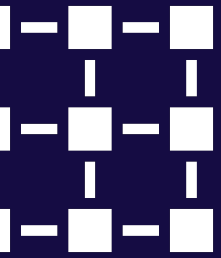
The Internals of Veilid

A New Distributed Application Framework

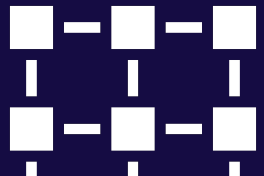
Christien 'DilDog' Rioux
Katelyn 'Medus4' Bowden

www.veilid.com



www.veilid.com

What is Veilid ?

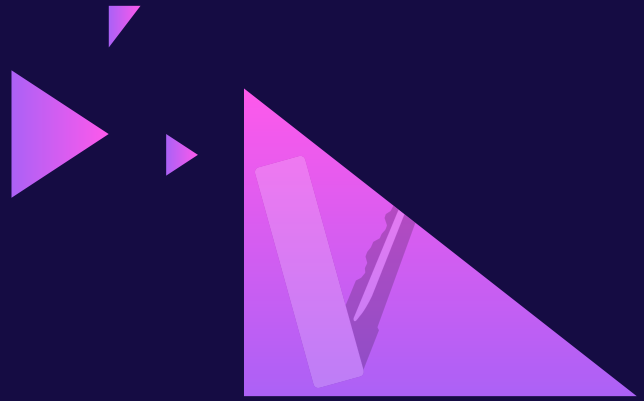


The Veilid Mission

We exist to develop, distribute, and maintain a privacy focused communication platform and protocol for the purposes of defending human and civil rights.

“Fight for the things that you care about, but do it in a way that will lead others to join you.” - Ruth Bader Ginsburg

Others have come before us...



Tor

Privacy-oriented Networking

IPFS

Distributed Data Storage

Other Efforts (Check them out too!)

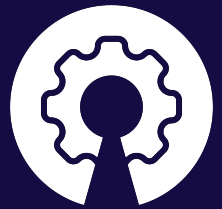
NOSTR - social media specific, still ‘federated’ relay vs client
Scuttlebutt.nz - social publication system, no ip privacy
Holepunch.to - similar app framework concept , no ip privacy
Not mentioned - a lot of ‘web3’ ‘dApps’ that require buying some ‘coin’

Veilid is an open-source peer-to-peer mobile-first networked application framework

Veilid is conceptually similar to IPFS + Tor, but faster and designed from the ground-up to provide all services over a privately routed network.

Veilid enables development of fully-distributed applications without a 'blockchain' or a 'transactional layer' at their base.

Veilid can be included as part of user-facing applications or run as a ‘headless node’ for power users who wish to help build the network.



Veilid Design Goals

www.veilid.com

Security First

Written In Rust
Memory and type-safety

Runs Everywhere

Linux, Mac, Windows
Android, iOS
and browser WASM

Standard Protocols

UDP
TCP
Websockets

All In Network

No External Services
Avoid DNS
No STUN/TURN

Privacy Focused

Strong cryptography
IP privacy is built-in
Nodes != Identity

Resilient

Low Latency
High Node Churn
Switching Networks

Building a community
of applications

Not everything
needs to be centralized

Stop being dependent
on corporate systems

Networking

www.veilid.com

Nodes

All Veilid applications running veilid-core are ‘nodes’ and they are all equal in the eyes of the network

No nodes are ‘special’

All nodes help each other out, regardless of the application hosting them

Nodes are only limited by the resources they bring and the configuration of the network they are on

Applications directly linking in veilid-core

Linux, Mac, Windows, Android, iOS, and Web Apps for Everyone
FFI and WASM Bindings for Flutter / Dart
Rust applications can directly use veilid-core
Native bindings for other languages are welcome!

Headless nodes running veilid-server

Linux, Mac and Windows for ‘power users’
Can be controlled via JSON API for simpler apps
Python development via veilid-python package
Admins and devs can use veilid-cli to control server

```
Node Id                                                [^] | Address
VLD0:6-FfH7TPb70U-JntwjHS7XqTCMK0lhVqPQ17dJuwlBM | UDP:170.64.186.46:5150
VLD0:7DyMbC1I4kHzLhvJom6YQAdfc2VAsFYvfF1V7ceMQkc | UDP:161.35.164.16:5150
VLD0:cqxHp4LUuFhKA-0E9UhSXu4FX5groDiqFUB-E6CPioc | UDP:159.89.163.27:5150
VLD0:m5OY1uhPTq2VWhpYJASmzATsKTC7eZBQmyNs6tRJMmA | UDP:159.223.237.84:5150
VLD0:oSrjFs_AEXvFQCDvpuyTNMs6nsMb8hEdlPkryJkdOUc | UDP:157.230.215.0:5150
VLD0:pmPFxZX8gyUCkH11etUXIx582TdSFBhjd7Y-TtizD0w | UDP:170.64.186.46:5150
Command>
Connected to [::1]:5959 | [|   ]+P-l | Down: 10.75KB/s Up: 6.30KB/s | No Tunnels |
```

www.veilid.com

Protocols

Low level protocols supported by Veilid are kept simple, to minimize complications

Everything uses framed RPC operations up to 64KB in size

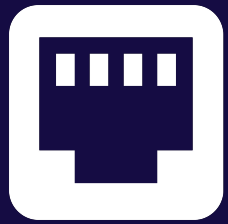
Protocol support is extensible and may add WebRTC and other specialized protocols in the future

DNS is only used one time during ‘bootstrap’ and not required

SSL is optional and only for HTTPS Websockets for Veilid Webapps

UDP

Fast, unsequenced, unreliable datagrams
Chunked into MTU-sized pieces and reassembled by Veilid
Support for out-of-order delivery
No retransmission or acknowledgment



