Anoma Network Architecture

TG X Thoth[a]

[a]Heliax AG

* *E-Mail: tg@heliax.dev*

Abstract

We present the Anoma network and software architecture, and its underlying mathematical and communication models that provide the communication infrastructure for our distributed systems protocols.
The architecture includes a distributed name and identity system using petnames, a trust and reputation mechanism based on service commitments, a modular transport and routing system, and a sovereign domain system that enables pluralistic interoperability.
Domains provide distributed immutable and mutable data storage and dissemination services, where each domain induces a peer-to-peer overlay with its own authentication, membership, topology, message dissemination and storage protocols.
The communication model is inspired by the actor model, where actors communicate via message passing in a distributed system. Message send allows expressing routing & transport selection preferences and constraints. Message delivery semantics are either unreliable or reliable casual delivery. Nodes maintain trust and reputation metrics as well as measurements about other nodes, which influences node selection in routing algorithms. Compositional cryptographic identities induce unicast, multicast (pub/sub), and anycast communication channels.

Keywords: distributed systems, peer-to-peer networks

Received 2024-MM-DD; Revised 2024-MM-DD; Published: 2024-MM-DD)

1. Introduction

1.1. Design overview.

1.1.1. Network architecture. The network architecture consists of sovereign domains that provide heterogenous and interoperable services in the network.

The basic set of network services offered by the system are a distributed identity and name system based on cryptographic identities, a distributed encrypted data storage system for immutable and mutable data, and a data dissemination and synchronization protocol with a publish/subscribe (pub/sub) interface for the update and synchronization of mutable data structures.

Each domain has its own peer-to-peer overlay network and determines its own membership, authentication mechanism, and network protocols. Domains offer publish/subscribe (pub/sub) and storage services for immutable and mutable data structures, which provide the underlying mechanisms for higher level protocols in the system. Domains are interoperable via a common protocol for external requests that allows non-members to query information from domains and thus enables heterogenous interoperability.

The network architecture allows domains with different synchronization models. A domain can either work with a single globally synchronized data structure, or choose to rely on Mergeable Replicated Data Types (MRDT) [KPSJ19] or Conflict-free Replicated Data Types (CRDT) **(author?)** [SPBZ11] that allow domains to synchronize data despite network partitions, and thus multiple instances of the same domain can coexist both on the core network and various edge networks. Such domains are considered *grassroots*, as defined by [Sha24].

Node-to-node connectivity is provided by a modular transport and routing system where different transport protocols offer different ordering, reliability and security guarantees, while the routing system supports unicast, multicast, and anycast communication based on cryptographic destination identities.

Asynchronous user-to-user communication is made possible by encrypted inboxes hosted by user-designated nodes.

The communication model is inspired by the actor model, where an *engine* in our system is a process that communicates via message passing.

A *node* is a set of engine instances in a single trust domain that run on a network-connected computer, and may participate in zero or more domains.

A *user* interacts with the network via *nodes* that allow the user to send and receive messages via user-facing engines, which are typically desktop or mobile applications with a user interface.

Each user, node, pub/sub topic, and domain is addressed by a cryptographic identity. A distributed identity and name system is responsible for creating and updating bindings between names and identities and addresses, such as node identity to transport addresses where a node is reachable at, domain identity to node identities handling join and external requests, pub/sub topic identity to publisher node identities, and user identity to inbox node identities.

2. System model

2.1. Engines. The communication model is inspired by the actor model, where processes, or actors, communicate via asynchronous message passing in a distributed system.

We refer to these processes as *engines* in our model.

Each running engine instance has local state, and may send and respond to messages, react to timer events, and spawn other engines.

A node consists of a set of running engine instances, has a cryptographic identity and a number of transport addresses. Nodes communicate with each other over authenticated and encrypted transport channels.

2.2. Communication patterns. Cryptographic identities induce unicast, multicast (publish/subscribe), and anycast communication channels.

- *Unicast*: direct messages between two engines.
- *Multicast*: messages sent to a pub/sub topic by one or more authorized publishers and delivered to all subscribers.
- *Anycast*: messages sent to any known member of a domain that is designated to serve requests.

2.3. Message routing. Destination routing is used for routing messagess based on cryptographic identities. The network architecture is layered and consists of the following components that enable this:

- *Transport*: responsible for establishing and maintaining encrypted point-to-point connections between nodes. It supports various existing transport protocols, as well as multiplexing node-to-node connections over multiple transport connections.
- *Routing*: responsible for routing messages to their destination identity (a user, node, topic, or domain) via a unicast, multicast, or anycast protocol that corresponds to the type of destination. It performs address resolution using locally available information maintained by the identity and name system, and uses the transport layer for sending unicast messages to nodes.
- *Name & address resolution*: responsible for maintaining a local database of known identities along with the self-signed bindings they advertise that allows resolving user, topic, and domain identities to node identities, and node identities to transport addresses.

