Electronics and Computer Science
Faculty of Physical Sciences and Engineering
University of Southampton

Author: Christopher Baines

April 29, 2014

Enhancing Tor Hidden Services

Project Supervisor: Dr Tim Chown
Second Examiner: Dr Soon Xin Ng

A project report submitted for the award of
MEng Computer Science

Abstract

Tor is an low latency, onion routing system that anonymizes TCP streams. One particular Tor feature is hidden services, these provide responder anonymity, this means the identity of the server providing the service, is hidden from the requester (user) of the service.

While modern web services, which can use techniques like anycast and DNS (domain name system) round robin to distribute clients across many servers, the main load for a Tor hidden service, will always go through a single node in the Tor network. This has implications for the availability and scalability of Tor Hidden Services, which has knock on implications anonymity of the hidden service, as downtime can possibly reveal information about its real world location. I aim to modify Tor to allow for distributed hidden services.

This was achieved, however a deterministic property was added to the introduction points, which can be used to attack the service. A approach to solve this is discussed.

Contents

Acknowledgements

Firstly I would like to thank my project supervisor, Tim Chown, for his support and input.

Secondly, I would like to thank those who I communicated with on both the Tor development mailing list (Nick Mathewson, Matthew Finkel, Paul Syverson, Lars Luthman and George Kadianakis), and the tor development IRC channel. This communication was instrumental in helping me complete this project.

1 Background

1.1 Tor

Tor is a low latency, decentralised, overlay network that attempts to anonymize TCP streams. Many applications can be used through Tor, such as web browsers, secure shell and instant messaging.

Tor achieves this by routing clients connections through a number of onion routers (OR's). This route is collectively called a circuit. The anonymity comes from each OR in the circuit only knowing the identity of the previous OR and next OR. This is slightly different at the ends, as the first OR in the circuit will see the identity of the client, and the last OR will send the traffic outside of the Tor network.

Standard Tor circuits have 3 OR's, the first is denoted the Guard, the second the Middle, and the 3rd and last, the Exit. Each client keeps a list of guards that it will use for long periods to reduce the risk of compromise. The middle node knows the identity of the guard, and the exit, but not the client, or the destination. The exit can see the traffic coming from the middle, and going to the destination, but cannot determine the identity of the guard or client. See appendix C for a set of diagrams which explain the process of connecting to the Internet through the Tor network.

1.2 Tor Hidden Services

As well as providing anonymity for clients, Tor can also provide responder anonymity, this means that the user cannot determine the identity or location of the service. Tor hidden services also have the property that the service cannot determine the identity or location of the user. Tor Hidden Services are first described in the Tor design paper [6].

1.2.1 Network Level Outline

These are two important roles that nodes within the Tor network can function as with regard to hidden services, introduction points (IP) and rendezvous points (RP).

The introduction points are short to medium term nodes within the Tor network, chosen by the hidden service. There identity is made available through several Tor hidden service directory servers. Clients use them to arrange a rendezvous location within the network. Those connecting to a hidden service know the identity of its introduction points, but it should be difficult for the introduction points to learn what service they are handling. The introduction points also handle none of the actual traffic for the service.

The rendezvous points are chosen by the client, there is only one chosen per attempt to connect to the hidden service. The client asks the introduction point, to ask the hidden service, to connect to the chosen rendezvous point. Once the service does so (this is done through a 3 hop circuit), the rendezvous point joins the circuits (from the client and the service) allowing for direct (6 hops through the Tor network) communication between the client and the service.

1.2.2 Creating and Publishing a Hidden Service

Hidden service creation revolves around generating a public/private keypair, this is used as the basis of the services identity. [13, 1.2]

A diagrammatic overview the process described in the following paragraphs is available in appendix D.

Introduction Points (IP's) Several introduction points are randomly selected from the set of suitable nodes, each is connected to using a Tor circuit. These nodes are asked to become introduction points, and sent a public key, that will be used by clients to connect. In older versions of Tor using v0 service descriptors, this was the hidden service public key. With v2 descriptors, a key (referred to as the 'service key') is created per introduction point. This helps prevent the introduction point from identifying what service it is handling introductions for, as it can no longer match the key directly to the service. [13, 1.2]

Service Descriptor Clients connect to Tor hidden services using information in the service descriptor , this is fetched from appropriate hidden service directory (HSDir). Descriptors of type v0 were used in versions prior to 0.2.2.1-alpha, and there is a hypothetical v1 descriptor that was never used. From now on will just discuss v2 type descriptors. [13, 1.3]

The service descriptor, or just descriptor for short includes the hidden services public key, a creation timestamp, and a (possibly empty) set of introduction points (along with there corresponding keys "service" keys). [13, 1.3]

The service descriptor is identified by a descriptor-id. This is a 160 bit value, "formatted as 32 base32 characters" [13, 1.3] , derived from the permanent-id of the service, that is the address clients use to connect and a time-period which in turn, is derived from the system time (in such a way that its value changes every 24 hours). In addition to this, there is a replica number included, which allows for the creation of multiple descriptors, for the same service, with different descriptor id's. [13, 1.3]

This service descriptor is stored and made available by a changing set of 6 hidden service directories. Two descriptors are created, differing only in the replica number (to give different descriptor id's). The hidden service directories identity digests are

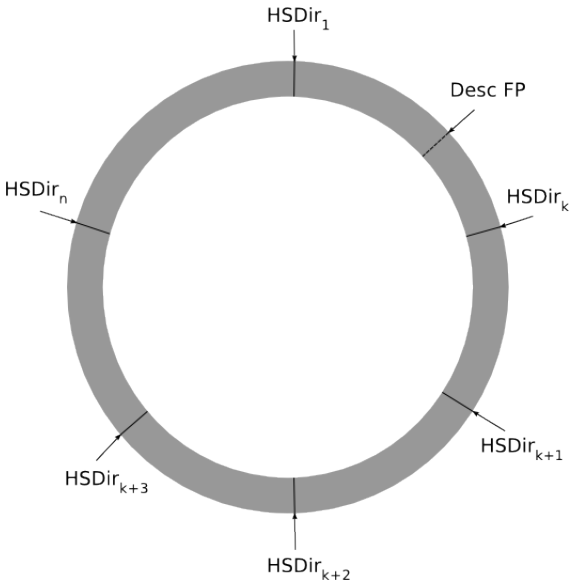


Figure 1: HSDir Hash Ring

put in a circular list (see Figure 1), the three hidden service directories following the descriptor-id's of the two service descriptors are deemed responsible and are sent the respective descriptors. [13, 1.3]

This process is repeated once an hour, or immediately if the content of the descriptor changes e.g. the set of introduction points changes. [13, 1.3]

1.2.3 Connecting to a Hidden Service

First the client takes the hidden service address, this is in the form "z.onion" where z is derived from the hidden service's public key. Like service descriptors, onion proxy's (term used for a Tor node that just acts as a client, abbreviated to OP) act differently depending on the version, I will just discuss the fetching of v2 descriptors which occurs in versions $\geq$ 0.2.1.10-alpha. [13, 1.5] A diagrammatic overview of this process is available in section E.

Descriptor Lookup The clients OP can calculate the currently responsible set of hidden service directories by using the same process used by the services OP described above, note that the permanent-id is the z part of the "z.onion" address. The clients OP then queries the responsible directories, one by one, until the service descriptor is received. [13, 1.5]

Rendezvous Point A OR is randomly chosen, and connected to using a Tor circuit, it is given a random 20 byte value, the "rendezvous cookie", which it will use later to perform some basic authentication of the hidden service. [13, 1.7]

Client OP to Introduction Point Tor clients select introduction points from the service descriptor randomly. This is sent an encrypted message to the hidden service, using the public key specific to this introduction point in the service descriptor. This message contains the identity of the rendezvous point and the rendezvous cookie value. [13, 1.8]

Introduction Point to Hidden Service If the introduction point has a circuit with the corresponding public key included in the above message, it forwards the communication on to the hidden service. [13, 1.9]

The Rendezvous The hidden service then connects to the introduction point, providing the cookie passed to it by the client (through the IP). If the RP recognises the cookie, it connects the two circuits. Once this is complete, the client can establish a TCP connection to the hidden service through Tor. [13, 1.10]

1.3 Previous Work

Limited work has been done on distributed hidden services. It is discussed briefly in the 4th paragraph of the 'Hidden Service Scaling' section of [1].

In terms of existing techniques used on the internet to this effect, DNS (domain name system) round robin [3] and anycast [1] have been used to direct clients to

[1] https://blog.torproject.org/blog/hidden-services-need-some-love

different servers. The aim of this is to provide a technique similar to anycast, or DNS round robin that works for Tor hidden services.