TCP

Sequenced, reliable streams
In-line framing
All the usual TCP guarantees

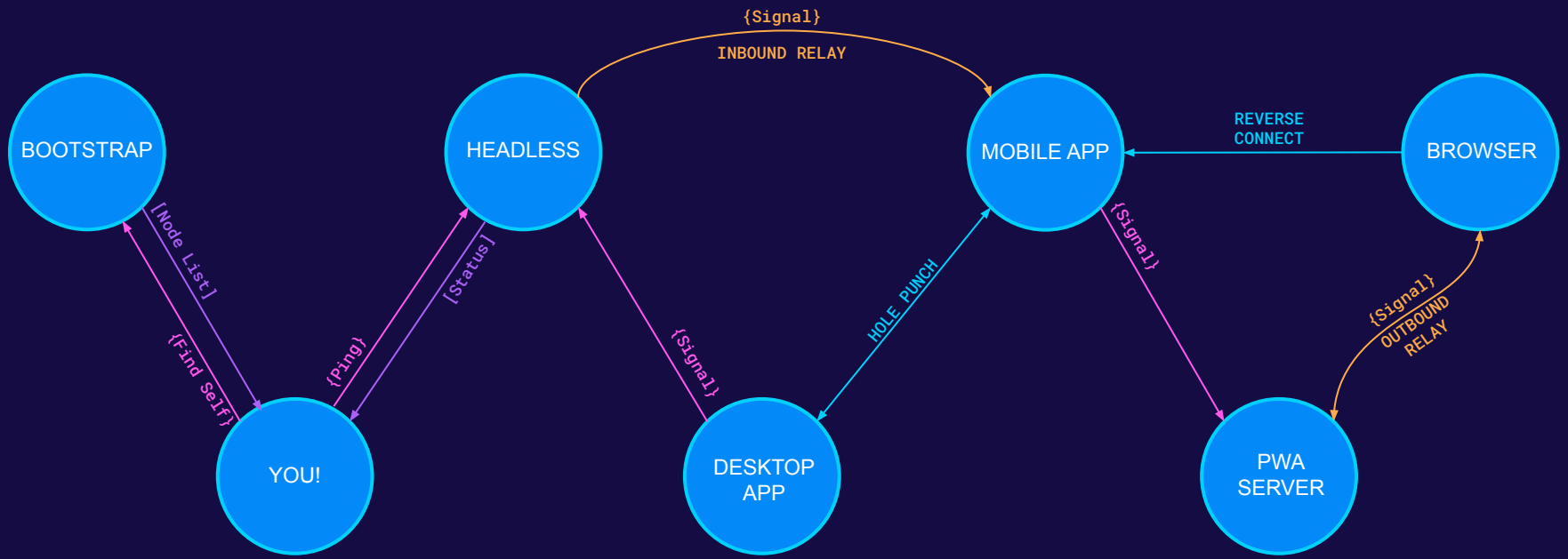
Websockets

Sequenced, reliable streams
Support for HTTP and HTTPS delivery
All nodes speak Websockets
Browsers can directly contact any other node on the network

Network Topology

Every node has a 256-bit public key ‘node id’.
Nodes arrange their routing table with a ‘distance’ metric
Routing tables are ‘buckets’ like Kademlia DHT

www.veilid.com



Veilid

Bootstrapping

Bootstrap nodes aren't 'special'. Any node can bootstrap a Veilid network.
Networks can be 'keyed' to keep nodes off that don't have the key.
You can join the 'big Veilid network' or make your own isolated network.

www.veilid.com

Ask Bootstraps To 'Find Self'

A single initial DNS TXT record request returns some bootstrap nodes that are known to exist. Those are asked to return nodes that are 'close' to your own node.

Public Address Detection

Nodes are often behind various forms of NAT. Validating one's own public 'Dial Info' is essential for publishing one's Node Info and answering Find Node requests.

Relay Configuration

Low-capability network classes may require the use of Inbound or Outbound relays in order to achieve reachability Nodes help each other out to the best of their ability and incur no penalty for not being able to assist other nodes

Peer Minimum Refresh

Nodes in your routing table are asked to return nodes that are near you as well. Finding nodes close to your own is always harder than finding nodes far away, so we focus on that with our requests.

Network Class Detection

Determining NAT type and what mechanisms can be used to achieve connectivity. Direct connection techniques like reverse connections and UDP hole punching may be inappropriate for some network classes.

Ping Validation

Nodes come and go, change address, and are unreliable. Checking routing table nodes for proof-of-life is done with exponential backoff. Nodes are removed from the routing table on a LIFO basis.

www.veilid.com

All devices are welcome
and treated fairly

You can use the public
Veilid Network or build your own

Nodes help each other like
mutual aid for connectivity

Veilid

Cryptography

VLD0

Strong, appropriate, cryptography choices are essential to the functioning of Veilid.

Veilid provides applications guarantees about how data is handled on the wire and at rest.

Cryptosystems were chosen that work well together and provide a balance of speed and cryptographic hardness

Authentication is Ed25519

Elliptic curve25519 was chosen to provide public/private key authentication and signing capabilities

Key Exchange is x25519

Curve25519 has a DH function that allows nodes to generate a symmetric key to communicate privately

Encryption is XChaCha20-Poly1305

ChaCha20 with a 192-bit extended nonce is a fast authenticated stream cipher with associated data (AEAD)

Message Digest is BLAKE3

BLAKE3 is a extremely fast cryptographic hash that is highly parallelizable and as strong as SHA3-256 and over 17 times faster

Key Derivation is Argon2

Password hash generation should be slow and resistant to GPU attacks
Argon2 was the winner of the 2015 Password Hashing Competition

www.veilid.com

Upgrading

Nothing lasts forever

And cryptography is no exception. As computing power improves and cryptographic attacks evolve, weaknesses in cryptosystems are inevitable

Veilid has ensured that upgrading to newer cryptosystems is streamlined and minimally invasive to app developers, and handled transparently at the node level

Multiple Routing Tables

Because changing cryptosystems changes node ids, there will be different distance measurements between nodes, necessitating a separate routing table per cryptosystem. We support this today.

