HOPR - a Decentralized and Metadata-Private Messaging Protocol with Incentives

Amira Bouguera, Robert Kiel, Dr. Sebastian Bürgel

April 2021, V2

Abstract

1 Introduction

Internet privacy, also commonly referred to as online privacy, is a subset of data privacy and a fundamental human right. We consider privacy to revolve around control, use and disclosure of one's personally identifiable information. However, our increasingly technologically-driven world puts great pressure on privacy. This is why several actors in the space are continuesly developinng different solutions to achieve anonymous communication and thus leveraging internet privacy.

1.1 State of the art

1.1.1 Privacy protocols

VPN or Virtual Private Network is a privacy technology that allows creating a secure connection on the internet by building an encrypted tunnel between the client and server. VPN also provides a new IP address (the one of the proxy) to help bypass censership and geolocalization blocks. Due to the centralized trust model VPNs has, they suffer from inherent weaknesses like being a single point of failure. They are also vulnerable to powerful network eavedropers who can track the routed network traffic.

Those privacy concerns have motivated the creation of a new model of VPNs, dVPNs (Decentralized Virtual Private Networks) with no central authority and backed by blockchain technology. Orchid, Sentinel and Mysterium are the most known projects in the blockchain ecosystem providing dVPN solutions. These projects however lack privacy, performance and reliability guarantees due to the fact that regular internet users can be both bandwidth providers and normal users. One of the main reasons behind using decentralized VPN instead of a centralized one is to implement a no traffic logging policy as a way to increase

privacy, this however introduces the risk of using the service illegally (terrorism, drug smugling, child pornography...etc) which holds service providers reliable to government authorities without them having any legal protection as large VPN providers would have. There have also been some incidents reported, where unaware dVPN users have been (ab)used as exit nodes through which DDoS attacks were performed. Similarly to VPN, users have no guarantee whether a dVPN might inspect, log, and share any of their traffic. A promising project called VPN^0 started by the Brave team which provides better privacy guarantees by leveraging zero knowledge proofs to hide traffic content to relay nodes with traffic accounting and traffic blaming capabilities as a way to combat the weaknesses of other dVPN solutions. This project is still research oriented though.

Another existing solution for network privacy is Tor, an open-source software for enabling anonymous communication. Tor is based on onion routing which encapsulates messages in layers of encryption and transmits them through a series of network nodes called onion routers. Tor however is susceptible to end-to-end correlation attacks conducted by an adversary who can eavesdrop the communication channels.These attacks reveal a wide range of information like the identity of the communicating peers. Another project based on onion routing is I2P peer-to-peer network. I2P has different design choices from those of Tor:

- Packet switched instead of circuit switched: Tor allocates connection to long lived circuits, this allocation does not change until either the connection or circuit closes. On the other hand, routers in I2P maintain multiple tunnels per destination which increases significantly the scalability and resilience against failures since packets are used in parallel.

- Unidirectional instead of bidirectional tunnels: which makes deanonymization harder since tunnel participants see half as much data in unidirectional tunnels and need two sets of peers to be profiled.

- Peer profiles instead of directory authorities: I2P's network information is stored in a DHT (information in the DHT is inherently untrusted) while Tor's relay network is managed by a set of nine Directory Authorities.

I2P are vulnerable to eclipse attacks since no I2P router has a full view of the global network (similar to other peer-to-peer networks) and they also protect against only local adversaries (like Tor) and thus vulnerable to timing, intersection and traffic analysis attacks. I2P have also showed to be vulnerable to sybil and predecessor attacks inspite of the different contermeasures implemented to defeat them.

