Technical Documentation

I2P Technical Overview

Intro and Threat Analysis
Overview of the network and its capabilities

Specifications and Protocol Stack
Network architecture

Roadmap and Proposals
Development roadmaps

I2P Application Layer

App Development Overview and Guide
Tagline here

App Layer API and Protocols
High-level, easy to use APIs for applications written in any language to send and receive data

End-to-End Transport API and Protocols
The end-to-end protocols used by clients for reliable and unreliable communication.

Client-to-Router Interface API and Protocol
The end-to-end protocols used by clients for reliable and unreliable communication

I2P Network Protocol Documentation

End-to-End Encryption
How client messages are end-to-end encrypted by the router

Network Database
Distributed storage and retrieval of information about routers and clients

Router Message Protocol
I2P is a message-oriented router. The messages sent between routers are defined by the I2NP protocol.

Transport Layer
The protocols for direct (point-to-point) router to router communication

Tunnels
Selecting peers, requesting tunnels through peers, encrypting and routing messages through these tunnels.

Other Router Topics
Tagline here

I2P Research Documentation

Academic Research
Tagline here

Academic Papers and Peer Review
Tagline here

Open Research Topics
Tagline here

I2P Metrics
Tagline here

Vulnerability Response Process
Tagline here

Technical Introduction

Intro

I2P is a scalable, self organizing, resilient packet switched anonymous network layer, upon which any number of different anonymity or security conscious applications can operate. Each of these applications may make their own anonymity, latency, and throughput tradeoffs without worrying about the proper implementation of a free route mixnet, allowing them to blend their activity with the larger anonymity set of users already running on top of I2P.

Applications available already provide the full range of typical Internet activities - **anonymous** web browsing, web hosting, chat, file sharing, e-mail, blogging and content syndication, newsgroups, as well as several other applications under development.

- Web browsing: using any existing browser that supports using a proxy.
- Chat: IRC, Jabber, I2P-Messenger.
- File sharing: I2PSnark, Robert, iMule, I2Phex, PyBit, I2P-bt and others.
- E-mail: susimail and I2P-Bote.
- Blog: using e.g. the pebble plugin or the distributed blogging software Syndie.
- Distributed Data Store: Save your data redundantly in the Tahoe-LAFS cloud over I2P.
- Newsgroups: using any newsgroup reader that supports using a proxy.

Unlike web sites hosted within content distribution networks like Freenet or GNUnet, the services hosted on I2P are fully interactive - there are traditional web-style search engines, bulletin boards, blogs you can comment on, database driven sites, and bridges to query static systems like Freenet without needing to install it locally.

With all of these anonymity enabled applications, I2P takes on the role of the message oriented middleware - applications say that they want to send some data to a cryptographic identifier (a "destination") and I2P takes care of making sure it gets there securely and anonymously. I2P also bundles a simple streaming library to allow I2P's anonymous best-effort messages to transfer as reliable, in-order streams, transparently offering a TCP based congestion control algorithm tuned for the high bandwidth delay product of the network. While there have been several simple SOCKS proxies available to tie existing applications into the network, their value has been limited as nearly every application routinely exposes what, in an anonymous context, is sensitive information. The only safe way to go is to fully audit an application to ensure proper operation and to assist in that we provide a series of APIs in various languages which can be used to make the most out of the network.

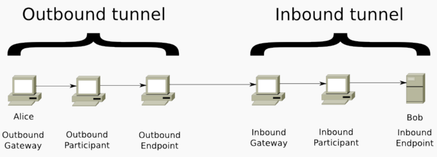
I2P is not a research project - academic, commercial, or governmental, but is instead an engineering effort aimed at doing whatever is necessary to provide a sufficient level of anonymity to those who need it. It has been in active development since early 2003 with one full time developer and a dedicated group of part time contributors from all over the world. All of the work done on I2P is open source and freely available on the website, with the majority of the code released outright into the public domain, though making use of a few cryptographic routines under BSD-style licenses. The people working on I2P do not control what people release client applications under, and there are several GPL'ed applications available (I2PTunnel, susimail, I2PSnark, I2P-Bote, I2Phex and others.). Funding for I2P comes entirely from donations, and does not receive any tax breaks in any jurisdiction at this time, as many of the developers are themselves anonymous.

Operation

Overview

To understand I2P's operation, it is essential to understand a few key concepts. First, I2P makes a strict separation between the software participating in the network (a "router") and the anonymous endpoints ("destinations") associated with individual applications. The fact that someone is running I2P is not usually a secret. What is hidden is information on what the user is doing, if anything at all, as well as what router a particular destination is connected to. End users will typically have several local destinations on their router - for instance, one proxying in to IRC servers, another supporting the user's anonymous webserver ("I2P Site"), another for an I2Phex instance, another for torrents, etc.

Another critical concept to understand is the "tunnel". A tunnel is a directed path through an explicitly selected list of routers. Layered encryption is used, so each of the routers can only decrypt a single layer. The decrypted information contains the IP of the next router, along with the encrypted information to be forwarded. Each tunnel has a starting point (the first router, also known as "gateway") and an end point. Messages can be sent only in one way. To send messages back, another tunnel is required.



Two types of tunnels exist: **"outbound" tunnels** send messages away from the tunnel creator, while **"inbound" tunnels** bring messages to the tunnel creator. Combining these two tunnels allows users to send messages to each other. The sender ("Alice" in the above image) sets up an outbound tunnel, while the receiver ("Bob" in the above image) creates an inbound tunnel. The gateway of an inbound tunnel can receive messages from any other user and will send them on until the endpoint ("Bob"). The endpoint of the outbound tunnel will need to send the message on to the gateway of the inbound tunnel. To do this, the sender ("Alice") adds instructions to her encrypted message. Once the endpoint of the outbound tunnel decrypts the message, it will have instructions to forward the message to the correct inbound gateway (the gateway to "Bob").

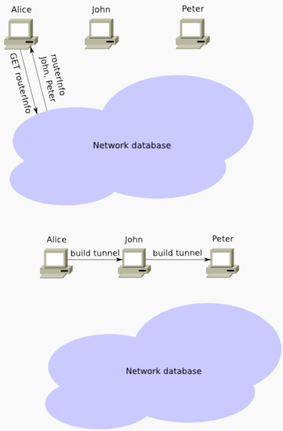
A third critical concept to understand is I2P's **"network database"** (or "netDb") - a pair of algorithms used to share network metadata. The two types of metadata carried are **"routerInfo"** and **"leaseSets"** - the routerInfo gives routers the data necessary for contacting a particular router (their public keys, transport addresses, etc), while the leaseSet gives routers the information necessary for contacting a particular destination. A leaseSet contains a number of "leases". Each of this leases specifies a tunnel gateway, which allows reaching a specific destination. The full information contained in a lease:Inbound gateway for a tunnel that allows reaching a specific destination

- Time when a tunnel expires
- Pair of public keys to be able to encrypt messages (to send through the tunnel and reach the destination).

Routers themselves send their routerInfo to the netDb directly, while leaseSets are sent through outbound tunnels (leaseSets need to be sent anonymously, to avoid correlating a router with his leaseSets).

We can combine the above concepts to build successful connections in the network.

To build up her own inbound and outbound tunnels, Alice does a lookup in the netDb to collect routerInfo. This way, she gathers lists of peers she can use as hops in her tunnels. She can then send a build message to the first hop, requesting the construction of a tunnel and asking that router to send the construction message onward, until the tunnel has been constructed.