Typed Keys

Cryptographic keys, signatures, and hashes are all tagged with their cryptosystem to ensure that we know exactly how they were generated and how they should be used and persisted

Migration Support

Reading persisted data will automatically use the correct cryptosystem and will default to always writing it back using the newest/best cryptosystem. This allows for data to be easily migrated just by reading it and writing it back to storage

Simultaneous Cryptosystems

While transitioning cryptosystems, nodes can respond to other nodes using either the old system or the new one, or both.

Secure Storage

Device-level secret storage APIs are available for all platforms

Encrypted table store APIs are exposed to applications to make safe data storage easy

Device data keys can also be password protected

Apps never need to write anything to disk unencrypted

Veilid

ProtectedStore

Device-level Secret Storage

MacOS / iOS Keychain
Android Keystore
Windows Protected Storage
Linux Secret Service

★New Rust Crate: keyring-manager

TableStore

Encrypted Key-Value Database

SQLITE on Native
IndexedDB in Browser
Device Key can be protected
from backup dumping attacks

★New Rust Crate: keyvaluedb

RecordStore

Distributed Hash Table Storage

Encrypted + Authenticated
Subkey support
LRU distributed cache
Per-key multi-writer schemas

BlockStore

Content-addressable Data Distribution

Take What You Give model
Connect and share cloud storage
Bittorrent-like sharding

“COMING SOON”

On The Wire

Data is encrypted at rest and on the wire
Everything is authenticated and encrypted between nodes
All node information is signed

www.veilid.com

All Protocols Same Encryption

Each low-level protocol uses the same message and receipt encapsulation. No protocol is special and all protocols offer the same safety guarantees.

Encrypted And Signed

Messages between nodes are signed by the sender and encrypted for only the receiver. Messages can be relayed without decryption and authentication covers the entire contents including headers.

Everything Is Timestamped

Envelopes include timestamps and unique nonces and reject old or replayed messages.

Node Information Is Signed

When a node publishes routing table entries they are signed. No node can lie about another node's dial info, capabilities, availability, or replay old node info when newer info is available.

www.veilid.com

Everything is
end-to-end encrypted

All storage is
encrypted at rest

Your data is protected
even if you lose your device

RPC Protocol

RPC Schema

Strong, appropriate, cryptography choices are essential to the functioning of Veilid.

Veilid provides applications guarantees about how data is handled on the wire and at rest.

Cryptosystems were chosen that work well together and provide a balance of speed and cryptographic hardness

Schema Language is Cap'n Proto

Cap'n Proto is designed for deserialization speed and schema evolution. Flexible and well supported in Rust

RPC is fully in-schema and documented

Both 'Question/Answer' and 'Statement' RPC modes are supported. All schema fields are documented.

RPC fully supports Private Routing

All private routing structures are expressed in the RPC schema itself, no magic encrypted blobs.

Schema Evolution is built-in

Fields can be added and removed with full backward and forward compatibility. New features won't break older Veilid nodes.

RPC Schema is cryptography-independent

As cryptosystems change, the language spoken by Veilid nodes remains the same.

www.veilid.com

FindNodeQ

Finding Nodes from other nodes' routing tables is a functional primitive for Veilid networking

A node that sends a FindNodeQ RPC question will receive a FindNodeA RPC answer within the allowed RPC latency

The question asks a node to find nodes 'close' to a key in hash space that meet some capability criteria

The answer returns a list of nodes and their Signed Node Info

```
struct Operation @0xbf2811c435403c3b {
    opId                    @0  :UInt64;
    senderPeerInfo          @1  :PeerInfo;
    targetNodeInfoTs        @2  :UInt64;
    kind :union {
        question            @3  :Question;
        statement           @4  :Statement;
        answer              @5  :Answer;
    }
}
```

```
# Things that want an answer
struct Question @0xd8510bc33492ef70 {
    respondTo :union {
        sender              @0  :Void;
        privateRoute        @1  :PrivateRoute;
    }
    detail :union {
        # Direct operations
        statusQ             @2  :OperationStatusQ;
        findNodeQ           @3  :OperationFindNodeQ;

        # Routable operations
        appCallQ            @4  :OperationAppCallQ;
        getValueQ           @5  :OperationGetValueQ;
        setValueQ           @6  :OperationSetValueQ;
        watchValueQ         @7  :OperationWatchValueQ;
```

```
struct OperationFindNodeQ @0xfdef788fe9623bcd {
    nodeId                  @0  :TypedKey;
    capabilities            @1  :List(Capability);
}
```

```
struct TypedKey @0xe2d567a9f1e61b29 {
    kind                    @0  :CryptoKind;
    key                     @1  :PublicKey;
}
```