Previous research has been done on adding aditional layers of protection on top of the introduction points, to help prevent attacks that aim to make the service unavailable [12], but while the goals of this project are similar, the area of Tor hidden services that is focused on differs.

Regarding scalability, there has been previous work done [9][8] regarding the performance of Tor hidden services, but this focuses on the overhead of the Tor network, rather than the lack of distributed hidden services.

With the current version of Tor, it is possible to run the same hidden service, on multiple machines. However, most, if not all of the traffic will come to one of these machines (just one instance of the hidden service). This is due to the services instances overriding each others descriptors on the HSDir's.

For example, you setup a Tor Hidden Service on machine A, you copy the key to another machine, and configure Tor on that machine to serve a hidden service. When you start Tor on the second machine (B), it will choose some introduction points, create a hidden service descriptor, and upload this to the same set of hidden service directory servers. This will override the previous descriptor, preventing any new clients connecting to machine A, instead they will connect to B. Clients that have cached the descriptor will continue to connect to A until they re-fetch the descriptor.

Furthermore, if that instance was to fail, the service would be unreachable for new clients, until an active instance published its descriptor.

2 Requirements

The goal in this project is to modify Tor to allow for distributed hidden services. The lack of distributed hidden services in Tor is currently a inherent restriction in the protocol that limits the reliability and scalability of hidden services can achieve.

The reliability of the service is limited because the protocol only allows for one node in the network to be providing access to the service. This means that any failure in this node, or in the surrounding infrastructure could impact the availability of the service. This has a knock on effect on the anonymity of the service, as if an attacker is monitoring the availability of the service, and monitoring events on the internet, for example power outages in data centers, or other networking events, then they can attempt to identify the location of the service by correlating these events to determine some information regarding the location of the service.

The limitation in the scalability in the service comes again from the single node providing access to the service. All the rendezvous requests and traffic to and from the service must go through this node, meaning that the service is limited to the number of clients that can be supported by this single node. Note that this is specific to the load of Tor, and not the service itself, as it is possible to have the service hosted on another machine or set of machines and have Tor direct the traffic to them.

With the functionality added during this project, I aim to preserve or if possible, improve the anonymity properties offered by Tor hidden services. Enabling more people to use Tor has a direct benefit in terms of the anonymity that Tor provides to users [5].

2.1 Primary Goals

2.1.1 Allow for the distribution of connections to a hidden service

This is the core goal, as achieving this would allow for the horizontal scaling (scaling through adding nodes to the system) of hidden services, by distributing the load across multiple machines.

2.1.2 A service must be accessible if one or more instances are in operation

In conjunction with the previous goal, this provides a means for achieving increased availability, the service can be hosted from multiple geographical locations, reducing the probability of all of the service instances going offline (e.g. due to power or network outage).

2.2 Secondary Goals

2.2.1 Obscure the number of hidden service instances

The number of instances needs only to be known to the hidden service operator.

2.2.2 Obscure the state of each service instance

The state (operational or not) service instance can help to compromise the anonymity offered by a hidden service if available.

2.3 Usability Goals

2.3.1 Simple Initial Setup for one instance Hidden Services

It should not get more difficult to run a hidden service with just one instance.

2.3.2 Simple Addition of Instances

This is from the view of the operator, so a complex technical solution would satisfy this goal, as long as it is still simple to use.

2.3.3 Simple Removal of Instances

This is from the view of the operator, so a complex technical solution would satisfy this goal, as long as it is still simple to use.

2.3.4 Well Defined Failure Modes

Distributed systems are complicated, so care must be taken such that the system remains usable, in terms of the service operator solving problems relating to services that they operate.

2.4 Overview

As well as the goals listed above that are specifically about the modifications I am making to Tor, there are some existing functionality, most importantly, the anonymity that a hidden service provides that is key.

I will concentrate on the primary goals, but I will also attempt to satisfy the secondary goals, usability goals, and maintain the existing anonymity properties of Tor Hidden Services.

3 Analysis

There are many options to consider when designing a system to satisfy the goals above, the following sections describe some of these options.

3.1 Centralised or Decentralised

The Tor network itself is a partially centralised network, but any coordination between hidden service instances could be done either in a centralised, decentralised, or partial way.

3.2 Instance Communication

The instances of a hidden service could either not communicate at all, communicate indirectly, or communicate directly. There are aspects of hidden services that do require some coordination, most notably the descriptor, as the descriptors available to clients must allow clients to connect to all of the hidden service instances.

Not communicating at all, might require the hidden service directories to combine multiple descriptors in to one descriptor. An example of indirect communication might be for the hidden service to lookup existing introduction points, and connect to those. Direct communication includes instances connecting to each other just over the internet, or through the Tor network.

3.3 Descriptor

With multiple instances this could be created by a single coordinator instance, each instance could submit their descriptor to a different hidden service directory, or the hidden service directories could build a descriptor out of information provided by all the instances.

3.4 Key Distribution

Currently, the identity of a hidden service instance is directly linked with the possession of the private part of the hidden service public/private keypair. While currently, the main service key is not used when establishing introduction points (it was in previous versions of Tor, but is no longer), the main service key is needed to publish a descriptor, and so a instance without this key cannot change the descriptor (which includes the list of introduction points).

The service key's for the introduction points might also need to be shared, depending on the design. Note that if each service instance can derive a key for a introduction point, and it is crucially the same as the key derived by all the other service instances for that introduction point, this is satisfied. [13, 1.2]

3.5 Load Distribution

Currently, all the load is directed at a single instance. The load can be distributed at the significant parts of the current connection process. For example, the descriptor lookup (could return different results to different clients) and for the introduction point connection, connecting to different introduction points could direct the traffic

at different instances, or the introduction point could select the instance to which the client would connect.

3.5.1 Existing Load Balancing Tools and Techniques

Note that load balancing within the Tor protocol is not designed to replace the use of existing load balancing tools. Load balancing within the Tor network allows for improved reliability, as you can avoid having a single point of failure. It might be desirable in some cases to use standard load balancers between the instances of the Tor hidden service, and the application as this could allow you to take advantage of functionality offered by the load balancer, and improve scalability and reliability without having to run additional tor nodes.

4 Design

Working from the design considerations in the previous section, the following section contains 4 proposals consisting of different combination of the choices discussed above. The foundations for all but the first proposal are from Nick Mathewson's correspondence on the tor-dev mailing list, which can be found here [10].

Also covered is how these choices effect a subset of the goals.

4.1 Goals

I will be evaluating the designs below against the goals described both in the requirements section and again (for quick reference) below.

Primary Goals

Goal 1
Allow for the distribution of connections to a hidden service

Goal 2
A service should be accessible if one or more instances are in operation

Secondary Goals

Goal 1
Obscure the number of hidden service instances

Goal 2
Obscure the state of each service instance

4.2 Proposals

4.2.1 Proposal 1 (HSDir Coordinated Shared Introduction Points)

Changes Required Alter the hidden service, such that when initialised with an existing key and in a state where it has no current introduction points, it will attempt to get the introduction points for its own service (from a HSDir), and connect to those introduction points. This is rather than picking a random set of introduction points to connect to.

As hidden services currently create a key per introduction point, this will have to be changed to the hidden service's public key, such that this is the only information needed by another instance. This was used in the v0 hidden service descriptor.

Alter the introduction points such that they accept multiple connections with a common key (currently the IP would drop additional connections), and that they pass introduction messages to one of the connections with a matching key.

Primary Goals

Goal 1: Achieved
Introduction points will distribute connections to all connected instances.

Goal 2: Achieved
Introduction points will only have circuits for available instances, closing any that are not available.

Secondary Goals

Goal 1: Fail (partial)
While clients cannot determine the number of hidden service instances, the introduction point knows as it can detect when the circuits to those instances fail, or they don't respond to requests.

Goal 2 Fail (partial)
The number of circuits that the introduction point has for that service is equal to the number of instances for that service.

4.2.2 Proposal 2 (Multiple Service Descriptors with HSDir Selection)

Changes Required Alter the HSDir's such that they accept multiple descriptors for the same service, each descriptor could differ in the introduction points. Upon a request, a HSDir would return one of the set of known descriptors for that service.

Primary Goals

Goal 1: Achieved
As each client gets one descriptor out of a set of possible descriptors from the HSDir, different clients will connect to different instances.

Goal 2: Fail
If an instance goes down, that service will remain inaccessible until the client refetches the descriptor, and gets a descriptor corresponding with a working instance.

