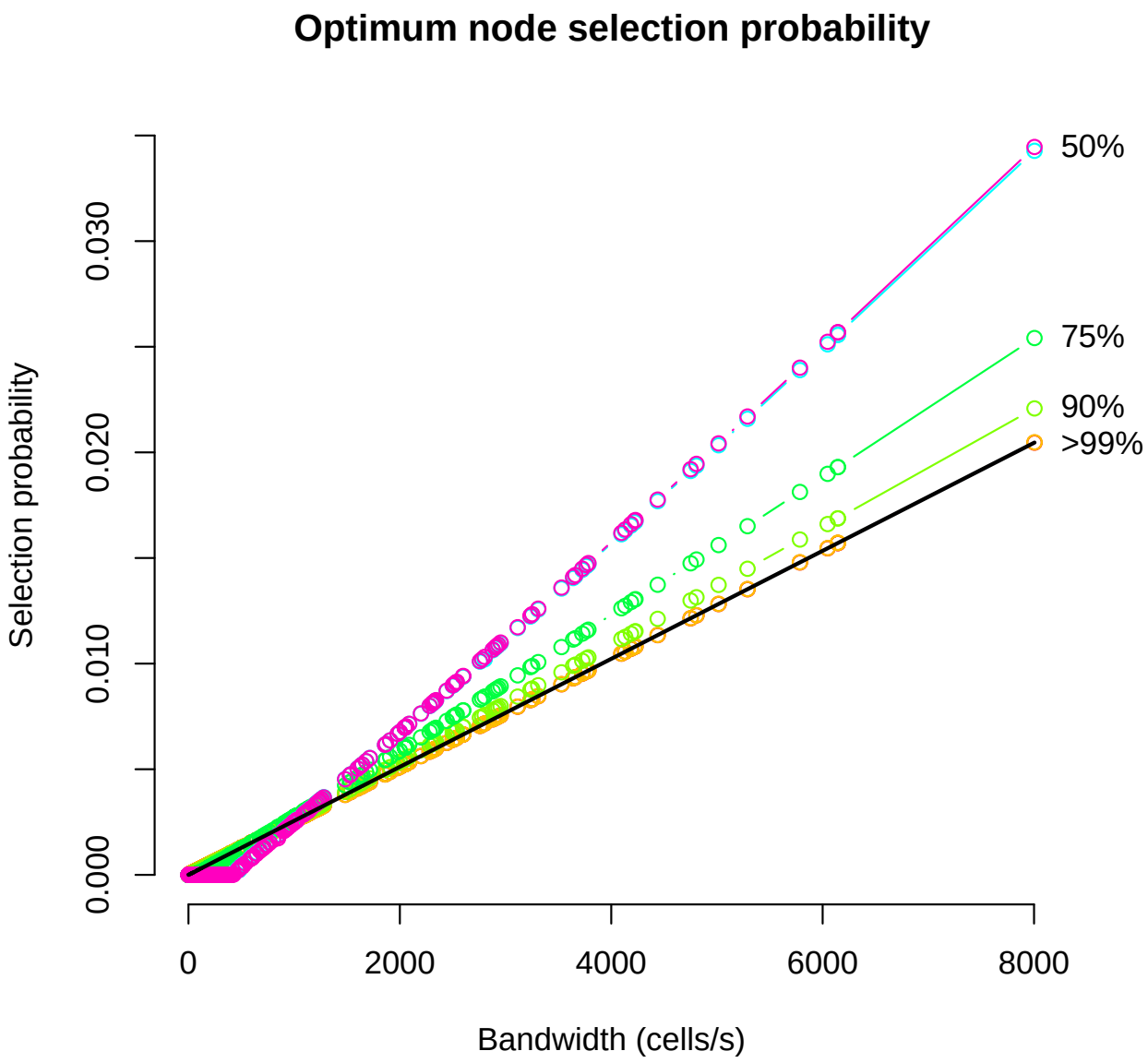

Figure 1: Optimum relay selection probabilities for a variety of network loads. Tor is currently at around 50% utilization. The relay selection probabilities currently used by Tor are shown in black.