www.veilid.com

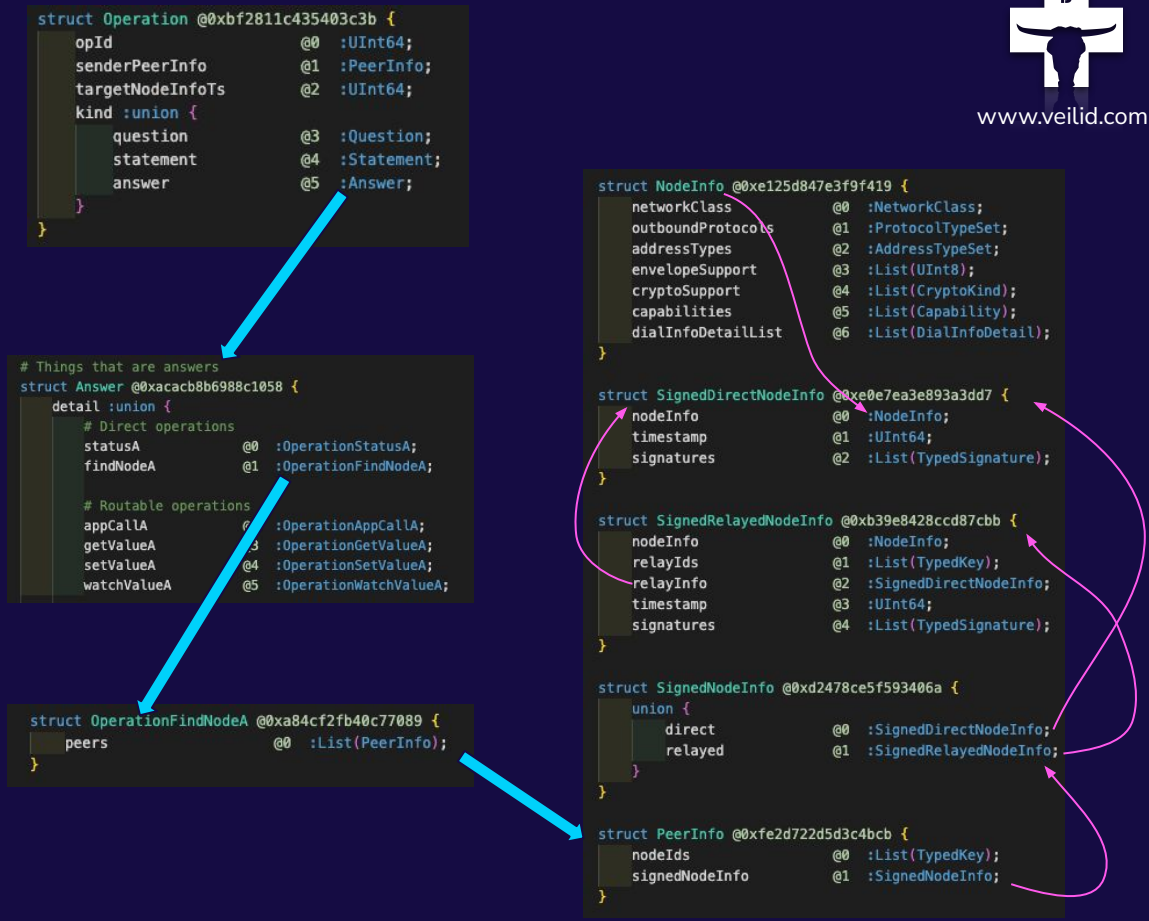
FindNodeA

Finding Nodes from other nodes' routing tables is a functional primitive for Veilid networking

A node that sends a FindNodeQ RPC question will receive a FindNodeA RPC answer within the allowed RPC latency

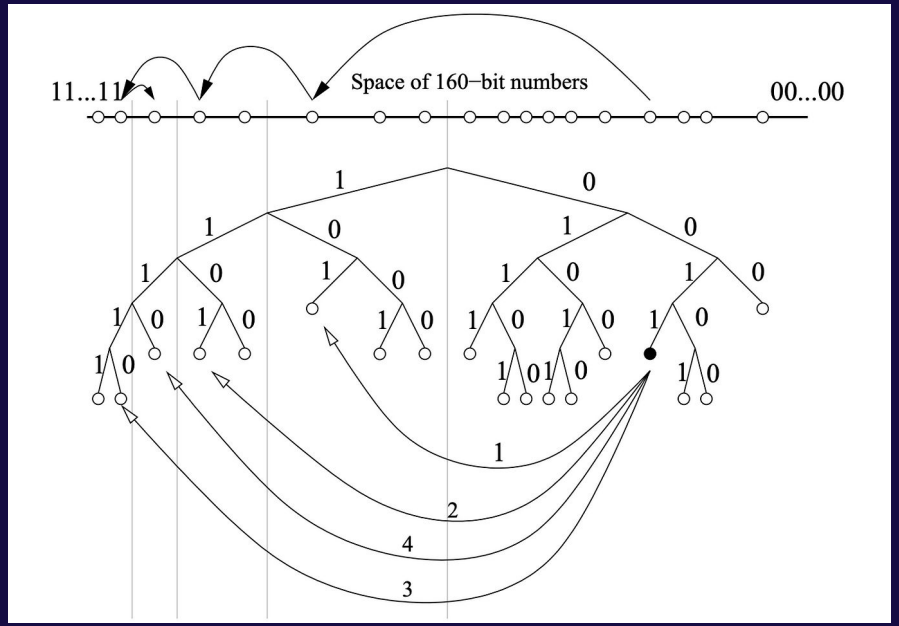
The question asks a node to find nodes 'close' to a key in hash space that meet some capability criteria

The answer returns a list of nodes and their Signed Node Info



Distributed Hash Table

Distributed Hash Tables are a way of storing data in records that have keys that are close to nodes in the network



DHT Is Just ‘Search’

It may look complicated, but all of the DHT algorithms out there are just ‘search’ algorithms. Finding data that is stored on some node somewhere out there.

Improving Search

We built a better DHT by making both search and data locality more relevant. Veilid synchronizes popular data when nodes come and go from the network.

DHT Schema

Veilid DHT is built using GetValue and SetValue RPC operations. Nodes can opt out of DHT storage if they do not want to participate.

Veilid DHT records have schemas that define subkeys that are individually addressable and can have multiple writers

DHT record subkeys have sequence numbers and are eventually consistent across multiple writes and background synchronizations

Veilid Default DHT Schema - DFLT

KEY VLD0:5K43_vkZ6-hvricy-ks9thswlNOfPUQgsNWTfhk4zuQ	
Subkey	**Description**
0	Owner Value 0
1	Owner Value 1
...	...

Schema			
Name	**Offset**	**Length**	**Field Description**
kind	0	4	Schema Format: "DFLT"
o_cnt	4	2	Owner Subkey Count

Veilid Simple DHT Schema - SMPL

KEY VLD0:5K43_vkZ6-hvricy-ks9thswlNOfPUQgsNWTfhk4zuQ	
Subkey	**Description**
0..o_cnt	Value
+..+m1_cnt	Member 1 Value
+..+m2_cnt	Member 2 Value
+..+m3_cnt	Member 3 Value
...	...