Secondary Goals

Goal 1: Fail
This can be determined with only Tor clients, as requesting the service descriptor a large number of times, and then counting the number of unique descriptors received would give you the number of instances.

Goal 2 Fail
If you cannot contact a hidden service through one descriptor, but it is available through another descriptor (with a different set of introduction points, then it is probably that the corresponding instance for the first descriptor has failed.

4.2.3 Proposal 3 (Combined Service Descriptors)

Changes Required The hidden service instances coordinate to produce a descriptor that contains all the introduction points for each instance, at which point it is published to the HSDir's.

A more complete design that fits in to this category is described in this proposal [7].

Primary Goals

Goal 1: Achieved
Depending on the introduction point chosen by the client, they will connect to a different instance.

Goal 2: Achieved

Clients will try each introduction point until a successful connection is established, so if there is one instance available, a client will try to connect to it.

Secondary Goals

Goal 1: Achieved

An upper bound is set by the number of introduction points in the descriptor, but depending on how many introduction points per instance, for n introduction points, you could have i introduction points, where $1 < i \leq n$.

Goal 2 Fail

If you try to continually connect to the service through all of the introduction points, and the service becomes inaccessible through a subset of those introduction points, it is probable that the subset of introduction points corresponds with a failed service instance.

4.2.4 Proposal 4 (HSDir Combined Service Descriptors)

Changes Required Alter the HSDir's such that they accept multiple descriptors for the same service, each descriptor could differ in the introduction points. Upon a request, a HSDir would return a descriptor that combines the introduction points of all the descriptors that it has revived.

Primary Goals

Goal 1: Achieved

Depending on the introduction point chosen by the client, they will connect to a different instance.

Goal 2: Achieved

Clients will try each introduction point until a successful connection is established, so if there is one instance available, a client will try to connect to it.

Secondary Goals

Goal 1: Fail (partial)

The HSDir's can tell how many instances there are by the number of unique descriptors that they recieve.

Goal 2 Fail

If you try to continually connect to the service through all of the introduction points, and the service becomes inaccessible through a subset of those introduction points, it is probable that the subset of introduction points corresponds with a failed service instance.

4.3 Analysis

With the elimination of proposal 2 due to it's failure for the second goal, there are three remaining proposals. One dividing factor is that proposal one is mainly centered

around the introduction points, and proposals three and four center around the hidden service directory servers.

With regards to future improvements, proposal one has more flexibility, as it has the client distribution on the introduction points, and not the directory servers. This has the advantage that the introduction points are in constant communication (have an open circuit) with each instance of the hidden service, whereas the hidden service directory servers are only contacted periodically or when necessary. This communication could allow for more advanced functionality in the future.

The coordination required between instances in proposal three could add much complexity to the design and implementation. Some functionality that could be provided by the introduction points, if connected to by multiple instances would also be impossible with this architecture.

Proposal four fails on the 2nd secondary goal, and while this is also the case for proposal one, in the case of proposal one, only the introduction points have this information, whereas for proposal four, this information can be acquired by anyone capable of interacting with the network.

4.3.1 Short Term Synchronisation for Proposal 1

Proposal one does not address the short term synchronisation of the set of introduction points. Assuming that you at some point have several instances connected to the same introduction points, a few events could trigger a particular instance to deviate from this set.

The most trivial event, would be the failure of one of the introduction points. A new introduction point would need to be chosen, which normally tor does randomly. If this took place, it would most probably result in all the instances using different introduction points. Like with the other design decisions, there are several different architectural techniques for solving problems such as this. Continuing with the loosely connected architecture, I designed an approach that allows each instance to pick a common new introduction point.

This works as follows. Assumptions:

Network Knowledge
: All nodes have knowledge of the other nodes in the network. For most of the time, this is the same between all the nodes.

Network State Information
: Information about the network state is local, depending on that nodes interaction with the rest of the network.

Consistant Pseudorandom Number
: Consistent and secure pseudo random numbers can be accessed by all nodes.

Introduction Point Suitability
: If a node can be used as an introduction point by one instance, it can be used by all instances.

There are a set of nodes that could act as introduction points. The set is assumed to differ between nodes due to state information. Each node accesses a consistent pseudorandom number (see assumption 3). The introduction points are then arranged

in to a ring by the identity. The introduction point with the identity $\geq$ the random number is selected.

Now for an initial selection, this might differ due to network state in the case where the chosen node is down (but not everyone knows this to be the case). If this happens, all the nodes which did not eliminate this node earlier, will now attempt contact, fail, then resume the process with the failed node eliminated. This process will iterate on each node, until all nodes have successfully established the necessary number of introduction points (or until all nodes in the network have been exhausted).

This method takes inspiration from the selection process used for hs directory servers and nickhopper's [2] suggestion of using a pseudo random function for selection of introduction points.

Currently, this cannot be implemented fully due to the lack a consistent and secure pseduorandom number generation. However, this may be a future feature, as a similar problem exists already in the hidden service architecture with the hidden service directories [4] (described in [2]).

4.4 Conclusion

In conclusion, I choose to proceed with an implementation of proposal one, as I feel for the reasons above that it holds the most promise with regards to the currently described functionality, and possible future functionality.

[2]Fri Dec 13 2013 - 18:33 #tor-dev on OFTC

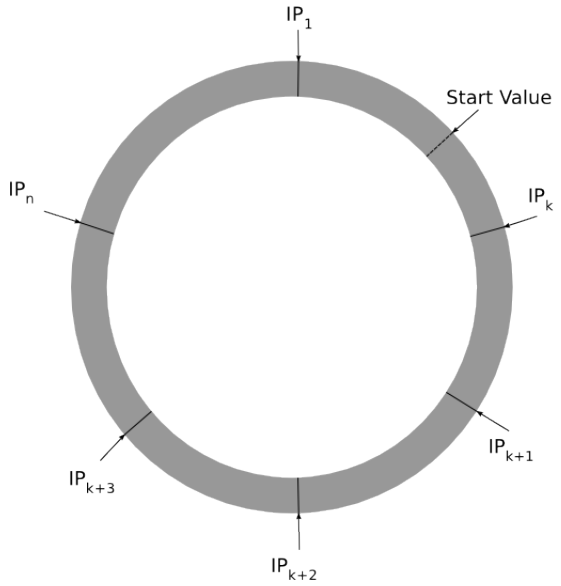


Figure 2: Introduction Point Hash Ring

5 Implementation

During the implementation phase of the project, I worked on tor [3] which is currently the most common Tor implementation (written in c).

For doing testing during development, and also for evaluation purposes, I needed a way of running a number of tor nodes, with some configuration with some way to also change and monitor their state. For this I chose to use a tool called chutney, it was not capable of the monitoring or changing of the network, so these were features which I added.

5.1 Modifications to tor

5.1.1 Remove the restriction on connections to an Introduction Point

In the specification [13, 1.2], when Introduction Points get a new connection from a hidden service, they should close any existing connections from that hidden service, I added a configuration option to disable this behaviour.

5.1.2 Remove Introduction Point Specific Keys

Normally, when tor connects to an introduction point, it would generate a key specific to that introduction point. This key is used to secure the connection between clients connecting to that introduction point, and the hidden service. The public part is published in the descriptor, such that clients can download it and secure the traffic. This presents difficulties for a loosely coordinated model, as the key needs to be known by each of the nodes.

The solution to this would be to generate the key in a deterministic way, based on the hidden service key (known to all nodes) and the identity of the introduction point. I choose not to do this, for several reasons. Practically, doing this well and correctly would have been a long and complex task, which would have provided little benefit to the overall project, as I can approximate this by simply using the long term key instead. Also, any work I did on the key infrastructure would probably be rendered obsolete, as the specification regarding the cryptography of hidden services looks to be soon to change.

5.1.3 Use one to many circuits when passing Introduction Requests

Instead of just looking for one circuit with a valid key, look for all circuits, then choose between them.

I implemented two different methods of distributing the rendezvous requests between the valid circuits. The first and simplest is a random approach. The second uses a round robin approach.

5.1.4 Initial Descriptor Check

When the service currently has no introduction points, instead of immediately choosing introduction points randomly, check for an existing descriptor, if one is found, use the introduction points that it contains. If a descriptor cannot be found, connect to a random set of introduction points.

[3] sometimes referred to as little-t tor, to separate it from the Tor project

5.1.5 Introduction Point Instance Selection

A random algorithm is currently in use for selecting what circuit to pass on a request through. A round robin algorithm was also implemented, and this can be activated through a configuration change.