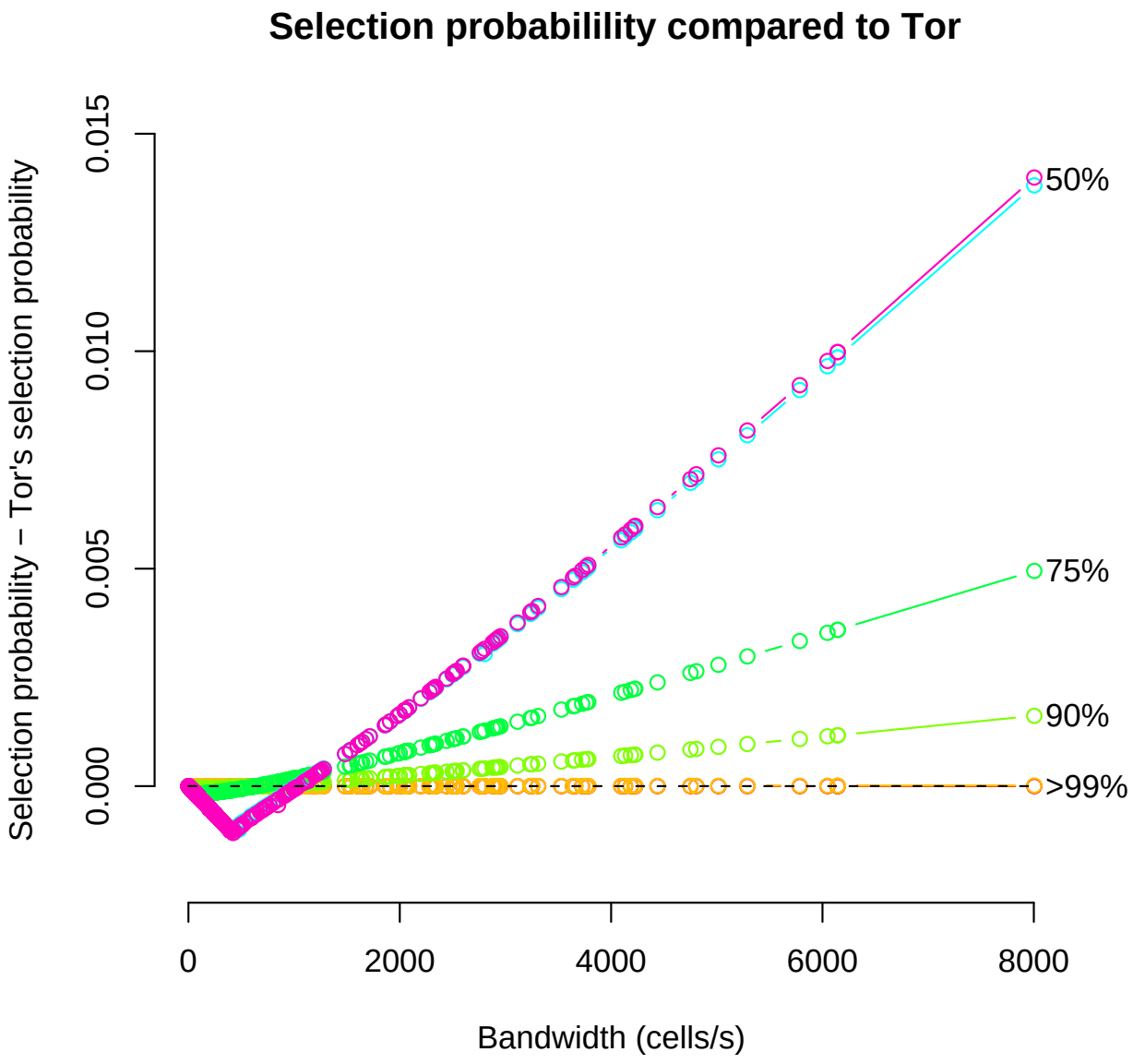

Figure 2: Difference between Tor's current relay selection probabilities and the optimum, for a variety of network loads. For Tor's current network load ($\approx 50\%$) shown in pink, the slowest relays are not used at all, and the slower relays are favoured less.

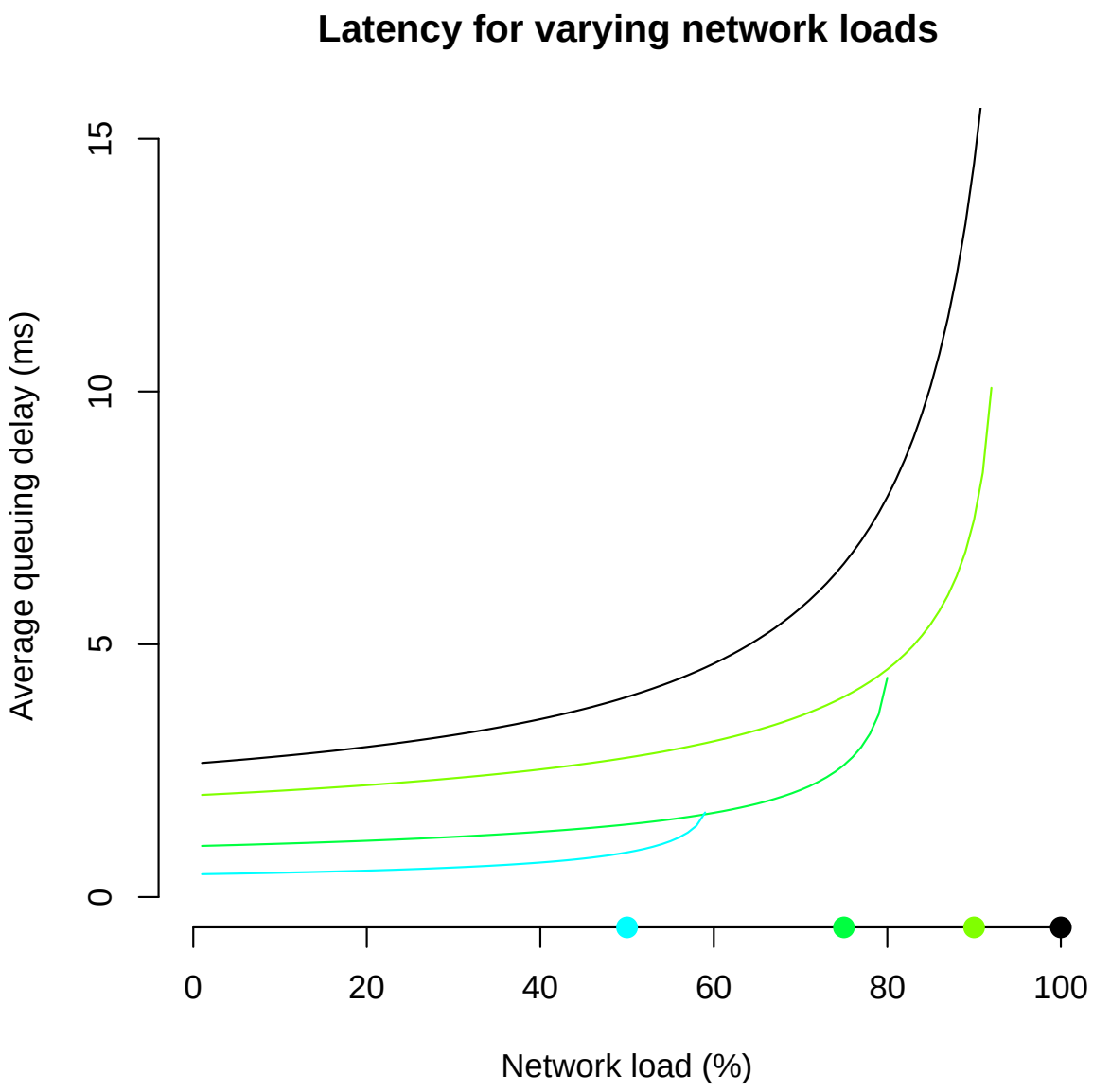

Figure 3: Average network latency against network load. Three relay selection probabilities are shown, optimized for 50%, 75%, and 90% network load. The Tor relay selection algorithm is also included (black). The dots on the x axis show the level of network load at which the relay selection probability distributions are optimized for. The line is cut off when the model predicts that at least one relay will have an infinite queue length, which occurs before load = capacity for all relay selection algorithms except for Tor's current one.

each Tor relay opportunistically monitor the data rates that it achieves when communicating with other Tor relays. Since currently Tor uses a clique topology, given enough time, all relays will communicate with all other Tor relays. If each Tor relay reports their measurements back to the directory authorities, then the median report should be a good estimate of that relay's bandwidth. As a bonus, this estimate should be difficult to game, when compared to the current approach of self-advertising bandwidth capacity.