Schema			
Name	**Offset**	**Length**	**Field Description**
kind	0	4	Schema Format: "SMPL"
o_cnt	4	2	Owner Subkey Count
m1_key	6	32	Member 1 Public Key
m1_cnt	38	2	Member 1 Subkey Count
m2_key	40	32	Member 2 Public Key
m2_cnt	72	2	Member 2 Subkey Count
m3_key	74	32	Member 3 Public Key
m3_cnt	106	2	Member 3 Subkey Count
	...	...	...

www.veilid.com

www.veilid.com

The DHT gives you full control over your data

Our DHT is not based on a blockchain or a coin

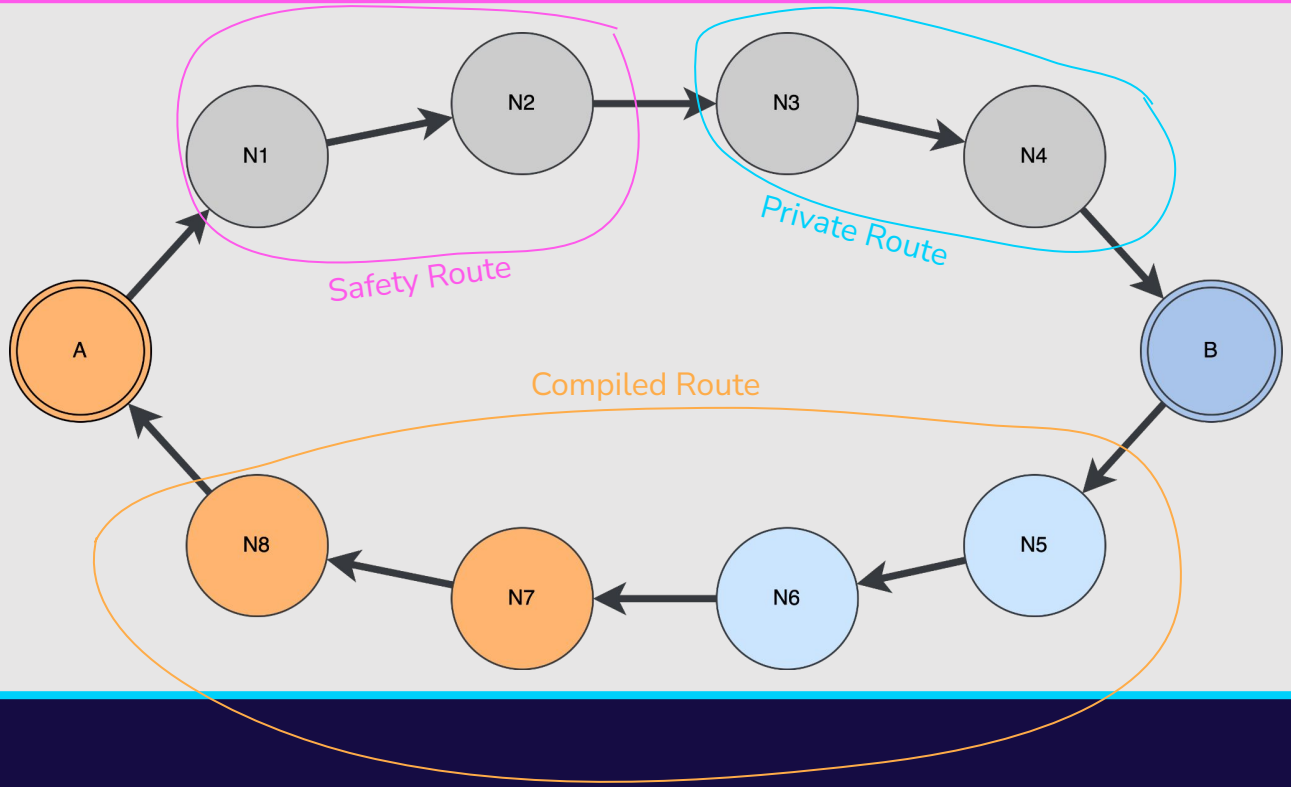
Popular data becomes more available automatically

Private Routing

Private And Safety Routes

Veilid Routes are a combination of source and destination private routing. Because no node can trust any other node to pick the whole route, both source and destination must participate

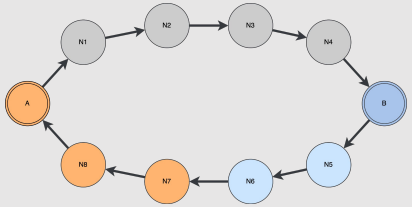
www.veilid.com



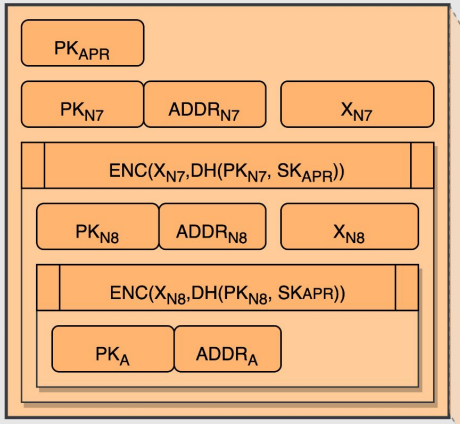
Veilid

Compiled Routes

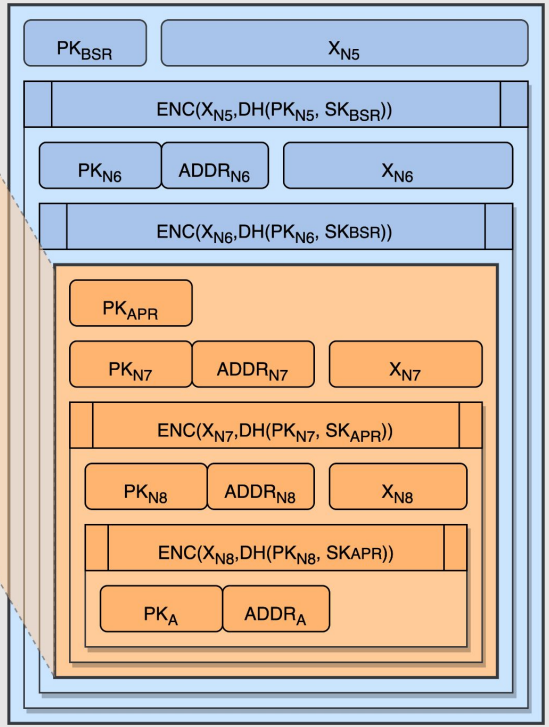
Private Routes are published as a ‘private destination’ and Safety Routes are allocated locally and combined together with a Private Route to form a Compiled Route.