Mixnets are overlay networks of mix nodes that route messages anonymously similarly to Tor. First mixnet paper in 1981 by David Chaum used a cascade topology where each node receives a batch of encrypted messages, decrypts, randomly permutes packets, and transfers them in parallel. Cascade topology

makes it easy to prove the anonymity properties of a given mixnet design for a particular mix, however, it does not scale well with respect to increasing mixnet traffic and is also susceptible to traffic and active attacks. Since then, research has evolved to provide solutions with low latency while still providing high anonymity by using a method called cover traffic. Cover Traffic is designed to hide communication messages among random noise. An external adversary able to observe the message flow should not be able to discriminate communication messages from random noise messages which increases privacy. What differentiates mixnets from Tor is that mixnets are designed to provide metadata protection from global network adversaries by using cover traffic. Because mixnets add extra latency to network traffic, they are better-suited to applications that are not as sensitive to increased latency, such as messaging or email applications while applications like real-time video streaming are better suited for Tor.

One of the well known projects is Loopix. Loopix leverages cover traffic to resist traffic analysis while still achieving low- to mid-latency. To this end Loopix employs a mixing strategy that we call a Poisson Mix that is based on the independent delaying of messages, which makes the timings of packets unlinkable.

The goal of each one of these projects is to acheive low latency, low bandwidth overhead and strong anonymity or as we call it the anonymity trilemma. We present in the following a comparison table (from the Loopix paper) between different anonymous communication systems.

	Low Latency	Low Communication Overhead	Scalable Deployment	Asynchronous Messaging†	Active Attack Resistant	Offline Storage*	Resistance to GPA
Loopix	✓	✓	✓	✓	✓	✓	✓
Dissent [43]	✗	✗	✗	✗	✓	✗	✓
Vuvuzela [42]	✗	✗	✓	✗	✓	✗	✓
Stadium [41]	✗	✓	✓	✗	✓	✗	✓
Riposte [10]	✗	✗	✓	✗	✓	✗	✓
Atom [29]	✗	✓	✓	✗	✓	✗	✓
Riffle [30]	✓	✓	✗	✗	✓	✗	✓
AnonPoP [23]	✗	✓	✓	✗	✗	✓	✓
Tor [19]	✓	✓	✓	✓	✗	✗	✗

Figure 1: Comparison between anonymous communication systems

All the previous mentioned projects except VPN and dVPNs lack the economic incentive which could result in scaling issues and poor performance. Tor and I2P for example rely on donations and government funding only which covers the cost of running a node. This has discouraged volunteers to join the network and the number of routers in both networks hasn't increased much for the last few years. Mixnets are also based on a group of volunteer agents who lack incentives to participate. Some solutions have proposed adding digital coins to messages, such that each volunteer can extract only the digital coin designated as a payment for them. However, malicious volunteers can sabotage the system

by extracting and using their coins without performing their task which consists of forwarding anonymized messages since there is no verification whether the message arrives to its final destination or not. Bandwidth providers in dVPNs share their resources and are granted tokens accordingly as way of payment for their services. This is done using the blockchain technology. A good example of such technologies is Mysterium: an open source dVPN completely built upon a P2P architecture. Mysterium runs a smart contract on top of Ethereum to make sure that the VPN service is paid adequately.

1.1.2   Scalability Layer 2 protocols

Blockchain technology (mostly public blockchains like Bitcoin and Ethereum) suffers from a major scalability issue which is due to the fact that every node in the network needs to process every transaction, validate it and stores a copy of the entire state. The number of transactions Ethereum can process for example cannot exceed that of a single node which is currently 15 transactions per second.

There have been multiple solutions proposed to treat the scalability issue such as sharding and off-chain computation. Both of these solutions intend to create a second layer of computation in order to reduce the load on the blockchain mainnet.
Off chain solutions like Plasma, Truebit and state channels process transactions outside the Blockchain while still guaranteeing a sufficient level of security and finality. State channels are better known as "payment channels". In models like the "Lightning Network", a payment channel is opened between two parties by committing a funding transaction, followed by making any number of transactions that update the channel's funds without broadcasting those to the blockchain, then closing the channel by broadcasting the final version of the settlement transaction.