Experiments show that opportunistic bandwidth measurement has a better systematic error than Tor's current self-advertised measure, although has a poorer log-log correlation (0.48 vs. 0.57). The most accurate scheme is active probing of capacity, with a log-log correlation of 0.63, but this introduces network overhead.

All three schemes suffer from fairly poor accuracy. Perhaps this inaccuracy is due to some relays with high variance in bandwidth capacity? We need to explore this area more to understand why our estimates are not as good as they could be.

Impact: Low-medium.

Effort: Medium, since we still need to get a better sense of the correct network load to expect, and we need to experiment to see if the model actually matches reality.

Risk: Low, since we can always back out the changes.

Plan: More research remains here to figure out what algorithms will actually produce more accurate bandwidth estimates. As with Section 4.1 above, once we do have some better numbers, we can change the weights in the directory, and clients will immediately move to the better numbers. We should also experiment with augmenting our estimates with active probes from Mike's SpeedRacer tool.

4.3 Bandwidth might not even be the right metric to weight by

The current Tor network selection algorithm biases purely by bandwidth. This approach will sometimes cause high latency circuits due to multiple ocean crossings or otherwise congested links. An alternative approach would be to not only bias selection of relays based on bandwidth, but to also bias the selection of hops based on expected latency.

Micah Sherr is finishing his PhD thesis at Penn under Matt Blaze, exploring exactly this issue. In the past we've avoided any sort of path selection algorithm that requires pairwise measurements of the network, because communicating N^2 measurements to clients would take too much bandwidth. Micah solves this problem by using a *virtual coordinate system* – a three or four dimension space such that distance between relays in the virtual coordinate space corresponds to the network latency (or other metric) between them.

His experiments show that we could see a significant speedup in the Tor network if users choose their paths based on this new relay selection algorithm. More research remains, of course, but the initial results are very promising.

On the other hand, reducing the number of potential paths would also have anonymity consequences, and these would need to be carefully considered. For example, an attacker who wishes to monitor traffic could create several relays, on distinct /16 subnets, but with low latency between them. A Tor client trying to minimize latency would be more likely to select these relays for both entry than exit than it would otherwise. This particular problem could be mitigated by selecting entry and exit relay as normal, and only using latency measurements to select the middle relay.

Impact: Medium-high.

Effort: Medium-high, since we first need to sort out how effective the algorithm is, and then we need to figure out a migration plan.

Risk: Medium, since a new selection algorithm probably carries with it a new set of anonymity-breaking papers that will only come out a few years after we deploy.

Plan: Micah is going to write a design proposal for getting relays to compute and maintain their virtual coordinates based on latency. Once we deploy that, we'll have some actual data points, and we'll be in a better position to simulate whether the idea will help in reality. Counting deployment time, that means we probably won't have clients using this scheme until 2010.

4.4 Considering exit policy in relay selection

When selecting an exit relay for a circuit, the Tor client will build a list of all exit relays which can carry the desired stream, then select from them with a probability weighted by each relay's capacity[7]. This means that relays with more permissive exit policies will be candidates for more circuits, and hence will be more heavily loaded compared to relays with restrictive policies.

Figure 4 shows the exit relay capacity for a selection of port numbers. It can be clearly seen that there is a radical difference in the availability of relays for certain ports (generally those not in the default exit policy). Any traffic to these ports will be routed through a small number of exit relays, and if they have a permissive exit policy, they will likely become overloaded from all the other traffic they receive. The extent of this effect will depend on how much traffic in Tor is to ports which are not in the default exit policy.

The overloading of permissive exit relays can be counteracted by adjusting the selection probability of a relay based on its exit policy and knowledge of the global network load per-port. While it should improve performance, this modification will make it easier for malicious exit relays to select traffic they wish to monitor. For example, an exit relay which wants to attack SSH sessions can currently list only port 22 in its exit policy. Currently they will get a small amount of traffic compared to their capacity, but with the modification they will get a much larger share of SSH traffic. (On the other hand, a malicious exit relay could already do this by artificially inflating its advertised bandwidth capacity.)

To properly balance exit relay usage, it is necessary to know the usage of the Tor network, by port. McCoy *et al.* [6] have figures for protocol usage in Tor, but these figures were generated through deep packet inspection, rather than by port number. Furthermore, the exit relay they ran used the fairly permissive default exit policy. Therefore, their measurements will underestimate the relative traffic on ports which are present in the default exit policy, and are also present in more restrictive policies. To accurately estimate the Tor network usage by port, it is necessary to measure the network usage by port on one or more exit relays, while simultaneously recording the exit policy of all other exit relays considered usable.

We could instead imagine more crude approaches. For example, in Section 3.4 we suggest using a tool like SpeedRacer or SoaT to identify relays that are overloaded. We could then either instruct clients to avoid them entirely, or reduce the capacity associated with that relay in the directory status to reduce the attention the relay gets from clients. Then we could avoid the whole question of *why* the relays are overloaded. On the other hand, understanding the reasons for load hotspots can help us resolve them at the architectural level.

Impact: Low-medium.

Effort: Low-medium.

Risk: Low.

Plan: When we're gathering statistics for metrics, we should make a point of gathering some anonymized data about destination ports seen by a few exit relays. Then we will have better intuition about whether we should solve this by reweighting at the clients, reweighting in the directory status, or ignoring the issue entirely.

4.5 Older entry guards are overloaded

While the load on exit relays is skewed based on having an unusual exit policy, load on entry guards is skewed based on how long they've been in the network.