A's Private Route



B's Safety Route To A

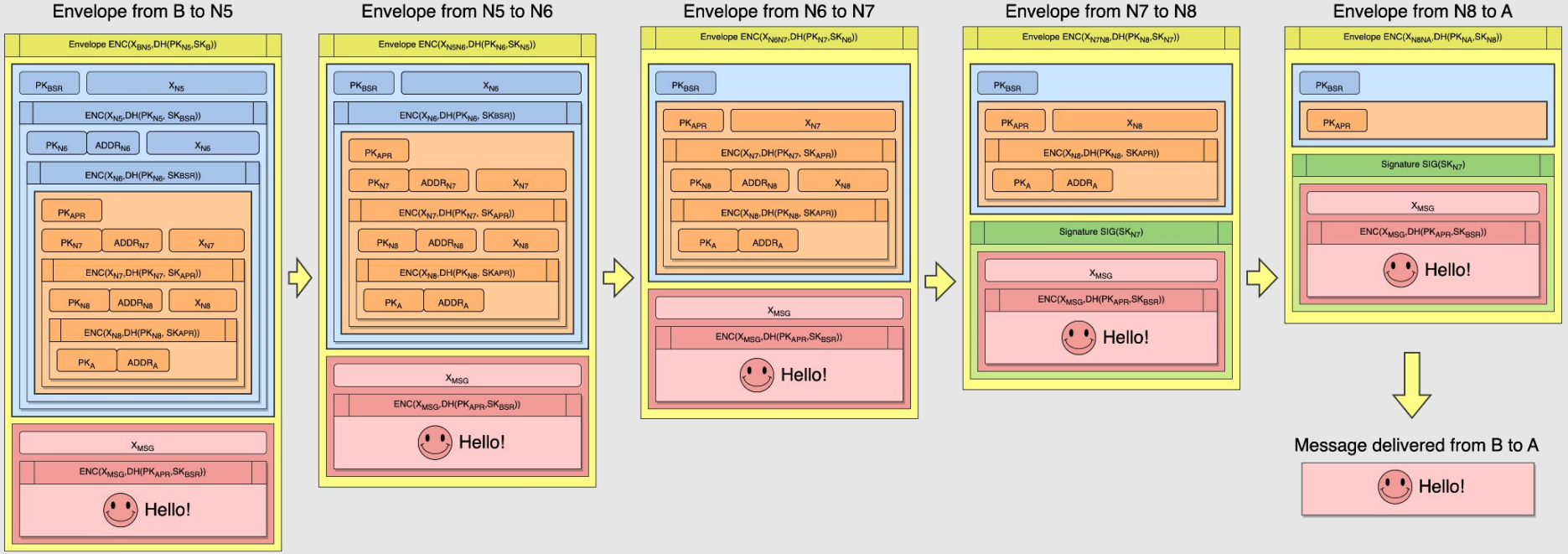


ENC=Authenticated Encryption With Nonce
DEC=Authenticated Decryption With Nonce
PK=Public Key
SK=Secret Key
DH=Diffie-Hellman Symmetric Key
ADDR=IP Address
X=Nonce

Secure Envelopes

Each node hop only knows about the next one
This is similar to onion routing, but assumes that the source is fully in control of the Safety Route and the destination is fully in control of the Private Route

www.veilid.com



Veilid

Toward The Future

Private routing is a balance of performance and security
Applications can make use of higher node hop counts if they desire
Future private routing advancements will be transparent to users

www.veilid.com

Per-Hop Payload Keying

Ensuring that there is nothing common between packets at each hop will reduce the risk of mass data collection being able to deanonymize routes.

Elimination of Hop Counting

Currently the protocol keeps an internal hop count that is not necessary. Efforts should be made to ensure that individual nodes don't know how far along in a route they are.

Simplify Directionality

Routes are currently bidirectional, but are allocated directionally. We may be able to simplify our allocation mechanism by enforcing bidirectionality. Bidirectional routes are faster, but directional routes could provide more anonymity.

Hop Caching

Route hop NodeInfo could be cached to save on-the-wire size as well as speed things up.

Increasing Hop Count

Currently the default is one hop chosen by the Safety Route, and one hop chosen by the Private Route, which leads to three hops total once compiled.

It may be important to increase hop count to 2 for users with critical safety needs and to protect from nation-state-level deanonymization where appropriate.

Existing research (on Tor) suggests that our existing hop count should be sufficient and provide comparable anonymity, but this should be revisited.