2.4. Message sending. Messages are sent to and routed based engine, topic, or domain destination identities. Message send allows expressing routing & transport selection preferences and constraints, such as reliability, ordering, and security considerations.

2.5. Message delivery. Unicast message delivery uses encrypted transport channels between nodes with optional reliability. Multicast communication uses a topic-based reliable publish/subscribe channels with optional content-based filters.

In both cases, reliable causal delivery causal delivery is ensured using a system of causal dependencies and implicit and explicit acknowledgements. This allows the routing system to detect lost messages, try to recover them, and delay delivery of messages until after their causal dependencies have been delivered.

2.6. Domains. A domain is a sovereign subnetwork that has an owner that defines the protocols and authentication mechanism used in the domain. Each domain has its own peer-to-peer overlay, but remain interoperable by using a common protocol for sending external requests to domain members, thereby enabling pluralistic interoperability between the various domains in the network.

Domains provide distributed data storage and dissemination services to participants, and a mechanism for non-members to query information.

They enable creating administrative and security boundaries between different subnetworks that protects domain members from external attacks, while ensuring that information can flow between them when authorized, via dedicated external facing nodes that serve external requests.

2.7. Immutable and mutable data. A distributed data storage system allows storing immutable and mutable data in the network. Immutable data is encrypted and content-addressed, while mutable data uses encrypted pub/sub channels that store and forward an append-only log of messages that form a causal DAG backed by a Blocklace [AS24] data structure. Pub/sub messages may contain partially ordered application messages, references to immutable data, or operations on Mergeable Replicated Data Types (MRDTs) [KPSJ19] and Conflict-free Replicated Data Types (CRDTs) [SPBZ11].

A system of acknowledgements and storage commitments allow distributed indexing of data, where each node builds an index that contain a mapping of content-addressed data identifiers to storage node identifiers, and may also be queried by other nodes to aid locating data in the network.

2.8. Identities, names, and addresses. Each user, node, domain, and pub/sub topic has its own cryptographic identity, which is also used as an address for sending messages.

A distributed name system offers secure, memorable, but not globally unique names, thereby considered a petname system [Sti05]. A self-signed *zone* is assigned to each identity in the system (i.e. to each user, node, topic, and domain), that contains a list of name to record mappings, where records may contain bindings between names, identities, and transport addresses, as well as pointers to other zones, enabling transitivity of names.

Cryptographic identities may be single or compositional that represent a single entity or a group of entities, respectively. In the latter case threshold cryptography is used. Compositional identities can be used to address a group of identities and for shared ownership of domains and pub/sub topics.

2.9. Trust, reputation, and service commitments. Each node collects performance measurements and maintains a reputation score about other nodes in the network that indicates to what extent a node fulfils the services it committed to. This information influences node selection in various protocols, and also be shared with other nodes in the network that may choose to take it into consideration to some extent, especially when shared by nodes deemed sufficiently trustworthy based on a locally assigned trust score.

{{#< include req.md >}}

{{#< include related.md >}}

3. Network Architecture

The network consists of sovereign domains that nodes of users may join and participate in, where each domain defines its own network protocols and authentication mechanism.

Domains offer distributed storage and pub/sub services, that allows domain members to publish, update, and retrieve both immutable and mutable data structures, part of which can be selectively shared with non-members.

In addition to the globally reachable and continously synchronized core network, a domain may also be instantiated on edge networks with opportunistic synchronization, in which case it acts as a *Grassroots System* [Sha24]. This is possible when the domain uses mergeable replicated data types that allow conflict-free data synchronization of concurrent updates.

3.1. Networks.

In the core network nodes typically have stable connectivity and publicly reachable IP addresses, and thus the expected churn rate is low/moderate, i.e. nodes rarely disconnect from and reconnect to the network, mostly due to occasional network or power disruptions. Node churn triggers topology changes that the network must react to.

Edge networks may be connected to the core network or other edge networks either permanently or intermittently, and have high churn rates, i.e. nodes running on mobile devices continuously come and go.

Due to the different dynamics of networks, different network protocols are required for unicast and multicast communication between nodes.

On edge networks mobile nodes of end users can communicate directly with each other. They can reach remote nodes on other networks via relays in the core network, which enable asynchronous communication and basic location privacy. Nodes with stronger location privacy requirements may use onion routing or a mix network.