When Alice wants to send a message to Bob, she first does a lookup in the netDb to find Bob's leaseSet, giving her his current inbound tunnel gateways. She then picks one of her outbound tunnels and sends the message down it with instructions for the outbound tunnel's endpoint to forward the message on to one of Bob's inbound tunnel gateways. When the outbound tunnel endpoint receives those instructions, it forwards the message as requested, and when Bob's inbound tunnel gateway receives it, it is forwarded down the tunnel to Bob's router. If Alice wants Bob to be able to reply to the message, she needs to transmit her own destination explicitly as part of the message itself. This can be done by introducing a higher-level layer, which is done in the streaming library. Alice may also cut down on the response time by bundling her most recent LeaseSet with the message so that Bob doesn't need to do a netDb lookup for it when he wants to reply, but this is optional.

And More on……https://geti2p.net/en/docs/how/tech-intro

Threat Model

Threat Model and Analysis

What do we mean by “anonymous”?

Your level of anonymity can be described as "how hard it is for someone to find out information you don't want them to know?" - who you are, where you are located, who you communicate with, or even when you communicate. "Perfect" anonymity is not a useful concept here - software will not make you indistinguishable from people that don't use computers or who are not on the Internet. Instead, we are working to provide sufficient anonymity to meet the real needs of whomever we can - from those simply browsing websites, to those exchanging data, to those fearful of discovery by powerful organizations or states.

The question of whether I2P provides sufficient anonymity for your particular needs is a hard one, but this page will hopefully assist in answering that question by exploring how I2P operates under various attacks so that you may decide whether it meets your needs.

We welcome further research and analysis on I2P's resistance to the threats described below. More review of existing literature (much of it focused on Tor) and original work focused on I2P is needed.

Network Topology Summary

I2P builds off the ideas of many other systems, but a few key points should be kept in mind when reviewing related literature:
I2P is a free route mixnet - the message creator explicitly defines the path that messages will be sent out (the outbound tunnel), and the message recipient explicitly defines the path that messages will be received on (the inbound tunnel).
I2P has no official entry and exit points - all peers fully participate in the mix, and there are no network layer in- or out-proxies (however, at the application layer, a few proxies do exist)
I2P is fully distributed - there are no central controls or authorities. One could modify some routers to operate mix cascades (building tunnels and giving out the keys necessary to control the forwarding at the tunnel endpoint) or directory based profiling and selection, all without breaking compatibility with the rest of the network, but doing so is of course not necessary (and may even harm one's anonymity).

We have documented plans to implement nontrivial delays and batching strategies whose existence is only known to the particular hop or tunnel gateway that receives the message, allowing a mostly low latency mixnet to provide cover traffic for higher latency communication (e.g. email). However we are aware that significant delays are required to provide meaningful protection, and that implementation of such delays will be a significant challenge. It is not clear at this time whether we will actually implement these delay features.

In theory, routers along the message path may inject an arbitrary number of hops before forwarding the message to the next peer, though the current implementation does not.

The Threat Model

I2P design started in 2003, not long after the advent of [Onion Routing], [Freenet], and [Tor]. Our design benefits substantially from the research published around that time. I2P uses several onion routing techniques, so we continue to benefit from the significant academic interest in Tor.

Taking from the attacks and analysis put forth in the anonymity literature (largely Traffic Analysis: Protocols, Attacks, Design Issues and Open Problems), the following briefly describes a wide variety of attacks as well as many of I2Ps defenses. We update this list to include new attacks as they are identified.

Included are some attacks that may be unique to I2P. We do not have good answers for all these attacks, however we continue to do research and improve our defenses.

In addition, many of these attacks are significantly easier than they should be, due to the modest size of the current network. While we are aware of some limitations that need to be addressed, I2P is designed to support hundreds of thousands, or millions, of participants. As we continue to spread the word and grow the network, these attacks will become much harder.

The network comparisons and "garlic" terminology pages may also be helpful to review.

Brute Force Attacks

A brute force attack can be mounted by a global passive or active adversary, watching all the messages pass between all of the nodes and attempting to correlate which message follows which path. Mounting this attack against I2P should be nontrivial, as all peers in the network are frequently sending messages (both end to end and network maintenance messages), plus an end to end message changes size and data along its path. In addition, the external adversary does not have access to the messages either, as inter-router communication is both encrypted and streamed (making two 1024 byte messages indistinguishable from one 2048 byte message).

However, a powerful attacker can use brute force to detect trends - if they can send 5GB to an I2P destination and monitor everyone's network connection, they can eliminate all peers who did not receive 5GB of data. Techniques to defeat this attack exist, but may be prohibitively expensive (see: Tarzan's mimics or constant rate traffic). Most users are not concerned with this attack, as the cost of mounting it are extreme (and often require illegal activity). However, the attack is still possible, for example by an observer at a large ISP or an Internet exchange point. Those who want to defend against it would want to take appropriate countermeasures, such as setting low bandwidth limits, and using unpublished or encrypted leasesets for I2P Sites. Other countermeasures, such as nontrivial delays and restricted routes, are not currently implemented.

As a partial defense against a single router or group of routers trying to route all the network's traffic, routers contain limits as to how many tunnels can be routed through a single peer. As the network grows, these limits are subject to further adjustment. Other mechanisms for peer rating, selection and avoidance are discussed on the peer selection page.

Timing Attacks

A I2P's messages are unidirectional and do not necessarily imply that a reply will be sent. However, applications on top of I2P will most likely have recognizable patterns within the frequency of their messages - for instance, an HTTP request will be a small message with a large sequence of reply messages containing the HTTP response. Using this data as well as a broad view of the network topology, an attacker may be able to disqualify some links as being too slow to have passed the message along.

This sort of attack is powerful, but its applicability to I2P is non obvious, as the variation on message delays due to queuing, message processing, and throttling will often meet or exceed the time of passing a message along a single link - even when the attacker knows that a reply will be sent as soon as the message is received. There are some scenarios which will expose fairly automatic replies though - the streaming library does (with the SYN+ACK) as does the message mode of guaranteed delivery (with the DataMessage+DeliveryStatusMessage).

Without protocol scrubbing or higher latency, global active adversaries can gain substantial information. As such, people concerned with these attacks could increase the latency (using nontrivial delays or batching strategies), include protocol scrubbing, or other advanced tunnel routing techniques, but these are unimplemented in I2P.

References: Low-Resource Routing Attacks Against Anonymous Systems

Intersection Attacks

Intersection attacks against low latency systems are extremely powerful - periodically make contact with the target and keep track of what peers are on the network. Over time, as node churn occurs the attacker will gain significant information about the target by simply intersecting the sets of peers that are online when a message successfully goes through. The cost of this attack is significant as the network grows, but may be feasible in some scenarios.

In summary, if an attacker is at both ends of your tunnel at the same time, he may be successful. I2P does not have a full defense to this for low latency communication. This is an inherent weakness of low-latency onion routing. Tor provides a similar disclaimer.

- Partial defenses implemented in I2P
- strict ordering of peers
- peer profiling and selection from a small group that changes slowly
- Limits on the number of tunnels routed through a single peer
- Prevention of peers from the same /16 IP range from being members of a single tunnel
- For I2P Sites or other hosted services, we support simultaneous hosting on multiple routers, or multihoming

Even in total, these defenses are not a complete solution. Also, we have made some design choices that may significantly increase our vulnerability:
- We do not use low-bandwidth "guard nodes”
- We use tunnel pools comprised of several tunnels, and traffic can shift from tunnel to tunnel.

Tunnels are not long-lived; new tunnels are built every 10 minutes.
Tunnel lengths are configurable. While 3-hop tunnels are recommended for full protection, several applications and services use 2-hop tunnels by default.In the future, it could for peers who can afford significant delays (per nontrivial delays and batching strategies). In addition, this is only relevant for destinations that other people know about - a private group whose destination is only known to trusted peers does not have to worry, as an adversary can't "ping" them to mount the attack.