1.2   The HOPR vision

HOPR is a decentralized incentivized mixnet that leverages privacy by design protocols. HOPR aims to protect people's metadata privacy and give them the freedom to use internet services safely and privately. HOPR runs on top of the Ethereum blockchain and uses three mechanisms to ensure users privacy via incentivisation: sphinx packet format and packet mixing, proof-of-relay and probabilistic payments.

2   Threat Model

Although we assume that nodes in the HOPR network can communicate reliably, the network can still be damaged by malicious actors and node failures. We assume a threat model with byzantine nodes with either the ability to observe all network traffic and launch network attacks or inject, drop or delay

messages.

There are different attack vectors which could threaten the security of HOPR network, in the following section we mention these attacks and the mitigation methods used by HOPR to resist them:

- **Sybil attacks:** An attacker uses a single node to forge multiple identities in the network, thereby bringing network redundancy and reducing system security. This attack is expensive to conduct in practice since the attacker must stake lots of HOPR tokens within each malicious node they create in order to increase their probability of being chosen as a relayer and thus attacking the network.

- **Eclipse attacks:** The attacker seeks to isolate and attack or manipulate a specific user that is part of the network. This is a common attack in peer-to-peer networks since nodes have a hard time identifying malicious ones as they don't have a global view of the whole network. The cost of launching an eclipse attack is high since HOPR nodes are constantly challenging other peers and keeping a reputation score for each node.

- **Camouflage attacks:** A malicious node pretends to be an honest one most of the time. When its reputation value reaches a high level, it occasionally attacks the system. Since the attacker needs a long time to gain enough reputation score and be selected. Based on this, the system can still perform well.

- **Observe-Act Attack:** The attacker observes the reputation score distribution of honest nodes, then control malicious nodes to act and have the same reputation score in order to increase the probability that most malicious nodes are chosen as relayers. This attack however reveals the identities of malicious nodes which conduct this attack and their reputation will be reduced if not loose their stake.

Security Goals

In addition to resisting the previous attacks, the HOPR protocol has been defined to meet these security goals which are inherited from the Sphinx packet format:

- Sender-receiver unlinkability: The inability of the adversary to distinguish whether $\{S_1 \rightarrow R_1, S_2 \rightarrow R_2\}$ or $\{S_1 \rightarrow R_2, S_2 \rightarrow R_1\}$ for any concurrently online honest senders S_1, S_2 and honest receivers R_1, R_2 of the adversary's choice.

- Resistance to active attacks: Resistance to active attacks like tagging and replay attacks where the adversary modifies and re-injects messages to extract information about their destinations or content.

3 Sphinx Packet Format

HOPR uses the SPHINX packet format to encapsulate and route data packets in order to provide privacy features such as sender and receiver unlinkability. Sphinx is a cryptographic message format used to relay anonymized messages within a mix network. A sphinx packet consists of two parts:

1. Header:

   - Key derivation
   - Routing information
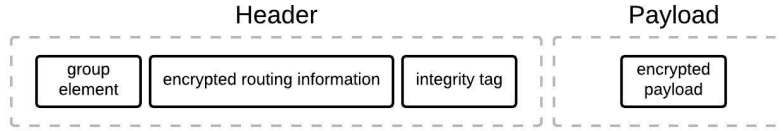   - Integrity protection

2. Body:

   - Onion-Encrypted payload

Figure 2: Sphinx packet format

Key derivation The sender (A) picks a random $x \in \mathbb{Z}_q^*$ that is used to derive new keys for every packet.

(A) randomly picks a path consisting of intermediate nodes (B), (C),(D) [see section path-finding] and the final destination of the packet (Z)

(A) performs an offline Diffie-Hellman key exchange with each of these nodes and derives shared keys with each of these nodes.