3.2. Communication primitives.

The basic communication primitive is unicast communication between engines, upon which multicast and anycast communication protocols are built.

Unicast communication is direct communication between nodes or users. Direct node-to-node communication can happen over various transport protocols that provide different reliablity, ordering, security, and location privacy guarantees. When connecting to a remote node, a transport selection mechanism selects an appropriate transport protocol based on constraints and preferences that may be specified per node or even per message. User-to-user communication is done via relays and is asynchronous and encrpted.

Multicast communication is done via topic-based publish/-subscribe (pub/sub) protocols inside domains, while anycast requests can be sent by external nodes to domain members who serve external requests.

Routing is based on the destination address in each message, which is the user or node identity for unicast, the pub/sub topic identity for multicast, and the domain identity for anycast communication.

3.3. Nodes.

The network consists of nodes that may communicate with each other either directly or via relays, and may participate in any number of domains.

Each node is addressed by a cryptographic identity, and has a number of transport addresses where it is reachable at.

As part of a decentralized discovery and address resolution mechanism, each node issues a self-signed, versioned *Node Advertisement* that contains the current transport addresses of the node The node publishes and updates these advertisements in a pub/sub topic that corresponds to the node identity.

```
#| label: NodeAdvert
#| fig-cap: Node Advertisement

type NodeAdvert :=
  mkNodeAdvert {
    id : NodeID;
    addrs : Set (Pair TransportAddress Priority);
    version : Nat;
    created : AbsTime;
    sig : Commitment;
  };
```

3.4. Users.

Users send and receive messages, publish, and retrieve data via nodes in the network. Typically, users run a node on each of their devices, where each node controlled by the user has access to part of the user's messages and data, have access to all domains the user is a member of, and have its own set of pub/sub topic subscriptions.

Each *user* has its own cryptographic identity, which allows users to establish cryptographic channels among them, even asynchronously with the help of relay nodes. Users may also join domains, which allows them to access and publish immutable data blobs, as well as read and update mutable data structures.

A `UserAdvert` ?] allows a user to designate relay nodes for receiving encrypted messages asynchronously.

Users may opt to use relays to send and receive messages that store and forward encrypted messages for them. This enables secure asynchronous messaging, and offers basic location privacy for users, as they can advertise the relay as their point of contact to other nodes instead of revealing their own IP address.

For stronger location privacy guarantees, users may opt to use multiple relays, connect to relays using privacy-preserving transports, such as onion routing offered by the Tor network, or use a mix network that offers pub/sub and storage services.

```
#| label: UserAdvert
#| fig-cap: User Advertisement

type UserAdvert :=
  mkNodeAdvert {
    id : UserID;
    nodes : Set NodeAdvert;
    relays : Set NodeAdvert;
    prekeys : Set PreKey;
    version : Nat;
    created : AbsTime;
    sig : Commitment;
  };
```

3.5. Domains. The network consists of sovereign domains. A domain is a sub-network with its own *cryptographic identity*. Domains enable *pluralistic interoperability*, i.e. different domains may run different protocols internally, while at the same time able to respond to external requests, which uses a common protocol understood by every domain.

A domain may be instantiated in one or more interconnected overlay networks that may be partitioned occasionally when the protocols used in the domain allow so. This enables both single-instance domains in the core network with globally synchronized data structures, as well as grassroots [Sha24] domains, which may have multiple instances simultaneously in the core network and on various edge networks and may often be partitioned, hence they employ mergeable data structures for data synchronization, such as CRDTs [SPBZ11] and MRDTs [KPSJ19].

Domain creation is permissionless, it only requires a *Domain Advertisement* **?**] signed by the domain owner(s), which contains the domain configuration, and may be distributed to current and invited members either via messages in the network, or out-of-band. The *Domain Advertisement* specifies the authentication mechanism used to join the domain, the network protocols used in the domain, and the list of nodes that serve join and external requests.

```
#| label: DomainAdvert
#| fig-cap: Domain Advertisement

type DomainAdvert :=
  mkDomainAdvert {
    id : DomainID;
    join_req : Set NodeAdvert;
    ext_req : Set NodeAdvert;
    protocols : Map String Protocol;
    version : Nat;
    created : AbsTime;
    sig : Commitment;
  };
```

The exact set of protocols used in a domain is defined by the `DomainAdvert`, it typically consists of a membership protocol, a topic-based publish-subscribe protocol, and a distributed storage protocol.