5.1.6 Selecting New Introduction Points

Previously, introduction points were selected randomly. Now the algorithm described in section 4.3.1 is used, with the service digest standing in for a pseudo random number.

5.1.7 Introduction Point Reconnection

Previously, Tor would select a new introduction point if it lost a connection to a current introduction point. Loosing a connection could happen if a problem occurs at either the introduction point, or any of the relays in the 3 hop circuit used to connect to it.

This behaviour can lead to an unnecessary change of introduction point, as if a single relay in the circuit fails, a completely new introduction point is established, and the descriptor updated.

Now Tor will attempt to reconnect to the introduction point a number of times before looking to replace it.

5.2 Modifications to chutney

5.2.1 General Architectural Changes

To enable more flexible testing in chutney, I moved some of the control (e.g. starting nodes) from the library itself in to the test scripts. This flexibility allowed for more modular code (code can easily be shared between tests using modules).

5.2.2 Event based testing in Chutney

Allow for events in Chutney, this will be done by connecting a Tor controller to the nodes such that events can be triggered when there state changes. This will allow for quicker and more reproducible tests.

I chose to use the stem library for interacting with Tor, because it is written in python, and is one of the more popular Tor controller libraries. I add to chutney the ability to get at a stem controller for each node, such that these can be used in the test script

5.2.3 Dynamic Nodes in Chutney

Chutney does not support shunting down any nodes in the network once it has started, this is required for testing the behaviour when the set of available introduction points change.

To implement this, I moved the starting of the nodes to the test script, and added a start function that gets called when the script is started. This allowed the test script to start and stop nodes at any point during the test.

5.2.4 Other Smaller Changes

argparse Use a library to parse the command line arguments allowing for a –quiet option to suppress the test output, and allowing for the use of one or more files to start or configure multiple tests.

6 Testing

Due to the modifications mainly altering the network behaviour, testing was done at this level (using a complete network). Using chutney, several tests were written each aiming to rigorously test a specific part of the system.

6.1 General Test Setup

The nodes denoted hidden service nodes are setup with a common service key during the configuration phase.

Some tests use the clients in groups. This is due to the client maintaining a connection with the hidden service, so a group of clients can only be expected to reach instances which were active when the initial connections were made. Testing in this manor ensures that the service remains accessible irregardless of when the client initially connected.

6.1.1 Test Primitives

The Hidden Services All of the tests use hidden services, these services are all web servers operating through virtual port 80. They are attached to various ports on the host system to which tor is directed to forward connections. These web servers are run using Twisted directly from the python test itself. They each only have one response, the node number of the hidden service they are behind. This is used to establish what instance a client has connected to.

Instance Coverage Testing To check which instances are accessible, a number of clients attempt to connect to the service. The responses are all collected, and they indicate which instances were accessed.

Tracking Introduction Points All of the tests require the monitoring of which nodes each instance is using as introduction points.

Each test is written as a general test that can be run with a configurable number of instances, and the tests described below are specific configurations of the general tests, chosen to exercise relevant parts of the implementation being tested.

6.2 Startup (hs-start-3)

This test establishes 3 instances of the hidden service. The clients are divided in to 3 groups, one for each instance.

The instances are started one by one, after starting each instance, the introduction points of all active instances are checked to ensure that they are consistent. Connections are then made through each client group corresponding with an active instance (including the one just started), where the expectation is to be able to reach all instances that were active when the client group was originally used.

6.3 Introduction Point Failure (hs(-c-)intro-fail-n)

These tests (hs-intro-fail-2, hs-intro-fail-3, hs-dual-intro-fail-3, hs-tripple-intro-fail-3) start a network, and then stop c introduction points, checking that c new common

introduction points are established, and that all instances are still reachable by clients.

6.4 Introduction Point Candidate Failure (hs-intro-select-2)

This tests the hidden services response to a candidate introduction point being uncontactable. Tor is configured to output the next introduction points that it will choose, the next is then disabled, and then one of the current introduction points is disabled. The test checks that a new introduction point is successfully selected, and that all instances remain reachable.

6.5 Stopping (hs-stop-3)

This tests the process of stopping an instance, it checks that once the instance is stopped, all connections still succeed, and are routed to all remaining instances.

6.6 Test Coverage

These tests in combination cover the original goals. The startup and stopping tests cover the basic network setup and takedown. If these tests are passed, it demonstrates that it is possible to establish a distributed hidden service, and remove and add nodes while maintaining service operation.

The introduction point failure and candidate failure seek to establish whether a good state for the distributed hidden service can be maintained during normal network function. This normal function includes that failures can occur in several parts of the network, the introduction points directly and the other relay nodes used in communication with the introduction points.

7 Evaluation

7.1 Analysis and Design

Analysis began with attempting to determine the current state of distributed hidden services for Tor. Interaction with existing developers helped in this regard, and some simple testing confirmed the behaviour.

The analysis of the best course of action with respect of the goals was quite broad, as there was many different options, each with advantages, disadvantages and other ramifications.

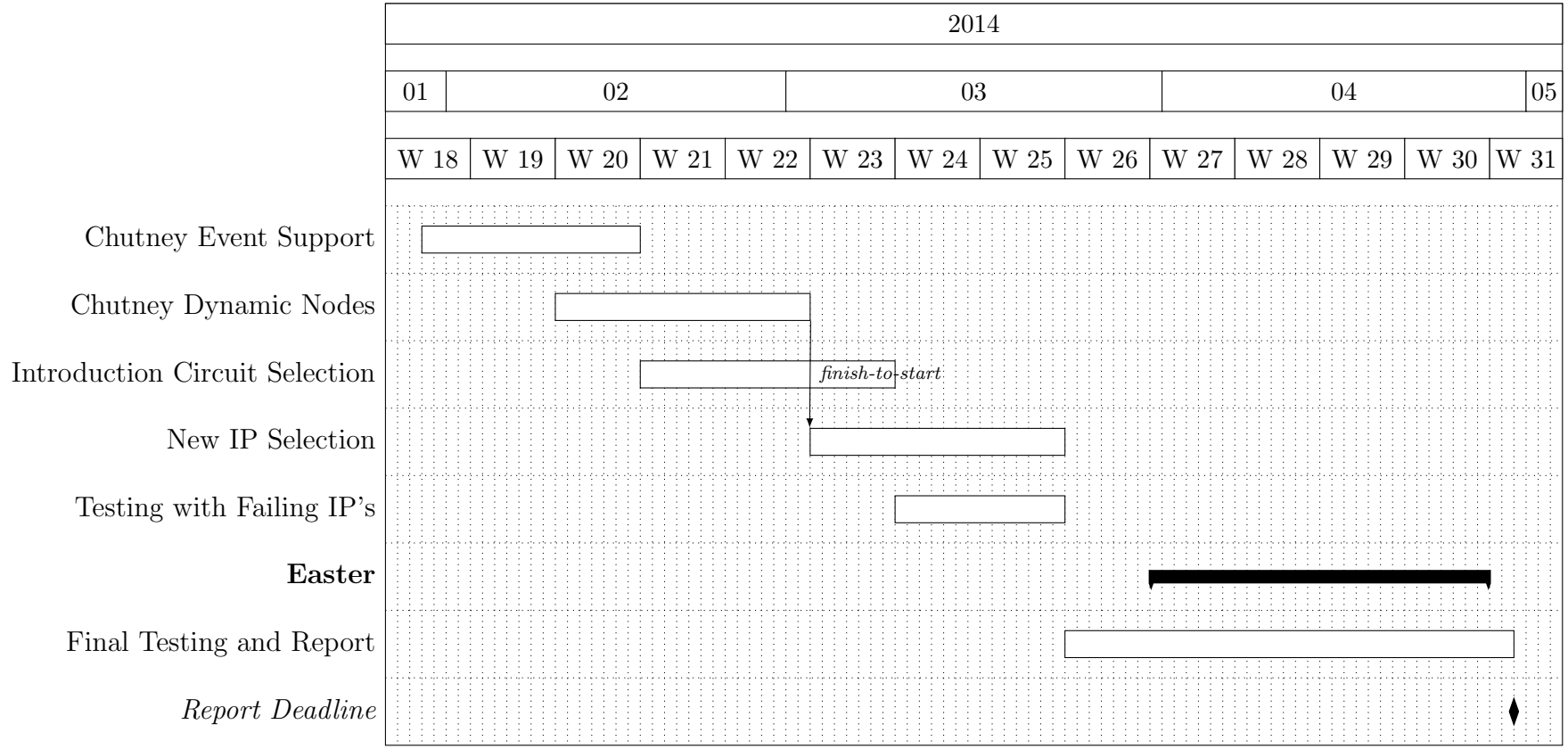
One error within the analysis was the failure to notice the nature of the introduction point keys, as this was only identified during the implementation phase. This only served to slow development slightly.