www.veilid.com

IP Privacy means your
location is safe too

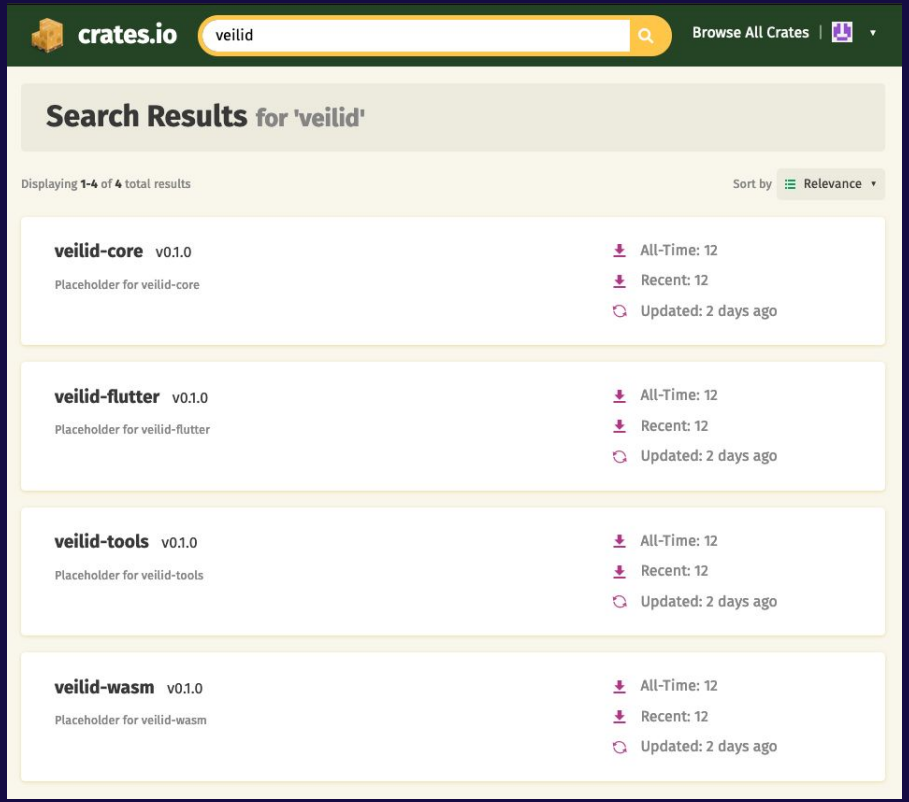
Users don’t have to
do anything to use it

No IP address means no
tracking, collection, or correlation

Veilid

Veilid 🦀 Rust

Veilid is written in Rust
Crates are published and you can use them today!



www.veilid.com

Power User Quick Start

Just read the README.md and clone the repository from GitLab and get started right away!

www.veilid.com

```
wget -O- https://packages.veilid.com/keys/veilid-packages-key.public | gpg --dearmor > /usr/share/keyrings/veilid-packages-keyring.gpg

echo "deb [arch=amd64 signed-by=/usr/share/keyrings/veilid-packages-keyring.gpg] http://packages.veilid.net/apt stable main" > /etc/apt/sources.list.d/veilid.list
```

```
yum-config-manager --add-repo https://packages.veilid.net/rpm/veilid-rpm-test-repo.repo
```

```
git clone --recurse-submodules git@gitlab.com:veilid/veilid.git
```

Veilid Server

In order to run the `veilid-server` locally:

```
cd ./veilid-server
cargo run
```

In order to see what options are available:

```
cargo run -- --help
```

Veilid CLI

In order to connect to your local `veilid-server`:

```
cd ./veilid-cli
cargo run
```

Similar to `veilid-server`, you may see CLI options by typing:

```
cargo run -- --help
```

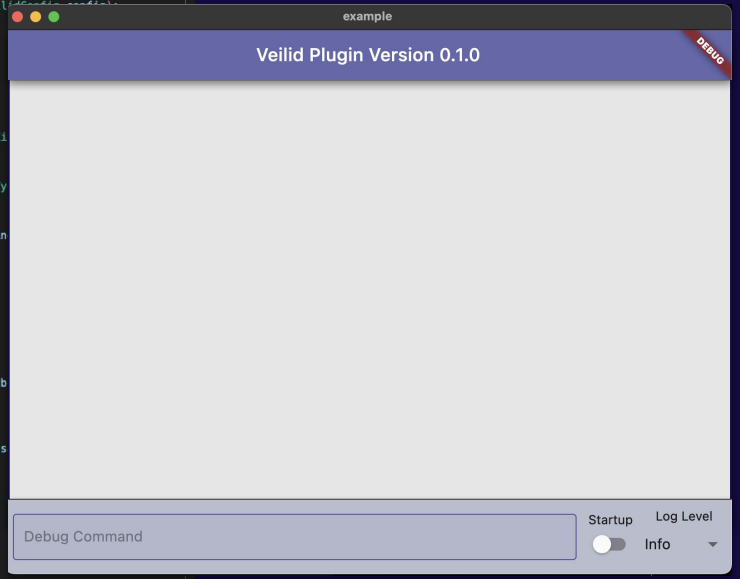
```
Node: VLD0:61EClozRZnsFiXmfbgZegCyVcSJ2x_CPg6ZV8j_19n0
Client Commands:
    exit/quit                       exit the client
    disconnect                      disconnect the client from the Veilid node
    shutdown                        shut the server down
    change_log_level <layer> <level>  change the log level for a tracing layer
                                    layers include:
                                        all, terminal, system, api, file, otlp
                                    levels include:
                                        error, warn, info, debug, trace
    enable [flag]                   set a flag
    disable [flag]                  unset a flag
                                    valid flags in include:
                                        app_messages

Server Debug Commands:
    buckets [dead|reliable]
    dialinfo
    entries [dead|reliable]
    entry <node>
    nodeinfo
    config [configkey [new value]]
    txtrecord
    keypair
    purge <buckets|connections|routes>
    attach
    detach
    restart network
    contact <node>[<modifiers>]
    ping <destination>
    route allocate [ord|*ord] [rel] [<count>] [in|out]
          release <route>
          publish <route> [full]
          unpublish <route>
          print <route>
          list
          import <blob>
          test <route>

Node Id                                          [^]  | Address                 [ ] | Ping [ ] | Down [ ] | Up [ ]
VLD0:3lmRWscDLHFlQj3oqTFLLxj7GfXrmm19faB0biw3A_o     | UDP:157.230.183.173:5150      | 23.06ms  | 1.15KB/s | 1.13KB/s
VLD0:6-FfH7TPb70U-JntwjHS7XqTCMK0lhVqPQ17dJuwlBM     | UDP:170.64.186.46:5150        | 250.58ms | 1.27KB/s | 1.13KB/s
VLD0:7DyMbC1I4kHzLhvJom6YQAdfc2VAsFYvfF1V7ceMQkc     | TCP:161.35.164.16:5150        | 92.37ms  | 1.49KB/s | 1.19KB/s
VLD0:cqxHp4LUuFhKA-0E9UhSXu4FXSgroDiqFU8-E6CPioc     | UDP:159.89.163.27:5150        | 277.60ms | 1.27KB/s | 1.13KB/s
VLD0:ex0-tbt5vpjY5cfOoBrXYfFL3cUKSwniUszcP0EpK_Y     | UDP:157.230.183.173:5150      | 41.56ms  | 1.15KB/s | 1.13KB/s
VLD0:jZKJhPeS3VWO5MvxB3WluqUBf8iKmmrGJxi7jMjmLpM     | UDP:99.88.44.84:39283         | 87.34ms  | 1.28KB/s | 1.13KB/s
Command> help                                                                                                  <Detach>
Connected to [::1]:5959 | [!!  ]+P-l | Down: 29.49KB/s Up: 15.29KB/s | No Tunnels |                | veilid-cli v0.1.0
```