(A) computes a sequence of r tuples (in our case r=4)

$$(a_0, s_0, b_0), \ldots\ldots\ldots\ldots, (a_{r-1}, s_{r-1}, b_{r-1})$$

as follows:

- $a_0 = g^x, s_0 = y_B^x, b_0 = h(a_0, s_0)$

- $a_1 = g^{xb_0}, s_1 = y_C^{xb_0}, b_1 = h(a_1, s_1)$

- $a_2 = g^{xb_0b_1}, s_2 = y_D^{xb_0b_1}, b_2 = h(a_2, s_2)$

- $a_3 = g^{xb_0b_1b_2}, s_3 = y_D^{xb_0b_1b_2}, b_3 = h(a_3, s_3)$

Where y_B, y_C, y_D, y_Z are the public keys of the nodes B, C, D which we assume to be available to A . The a_i are the group elements which, when combined with the nodes' public keys, allows computing a shared key for each via Diffie-Hellman (DH) key exchange, and so the first node in the user-chosen route can forward the packet to the next, and only that mix-node can decrypt it. The element g is a generator of G which is a prime order cyclic group satisfying the Decisional DH Assumption. The used cyclic group in the HOPR Sphinx implementation is an elliptic curve group on the Ethereum secp256k1 curve and thus operations will be done on the elliptic curve. The s_i are the DH shared secrets, b_i are the blinding factors and h is a hash function which is used to compute blinding factors. We use BLAKE2s hash function in HOPR which is a cryptographic hash function faster than SHA-2 and SHA-3, yet is at least as secure as SHA-3 and produces digests of any size between 1 and 32 bytes.

Routing information   Each node on the path needs to know the next downstream node. Therefore, the sender (A) generates routing information β_i for (B), (C) and (D) as well as message END to tell (Z) that it is the final receiver of the message.
As (A) has a shared secret with each of the nodes along the path, it is able to derive blindings for each of them.
Once (B) receives the packet, it derives the shared key s_0 by computing

$$s_0 = (a_0)^b = (g^x)^b = (g^b)^x = y_B^x$$

and removes its blindings. This allows (B) to unblind the routing info that tells (B) the public key of the next downstream node (C) and deletes the routing information from the header. Afterwards, it fills the empty space with its own blinding.
(B) also removes one layer of encryption from the payload. Same happens at (C) and (D): key derivation, unblinding, deleting, shifting, decryption and blinding. In addition to all these steps, the final mix-header to the destination node (Z) must include a final destination address that symbolizes the end of the path and tells that it's the recipient and an Identifier used to generate reply messages back to the sender.

Some padding is added at each mix stage, in order to keep the length of the message invariant at each hop. In the original Sphinx paper, a zero-byte padding is added which could reveal information about the path length and the final destination and thus breaking the SPHINX security properties. As a countermeasure in HOPR, We pad the routing information with random data.

Integrity   Each node along the path receives an authentication tag γ_i in the form of a message authentication code (MAC) which is encoded in the header, we use a MAC based on a hash function BLAKE2s.
By using the derived shared secret s_i, each node is able to recompute the au-

thentication tag and check the integrity of the received packet as follows:

$$\gamma_i = HMAC(s_i, \beta_i)$$

This integrity check allows to verify whether or not the header was modified.

Encrypt & Decrypt The payload is where the actual message is hidden and is computed in different layers using a block cipher encryption algorithm and is decrypted at each stage of mixing. LIONESS implementation based on the ChaCha20 stream cipher and the BLAKE2 hash algorithm is being used for encryption and decryption of the payload in HOPR's implementation of the SPHINX packet format. Any attempt of modification by an attacker would result in message distruction and thus is would be irrecoverable.

4 HOPR incentivization mechanism

HOPR incentivizes nodes in order to achieve correct transformation and delivery of mixnet packets. This is accomplished using a mechanism called "Proof-Of-Relay" with the following layer 2 solutions which are both cost effective and privacy preserving.

4.1 Probabilistic payments

In traditional payment channels, two parties A and B lock some funds within a smart contract, make multiple transactions off-chain and only commit the aggregation on-chain.