In a domain, member's nodes establish direct connections among each other. Domains with stronger privacy requirements where members require location privacy may opt to use a mix network that provides pub/sub and storage services.

3.5.1. Pub/sub topics. A topic-based publish/subscribe (pub/sub) protocol allows domain members to subscribe to topics of interest, where authorized publishers for a given topic may publish messages. For each topic subscribers may also define additional content-based filters according to their interests.

Topic creation and subscription are permissionless, while publishing is permissioned. In addition, publishers may encrypt messages published in topics, to enable end-to-end security between publishers and subscribers when relays are added to the topic to aid content distribution and availability.

A topic is defined by a signed *Topic Advertisement*.

```
type TopicAdvert :=
  mkTopicAdvert {
    id : TopicID;
    publishers : List MemberID;
    relays : List NodeID;
    tags : List String;
    version : Nat;
    created : AbsTime;
    sig : Commitment;
  };
```

3.5.2. Pub/sub messages. Pub/sub messages published in a topic must be signed by an authorized publisher and may either contain operations on a MRDT or CRDT, a no-op that serves as an explicit acknowledgement of previous messages, or a reference to an immutable storage object.

Before a message is accepted or forwarded by a subscriber it must pass a number of validation tests which ensure that publishers follow the rules set by the topic owner in the *Topic Advertisement* **?**]: - It must be signed by an authorized publisher - It must contain a valid operation the publisher is authorized to perform - It must form a linear history with the publisher's own previous messages

Messages in a topic form a Blocklace [AS24] data structure that represent a DAG of causal history of messages published in the topic. Each message references the publisher's own previous message, as well as the last locally known message for each publisher. This establishes a partial order and allows subscribers to detect missed messages, since the DAG is connected and all messages are eventually delivered to all subscribers. Missing messages can be recovered by sending explicit requests to other subscribers or publishers in the topic.

The message history recorded in the DAG allows subscribers to detect malicious publishers that send different messages to different nodes by publishing multiple independent messages that do not depend on or reference each other, as publishers are required to publish a linear history of messages to prevent such equivocations that can create ambiguity and inconsistency.

Messages may also specify causal dependencies on other messages from the same topic, which must be delivered before a subscriber can process the message.

TODO: DAG diagram

3.5.3. Immutable data. Immutable data is stored by a distributed blob storage protocol, where blobs are binary data objects that are first chunked to smaller, more manageble pieces, then encrypted using convergent encryption, and finally organized in a Merkle tree, where leaf nodes contain data chunks and internal nodes contain pointers to either leaf nodes. or other internal nodes. Multiple levels of internal nodes may be necessary when the chunk size is smaller than the size of all chunk hashes.

The blob identifier is the root chunk in the Merkle tree, which is necessary to request the blob from the network, and allows traversing the Merkle tree but does not allow decryption. To be able to decrypt the blob, a decryption key must be passed along with the root chunk ID.

Nodes may issue *storage commitments*, that allows them to commit to storing a blob for a specific amount of time. These storage commitments are then published in pub/sub topics or shared directly, which allows nodes to build a local index of known blobs along with the nodes that store it.

When requesting a blob, the request is sent to one of the locally known storage nodes. If none is available, then a random walk is performed along the domain overlay to locate the blob. When successful, one or more storage commitments are returned to the requestor.

3.5.4. Mutable data. The Blocklace itself is a CRDT, an append-only log of messages, which serves as a basis for constructing more complex mergeable data structures: each topic corresponds to a single data structure that comes with a data type definition in the *Topic Advertisement* ?], which defines the set of allowed operations on the data structure (e.g. a *counter* may have a single *increment* operation, or a *grow-only set* an *add* operation).

Operations on mutable data structures may reference immutable data blobs, this allows e.g. constructing a mutable *set* of immutable blobs.

In a domain, the set of available mutable data structures are published in an index, which contains a map of referencable data *paths* to cryptographic *topic IDs* that contain the data structure.

This allows a unified mechanism for defining, accessing, and updating mutable data structures used by the domain. Mutable data structures are used by the membership protocol for the set of domain members, and by the name system to map names to identities.

3.5.5. Join and external requests. The *Domain Advertisement* ?] serves the purpose of advertising a list of nodes that serve join requests and external requests from non-members.

Join requests may require an authentication mechanism, such as a pre-shared key, public key certificate, or zero knowledge proof.

External requests are used by non-members to request information from the domain without joining, and may require an authentication token, such as a hash-based message authentication code (MAC).

Anycast routing is used to send messages to any member of a domain that is designated to serve join or external requests.