Reference: One Cell Enough

Specifications

This page provides the specifications for various components of the I2P network and router software. These are living documents, and the specifications are updated as modifications are made to the network and software. The proposal documents that track changes to these specifications can be viewed here.

•"Last updated" is the last date when the specification given within a document was altered in any way, except for changes to the "accurate for" information.
•The "accurate for" column gives the version of the I2P network and reference Java implementation that the document is verified to be valid for. Because the documents are usually only updated when changes are made, the listed versions can sometimes be several releases behind. This does not mean that documents with old listed versions are necessarily inaccurate, but small differences may creep in during the course of development. Periodic reviews are conducted to update the "accurate for" information.

The I2P Project is committed to maintaining accurate, current documentation. If you find any inaccuracies in the documents linked below, please enter a ticket identifying the problem.

Title	Category	Last updated	Accurate for	Link
Common structures Specification	Design	2021-04	0.9.49	HTML I TXT
Low-level Cryptography Specification	Design	2020-09	0.9.47	HTML I TXT
Tunnel Creation Specification	Design	July 2019	0.9.41	HTML I TXT
Tunnel Message Specification	Design	2021-01	0.9.49	HTML I TXT
NTCP 2	Transports	2021-03	0.9.50	HTML I TXT
SSU Protocol Specification	Transports	2021-06	0.9.50	HTML I TXT
Datagram Specification	Protocols	February 2019	0.9.39	HTML I TXT
ECIES-X25519-AEAD-Ratchet	Protocols	2020-11	0.9.47	HTML I TXT
Encrypted LeaseSet Specification	Protocols	June 2019	0.9.41	HTML I TXT
I2CP Specification	Protocols	2020-11	0.9.48	HTML I TXT
I2NP Specification	Protocols	2021-07	0.9.51	HTML I TXT
Streaming Library Specification	Protocols	May 2020	0.9.46	HTML I TXT
Access Filter Format Specification		April 2019	0.9.40	HTML I TXT
Addressbook Subscription Feed Commands		2021-01	0.9.49	HTML I TXT
B32 for Encrypted Leasesets		2020-08	0.9.47	HTML I TXT
Blockfile and Hosts Database Specification		2020-09	0.9.47	HTML I TXT
Configuration File Specification		March 2020	0.9.45	HTML I TXT
ECIES-X25519 Router Messages		None	None	HTML I TXT
ECIES-X25519 Tunnel Creation		None	None	HTML I TXT
GeoIP File Specification		December 2013	0.9.9	HTML I TXT
Plugin Specification		November 2019	0.9.43	HTML I TXT
Red25519 Signature Scheme		2020-08	0.9.47	HTML I TXT
Software Update Specification		June 2021	0.9.51	HTML I TXT

Other Specification Documents

These will eventually be migrated to the new specifications system.

- ElGamal/AES+SessionTags
- NTCP
- I2PControl
- SAM v3
- BOB
- Bittorrent
- Naming and Address Book

Protocol Stack Chart

Here is the protocol stack for I2P. See also the Index to Technical Documentation.

Each of the layers in the stack provides extra capabilities. The capabilities are listed below, starting at the bottom of the protocol stack.

- **Internet Layer:**

IP: Internet Protocol, allow addressing hosts on the regular internet and routing packets across the internet using best-effort delivery.

- **Transport Layer:**
  TCP: Transmission Control Protocol, allow reliable, in-order delivery of packets across the internet.
  UDP: User Datagram Protocol, allow unreliable, out-of-order delivery of packets across the internet.
- **I2P Transport Layer:** provide encrypted connections between 2 I2P routers. These are not anonymous yet, this is strictly a hop-to-hop connection. Two protocols are implemented to provide these capabilities. NTCP builds on top of TCP, while SSU uses UDP.
  NTCP: NIO-based TCP
  SSU: Secure Semi-reliable UDP
- **I2P Tunnel Layer:** provide full encrypted tunnel connections.
  Tunnel messages: tunnel messages are large messages containing encrypted I2NP (see below) messages and encrypted instructions for their delivery. The encryption is layered. The first hop will decrypt the tunnel message and read a part. Another part can still be encrypted (with another key), so it will be forwarded.
  I2NP messages: I2P Network Protocol messages are used to pass messages through multiple routers. These I2NP messages are combined in tunnel messages.
- **I2P Garlic Layer:** provide encrypted and anonymous end-to-end I2P message delivery.
  I2NP messages: I2P Network Protocol messages are wrapped in each other and used to ensure encryption between two tunnels and are passed along from source to destination, keeping both anonymous.

The following layers are strictly speaking no longer part of the I2P Protocol stack, they are not part of the core 'I2P router' functionality. However, each of these layers adds additional functionality, to allow applications simple and convenient I2P usage.

- **I2P Client Layer:** allow any client to use I2P functionality, without requiring the direct use of the router API.
  I2CP: I2P Client Protocol, allows secure and asynchronous messaging over I2P by communicating messages over the I2CP TCP socket.
- **I2P End-to-end Transport Layer:** allow TCP- or UDP-like functionality on top of I2P.
  Streaming Library: an implementation of TCP-like streams over I2P. This allows easier porting of existing applications to I2P.
  Datagram Library: an implementation of UDP-like messages over I2P. This allows easier porting of existing applications to I2P.
- **I2P Application Interface Layer:** additional (optional) libraries allowing easier implementations on top of I2P.
  I2PTunnel
  SAM/SAMv2/SAMv3(*), BOB
  I2P Application Proxy Layer: proxy systems.
  HTTP Client/Server, IRC Client, SOCKS, Streamr

Finally, what could be considered the **'I2P application layer'**, is a large number of applications on top of I2P. We can order this based on the I2P stack layer they use.

- **Streaming/datagram applications**: i2psnark, Syndie, i2phex…
- **SAM/BOB applications**: IMule, i2p-bt, i2prufus, Robert…
- **Other I2P applications**: Syndie, EepGet, plugins...
- **Regular applications**: Jetty, Apache, Monotone, CVS, browsers, e-mail...

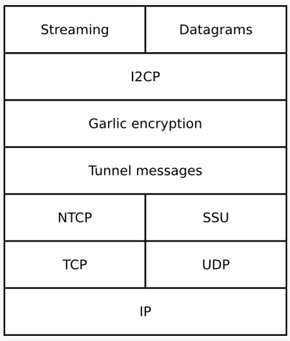


Figure 1: The layers in the I2P Network stack.

* Note: SAM/SAMv2 can use both the streaming lib and datagrams.

Development Roadmaps

Development Roadmaps

This page was last updated in 2021-09.

This is the official project roadmap for the desktop and Android Java I2P releases only. Some related tasks for resources such as the website and plugins may be included.

For details and discussion on specific items, search on gitlab or zzz.i2p. For contents of past releases, see the release notes. For other project goals, see the meeting notes.

We do not maintain separate unstable and stable branches or releases. We have a single, stable release path. Our normal release cycle is 13 weeks, with releases in February, May, August, and November.

Older releases are at the bottom of the page.