HOPR uses a concept called *acknowledgements* which allows every node to creates a message that acknowledges the processing of the packet to the previous node. This acknowledgement contains the cryptographic material to unlock the payout for the previous node. Note that the acknowledgement is always sent to the previous node - even if there was no payment.
The fact that we are using payment channels implies that the last HOPR acknowledgement contains all previous incentives plus the incentive for the most recent interaction

$$value_(ACK_n) = \sum_{i=1}^{n} fee_{packet_i}$$

where n is the total number of mixnet packets transformed.

If B received ACK_n before sending $packet_{n-1}$, it has no incentive to process $packet_{n-1}$ rather than $packet_{n-2}$.

To avoid this limitation of traditional payment channels, HOPR utilizes probabilistic payments

In probabilistic payments, the payouts use a concept called "tickets", a ticket can be either a win or a loss with a certain winning probability. This means nodes are incentivized to continue relaying packets as they don't know which ticket is a win.

HOPR uses a custom-made layer 2 solution. It is inspired by payment channels and probabilistic payments where incentives can be claimed independently:

$$value(ACK_i) = value(ACK_j) \quad for \quad i, j \in \{1, n\}$$

Hence, there is no added value in pretending packet loss or intentionally changing the order in which packets are processed.

4.2 Proof Of Relay

HOPR incentivizes packet transformation and delivery using a mechanism called "Proof-Of-Relay". This mechanism makes sure nodes relay services are verifiable.

Construction

- Every packet is sent together with a ticket.

- Each ticket contains a challenge.

- The validity of a ticket can only be checked on reception of the packet but the on-chain logic enforces a solution to the challenge stated in the ticket.

Since "Proof-Of-Relay" is used to make the relay services of nodes verifiable, it is the duty of each node to check that given challenges are derivable from the given and the expected information. Packets with inappropriate challenges should be dropped as they might not lead to winning tickets. Therefore, the sender of the packet also provides a hint of the expected value that a node is supposed to get from the next downstream node (as explained in the ticket section).

5 Tickets

In the HOPR protocol, nodes that have staked funds within a payment channel can issue tickets that are used for payment to other nodes. Tickets are used for probabilistic payments; every ticket is bound to a specific payment channel and cannot be spent elsewhere. They are redeemable at most once and they lose their value when the channel is closed or when the commitment is reset. A commitment is secret on-chain value that is used to verify whether a ticket is a win or not when it's revealed in order to redeem it.

5.1 Ticket issuance

A ticket can be issued once two nodes have established a payment channel with each other which means that at least one of them has locked HOPR tokens.
The ticket issuer A (A could also be the packet creator) selects the winning probability of the ticket and the relay fee to use and sets amount to:

$$\delta := \frac{L \times F}{P_w}$$

where δ is the amount of HOPR tokens set in the ticket, L is the path length, F is the relay fee and P_w is the winning probability.

The issuer (A) issues a ticket for the next downstream node, the challenge is given together with the routing information by the packet.
(A) does not know whether the ticket is a win or not.
(A) sets content of a ticket to:

$$t = (R, C, \alpha_c, \delta, P_w, \zeta, I)$$

where R is the recipient, C is the challenge, α_c is the account.counter, ζ is the channel.iteration and I is the index.
(A) then signs the ticket with its private key and sends $T := (t, Sig_I(t))$ to the recipient together with a mixnet packet.
The data for a ticket is signed by the issuer, and a ticket is the data followed by the signature:

$$T := (T_D, Sig_I(T_D))$$

where
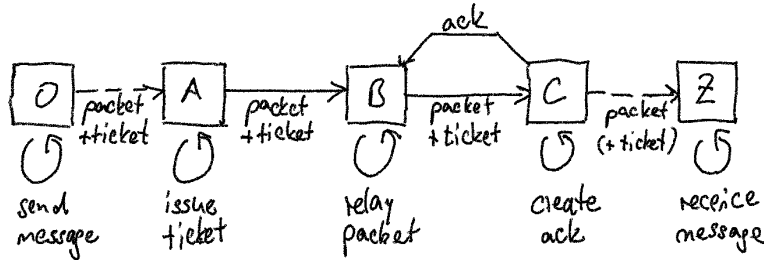
$$T_D := (\delta, P_w, R, I, C, \zeta, c_{Id}, tag, V)$$