Since Tor clients choose a small number of entry guards and keep them for several months, a relay that's been listed with the Guard flag for a long time will accumulate an increasing number of clients. A relay that just earned its Guard flag for the first time will see very few clients.

To combat this skew, clients should rotate entry guards every so often. We need to look at network performance metrics and discern how long it takes for the skew to become noticeable – it might be that

[7]The actual algorithm is slightly more complex: in particular, exit relays which are also guard relays will be weighted less, and some circuits are created preemptively without any destination port in mind.

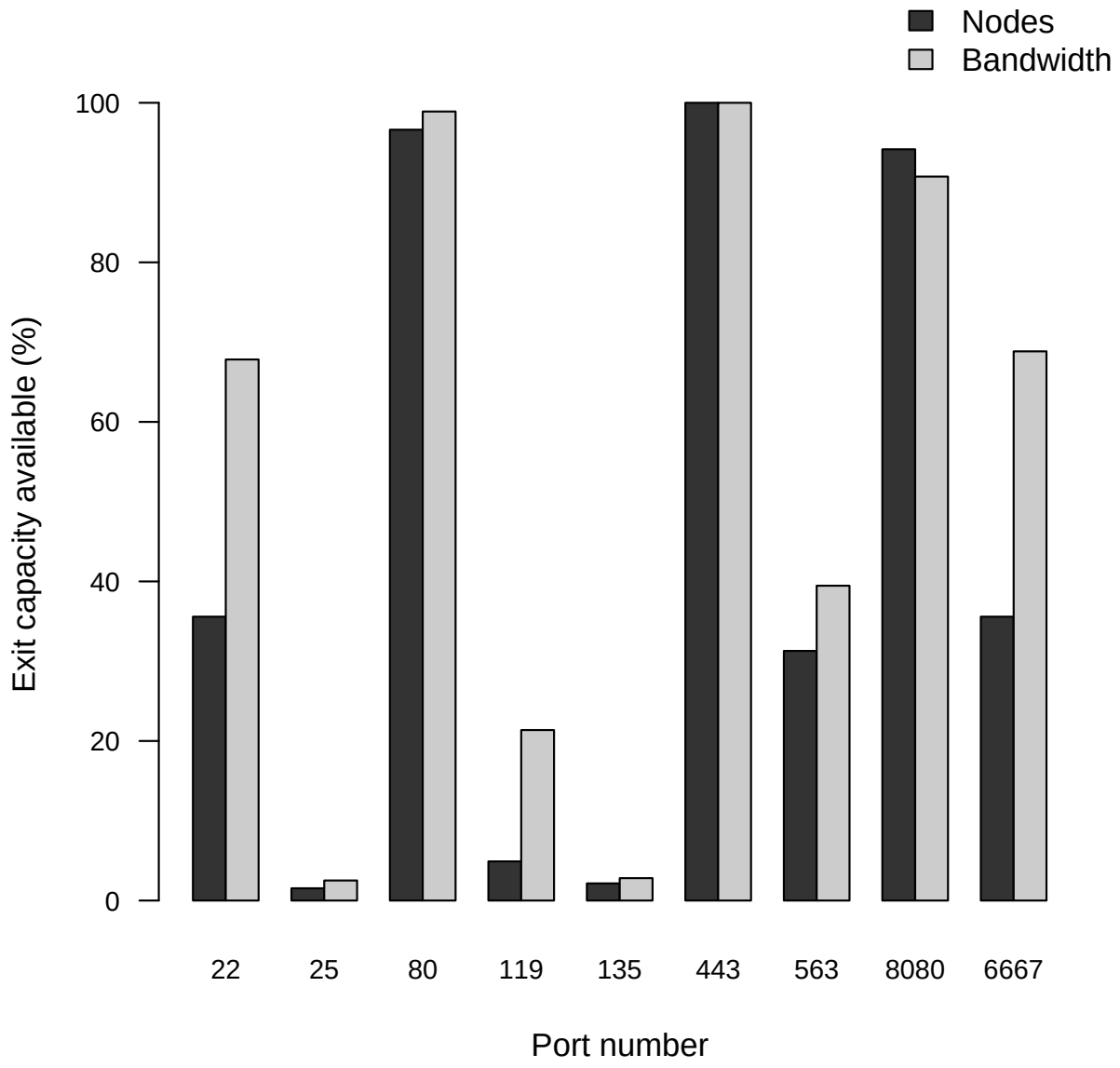

Figure 4: Exit relay capacity, in terms of number of relays and advertised bandwidth for a selection of port numbers.

rotating to a new guard after a week or two is enough to substantially resolve the problem. We also need to consider the added risk that higher guard churn poses versus the original attack they were designed to thwart [11], but I think a few weeks should still be plenty high.

At the same time, there are fewer relays with the Guard flag than there should be. While the Exit flag really is a function of the relay's exit policy, the required properties for entry guards are much more vague: we want them to be "fast enough", and we want them to be "likely to be around for a while more". I think the requirements currently are too strict. This scarcity of entry guards in turn influences the anonymity the Tor network can provide, since there are fewer potential entry points into the network.

Impact: High.

Effort: Low.

Risk: Low.

Plan: We should do it, early in Tor 0.2.2.x. We'll need proposals first, both for the "dropping old guards" plan (to assess the tradeoff from the anonymity risk) and for the "opening up the guard criteria" plan.

5 Clients need to handle variable latency and failures better

The next issue we need to tackle is that Tor clients aren't as good as they should be at handling high or variable latency and connection failures. First, we need ways to smooth out the latency that clients see. Then, for the cases where we can't smooth it out enough, we need better heuristics for clients to automatically shift away from bad circuits, and other tricks for them to dynamically adapt their behavior.

5.1 Our round-robin and rate limiting is too granular

Tor's rate limiting uses a token bucket approach to enforce a long-term average rate of incoming and outgoing bytes, while still permitting short-term bursts above the allowed bandwidth. Each token represents permission to send another byte onto the network (or read from the network). Every second new tokens are added, up to some cap (the *bucket size*).