0.9.48

Released: December 1, 2020

- ECIES router tunnel build record
- Avoid old DSA-SHA1 routers
- Block same-country connections when in hidden mode
- Deprecate BOB
- Drop support for Xenial
- Ratchet efficiency improvements and memory reduction
- Randomize SSU intro key
- Enable system tray for Linux KDE and LXDE
- More SSU performance improvements
- Continue transition to Git
- Operators guides for reseed services
- Windows Installer "Install as Windows Service" bugfixes and improvements.
- Implement controlled vocabuary as part of Information Architecture improvements
- Alternate destination header/meta tag for web sites offering I2P mirrors
- Snark in the Browser: Use torrents as alternates sources for resources embedded in an I2P Site
- Snark in the Browser: Demo a torrent-backed web page
- Finish ji2p-cluster which adds the k8s part of the code
- Publish reasonable contact information for infrastructure admins

0.9.49

Released: February 17, 2021

- SSU send individual fragments
- SSU Westwood+
- SSU fast retransmit
- SSU fix partial acks
- ECIES router encrypted messages
- Start rekeying routers to ECIES
- Start work on new tunnel build message (proposal 157)
- More SSU performance improvements
- i2psnark webseed support
- Start work on i2psnark hybrid v2 support
- Move web resources to wars
- Move resources to jars
- Fix Gradle build
- Hidden mode fixes
- Change DoH to RFC 8484
- Fix "Start on Boot" support on Android
- Add support for copying b32 addresses from the tunnels panel on I2P for Android client
- Add SAMv3 Support to I2P for Android
- Revise CSS on the default I2P Site to resemble console Light theme
- Document setup/configuration of default I2P site on the project site
- Add icons and symbols used in I2P router console Light theme to router console Dark theme
- Complete transition to Git
- Donation page redesign and backend (deployment)
- Review and update information about VCS, Code Repositories, and Mirrors across the entire website.

And more on…..https://geti2p.net/en/get-involved/roadmap

Application Development Overview and Guide

Why write I2P-specific code?

There are multiple ways to use applications in I2P. Using I2PTunnel, you can use regular applications without needing to program explicit I2P support. This is very effective for client-server scenario's, where you need to connect to a single website. You can simply create a tunnel using I2PTunnel to connect to that website, as shown in Figure 1.

If your application is distributed, it will require connections to a large amount of peers. Using I2PTunnel, you will need to create a new tunnel for each peer you want to contact, as shown in Figure 2. This process can of course be automated, but running a lot of I2PTunnel instances creates a large amount of overhead. In addition, with many protocols you will need to force everyone to use the same set of ports for all peers - e.g. if you want to reliably run DCC chat, everyone needs to agree that port 10001 is Alice, port 10002 is Bob, port 10003 is Charlie, and so on, since the protocol includes TCP/IP specific information (host and port).

General network applications often send a lot of additional data that could be used to identify users. Hostnames, port numbers, time zones, character sets, etc. are often sent without informing the user. As such, designing the network protocol specifically with anonymity in mind can avoid compromising user identities.

There are also efficiency considerations to review when determining how to interact on top of I2P. The streaming library and things built on top of it operate with handshakes similar to TCP, while the core I2P protocols (I2NP and I2CP) are strictly message based (like UDP or in some instances raw IP). The important distinction is that with I2P, communication is operating over a long fat network - each end to end message will have nontrivial latencies, but may contain payloads of up to several KB. An application that needs a simple request and response can get rid of any state and drop the latency incurred by the startup and teardown handshakes by using (best effort) datagrams without having to worry about MTU detection or fragmentation of messages.

Figure 1: Creating a server-client connection using I2PTunnel only requires creating a single tunnel.

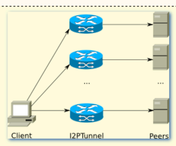


Figure 2: Setting up connections for a peer-to-peer applications requires a very large amount of tunnels.

In summary, a number of reasons to write I2P-specific code:
- Creating a large amount of I2PTunnel instances consumes a non-trivial amount of resources, which is problematic for distributed applications (a new tunnel is required for each peer).
- General network protocols often send a lot of additional data that can be used to identify users. Programming specifically for I2P allows the creation of a network protocol that does not leak such information, keeping users anonymous and secure.
- Network protocols designed for use on the regular internet can be inefficient on I2P, which is a network with a much higher latency.

I2P supports a standard plugins interface for developers so that applications may be easily integrated and distributed.

Applications written in Java and accessible/runnable using an HTML interface via the standard webapps/app.war may be considered for inclusion in the i2p distribution.

Important Concepts

There are a few changes that require adjusting to when using I2P:

Destination ~= host+port

An application running on I2P sends messages from and receives messages to a unique cryptographically secure end point - a "destination". In TCP or UDP terms, a destination could (largely) be considered the equivalent of a hostname plus port number pair, though there are a few differences.

- An I2P destination itself is a cryptographic construct - all data sent to one is encrypted as if there were universal deployment of IPsec with the (anonymized) location of the end point signed as if there were universal deployment of DNSSEC.
- I2P destinations are mobile identifiers - they can be moved from one I2P router to another (or it can even "multihome" - operate on multiple routers at once). This is quite different from the TCP or UDP world where a single end point (port) must stay on a single host.
- I2P destinations are ugly and large - behind the scenes, they contain a 2048 bit ElGamal public key for encryption, a 1024 bit DSA public key for signing, and a variable size certificate, which may contain proof of work or blinded data.
  There are existing ways to refer to these large and ugly destinations by short and pretty names (e.g. "irc.duck.i2p"), but those techniques do not guarantee globally uniqueness (since they're stored locally in a database on each person's machine) and the current mechanism is not especially scalable nor secure (updates to the host list are managed using "subscriptions" to naming services). There may be some secure, human readable, scalable, and globally unique, naming system some day, but applications shouldn't depend upon it being in place, since there are those who don't think such a beast is possible. Further information on the naming system is available.

While most applications do not need to distinguish protocols and ports, I2P *does* support them. Complex applications may specify a protocol, from port, and to port, on a per-message basis, to multiplex traffic on a single destination. See the datagram page for details. Simple applications operate by listening for "all protocols" on "all ports" of a destination.

Anonymity and confidentiality

I2P has transparent end to end encryption and authentication for all data passed over the network - if Bob sends to Alice's destination, only Alice's destination can receive it, and if Bob is using the datagrams or streaming library, Alice knows for certain that Bob's destination is the one who sent the data.

Of course, I2P transparently anonymizes the data sent between Alice and Bob, but it does nothing to anonymize the content of what they send. For instance, if Alice sends Bob a form with her full name, government IDs, and credit card numbers, there is nothing I2P can do. As such, protocols and applications should keep in mind what information they are trying to protect and what information they are willing to expose.

I2P datagrams can be up to several KB

Applications that use I2P datagrams (either raw or repliable ones) can essentially be thought of in terms of UDP - the datagrams are unordered, best effort, and connectionless - but unlike UDP, applications don't need to worry about MTU detection and can simply fire off large datagrams. While the upper limit is nominally 32 KB, the message is fragmented for transport, thus dropping the reliability of the whole. Datagrams over about 10 KB are not currently recommended. See the datagram page for details. For many applications, 10 KB of data is sufficient for an entire request or response, allowing them to transparently operate in I2P as a UDP-like application without having to write fragmentation, resends, etc.

Development Options

There are several means of sending data over I2P, each with their own pros and cons. The streaming lib is the recommended interface, used by the majority of I2P applications.

Streaming Lib

The full streaming library is now the standard interface. It allows programming using TCP-like sockets, as explained in the Streaming development guide.

BOB

BOB is the Basic Open Bridge, allowing an application in any language to make streaming connections to and from I2P. At this point in time it lacks UDP support, but UDP support is planned in the near future. BOB also contains several tools, such as destination key generation, and verification that an address conforms to I2P specifications. Up to date info and applications that use BOB can be found at this I2P Site.