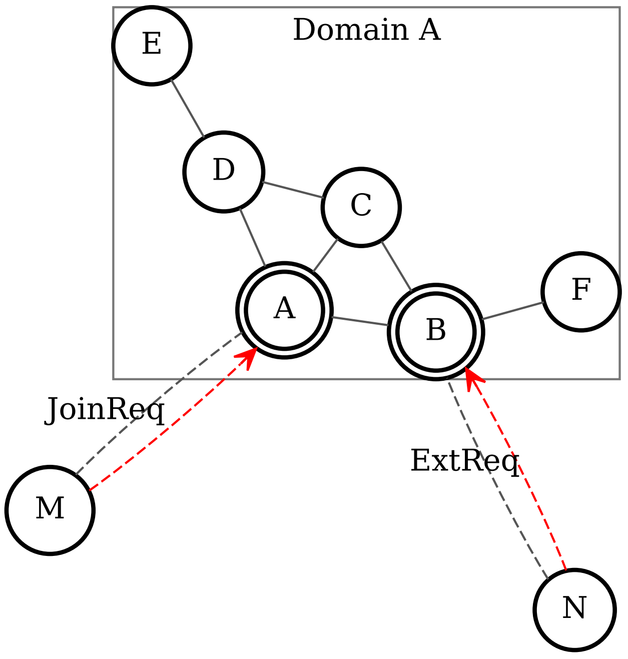

Figure 1. Domain overlay (A-F). A & B handle external requests from N & M

3.5.6. Membership. The domain's *membership* is the set of nodes that have joined and are allowed to participate in the network protocols of the domain.

Domains employ a full membership protocol, which have a number of advantages over partial membership protocols: it avoids many attack vectors by Byzantine members that partial membership protocols suffer from, such as view poisoning and eclipse attacks, which are costly to defend against (e.g. Brahms [BGK$^+$09], a partial membership protocol, uses proof-of-work to limit view pushes a member is allowed to perform).

Full membership also reduces protocol complexity, and offers faster convergence for membership updates at the expense of storing more state and a decrease in scalability. However, it can still support tens of thousands of members, as shown by Fireflies [JRVJ15].

Membership is stored as a mutable data structure in the domain as a permissioned CRDT map where each member can update only its own record. The membership record contains the *Node Advertisement* of the member, as well as its pub/sub topic subscriptions with optional content filters per topic.

Updates to the CRDT map are disseminated via a pub/sub topic that all members subscribe to. Periodic acknowledgement messages in this topic serve as keep-alive messages that allow other members to determine when a node was last active, and thus serves as a mechanism to find recently online nodes in the domain overlay.