7.2 Implementation and Testing

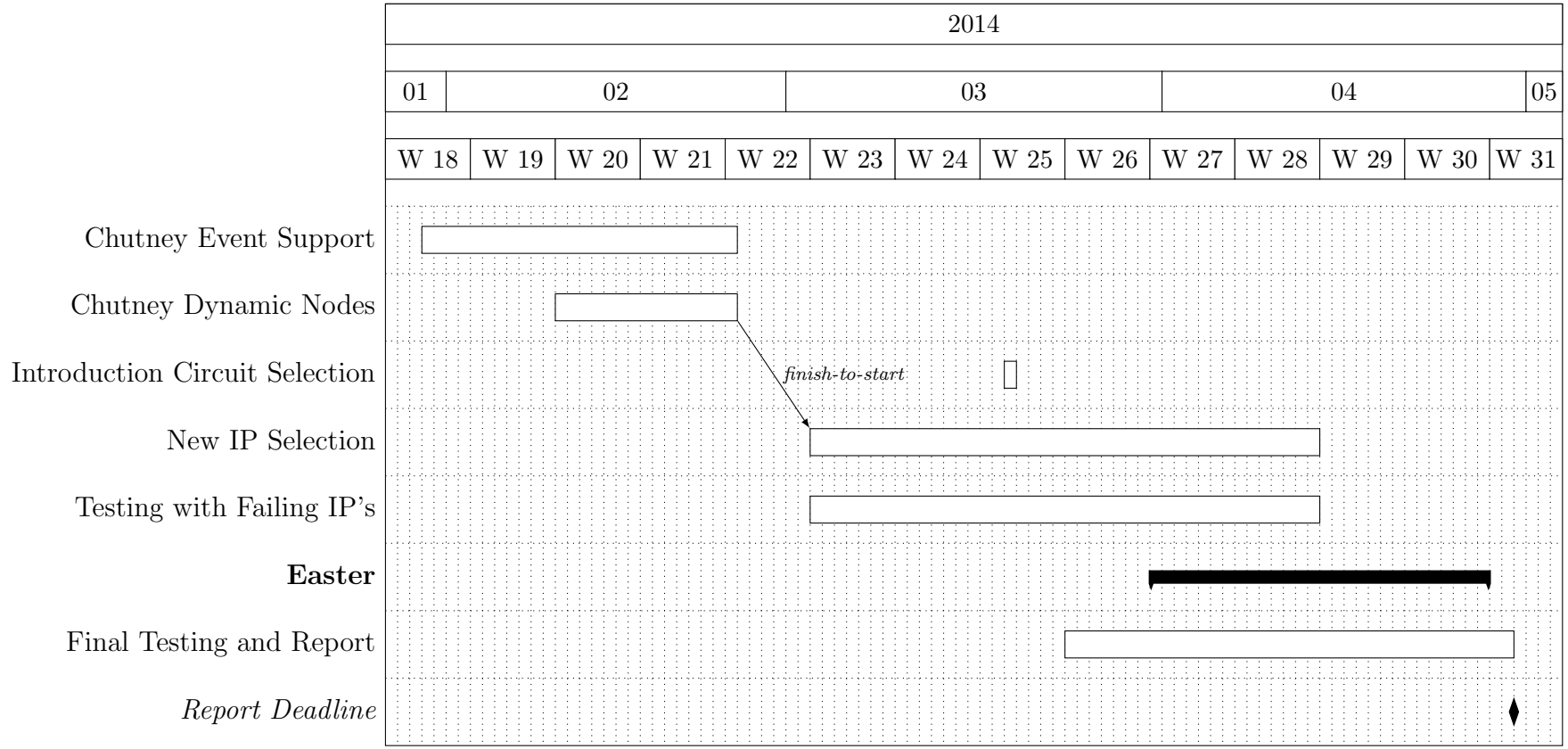
To reduce uncertainty and risk during the later parts of the project, key components were implemented early, and key ideas were tested early.

This approach was effective, as it lead to discovering some mistakes early, for example a misreading of the rend-spec document regarding the service keys. It also helped with the familiarity with the codebase, as at over 100,000 lines of code, it took time to become familiar with the complex existing behaviour.

7.3 Semester 2 Plan



7.4 Semester 2 Actual



The event and dynamic node support in chutney was implemented roughly on schedule, but this functionality was improved and refined as the project developed.

The Introduction Circuit Selection task took far less time than originally allocated as the desired functionality was scaled back, due to the issues that arose when beginning the implementation. The originally plan was to add a weighted round robin load balancing system on the introduction points. However, I later chose to not do this, as it became apparent that this alone would not help balance the load across the application instances.

This is due to the application being separate from the Tor relay which will be effectively accepting and rejecting clients. The Tor relay has no way of monitoring application load, and therefore cannot know when it needs to reject clients. One way of resolving this would be to have the application communicating with the relay over the control port to inform it when it wants to reject clients.

The introduction point selection ended up taking longer than expected due to mostly one major issue with unexpected network effects confusing the testing. The test would stop an introduction point that each instance was connected to, however this change would sometimes result in some of the instances disconnecting from some of the other introduction points to which they were connected. Initially this was investigated on a individual node by node basis, however, after getting the information out of Tor regarding the circuit states, it became apparent that this was in fact a problem not on the node, but caused by the network. The additional, unexplained disconnections were due to the circuits for those introduction points, using the introduction point that was deliberately disabled as a guard or middle relay in that circuit.

Far more rigorous testing was also performed on this section that originally planned, as it became apparent that testing multiple failures would be required to reasonably test the new functionality.

The final testing and report phase was on schedule.

7.5 Fulfillment of Goals

The implementation meets all the primary goals (as demonstrated by the tests), is promising concerning the secondary goals (in terms of future scope and compared to other considered solutions).

7.5.1 Usability Goals

The usability goals are an important addition to the technical goals, as this additional functionality needs to be accessible to those who might want to use it. Firstly, running a single instance hidden service has not changed, satisfying the first goal.

Setting up a distributed hidden service is very similar to setting up a single instance hidden service. The only difference being you need to start the second instance with the key generated by the first.

Removing an instance is even simpler, as you just stop tor (as you would do for a single instance service).

In the case of possible failures, information about the service is logged, and if any problems arise, this will be noted in the logs.

7.6 Known Issues

There are two major known issues that have been introduced by the changes made.

7.6.1 Use of Service Keys

Previously, Tor did use the public key for the Introduction Point connection. However, this was changed to single use service keys to help prevent the introduction point identifying the service. Due to the random nature of the key generation, the single use keys in there current form could not be used when multiple instances of a single service were connecting to a introduction point. Therefore, the previous behaviour (using the public key) was restored.

For large services, this might not be much of an issue, as it would be trivial for the introduction points to discover the service identity, and then lookup the introduction points from the hidden service directory servers. However, if a return to single use keys was desired, it may be possible to leverage the cryptography changes in the new rendezvous specification to achieved this [11].

7.6.2 Predictability of Introduction Points

This is a more serious issue that dramatically increases the chance that an attacker can position nodes under their control as introduction points for a service. However, this is less probable than the similar attack on hidden service directories, as the hidden service directories change regularly. Though for the same reason, this attack against the introduction points is more severe due to the introduction points changing less frequently.

To exploit this, an attacker would determine what region of the hash ring, that is used to select introduction points is being used. They would then create potential introduction points, with identities that fall in to this region of the hash ring. This increases the chance that these nodes will be selected as introduction points in the future. If these do get selected, the attacker can disable access to the service by not forwarding introduction requests onwards.

7.7 Deployment and Transition

To use this functionality, currently you would face one major issue. That is that all Tor nodes not running with the modifications made in this project would not allow simultaneous connections from multiple introduction points.

If these changes were adopted by the Tor project, there would be a couple of options for deploying this new functionality. Assuming that like previous functionality changes in the network, it would be necessary to have the network composed of both nodes capable of operating as introduction points accepting multiple connections, and not accepting multiple connections. Services looking to use this functionality would need to only select introduction points capable of accepting multiple connections.

The service could try to establish for each introduction point, if it is capable by connecting to it through two circuits using the same, key. If the first circuit is closed when the second is established, then the introduction points is unsuitable. This might have severe performance implications, as it would significantly increase the time to select introduction points, especially if the proportion of capable nodes in the network is low.