SAM, SAM V2, SAM V3

SAM is not recommended. SAM V2 is okay, SAM V3 is recommended.
SAM is the Simple Anonymous Messaging protocol, allowing an application written in any language to talk to a SAM bridge through a plain TCP socket and have that bridge multiplex all of its I2P traffic, transparently coordinating the encryption/decryption and event based handling. SAM supports three styles of operation:
- streams, for when Alice and Bob want to send data to each other reliably and in order
- repliable datagrams, for when Alice wants to send Bob a message that Bob can reply to
- raw datagrams, for when Alice wants to squeeze the most bandwidth and performance as possible, and Bob doesn't care whether the data's sender is authenticated or not (e.g. the data transferred is self authenticating)

SAM V3 aims at the same goal as SAM and SAM V2, but does not require multiplexing/demultiplexing. Each I2P stream is handled by its own socket between the application and the SAM bridge. Besides, datagrams can be sent and received by the application through datagram communications with the SAM bridge.

SAM V2 is a new version used by imule that fixes some of the problems in SAM.

SAM V3 is used by imule since version 1.4.0.

And More on……https://geti2p.net/en/get-involved/develop/applications

Naming and Address Book

Overview

I2P ships with a generic naming library and a base implementation designed to work off a local name to destination mapping, as well as an add-on application called the address book. I2P also supports Base32 hostnames similar to Tor's .onion addresses.

The address book is a web-of-trust driven secure, distributed, and human readable naming system, sacrificing only the call for all human readable names to be globally unique by mandating only local uniqueness. While all messages in I2P are cryptographically addressed by their destination, different people can have local address book entries for "Alice" which refer to different destinations. People can still discover new names by importing published address books of peers specified in their web of trust, by adding in the entries provided through a third party, or (if some people organize a series of published address books using a first come first serve registration system) people can choose to treat these address books as name servers, emulating traditional DNS.

NOTE: For the reasoning behind the I2P naming system, common arguments against it and possible alternatives see the naming discussion page.

Naming System Components

There is no central naming authority in I2P. All hostnames are local.
The naming system is quite simple and most of it is implemented in applications external to the router, but bundled with the I2P distribution. The components are:

1. The local naming service which does lookups and also handles Base32 hostnames.
2. The HTTP proxy which asks the router for lookups and points the user to remote jump services to assist with failed lookups.
3. HTTP host-add forms which allow users to add hosts to their local hosts.txt
4. HTTP jump services which provide their own lookups and redirection.
5. The address book application which merges external host lists, retrieved via HTTP, with the local list.
6. The SusiDNS application which is a simple web front-end for address book configuration and viewing of the local host lists.

Naming Services

All destinations in I2P are 516-byte (or longer) keys. (To be more precise, it is a 256-byte public key plus a 128-byte signing key plus a 3-or-more byte certificate, which in Base64 representation is 516 or more bytes. Non-null Certificates are in use now for signature type indication. Therefore, certificates in recently-generated destinations are more than 3 bytes.

If an application (i2ptunnel or the HTTP proxy) wishes to access a destination by name, the router does a very simple local lookup to resolve that name.

Hosts.txt Naming Service

The hosts.txt Naming Service does a simple linear search through text files. This naming service was the default until release 0.8.8 when it was replaced by the Blockfile Naming Service. The hosts.txt format had become too slow after the file grew to thousands of entries.
It does a linear search through three local files, in order, to look up host names and convert them to a 516-byte destination key. Each file is in a simple configuration file format, with hostname=base64, one per line. The files are:

1. privatehosts.txt
2. userhosts.txt
3. hosts.txt

Blockfile Naming Service

The Blockfile Naming Service stores multiple "address books" in a single database file named hostsdb.blockfile. This Naming Service is the default since release 0.8.8.

A blockfile is simply on-disk storage of multiple sorted maps (key-value pairs), implemented as skiplists. The blockfile format is specified on the Blockfile page. It provides fast Destination lookup in a compact format. While the blockfile overhead is substantial, the destinations are stored in binary rather than in Base 64 as in the hosts.txt format. In addition, the blockfile provides the capability of arbitrary metadata storage (such as added date, source, and comments) for each entry to implement advanced address book features. The blockfile storage requirement is a modest increase over the hosts.txt format, and the blockfile provides approximately 10x reduction in lookup times.

On creation, the naming service imports entries from the three files used by the hosts.txt Naming Service. The blockfile mimics the previous implementation by maintaining three maps that are searched in-order, named privatehosts.txt, userhosts.txt, and hosts.txt. It also maintains a reverse-lookup map to implement rapid reverse lookups.

Other Naming Service Facilities

The lookup is case-insensitive. The first match is used, and conflicts are not detected. There is no enforcement of naming rules in lookups. Lookups are cached for a few minutes. Base 32 resolution is described below. For a full description of the Naming Service API see the Naming Service Javadocs. This API was significantly expanded in release 0.8.7 to provide adds and removes, storage of arbitrary properties with the hostname, and other features.

Alternatives and Experimental Naming Services

The naming service is specified with the configuration property `i2p.naming.impl=class`. Other implementations are possible. For example, there is an experimental facility for real-time lookups (a la DNS) over the network within the router. For more information see the alternatives on the discussion page.

The HTTP proxy does a lookup via the router for all hostnames ending in '.i2p'. Otherwise, it forwards the request to a configured HTTP outproxy. Thus, in practice, all HTTP (I2P Site) hostnames must end in the pseudo-Top Level Domain '.i2p'.

We have applied to reserve the .i2p TLD following the procedures specified in RFC 6761.

If the router fails to resolve the hostname, the HTTP proxy returns an error page to the user with links to several "jump" services. See below for details.

Address Book

Incoming Subscriptions and Merging

The address book application periodically retrieves other users' hosts.txt files and merges them with the local hosts.txt, after several checks. Naming conflicts are resolved on a first-come first-served basis.

Subscribing to another user's hosts.txt file involves giving them a certain amount of trust. You do not want them, for example, 'hijacking' a new site by quickly entering in their own key for a new site before passing the new host/key entry to you.

For this reason, the only subscription configured by default is `http://i2p-projekt.i2p/hosts.txt (http://udhdrtrcetjm5sxzskjyr5ztpeszydbh4dpl3pl4utgqqw2v4jna.b32.i2p/hosts.txt)`, which contains a copy of the hosts.txt included in the I2P release. Users must configure additional subscriptions in their local address book application (via subscriptions.txt or SusiDNS).

Some other public address book subscription links:
- http://i2host.i2p.xyz/cgi-bin/i2hostetag
- http://stats.i2p/cgi-bin/newhosts.txt

The operators of these services may have various policies for listing hosts. Presence on this list does not imply endorsement.

And More....

Address Book Subscription Feed Commands

Overview

This specification extends the address subscription feed with commands, to enable name servers to broadcast entry updates from hostname holders. Implemented in 0.9.26, originally proposed in proposal 112.

Motivation

Previously, the hosts.txt subscription servers just sent data in a hosts.txt format, which is as follows:
example.i2p=b64destination
There are several problems with this:

- Hostname holders cannot update the Destination associated with their hostnames (in order to e.g. upgrade the signing key to a stronger type).
- Hostname holders cannot relinquish their hostnames arbitrarily; they must give the corresponding Destination private keys directly to the new holder.
- There is no way to authenticate that a subdomain is controlled by the corresponding base hostname; this is currently only enforced individually by some name servers.

Design

This specification adds a number of command lines to the hosts.txt format. With these commands, name servers can extend their services to provide a number of additional features. Clients that implement this specification will be able to listen for these features through the regular subscription process.

All command lines must be signed by the corresponding Destination. This ensures that changes are only made at the request of the hostname holder.

Security Implications

This specification does not affect anonymity.

There is an increase in the risk associated with losing control of a Destination key, as someone who obtains it can use these commands to make changes to any associated hostnames. But this is no more of a problem than the status quo, where someone who obtains a Destination can impersonate a hostname and (partially) take over its traffic. The increased risk is also balanced our by giving hostname holders the ability to change the Destination associated with a hostname, in the event that they believe the Destination has been compromised; this is impossible with the current system.