So Tor relays that have cells buffered waiting to go out onto the network will wait until the new second arrives, and then deliver as many cells as they can. In practice, this behavior results in traffic 'bumps' at the beginning of each second, with little network traffic the rest of the time. Mike and Karsten have been collecting data from circuit extension times (how long it takes to establish each hop of a circuit); the bumps are easily seen in Figure 5.

Our original theory when designing Tor's rate limiting was that one-second granularity should be sufficient: cells will go out as quickly as possible while the bucket still has tokens, and once it's empty there's nothing we can do but wait until the next second for permission to send more cells.

We should explore refilling the buckets more often than once a second, for three reasons. First, we'll get a better intuition about how full the buffers really are: if we spread things out better, then we could reduce latency by perhaps multiple seconds. Second, spikes-and-silence is not friendly for TCP, so averaging the flows ourselves might mean much smoother network performance. Third, sub-second precision will give us more flexibility in our priority strategies from Section 2.1.

On the other hand, we don't want to go too far: cells are 512 bytes, so it isn't useful to think in units smaller than that. Also, every network write operation carries with it overhead from the TLS record, the TCP header, and the IP packet header. Finally, network transmission unit (MTU) sizes vary, but if we could use a larger packet on the wire and we don't, then we're not being as efficient as we could be.

Impact: Low-Medium.

Effort: Medium.

Risk: Low, unless we add in bad feedback effects and don't notice.

Plan: We should continue collecting metrics to get better intuition here. While we're designing priority stategies for Section 2.1, we should keep the option of higher-resolution rate-limiting in mind.

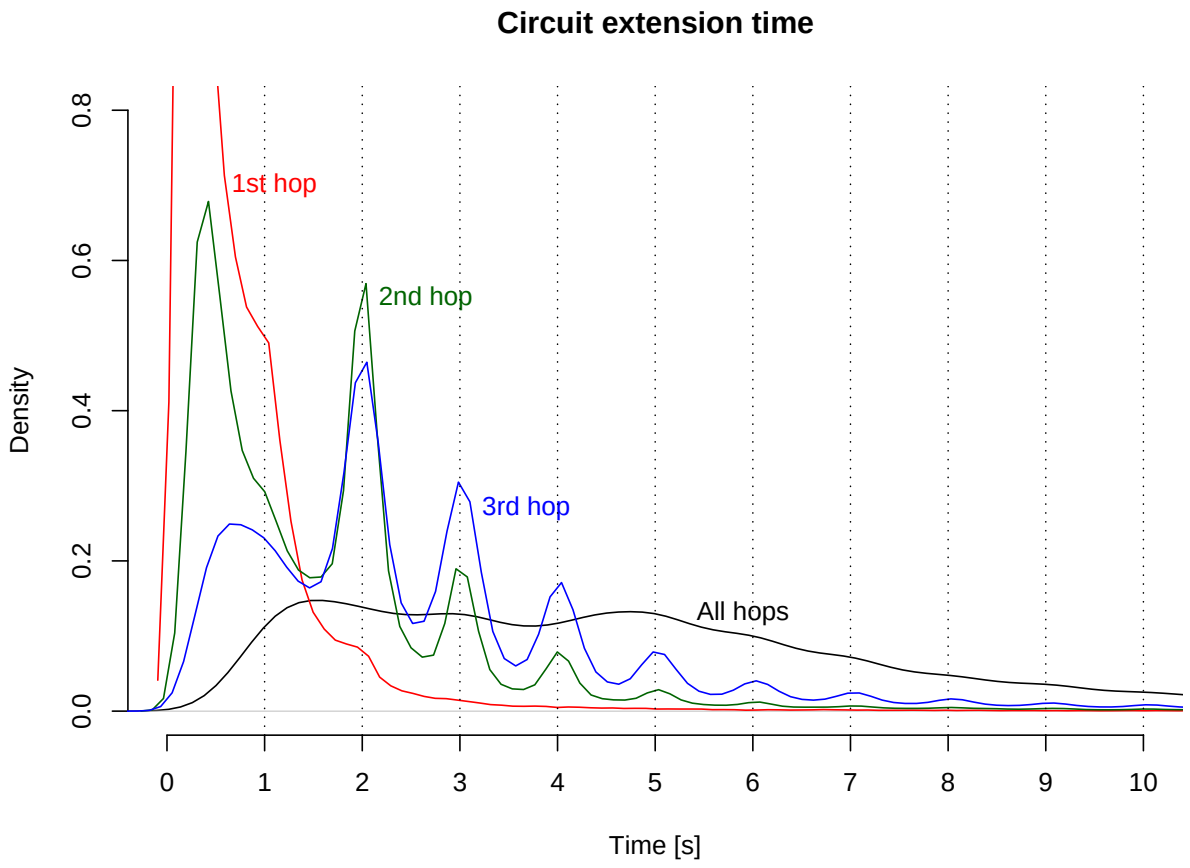

Figure 5: Number of seconds it takes to establish each hop of a 3-hop circuit. The higher density of samples around 2s, 3s, etc indicate that rate limiting at each relay is introducing extra delay into the responses.

5.2 Better timeouts for giving up on circuits and trying a new one

Some circuits are established very quickly, and some circuits take many seconds to form. The time it takes for the circuit to open can give us a hint about how well that circuit will perform for future traffic. We should discard extremely slow circuits early, so clients never have to even try them.

The question, though, is how to decide the right timeouts? If we set a static timeout in the clients, then choosing a number that's too low will cause clients to discard too many circuits. Worse, clients on really bad connections will never manage to establish a circuit. On the other hand, setting a number that's too high won't change the status quo much.

Fallon Chen worked during her Google-Summer-of-Code-2008 internship with us on collecting data about how long it takes for clients to establish circuits, and analyzing the data to decide what shape the distribution has (it appears to be a Pareto distribution). The goal is for clients to track their own circuit build times, and then be able to recognize if a circuit has taken longer than it should have compared to the previous circuits. That way clients with fast connections can discard not-quite-fast-enough circuits, whereas clients with slow connections can discard only the really-very-slow circuits. Not only do clients get better performance, but we can also dynamically adapt our paths away from overloaded relays.

Mike and Fallon wrote a proposal[8] explaining the details of how to collect the stats, how many data points the client needs before it has a good sense of the expected build times, and so on.