The other option is to specify the version or capability of a relay in this regard, either centrally, or during the communication with the prospective introduction point.

7.8 Comparison to existing tools and techniques

There are many existing software solutions and general techniques for distributing service load among several instances of that services. This project does not aim to replace these existing tools and techniques, but instead to provide some similar functionality within the Tor protocol.

In some circumstances, these protocol changes would be complemented by existing load balancing tools. Existing load balancers could be placed behind the Tor hidden service instances, to further distribute the load between additional servers, without adding any explicit extra overhead to the Tor network.

The approach designed here bears some similarities to both DNS round robin, and anycast. Clients will attempt to connect to a service using its .onion url, and like DNS round robin, will connect to one instance of that service. However, while DNS round robin distributes the load, in this model, the distribution takes place when the client attempts connection to the service. This difference does present some advantages over DNS round robin, as it removes any layered caching effects (described in [15]). However, similar effects of caching will be seen due to the caching of connections to hidden services by connecting relays.

While anycast normally distributes clients by geographical position, this has no use in Tor, as the circuits aim to route through different portions of the internet.

8 Conclusion

8.1 Project Achievements

I have successfully designed a protocol that achieves all of the primary goals of the project, and that is a good approach for the secondary goals. I then successfully implemented this in tor, and adapted chutney to be able to test these changes. The changes I made to chutney are also generally applicable to other functions within tor, and are not limited to this particular design or implementation of distributed hidden services.

Some issues were encountered that limit the usefulness of the implementation given here. This following section documents possible future improvements.

8.2 Future Work

8.2.1 Removing Introduction Point Predictability

Currently, with the modifications made, the next introduction point that each of the instances will choose, should they need to (e.g. to replace a failed introduction points) is predictable. This could be used by an attacker to take control of the introduction points for a service, and thereby preventing people from accessing it.

This new issue with the introduction points is very similar to a current issue with the hidden service directories, as they are currently predictable. Currently, discussion is underway concerning how to fix that issue [4], and it is very probably that any solution to the problem concerning the hidden service directories would be applicable to the introduction points also. This is due to the similarities in the algorithms now used to select both sets of nodes.

8.2.2 Introduction Point Load Balancing

Improving this requires a generic way for a service to inform Tor when it wants to reject or accept clients. This is such that Tor can inform the introduction point, and that introduction point can offer clients to multiple instances.

Once this is done, more complex algorithms can then be used on the introduction points to better balance the load between different instances of the service.

8.2.3 Better Introduction Point Circuit Routing

Tor allows for routing a circuit used to connect to an introduction point, through other introduction points. This can cause additional problems if the introduction point, which is also routed through fails, as it will break connectivity with both itself, and the other introduction point (which is functioning normally).

This could be avoided if tor only constructed circuits to introduction points that do not include other introduction points.

It could also avoid selecting introduction points where that node is used in a circuit to an existing introduction point, to again reduce the chance of loosing connection to two introduction points due to the failure of a single node common to both circuits.

8.2.4 Introduction Point Key Handling

Currently the key that is used to secure connections between the client and the hidden service through each introduction point is the hidden service key. This is due to the additional constraint that this key must be known to all nodes.

This provides the introduction point more information than it needs to know, and does know in the current single instance model.

To get back to the state where the key used for each introduction point is separate (and not the hidden service key), in the loosely connected model described, the key could be generated in a similar manor to which the introduction point is chosen. The introduction point is chosen using a deterministic, time dependant process.

This was not applicable to look at during the project, as the current cryptographic architecture for Tor hidden services was undergoing major changes at the time [11].

8.2.5 Operator Deployment Toolkit

In developing tor to support distributed hidden services, I tested by setting up each hidden service instance to be identifiable by responding uniquely to a request. By then connecting to the hidden service, I could verify that all the instances are contactable.

For those operating distributerd hidden services, it would be useful to have software that would perform the verification part of this, by connecting to each introduction point multiple times to verify that each instance is reachable.

References

[1] J. Abley and K. Lindqvist. Operation of Anycast Services. RFC 4786 (Best Current Practice), December 2006.

[2] Alex Biryukov, Ivan Pustogarov, and Ralf-Philipp Weinmann. Trawling for tor hidden services: Detection, measurement, deanonymization. In *Proceedings of the 2013 IEEE Symposium on Security and Privacy*, May 2013.

[3] T. Brisco. DNS Support for Load Balancing. RFC 1794 (Informational), April 1995.

[4] Roger Dingledine. The hsdirs for a hidden service should not be predictable indefinitely into the future. `https://trac.torproject.org/projects/tor/ticket/8244`, Feb 2013.

[5] Roger Dingledine and Nick Mathewson. Anonymity loves company: Usability and the network effect. In Ross Anderson, editor, *Proceedings of the Fifth Workshop on the Economics of Information Security (WEIS 2006)*, June 2006.

[6] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The second-generation onion router. In *Proceedings of the 13th USENIX Security Symposium*, August 2004.

[7] Matthew Finkel. [draft proposal] scalable hidden services. `https://gitweb.torproject.org/user/sysrqb/torspec.git/blob_plain/a2f8e0c87f4e6e62ad56dc943b95170e2b403429:/proposals/ideas/xxx-hs-scalability.txt`, October 2013.

[8] Jörg Lenhard, Karsten Loesing, and Guido Wirtz. Performance measurements of tor hidden services in low-bandwidth access networks. In Michel Abdalla, David Pointcheval, Pierre-Alain Fouque, and Damien Vergnaud, editors, *Proceedings of the 7th International Conference on Applied Cryptography and Network Security (ACNS 09), Paris-Rocquencourt, France, June 2-5, 2009*, volume 5536 of *Lecture Notes in Computer Science*, June 2009.

[9] Karsten Loesing, Werner Sandmann, Christian Wilms, and Guido Wirtz. Performance Measurements and Statistics of Tor Hidden Services. In *Proceedings of the 2008 International Symposium on Applications and the Internet (SAINT)*. IEEE CS Press, July 2008.

[10] Nick Mathewson. [tor-dev] hidden service scaling. `https://lists.torproject.org/pipermail/tor-dev/2013-October/005566.html`, October 2013.

[11] Nick Mathewson. Next-generation hidden services in tor. `https://gitweb.torproject.org/torspec.git/blob_plain/97062e055d0992356ba61dfe605d40b10b90ccff:/proposals/224-rend-spec-ng.txt`, April 2014.

[12] Lasse Øverlier and Paul Syverson. Valet services: Improving hidden servers with a personal touch. In George Danezis and Philippe Golle, editors, *Proceedings of the Sixth Workshop on Privacy Enhancing Technologies (PET 2006)*, pages 223–244. Springer, June 2006.

[13] Nick Mathewson Roger Dingledine. Tor rendezvous specification. https://gitweb.torproject.org/torspec.git/blob_plain/6c974c54f190e9e0d75d37ec999f6a354d9fbc36:/rend-spec.txt, Sep 2013.

[14] Nick Mathewson Roger Dingledine. Tor protocol specification. https://gitweb.torproject.org/torspec.git/blob_plain/7901fc11a9ecc6e857bf860fecb5ed25bd073378:/rend-spec.txt, April 2014.

[15] Zhong Xu, Rong Huang, and Laxmi N Bhuyan. Load balancing of dns-based distributed web server systems with page caching. In *Parallel and Distributed Systems, 2004. ICPADS 2004. Proceedings. Tenth International Conference on*, pages 587–594. IEEE, 2004.

List of Abbreviations and Key Terms

circuit A sequence of OR's, through which traffic can be sent in fixed size cells

HSDir Flag denoting a OR to be a hidden service directory, used to describe OR's with this tag

instance In the context of Tor Hidden Services, used to refer to a node in the Tor network providing access to a hidden service

introduction point (IP) Allows clients to pass on rendezvous requests to the hidden service, without knowing its location

OP (onion proxy) like an onion router, but only handles local requests [14, 2]

OR Onion Router

Rendezvous Point (RP) Serves to relay traffic between the client and hidden service

service key described in the rend-spec as the key involved in establishing an introduction point, and publishing a descriptor (the public part of the service-key corresponding with each introduction point is included in the descriptor)

Tor The Onion Router

A Brief

Enhancing Tor Hidden Services

Christopher Baines
cb15g11@soton.ac.uk

Project Supervisor: Dr Tim Chown
tjc@ecs.soton.ac.uk

April 6, 2014

1 Description

Tor is an low latency, onion routing system. One particular Tor feature is hidden services, these provide responder anonymity, this means the identity of the server providing the service, is hidden from the requester (user) of the service.