Specifications

New Line Types

This specification does not affect anonymity.

There is an increase in the risk associated with losing control of a Destination key, as someone who obtains it can use these commands to make changes to any associated hostnames. But this is no more of a problem than the status quo, where someone who obtains a Destination can impersonate a hostname and (partially) take over its traffic. The increased risk is also balanced our by giving hostname holders the ability to change the Destination associated with a hostname, in the event that they believe the Destination has been compromised; this is impossible with the current system.

Plug-Ins Overview

General Information

I2P includes a plugin architecture to support easy development and installation of additional software.

There are now plugins available that support distributed email, blogs, IRC clients, distributed file storage, wikis, and more.

Benefits to i2p users and app developers:
- Easy distribution of applications
- Allows innovation and use of additional libraries without worrying about increasing the size of `i2pupdate.sud`
- Support large or special-purpose applications that would never be bundled with the I2P installation
- Cryptographically signed and verified applications
- Automatic updates of applications, just like for the router
- Separate initial install and update packages, if desired, for smaller update downloads
- One-click installation of applications. No more asking users to modify `wrapper.config` or `clients.config`
- Isolate applications from the base `$I2P` installation
- Automatic compatibility checking for I2P version, Java version, Jetty version, and previous installed application version
- Automatic link addition in console
- Automatic startup of application, including modifying classpath, without requiring a restart
- Automatic integration and startup of webapps into console Jetty instance
- Facilitate creation of 'app stores' like the one at plugins.i2p.xyz
- One-click uninstall
- Language and theme packs for the console
- Bring detailed application information to the router console
- Non-java applications also supported

Required I2P Version

0.7.12 or newer.

Installation

To install and start a plugin, copy the `.xpi2p` install link to the form at the bottom of configclients.jsp in your router console and click the "install plugin" button. After a plugin is installed and started, a link to the plugin will usually appear at the top of your summary bar.

To update a plugin to the latest version, just click the update button on configclients.jsp. There is also a button to check if the plugin has a more recent version, as well as a button to check for updates for all plugins. Plugins will be checked for updates automatically when updating to a new I2P release (not including dev builds).

Development

See the latest plugin specification and the plugin forum on zzz.i2p.
See also the sources for plugins developed by various people. Some plugins, such as snowman, were developed specifically as examples.

Developers wanted! Plugins are a great way to learn more about I2P or easily add some feature.

Getting Started

To create a plugin from an existing binary package you will need to get makeplugin.sh from the i2p.scripts branch in monotone.

Known Issues

Note that the router's plugin architecture does **NOT** currently provide any additional security isolation or sandboxing of plugins.
- Updates of a plugin with included jars (not wars) won't be recognized if the plugin was already run, as it requires class loader trickery to flush the class cache; a full router restart is required.
- The stop button may be displayed even if there is nothing to stop.
- Plugins running in a separate JVM create a `logs/` directory in `$CWD`.
- No initial keys are present, except for those of jrandom and zzz (using the same keys as for router update), so the first key seen for a signer is automatically accepted—there is no signing key authority.
- When deleting a plugin, the directory is not always deleted, especially on Windows.
- Installing a plugin requiring Java 1.6 on a Java 1.5 machine will result in a "plugin is corrupt" message if pack200 compression of the plugin file is used.
- Theme and translation plugins are untested.
- Disabling autostart doesn't always work.

Plug-In Specification

Managed Clients

Embedding the Router in Your Application

BitTorrent over I2P

I2PControl Plugin API

hostsdb.blockfile Format

Configuration File Format

Repeat “App Dev Overview” ?

https://geti2p.net/en/get-involved/develop/applications

Application Development Overview and Guide

Why write I2P-specific code?

There are multiple ways to use applications in I2P. Using I2PTunnel, you can use regular applications without needing to program explicit I2P support. This is very effective for client-server scenario's, where you need to connect to a single website. You can simply create a tunnel using I2PTunnel to connect to that website, as shown in Figure 1.

If your application is distributed, it will require connections to a large amount of peers. Using I2PTunnel, you will need to create a new tunnel for each peer you want to contact, as shown in Figure 2. This process can of course be automated, but running a lot of I2PTunnel instances creates a large amount of overhead. In addition, with many protocols you will need to force everyone to use the same set of ports for all peers - e.g. if you want to reliably run DCC chat, everyone needs to agree that port 10001 is Alice, port 10002 is Bob, port 10003 is Charlie, and so on, since the protocol includes TCP/IP specific information (host and port).

General network applications often send a lot of additional data that could be used to identify users. Hostnames, port numbers, time zones, character sets, etc. are often sent without informing the user. As such, designing the network protocol specifically with anonymity in mind can avoid compromising user identities.

There are also efficiency considerations to review when determining how to interact on top of I2P. The streaming library and things built on top of it operate with handshakes similar to TCP, while the core I2P protocols (I2NP and I2CP) are strictly message based (like UDP or in some instances raw IP). The important distinction is that with I2P, communication is operating over a long fat network - each end to end message will have nontrivial latencies, but may contain payloads of up to several KB. An application that needs a simple request and response can get rid of any state and drop the latency incurred by the startup and teardown handshakes by using (best effort) datagrams without having to worry about MTU detection or fragmentation of messages.

Figure 1: Creating a server-client connection using I2PTunnel only requires creating a single tunnel.

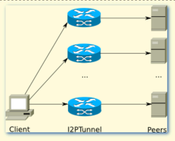


Figure 2: Setting up connections for a peer-to-peer applications requires a very large amount of tunnels.

In summary, a number of reasons to write I2P-specific code:
- Creating a large amount of I2PTunnel instances consumes a non-trivial amount of resources, which is problematic for distributed applications (a new tunnel is required for each peer).
- General network protocols often send a lot of additional data that can be used to identify users. Programming specifically for I2P allows the creation of a network protocol that does not leak such information, keeping users anonymous and secure.
- Network protocols designed for use on the regular internet can be inefficient on I2P, which is a network with a much higher latency.

I2P supports a standard plugins interface for developers so that applications may be easily integrated and distributed.

Applications written in Java and accessible/runnable using an HTML interface via the standard webapps/app.war may be considered for inclusion in the i2p distribution.

Important Concepts

There are a few changes that require adjusting to when using I2P:

Destination ~= host+port

An application running on I2P sends messages from and receives messages to a unique cryptographically secure end point - a "destination". In TCP or UDP terms, a destination could (largely) be considered the equivalent of a hostname plus port number pair, though there are a few differences.

- An I2P destination itself is a cryptographic construct - all data sent to one is encrypted as if there were universal deployment of IPsec with the (anonymized) location of the end point signed as if there were universal deployment of DNSSEC.
- I2P destinations are mobile identifiers - they can be moved from one I2P router to another (or it can even "multihome" - operate on multiple routers at once). This is quite different from the TCP or UDP world where a single end point (port) must stay on a single host.
- I2P destinations are ugly and large - behind the scenes, they contain a 2048 bit ElGamal public key for encryption, a 1024 bit DSA public key for signing, and a variable size certificate, which may contain proof of work or blinded data.
  There are existing ways to refer to these large and ugly destinations by short and pretty names (e.g. "irc.duck.i2p"), but those techniques do not guarantee globally uniqueness (since they're stored locally in a database on each person's machine) and the current mechanism is not especially scalable nor secure (updates to the host list are managed using "subscriptions" to naming services). There may be some secure, human readable, scalable, and globally unique, naming system some day, but applications shouldn't depend upon it being in place, since there are those who don't think such a beast is possible. Further information on the naming system is available.