Further, there's another case in Tor where adaptive timeouts would be smart: how long we wait in between trying to attach a stream to a given circuit and deciding that we should try a new circuit. Right now we have a crude and static "try 10 seconds on one, then try 15 seconds on another" algorithm, which is both way too high and way too low, depending on the context.

Impact: Medium.

Effort: Medium, but we're already part-way through it.

Risk: Low, unless we've mis-characterized the distribution of circuit extend times, in which case clients end up discarding too many circuits.

Plan: We should deploy the changes in clients in Tor 0.2.2.x to collect circuit times, and see how that goes. Then we should gather data about stream timeouts to build a plan for how to resolve the second piece.

5.3 If extending a circuit fails, try extending a few other places before abandoning the circuit.

Right now, when any extend operation fails, we abandon the entire circuit. As the reasoning goes, any other approach allows an attacker who controls some relays (or part of the network) to dictate our circuits (by declining to extend except to relays that he can successfully attack).

However, this reasoning is probably too paranoid. If we try at most three times for each hop, we greatly increase the odds that we can reuse the work we've already done, but we don't much increase the odds that an attacker will control the entire circuit.

Overall, this modification should cut down on the total number of extend attempts in the network. This result is particularly helpful since some of our other schemes in this document involve increasing that number.

Impact: Low.

Effort: Low.

Risk: Low-medium. We need to actually do some computations to confirm that the risk of whole-path compromise is as low as we think it is.

Plan: Do the computations, then write a proposal, then do it.

[8] https://svn.torproject.org/svn/tor/trunk/doc/spec/proposals/151-path-selection-improvements.txt

5.4 Bundle the first data cell with the begin cell

In Tor's current design, clients send a "relay begin" cell to specify the intended destination for our stream, and then wait for a "relay connected" cell to confirm the connection is established. Only then do they complete the SOCKS handshake with the local application, and start reading application traffic.

We could modify our local proxy protocol in the case of Privoxy or Polipo so it sends the web request to the SOCKS port during the handshake. Then we could optimistically include the first cell worth of application data in the original begin cell. This trick would allow us to cut out an entire network round-trip every time we establish a new connection through Tor. The result would be quicker page loads for users.

Alas, this trick would involve extending the SOCKS protocol, which isn't usually a polite strategy when it comes to interoperating with other applications. On the other hand, it should be possible to extend it in a backwards-compatible way: applications that don't know about the trick would still behave the same and still work fine (albeit in a degraded mode where they waste a network round-trip).

Impact: Medium.

Effort: Medium.

Risk: Low.

Plan: Overall, it seems like a delicate move, but with potentially quite a good payoff. I'm not convinced yet either way.

6 The network overhead may still be high for modem users

Even if we resolve all the other pieces of the performance question, there still remain some challenges posed uniquely by users with extremely low bandwidth – for example, users on modems or cell phones. We need to optimize the Tor protocols so they are efficient enough that Tor can be practical in this situation too.

6.1 We've made progress already at directory overhead

We've already made great progress at reducing directory overhead, both for bootstrapping and maintenance. Our blog post on the topic provides background and details[9].

Proposal 158 further reduces the directory overhead, and is scheduled to be deployed in Tor 0.2.2.x.[10]

Impact: Low for normal users, high for low-bandwidth users.

Effort: Medium, but we're already a lot of the way through it.

Risk: Low.

Plan: We should roll out proposal 158. Then we'll be in good shape for a while. The next directory overhead challenge will be in advertising many more relays; but first we need to get the relays.

6.2 Our TLS overhead can also be improved

OpenSSL will, by default, insert an empty TLS application record before any one which contains data. This is to prevent an attack, by which someone who has partial control over the plaintext of a TLS stream, can also confirm guesses as to the plaintext which he does not control. By including an empty application record, which incorporates a MAC, the attacker is made unable to control the CBC initialization vector, and hence does not have control of the input to the encryption function [7].

This application record does introduce an appreciable overhead. Most Tor cells are sent in application records of their own, giving application records of 512 bytes (cell) + 20 bytes (MAC) + 12 bytes (TLS padding) + 5 bytes (TLS application record header) = 549 bytes. The empty application records contain

[9] https://blog.torproject.org/blog/overhead-directory-info%3A-past%2C-present%2C-future

[10] https://svn.torproject.org/svn/tor/trunk/doc/spec/proposals/158-microdescriptors.txt

only 20 bytes (MAC) + 12 bytes (TLS padding) + 5 bytes (TLS application record header) = 37 bytes. There is also a 20 byte IP header and 32 byte TCP header.

Thus the overhead saved by removing the empty TLS application record itself is $37/(549+37+20+32) = 5.8\%$. This calculation is assuming that the same number of IP packets will be sent, because currently Tor sends packets, with only one cell, far smaller than the path MTU. If Tor were to pack cells optimally efficiently into packets, then removing the empty application records would also reduce the number of packets, and hence TCP/IP headers, that needed to be sent. The reduction in TCP/IP header overhead would be $37/(549 + 37) = 6.3\%$.

Of course, the empty application record was inserted for a reason – to prevent an attack on the CBC mode of operation used by TLS, so before removing it we must be confident the attack does not apply to Tor. Ben Laurie (one of the OpenSSL developers) concluded that in his opinion Tor could safely remove the insertion of empty TLS application records [5]. Steven was able to come up with only certificational weaknesses (discussed in the above analysis), which are expensive to exploit and give little information to the attacker.

Impact: Low.
Effort: Low.
Risk: Medium, since our initial analysis might be wrong.
Plan: Do it in the Tor 0.2.2.x or 0.2.3.x timeframe. Not critical.

7 Last thoughts

7.1 Lessons from economics

Imagine we implement all the solutions above, and it doubles the effective capacity of the Tor network. The naïve hypothesis is that users would then experience twice the throughput. Unfortunately this is not true, because it assumes that the number of users does not vary with bandwidth available. In fact, as the supply of the Tor network's bandwidth increases, there will be a corresponding increase in the demand for bandwidth from Tor users. Simple economics shows that performance of Tor and other anonymization networks is controlled by how the number of users scales with available bandwidth; this relationship can be represented by a demand curve.[11]

Figure 6 is the typical supply and demand graph from economics textbooks, except with long-term throughput per user substituted for price, and number of users substituted for quantity of goods sold. As the number of users increases, the bandwidth supplied by the network falls.