3.6. Communication diagrams.

TODO remove router from intra-node diagram

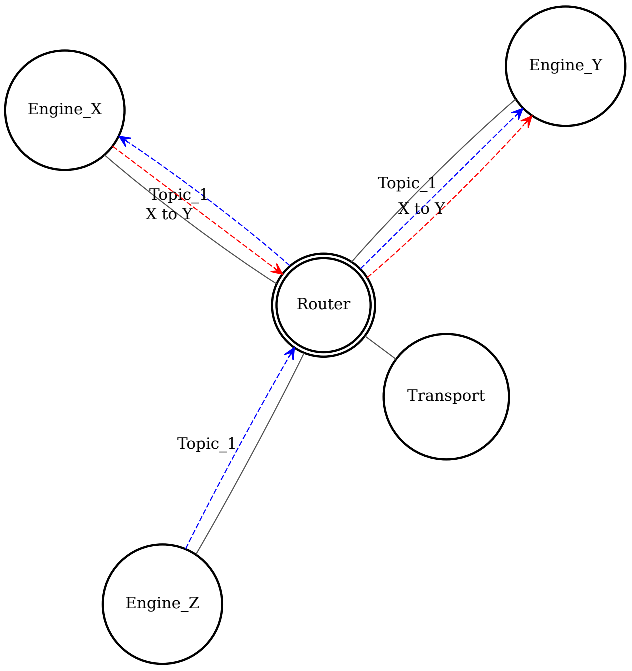

Figure 2. Intra-node communication

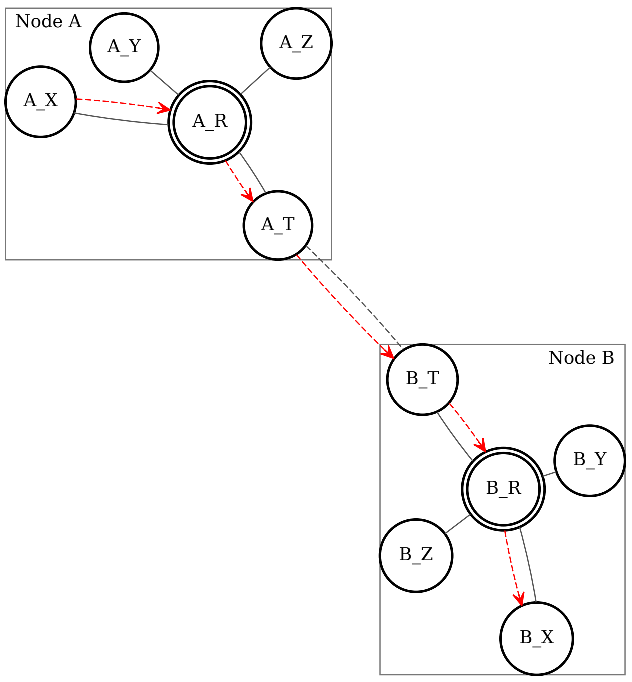

Figure 3. Unicast communication

4. Node architecture

Each node has a cryptographic identity that uniquely identifies it in the network. A node consists of a set of running engines. Local engines communicate with each other directly, while they reach remote nodes via the Router and Transport engines.

The Transport engine is responsible for maintaining authenticated and encrypted network connections between nodes via a modular transport system that supports various transport protocols with different reliability, latency, and privacy guarantees. Transport protocols initially supported include QUIC and TLS, later this can be extended with onion-routed, mix, mesh, and delay-tolerant network protocols.

The Router engine routes incoming and outgoing messages from and to local engines, based on the destination address of each message, which consists of a node and engine identity.

4.1. Addressing.

Nodes, pub/sub topics, and domains are addressed by a cryptographic identity. The node address is defined as the identity of the *Router Engine* of the node.

Engine identities are composed of the node identity and a per-node unique engine identifier.

In order to connect to a node, its transport addresses must be known locally. Similarly, for domains, one or more domain members' node identities must be known. For this purpose signed advertisement messages are used, which bind addresses together from different layers:

- *NodeAdvert*: binds a node address to transport addresses
- *DomainAdvert*: binds a domain address to node addresses

These are transmitted either as part of network protocols or out-of-band, and are stored by the *Network Identity Store Engine* of each node.

4.2. Transport.

The *Transport Engine* is responsible for establishing and maintaining network connections to other nodes via transport protocol modules, which provide authenticated and encrypted transport channels between nodes.

Transport maintains a pool of open connections to other nodes, and reuses them whenever a message needs to be sent to one of the connected nodes.

The modular transport architecture allows a node to support various transport protocols, such as protocols over IP, public-key addressed overlay networks, mesh networks, and delay-tolerant networks.

All transport protocols must provide authenticated and encrypted connections between nodes. Ordering and reliability guarantees may vary, an appropriate transport protocol is selected according to transport constraints and requirements specified in messages to transmitted.