While most applications do not need to distinguish protocols and ports, I2P *does* support them. Complex applications may specify a protocol, from port, and to port, on a per-message basis, to multiplex traffic on a single destination. See the datagram page for details. Simple applications operate by listening for "all protocols" on "all ports" of a destination.

Anonymity and confidentiality

I2P has transparent end to end encryption and authentication for all data passed over the network - if Bob sends to Alice's destination, only Alice's destination can receive it, and if Bob is using the datagrams or streaming library, Alice knows for certain that Bob's destination is the one who sent the data.

Of course, I2P transparently anonymizes the data sent between Alice and Bob, but it does nothing to anonymize the content of what they send. For instance, if Alice sends Bob a form with her full name, government IDs, and credit card numbers, there is nothing I2P can do. As such, protocols and applications should keep in mind what information they are trying to protect and what information they are willing to expose.

I2P datagrams can be up to several KB

Applications that use I2P datagrams (either raw or repliable ones) can essentially be thought of in terms of UDP - the datagrams are unordered, best effort, and connectionless - but unlike UDP, applications don't need to worry about MTU detection and can simply fire off large datagrams. While the upper limit is nominally 32 KB, the message is fragmented for transport, thus dropping the reliability of the whole. Datagrams over about 10 KB are not currently recommended. See the datagram page for details. For many applications, 10 KB of data is sufficient for an entire request or response, allowing them to transparently operate in I2P as a UDP-like application without having to write fragmentation, resends, etc.

Development Options

There are several means of sending data over I2P, each with their own pros and cons. The streaming lib is the recommended interface, used by the majority of I2P applications.

Streaming Lib

The full streaming library is now the standard interface. It allows programming using TCP-like sockets, as explained in the Streaming development guide.

BOB

BOB is the Basic Open Bridge, allowing an application in any language to make streaming connections to and from I2P. At this point in time it lacks UDP support, but UDP support is planned in the near future. BOB also contains several tools, such as destination key generation, and verification that an address conforms to I2P specifications. Up to date info and applications that use BOB can be found at this I2P Site.

SAM, SAM V2, SAM V3

SAM is not recommended. SAM V2 is okay, SAM V3 is recommended.
SAM is the Simple Anonymous Messaging protocol, allowing an application written in any language to talk to a SAM bridge through a plain TCP socket and have that bridge multiplex all of its I2P traffic, transparently coordinating the encryption/decryption and event based handling. SAM supports three styles of operation:
- streams, for when Alice and Bob want to send data to each other reliably and in order
- repliable datagrams, for when Alice wants to send Bob a message that Bob can reply to
- raw datagrams, for when Alice wants to squeeze the most bandwidth and performance as possible, and Bob doesn't care whether the data's sender is authenticated or not (e.g. the data transferred is self authenticating)

SAM V3 aims at the same goal as SAM and SAM V2, but does not require multiplexing/demultiplexing. Each I2P stream is handled by its own socket between the application and the SAM bridge. Besides, datagrams can be sent and received by the application through datagram communications with the SAM bridge.

SAM V2 is a new version used by imule that fixes some of the problems in SAM.

SAM V3 is used by imule since version 1.4.0.

And More on……https://geti2p.net/en/get-involved/develop/applications

I2PTunnel

I2PTunnel Configuration

SOCKS Proxy

HTTP Proxy

CONNECT Proxy

IRC Proxy

SOCKS IRC Proxy

* Note: SAM/SAMv2 can use both the streaming lib and datagrams.

Streamr Proxy

* Note: SAM/SAMv2 can use both the streaming lib and datagrams.

HTTP Bidir Proxy

Figure 1: The layers in the I2P Network stack.

* Note: SAM/SAMv2 can use both the streaming lib and datagrams.

SAM Protocol

Figure 1: The layers in the I2P Network stack.

* Note: SAM/SAMv2 can use both the streaming lib and datagrams.

SAMv2 Protocol

Figure 1: The layers in the I2P Network stack.

* Note: SAM/SAMv2 can use both the streaming lib and datagrams.

SAMv3 Protocol

Figure 1: The layers in the I2P Network stack.

* Note: SAM/SAMv2 can use both the streaming lib and datagrams.

BOB Protocol

Figure 1: The layers in the I2P Network stack.

* Note: SAM/SAMv2 can use both the streaming lib and datagrams.

Network Database

This page was last updated in August 2019 and is accurate for router version 0.9.42.

Overview

YI2P's netDb is a specialized distributed database, containing just two types of data - router contact information (**RouterInfos**) and destination contact information (**LeaseSets**). Each piece of data is signed by the appropriate party and verified by anyone who uses or stores it. In addition, the data has liveliness information within it, allowing irrelevant entries to be dropped, newer entries to replace older ones, and protection against certain classes of attack.

The netDb is distributed with a simple technique called "floodfill", where a subset of all routers, called "floodfill routers", maintains the distributed database.

Router Info

When an I2P router wants to contact another router, they need to know some key pieces of data - all of which are bundled up and signed by the router into a structure called the "RouterInfo", which is distributed with the SHA256 of the router's identity as the key. The structure itself contains:

- The router's identity (an encryption key, a signing key, and a certificate)
- The contact addresses at which it can be reached
- When this was published
- A set of arbitrary text options
- The signature of the above, generated by the identity's signing key

Expected Options

The following text options, while not strictly required, are expected to be present:

- **caps** (Capabilities flags - used to indicate floodfill participation, approximate bandwidth, and perceived reachability)
  - **f**: Floodfill
  - **H**: Hidden
  - **K**: Under 12 KBps shared bandwidth
  - **L**: 12 - 48 KBps shared bandwidth (default)
  - **M**: 48 - 64 KBps shared bandwidth
  - **N**: 64 - 128 KBps shared bandwidth
  - **O**: 128 - 256 KBps shared bandwidth
  - **P**: 256 - 2000 KBps shared bandwidth (as of release 0.9.20)
  - **R**: Reachable
  - **U**: Unreachable
  - **X**: Over 2000 KBps shared bandwidth (as of release 0.9.20)
- "Shared bandwidth" == (share %) * min(in bw, out bw)
- For compatibility with older routers, a router may publish multiple bandwidth letters, for example "PO".
- **coreVersion** (The core library version, always the same as the router version) (Never used, removed in release 0.9.24)
- **netId** = 2 (Basic network compatibility - A router will refuse to communicate with a peer having a different netId)
- **router.version** (Used to determine compatibility with newer features and messages)
- **stat_uptime** = 90m (Always sent as 90m, for compatibility with an older scheme where routers published their actual uptime, and only sent tunnel requests to peers whose uptime was more than 60m) (Unused since version 0.7.9, removed in release 0.9.24)

These values are used by other routers for basic decisions. Should we connect to this router? Should we attempt to route a tunnel through this router? The bandwidth capability flag, in particular, is used only to determine whether the router meets a minimum threshold for routing tunnels. Above the minimum threshold, the advertised bandwidth is not used or trusted anywhere in the router, except for display in the user interface and for debugging and network analysis.
Valid NetID numbers:

Usage	NetID Number
Reserved	0
Reserved	1
Current Network (default)	2
Reserved Future Networks	3 - 15
Forks and Test Networks	16 - 254
Reserved	255

Additional Options

Additional text options include a small number of statistics about the router's health, which are aggregated by sites such as stats.i2p for network performance analysis and debugging. These statistics were chosen to provide data crucial to the developers, such as tunnel build success rates, while balancing the need for such data with the side-effects that could result from revealing this data.
Current statistics are limited to:

- Exploratory tunnel build success, reject, and timeout rates
- 1 hour average number of participating tunnels

Floodfill routers publish additional data on the number of entries in their network database.