In drawing the supply curve, we have assumed the network's bandwidth is constant and shared equally over as many users as needed. The shape of the demand curve is much harder to even approximate, but for the sake of discussion, we have drawn three alternatives. The number of Tor users and the throughput they each get is the intersection between the supply and demand curves – the equilibrium. If the number of users is below this point, more users will join and the throughput per user will fall to the lowest tolerable level. Similarly, if the number of users is too high, some will be getting lower throughput than their minimum, so will give up, improving the network for the rest of the users.

Now assume Tor's bandwidth grows by 50% – the supply curve shifts, as shown in the figure. By comparing how the equilibrium moves, we can see how the shape of the demand curve affects the performance improvement that Tor users see. If the number of users is independent of performance, shown in curve A, then everyone gets a 50% improvement, which matches the naïve hypothesis. More realistically, the number of users increases, so the performance gain is less. The shallower the curve gets, the smaller the performance increase will be. For demand curve B, there is a 18% increase in the number of Tor users and a 27% increase

[11] The economics discussion is based on a blog post published in Light Blue Touchpaper [9]. The property discussed was also observed by Andreas Pfitzmann in response to a presentation at the PET Symposium [16].

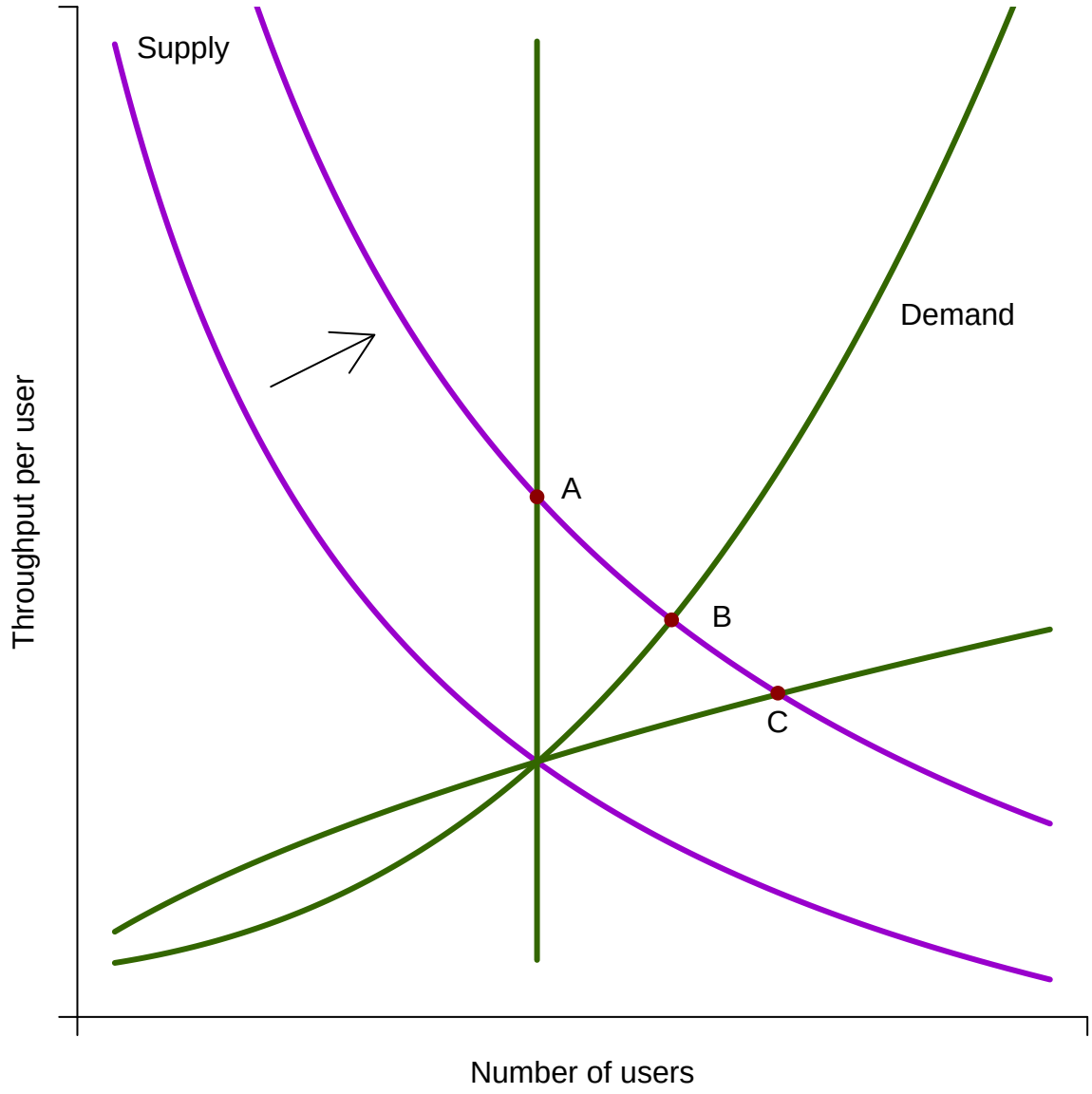

Figure 6: Hypothetical supply and demand curves for Tor network resources. As supply goes up, point A corresponds to no increase in users, whereas points B and C represent more users arriving to use up some of the new capacity.

in throughput. On the other hand, with curve C there are 33% more users and so only a 13% increase in throughput for each user.

The above analysis glosses over many topics. One interesting analysis is reaching equilibrium – in fact it could take some time between the network bandwidth changing and the user population reaching stability. If this period is sufficiently long and network bandwidth is sufficiently volatile it might never reach equilibrium. We might also consider effects which shift the demand curve. In normal economics, marketing makes people buy a product even though they considered it too expensive. Similarly, a Slashdot article or news of a privacy scandal could make Tor users more tolerant of the poor performance. Finally, the user perception of performance is an interesting and complex topic. In this analyis we assumed that performance is equivalent to throughput; but actually latency, packet loss, predictability, and their interaction with TCP/IP congestion control are important components too.