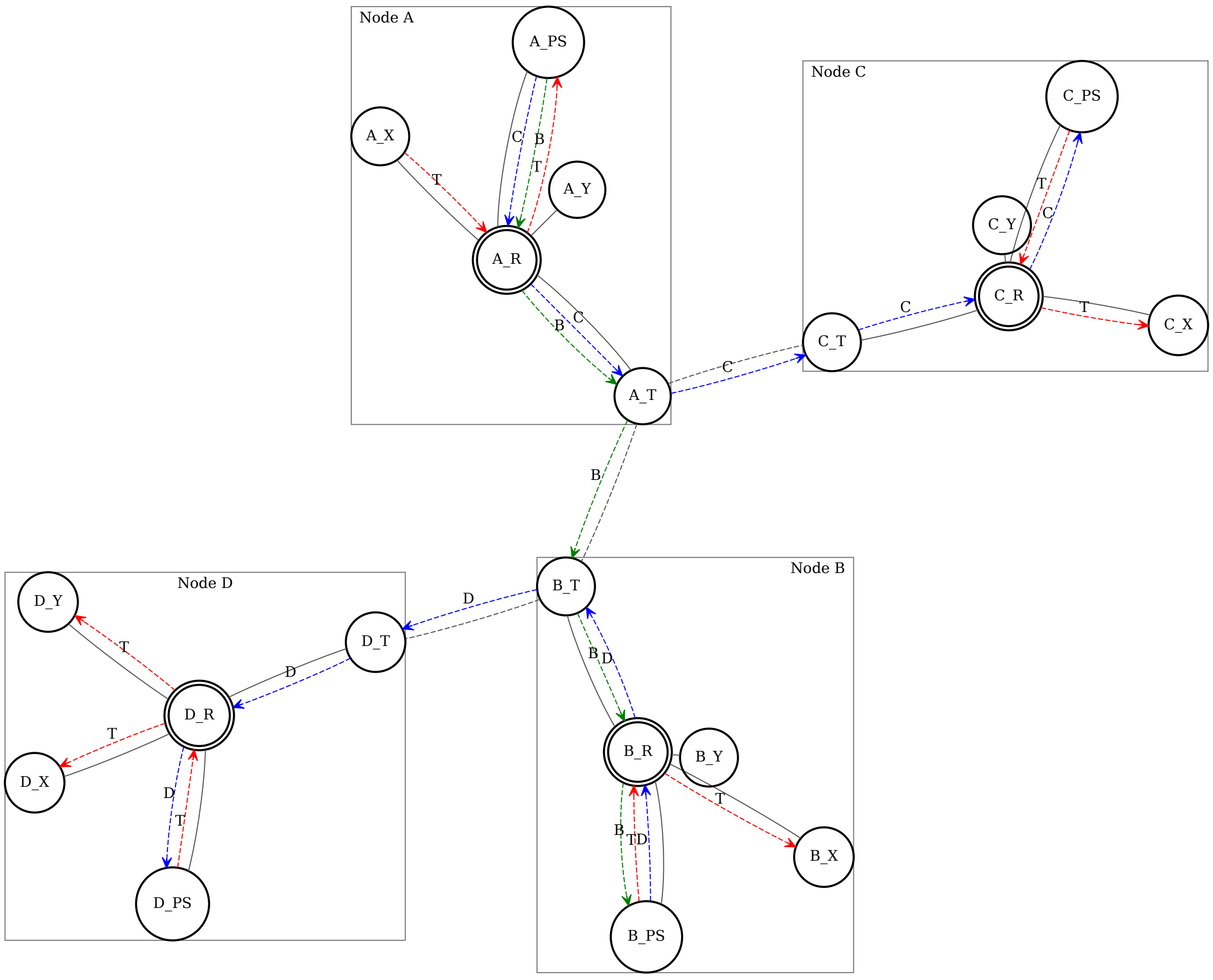

Figure 4. Multicast communication

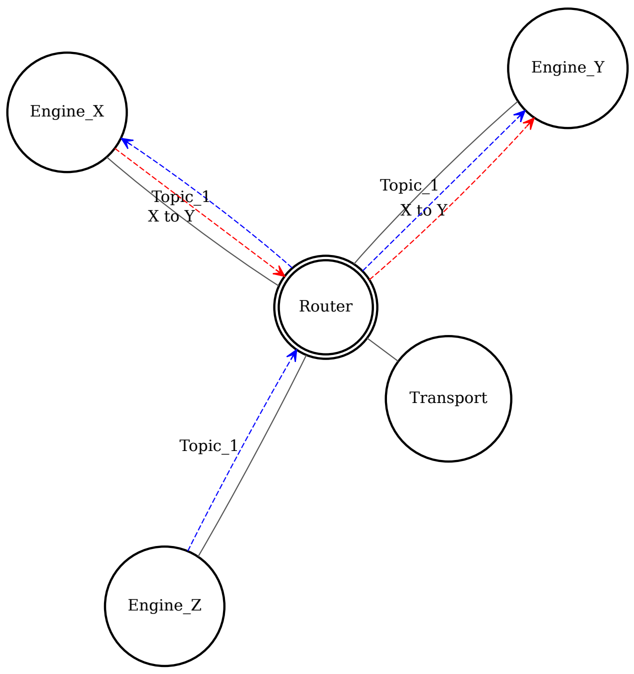

Figure 5. Intra-node communication

The preferred transport protocol is nQUIC [HAWSC19], as this provides the best security and lowest connection establishment overhead out of the initially supported set of protocols. Alternatively, QUIC-TLS [TT21] may be also used when nQUIC is not supported by an implementation. QUIC uses UDP, which may be blocked on 2-5% of networks [KT22], in which case TLS 1.3 [Res18] is available as a fallback option that runs over TCP.

Certain nodes may run in web browsers, which may use either the WebTransport [AJV24] or WebSocket [FM11] protocols. These allow HTTP connections to be upgraded to bidirectional streams over QUIC-TLS and TLS 1.3, respectively.

A *Transport Address* contains all information necessary for a transport module to establish a connection, including the long-term public key used by the transport protocol. In case of TLS, which is used by all of the above protocols, this is the public key of the self-signed X.509 certificate. *NodeAdvert* messages contain one or more transport addresses to facilitate connection establishment.