The data published can be seen in the router's user interface, but is not used or trusted within the router. As the network has matured, we have gradually removed most of the published statistics to improve anonymity, and we plan to remove more in future releases.

Family Options

As of release 0.9.24, routers may declare that they are part of a "family", operated by the same entity. Multiple routers in the same family will not be used in a single tunnel.
The family options are:

- **family** (The family name)
- **family.key** The signature type code of the family's Signing Public Key (in ASCII digits) concatenated with ':' concatenated with the Signing Public Key in base 64
- **family.sig** The signature of ((family name in UTF-8) concatenated with (32 byte router hash)) in base 64

RouterInfo Expiration

RouterInfos have no set expiration time. Each router is free to maintain its own local policy to trade off the frequency of RouterInfo lookups with memory or disk usage. In the current implementation, there are the following general policies:

- There is no expiration during the first hour of uptime, as the persistent stored data may be old.
- There is no expiration if there are 25 or less RouterInfos.
- As the number of local RouterInfos grows, the expiration time shrinks, in an attempt to maintain a reasonable number RouterInfos. The expiration time with less than 120 routers is 72 hours, while expiration time with 300 routers is around 30 hours.
- RouterInfos containing SSU introducers expire in about an hour, as the introducer list expires in about that time.
- Floodfills use a short expiration time (1 hour) for all local RouterInfos, as valid RouterInfos will be frequently republished to them.

And More on……https://geti2p.net/en/docs/how/network-database

Garlic Routing

Garlic Routing and “Garlic Terminology”

The terms "garlic routing" and "garlic encryption" are often used rather loosely when referring to I2P's technology. Here, we explain the history of the terms, the various meanings, and the usage of "garlic" methods in I2P.
"Garlic routing" was first coined by Michael J. Freedman in Roger Dingledine's Free Haven Master's thesis Section 8.1.1 (June 2000), as derived from Onion Routing.

"Garlic" may have been used originally by I2P developers because I2P implements a form of bundling as Freedman describes, or simply to emphasize general differences from Tor. The specific reasoning may be lost to history. Generally, when referring to I2P, the term "garlic" may mean one of three things

1. Layered Encryption
2. Bundling multiple messages together
3. ElGamal/AES Encryption

Unfortunately, I2P's usage of "garlic" terminology over the past seven years has not always been precise; therefore the reader is cautioned when encountering the term. Hopefully, the explanation below will make things clear.

Layered Encryption

Onion routing is a technique for building paths, or tunnels, through a series of peers, and then using that tunnel. Messages are repeatedly encrypted by the originator, and then decrypted by each hop. During the building phase, only the routing instructions for the next hop are exposed to each peer. During the operating phase, messages are passed through the tunnel, and the message and its routing instructions are only exposed to the endpoint of the tunnel.

This is similar to the way Mixmaster (see network comparisons) sends messages - taking a message, encrypting it to the recipient's public key, taking that encrypted message and encrypting it (along with instructions specifying the next hop), and then taking that resulting encrypted message and so on, until it has one layer of encryption per hop along the path.

In this sense, "garlic routing" as a general concept is identical to "onion routing". As implemented in I2P, of course, there are several differences from the implementation in Tor; see below. Even so, there are substantial similarities such that I2P benefits from a large amount of academic research on onion routing, Tor, and similar mixnets.

Bundling Multiple Messages

Michael Freedman defined "garlic routing" as an extension to onion routing, in which multiple messages are bundled together. He called each message a "bulb". All the messages, each with its own delivery instructions, are exposed at the endpoint. This allows the efficient bundling of an onion routing "reply block" with the original message.

This concept is implemented in I2P, as described below. Our term for garlic "bulbs" is "cloves". Any number of messages can be contained, instead of just a single message. This is a significant distinction from the onion routing implemented in Tor. However, it is only one of many major architectural differences between I2P and Tor; perhaps it is not, by itself, enough to justify a change in terminology.

Another difference from the method described by Freedman is that the path is unidirectional - there is no "turning point" as seen in onion routing or mixmaster reply blocks, which greatly simplifies the algorithm and allows for more flexible and reliable delivery.

ElGamal/AES Encryption

In some cases, "garlic encryption" may simply mean ElGamal/AES+SessionTag encryption (without multiple layers).

"Garlic" Methods in I2P

Now that we've defined various "garlic" terms, we can say that I2P uses garlic routing, bundling and encryption in three places:

1. For building and routing through tunnels (layered encryption)
2. For determining the success or failure of end to end message delivery (bundling)
3. For publishing some network database entries (dampening the probability of a successful traffic analysis attack) (ElGamal/AES)

There are also significant ways that this technique can be used to improve the performance of the network, exploiting transport latency/throughput tradeoffs, and branching data through redundant paths to increase reliability.

Tunnel Building and Routing

In I2P, tunnels are unidirectional. Each party builds two tunnels, one for outbound and one for inbound traffic. Therefore, four tunnels are required for a single round-trip message and reply.

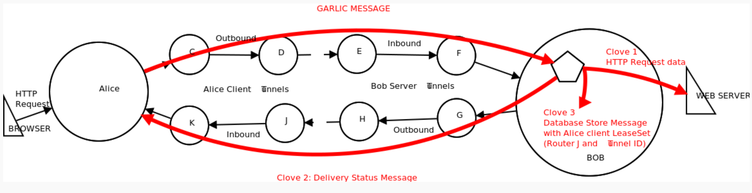
Tunnels are built, and then used, with layered encryption. This is described on the tunnel implementation page. Tunnel building details are defined on this page. We use ElGamal/AES+SessionTag for the encryption.

Tunnels are a general-purpose mechanism to transport all I2NP messages, and Garlic Messages are not used to build tunnels. We do not bundle multiple I2NP messages into a single Garlic Message for unwrapping at the outbound tunnel endpoint; the tunnel encryption is sufficient.

End-to-End Message Bundling

At the layer above tunnels, I2P delivers end-to-end messages between Destinations. Just as within a single tunnel, we use ElGamal/AES+SessionTag for the encryption. Each client message as delivered to the router through the I2CP interface becomes a single Garlic Clove with its own Delivery Instructions, inside a Garlic Message. Delivery Instructions may specify a Destination, Router, or Tunnel.

Generally, a Garlic Message will contain only one clove. However, the router will periodically bundle two additional cloves in the Garlic Message:



1. A Delivery Status Message, with Delivery Instructions specifying that it be sent back to the originating router as an acknowledgment. This is similar to the "reply block" or "reply onion" described in the references. It is used for determining the success or failure of end to end message delivery. The originating router may, upon failure to receive the Delivery Status Message within the expected time period, modify the routing to the far-end Destination, or take other actions.
2. A Database Store Message, containing a LeaseSet for the originating Destination, with Delivery Instructions specifying the far-end destination's router. By periodically bundling a LeaseSet, the router ensures that the far-end will be able to maintain communications. Otherwise the far-end would have to query a floodfill router for the network database entry, and all LeaseSets would have to be published to the network database, as explained on the network database page.

By default, the Delivery Status and Database Store Messages are bundled when the local LeaseSet changes, when additional Session Tags are delivered, or if the messages have not been bundled in the previous minute. As of release 0.9.2, the client may configure the default number of Session Tags to send and the low tag threshold for the current session. See the I2CP options specification for details. The session settings may also be overridden on a per-message basis. See the I2CP Send Message Expires specification for details.

Obviously, the additional messages are currently bundled for specific purposes, and not part of a general-purpose routing scheme.

As of release 0.9.12, the Delivery Status Message is wrapped in another Garlic Message by the originator so that the contents are encrypted and not visible to routers on the return path.

And More on……https://geti2p.net/en/docs/how/threat-model