So what does all this tell us?

The above discussion has argued that the speed of an anonymity network will converge on the slowest level that the most tolerant users will consider usable. This is problematic because there is significant variation in levels of tolerance between different users and different protocols. Most notably, file sharing users are subject to high profile legal threats, and do not require interactive traffic, so they will continue to use a network even if the performance is considerably lower than the usable level for web browsing.

In conventional markets, this type of problem is solved by differential pricing, for example different classes of seat on airline flights. In this model, several equilibrium points are allowed to form, and the one chosen will depend on the cost/benefit tradeoffs of the customers. A similar strategy could be used for Tor, allowing interactive web browsing users to get higher performance, while forcing bulk data transfer users to have lower performance (but still tolerable for them). Alternatively, the network could be configured to share resources in a manner such that the utility to each user is more equal. In this case, it will be acceptable to all users that a single equilibrium point is formed, because its level will no longer be characterized in terms of simple bandwidth.

Section 2 is an example of the former strategy. Web browsing users will be offered better performance, so we should attract more of them, but hopefully not so many that the performance returns to current levels. In constrast, bulk-traffic users will be given poorer performance, but since they are less sensitive to latency, it could be that they do not mind. Section 1 could be used to implement the latter strategy. If web-browsing users are more sensitive to latency than bandwidth, then we could optimize the network for latency rather than throughput.

7.2 The plan moving forward

Our next steps should be to work with funders and developers to turn this set of explanations and potential fixes into a roadmap: we need to lay out all the solutions, sort out the dependencies, assign developers to tasks, and get everything started.

At the same time, we need to continue to work on ways to measure changes in the network: without 'before' and 'after' snapshots, we'll have a much tougher time telling whether a given idea is actually working. Many of the plans here have a delay between when we roll out the change and when the clients and relays have upgraded enough for the change to be noticeable. Since our timeframe requires rolling out several solutions at the same time, an increased focus on metrics and measurements will be critical to keeping everything straight.

Lastly, we need to be aware that ramping up development on performance may need to push out or downgrade other items on our roadmap. So far, Tor has been focusing our development energy on the problems that funders are experiencing most severely at the time. This approach is good to make sure that we're always working on something that's actually important. But it also means that next year's critical items don't get as much attention as they should, and last year's critical items don't get as much maintenance as they should. Ultimately we need to work toward having consistent funding for core Tor development and maintenance as well as feature-oriented funding.

References

[1] Kent, S., and Seo, K. Security architecture for the internet protocol. RFC 4301, IETF, December 2005.

[2] Kiraly, C. Effect of Tor window size on performance. Email to or-dev@freehaven.net, February 2009. http://archives.seul.org/or/dev/Feb-2009/msg00000.html.

[3] Kiraly, C., G., B., and Lo Cigno, R. Solving performance issues in anonymization overlays with a L3 approach. Tech. Rep. DISI-08-041, University of Trento, September 2008. version 1.1, http://disi.unitn.it/locigno/preprints/TR-DISI-08-041.pdf.

[4] Kohno, T., Broido, A., and claffy, k. Remote physical device fingerprinting. In *IEEE Symposium on Security and Privacy* (Oakland, CA, US, May 2005), IEEE Computer Society, pp. 211–225.

[5] Laurie, B. On TLS empty record insertion. Email to or-dev@freehaven.net, in thread "Re: Empty TLS application records being injected in Tor streams", December 2008. http://archives.seul.org/or/dev/Dec-2008/msg00005.html.

[6] McCoy, D., Bauer, K., Grunwald, D., Kohno, T., and Sicker, D. Shining light in dark places: Understanding the Tor network. In *Proceedings of the Eighth International Symposium on Privacy Enhancing Technologies (PETS 2008)* (Leuven, Belgium, July 2008), N. Borisov and I. Goldberg, Eds., Springer, pp. 63–76.

[7] Möller, B. Security of CBC ciphersuites in SSL/TLS: Problems and countermeasures, May 2004. http://www.openssl.org/~bodo/tls-cbc.txt.

[8] Murdoch, S. J. Hot or not: Revealing hidden services by their clock skew. In *CCS '06: Proceedings of the 9th ACM Conference on Computer and Communications Security* (Alexandria, VA, US, October 2006), ACM Press, pp. 27–36.

[9] Murdoch, S. J. Economics of Tor performance. Light Blue Touchpaper, 18 July 2007. http://www.lightbluetouchpaper.org/2007/07/18/economics-of-tor-performance/.

[10] Murdoch, S. J., and Watson, R. N. M. Metrics for security and performance in low-latency anonymity networks. In *Proceedings of the Eighth International Symposium on Privacy Enhancing Technologies (PETS 2008)* (Leuven, Belgium, July 2008), N. Borisov and I. Goldberg, Eds., Springer, pp. 115–132.

[11] Øverlier, L., and Syverson, P. Locating hidden servers. In *Proceedings of the 2006 IEEE Symposium on Security and Privacy* (May 2006), IEEE CS.

[12] Pradhan, P., Kandula, S., Xu, W., Shaikh, A., and Nahum, E. Daytona: A user-level TCP stack, 2002. http://nms.lcs.mit.edu/~kandula/data/daytona.pdf.

[13] Reardon, J. Improving Tor using a TCP-over-DTLS tunnel. Master's thesis, University of Waterloo, September 2008. http://hdl.handle.net/10012/4011.

[14] Rescorla, E., and Modadugu, N. Datagram transport layer security. RFC 4347, IETF, April 2006.

[15] Snader, R., and Borisov, N. A tune-up for Tor: Improving security and performance in the Tor network. In *Network & Distributed System Security Symposium* (February 2008), Internet Society.

[16] Wendolsky, R., Herrmann, D., and Federrath, H. Performance comparison of low-latency anonymisation services from a user perspective. In *Proceedings of the Seventh Workshop on Privacy Enhancing Technologies (PET 2007)* (Ottawa, Canada, June 2007), N. Borisov and P. Golle, Eds., Springer.