4.2.1. QUIC. QUIC [IT21] allows stream multiplexing with ordered, reliable delivery for each stream independently, which reduces latency by removing head-of-line blocking across different streams. QUIC also supports unreliable, unordered delivery via a protocol extension [PKS22].

When using reliable, ordered communication, a stream is created for each source-destination engine address pair. Unreliable messages are useful for protocols such as multicast and gossip messages that do not require acknowledgement, as these protocols handle resilience themselves.

The most common transport security protocol used with QUIC is TLS [TT21]. However, TLS has the drawback of using an X.509 certificate, which is unnecessary for decentralized networks that do not rely on trusted Certificate Authorities (CA).

Hence nQUIC [HAWSC19] is the preferred alternative that uses the Noise Protocol Framework [Per18] with the IK pattern to secure connections.

4.2.2. Connection establishment. A signed *NodeAdvert* binds a node ID to one or more transport addresses. In order to establish a connection, a *NodeAdvert* must be known by the local node.

A transport protocol is chosen based on the transport constraints and requirements specificied in the message, with a fallback to local configuration associated with nodes or the defaults.

TODO:

- 0-RTT
- TLS cert verification

4.2.3. Serialization. Serialization (marshalling) is necessary before transmitting messages over the network.

Requirements:

- binary
- schema
- versioning
- fast
- suitable candidates
  - BARE
  - ProtoBuf

5. Concluding remarks

6. Future work

References

References

AJV24. Bernard Aboba, Nidhi Jaju, and Victor Vasiliev. Webtransport. 2024. (cit. on p. 8.)

AS24. Paulo Sérgio Almeida and Ehud Shapiro. The blocklace: A universal, byzantine fault-tolerant, conflict-free replicated data type, 2024. (cit. on pp. 2 and 4.)

BGK+09. Edward Bortnikov, Maxim Gurevich, Idit Keidar, Gabriel Kliot, and Alexander Shraer. Brahms: Byzantine resilient random membership sampling. *Computer Networks*, 53(13):2340–2359, 2009. (cit. on p. 5.)

FM11. Ian Fette and Alexey Melnikov. The websocket protocol. 2011. (cit. on p. 8.)

HAWSC19. Mathias Hall-Andersen, David Wong, Nick Sullivan, and Alishah Chator. nQUIC: Noise-based QUIC packet protection. Cryptology ePrint Archive, Paper 2019/028, 2019. https://eprint.iacr.org/2019/028. (cit. on p. 8.)

IT21. Jana Iyengar and Martin Thomson. Quic: A udp-based multiplexed and secure transport. 2021. (cit. on p. 8.)

JRVJ15. Håvard D Johansen, Robbert Van Renesse, Ymir Vigfusson, and Dag Johansen. Fireflies: A secure and scalable membership and gossip service. *ACM Transactions on Computer Systems (TOCS)*, 33(2):1–32, 2015. (cit. on p. 5.)

KPSJ19. Gowtham Kaki, Swarn Priya, KC Sivaramakrishnan, and Suresh Jagannathan. Mergeable replicated data types. *Proceedings of the ACM on Programming Languages*, 3(OOPSLA):1–29, 2019. (cit. on pp. 1, 2, and 4.)

KT22. Mirja Kühlewind and Brian Trammell. Applicability of the quic transport protocol. 2022. (cit. on p. 8.)

Per18. Trevor Perrin. The noise protocol framework. 2018. (cit. on p. 8.)

PKS22. Tommy Pauly, Eric Kinnear, and David Schinazi. An unreliable datagram extension to quic. 2022. (cit. on p. 8.)

Res18. Eric Rescorla. The transport layer security (tls) protocol version 1.3. 2018. (cit. on p. 8.)

Sha24. Ehud Shapiro. Grassroots systems: Concept, examples, implementation and applications, 2024. (cit. on pp. 1, 3, and 4.)

SPBZ11. Marc Shapiro, Nuno Preguiça, Carlos Baquero, and Marek Zawirski. Conflict-free replicated data types. In *13th International Conference on Stabilization, Safety, and Security of Distributed Systems*, SSS 2011, pages 386–400. Springer LNCS volume 6976, October 2011. (cit. on pp. 1, 2, and 4.)

Sti05. Marc Stiegler. Petname systems. *HP Laboratories, Mobile and Media Systems Laboratory, Palo Alto, Tech. Rep. HPL-2005-148*, 2005. (cit. on p. 2.)

TT21. Martin Thomson and Sean Turner. Using tls to secure quic. 2021. (cit. on p. 8.)