Veilid

Veilid 🐦 Flutter

Veilid has first-class FFI+JS Plugin support for Dart/Flutter and example code to get you started!

```
////////////////////////////////////
/// Veilid singleton factory

abstract class Veilid {
  static Veilid instance = getVeilid();

  void initializeVeilidCore(Map<String, dynamic> platformConfigJson);
  void changeLogLevel(String layer, VeilidConfigLogLevel logLevel);
  Future<Stream<VeilidUpdate>> startupVeilidCore(Veil
  Future<VeilidState> getVeilidState();
  Future<void> attach();
  Future<void> detach();
  Future<void> shutdownVeilidCore();

  // Crypto
  List<CryptoKind> validCryptoKinds();
  Future<VeilidCryptoSystem> getCryptoSystem(CryptoKi
  Future<VeilidCryptoSystem> bestCryptoSystem();
  Future<List<TypedKey>> verifySignatures(
      List<TypedKey> nodeIds, Uint8List data, List<Ty
  Future<List<TypedSignature>> generateSignatures(
      Uint8List data, List<TypedKeyPair> keyPairs);
  Future<TypedKeyPair> generateKeyPair(CryptoKind kin

  // Routing context
  Future<VeilidRoutingContext> routingContext();

  // Private route allocation
  Future<RouteBlob> newPrivateRoute();
  Future<RouteBlob> newCustomPrivateRoute(
      Stability stability, Sequencing sequencing);
  Future<String> importRemotePrivateRoute(Uint8List b
  Future<void> releasePrivateRoute(String key);

  // App calls
  Future<void> appCallReply(String id, Uint8List mess

  // TableStore
  Future<VeilidTableDB> openTableDB(String name, int
  Future<bool> deleteTableDB(String name);

  // Misc
  Timestamp now();
  String veilidVersionString();
  VeilidVersion veilidVersion();
  Future<String> debug(String command);
}
```



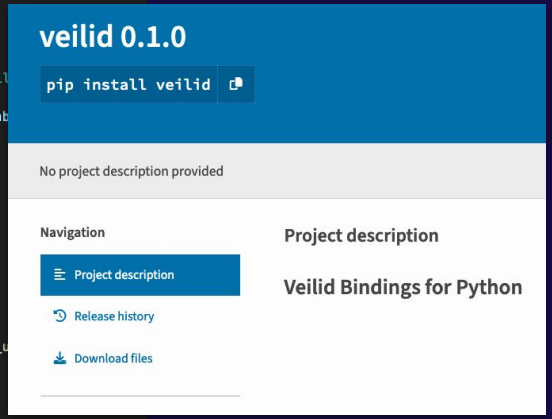
www.veilid.com

Veilid

Veilid 🐍 Python

Veilid has an easy no-compile way to get started learning the Veilid API with Python

```python
# Basic veilid tests

import socket

import pytest
import veilid

from .conftest import simple_update_callback


@pytest.mark.asyncio
async def test_connect(api_connection: veilid.VeilidAPI):
    pass


@pytest.mark.asyncio
async def test_get_node_id(api_connection: veilid.Veil
    state = await api_connection.get_state()
    node_ids = state.config.config.network.routing_tab

    assert len(node_ids) >= 1

    for node_id in node_ids:
        assert node_id[4] == ":"


@pytest.mark.asyncio
async def test_fail_connect():
    with pytest.raises(socket.gaierror) as exc:
        await veilid.json_api_connect(
            "fuahwelifuh32luhwafluehawea", 1, simple_u
        )

    assert exc.value.errno == socket.EAI_NONAME


@pytest.mark.asyncio
async def test_version(api_connection: veilid.VeilidAPI):
    v = await api_connection.veilid_version()
    print(f"veilid_version: {v.__dict__}")
    assert v.__dict__.keys() >= {"_major", "_minor", "_patch"}

    vstr = await api_connection.veilid_version_string()
    print(f"veilid_version_string: {vstr}")
```



www.veilid.com

Veilid

How You Can Help

Work With Us

Veilid is an open-source initiative, designed and implemented in the open. Come join our team and contribute to its growth! Be part of this!

www.veilid.com

Coders And Hackers

We can use more low-level programmers and protocol experts. Platform experts. We want this system to work well for everyone and be a strong foundation for general computing and application development.

Usability Experts

We want to make sure that Veilid and Veilid apps are accessible to everyone. Everyone should be able to make use of Veilid without even realizing they're doing it.

App Developers

You can get started writing a Veilid app today! Got a game idea? Want to port something from a centralized system to a decentralized one? Let's make this happen!

Open Source + Governance

Open source projects deserve to be managed in the open too. We've got an open RFC process for our design and an MPL-2.0 license that ensures that free and commercial entities can contribute safely and legally.

Find Us Online!

Web: www.veilid.com

Twitter: @veilidnetwork

Mastodon: @veilidnetwork

Discord: veilid.com/discord

GitLab: gitlab.com/veilid

See It Live Tonight

Release Party at 8pm!