In contrast to modern web services, which can use anycast and the domain name system to help with their scalability and redundancy, the main load for a Tor hidden service, will be just directed at one server. This also has implications for the anonymity of the hidden service, as downtime can possibly reveal information about its real world location.

2 Goals

I aim to modify Tor to allow for distributed hidden services. This will load balance across the different instances, and clients should be able to connect to the service if at least one instance is running.

I also aim to at least preserve the anonymity that hidden services provide, and hopefully increase it.

3 Scope

I aim to develop at least one working prototype, a full description of the changes made, and the reasoning behind them. While I aim to produce something that can be merged back in to the Tor project, this is outside my control, and therefore outside the scope of the project.

B Design Archive

Work done on both projects (tor and chutney) was committed to the respective projects git repositories. These have been bundled up for inclusion in the design archive.

To clone the repositories. Unpack the archive, you should find two files.

tor.bundle The bundle for the tor repository.

chutney.bundle The bundle for the chutney repository.

Then run:

```
git clone -b disths tor.bundle
git clone -b disths chutney.bundle
```

This will create two directories, tor and chutney containing the respective projects.

C Connecting to the Internet through the Tor Network

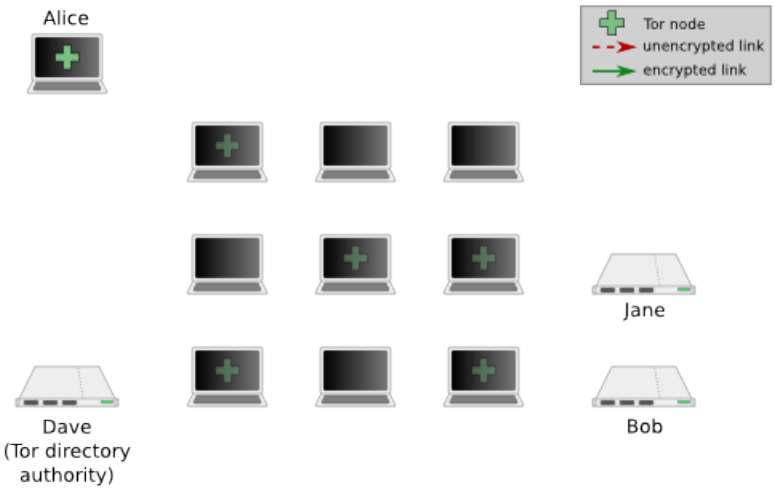


Figure 3: Initial State

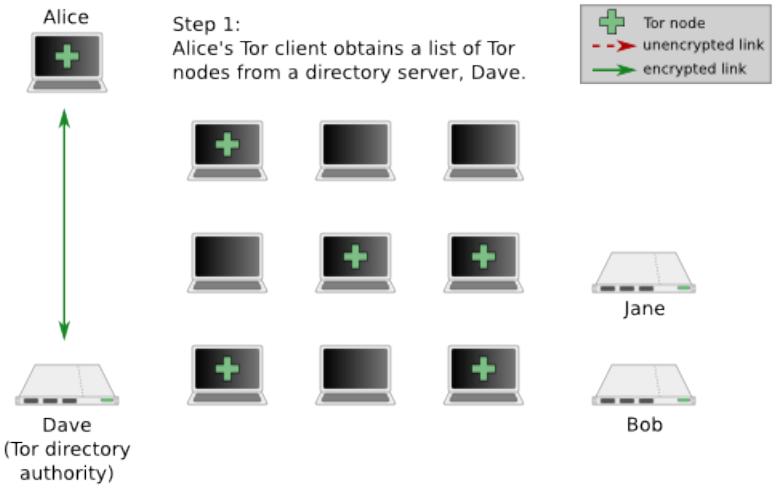


Figure 4: Step 1

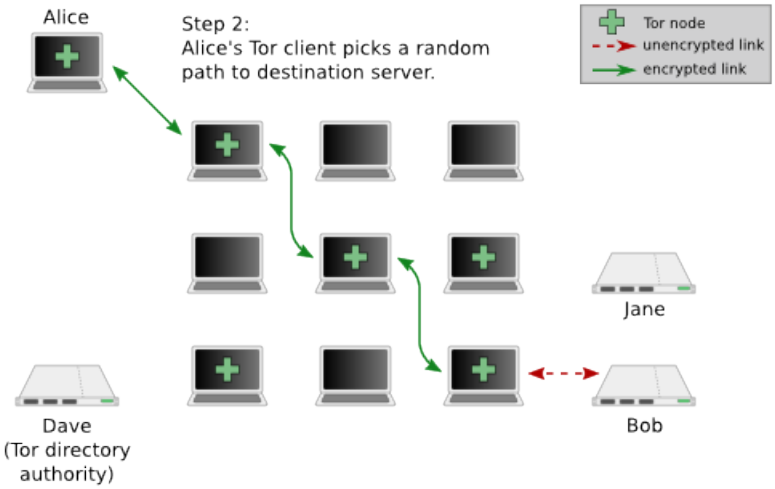


Figure 5: Step 2

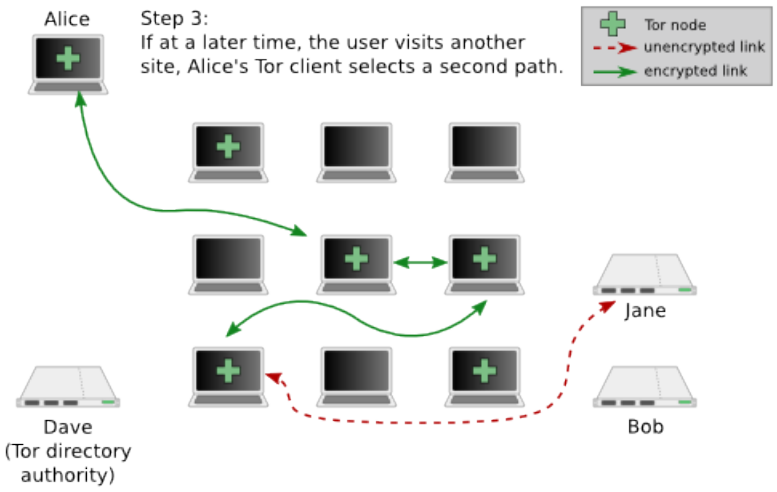


Figure 6: Step 3

D Hidden Service Setup

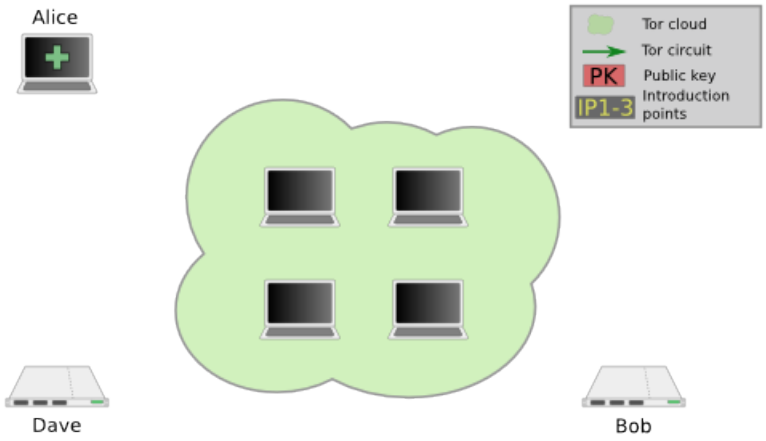


Figure 7: Initial State

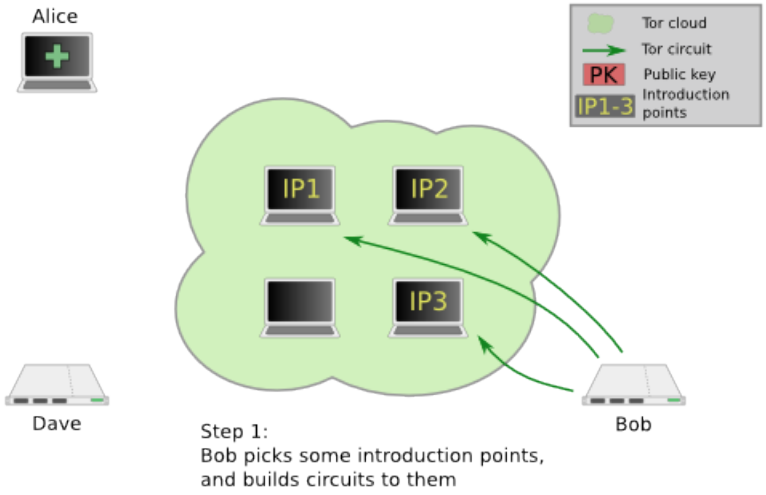


Figure 8: Step 1

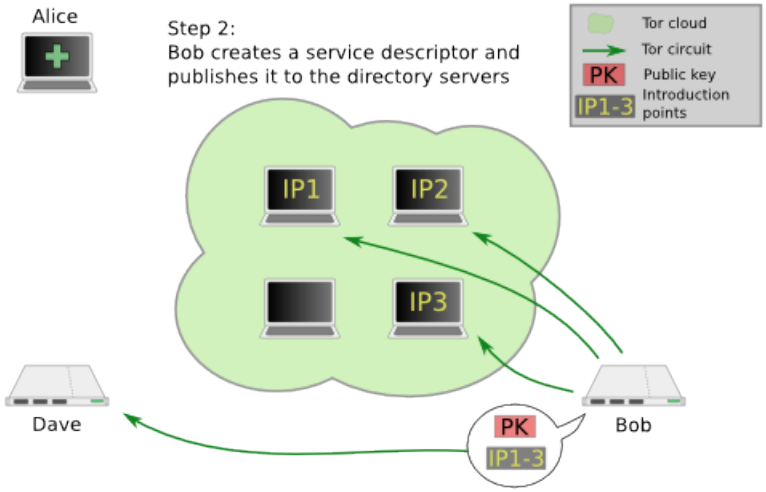


Figure 9: Step 2

E Hidden Service Connection

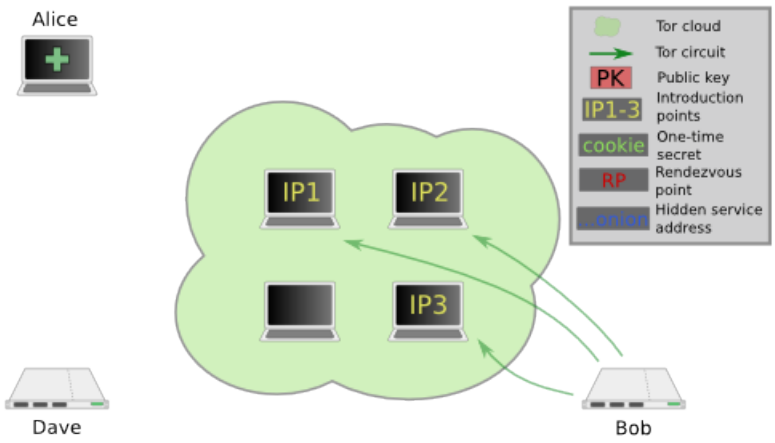


Figure 10: Initial State

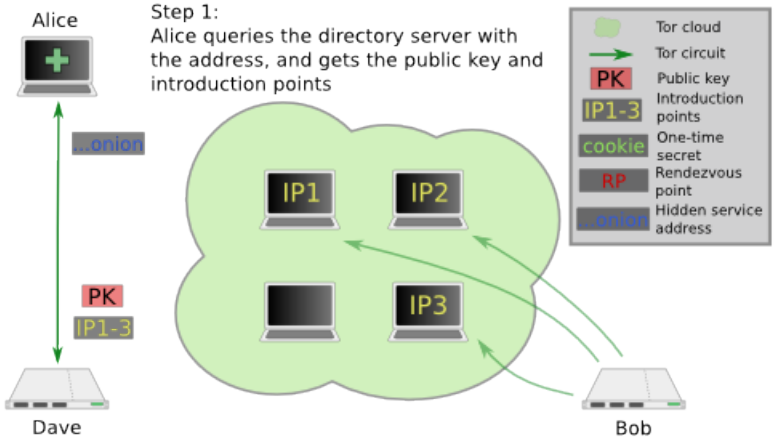


Figure 11: Step 1

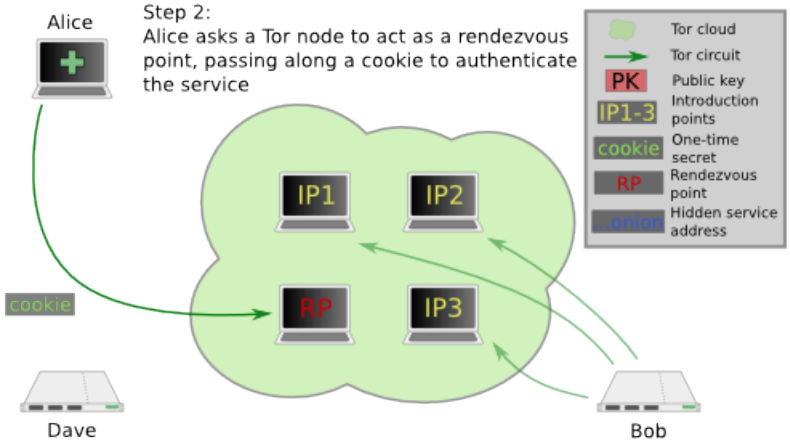


Figure 12: Step 2

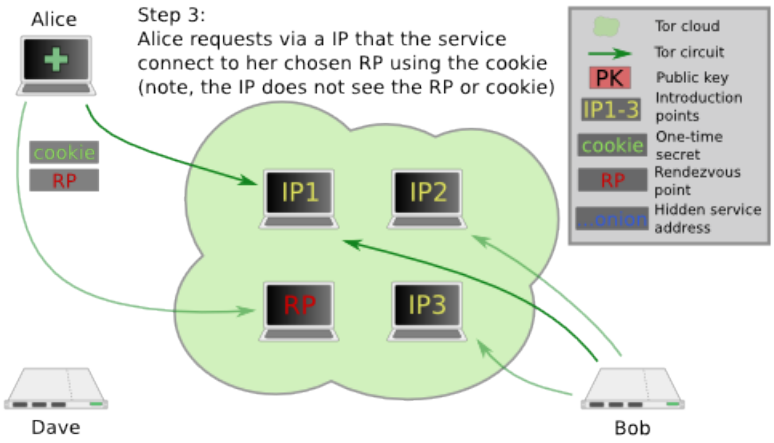


Figure 13: Step 3

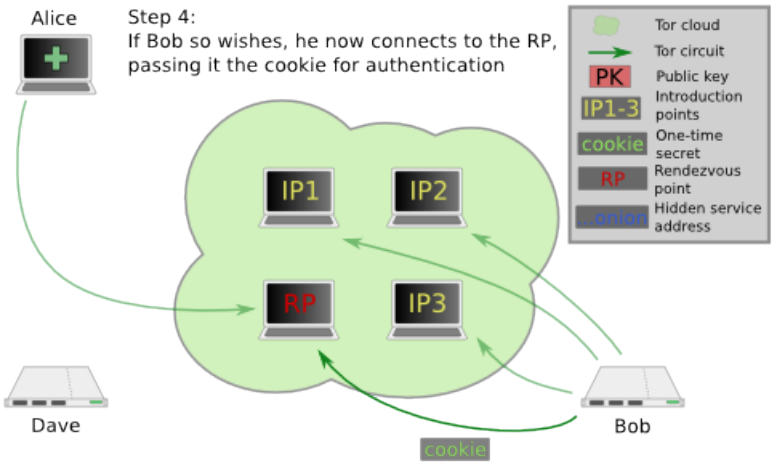


Figure 14: Step 4

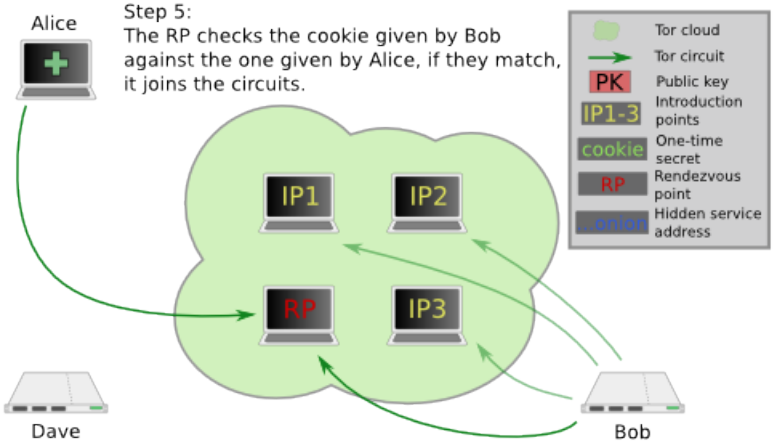


Figure 15: Step 5