Figure 3: Ticket workflow

5.1.1 Challenge

(A) creates a shared secret s_i with all the relay nodes in the channel (B-C-D-Z) by using an offline version of the Diffie-Hellman key exchange.
The shared secret s_i is used as a seed for a PRG (Pseudo Random Generator)

to create secret shares s_i, s'_i for each node along the route. Relayers compute s_i and get s'_{i+1} from the next downstream node.

The sender (A) creates $challenge_i := (s_i + s'_{i+1}) * G$ and a hint for B,C,D,Z (let's suppose these are the relay nodes and Z will be the final destination) a "hint" how the promised value s'_C, s'_D, s'_Z is going to look like. The value "hint" is computed as $hint_i := s'_{i+1} * G$

5.2 Ticket validation

Tickets are received together with packets which means that the recipient and the next downstream node share a secret s whose key shares s_i and s_{i+1} are derivable by those nodes.

Once (A) receives $s_{i+1}^{(1)}$ from (B) by the secret sharing, it can compute $response := s_i^{(0)} + s_{i+1}^{(1)}$ such that it verifies

$$response * G = ticket.challenge$$

Once the recipient transforms the packet, it is able to compute s_i. The recipient is now also able to extract the routing information from the packet. This includes a hint to the value s_{i+1} given as $hint_i := s_{i+1} * G$.

The unacknowledged ticket is stored in the database under the $hint$ to the promised value to make sure that the acknowledgement can be afterwards linked to the unacknowledged ticket.

Together with s_i, the node can verify that

$$ticket.challenge = s_i * G + hint_i$$

with

$$s_i * G + hint = s_i * G + s_{i+1} * G = (s_i + s_{i+1}) * G$$

This allows the recipient to verify that the promised value s_{i+1} indeed leads to a solution of the challenge given in the ticket. If this is not the case, then the node should drop the packet.

Without this check, the sender is able to intentionally create falsy challenges that lead to unredeemable tickets.

5.3 Ticket redemption

In order to unlock the ticket, the node stores it within its database until it receives an acknowledgement containing s_{i+1} from the next downstream node. The challenge can be computed from acknowledgement as $challenge := ack * G$. Once it receives an acknowledgement, it checks whether it stores an unacknowledged ticket for the received acknowledgement. If this is not the case, the node should drop the acknowledgement.

The node then computes the response to the challenge given in the ticket as

$$response := (s_i + s_{i+1}) * G$$

Additionally, the node retrieves the opening value *open* to the current on-chain commitment (used to verify whether a ticket is a win) and checks whether *response*, *open* leads to a winning ticket. This is the case if

$$H(ticketHash, response, open) < P_w$$

where

$$ticketHash := H(T_D)$$

If this is not the case, the node should drop the ticket. The final recipient of the packet does not receive a ticket because message reception is not incentivized by the HOPR protocol.

The node checks whether the information gained from the packet transformation is sufficient to fulfill the given challenge sent along with the ticket. It then replies with an acknowledgement that includes a response to the challenge.

The node also checks the following:

- The signature of the ticket issuer is valid:

$$ecrecover(Sig_{response}(0)) = address(T_D.challenge)$$

- The ticket has been already spent (replay protection):

$$channel.index < T_D.index$$

- Channel exists and is open.

- Channel balance towards the ticket recipient is sufficient to cover the costs for the ticket.

- Ticket index is strictly greater than the current value in the smart contract (reorder protection). This is valid because redeeming a *ticket* with $index = n$ requires $index < n + 1$ and sets $index := n + 1$. So the new index becomes greater than n.

6 Tickets aggregation

The word aggregation means the process of combining things or amounts into a single group. HOPR adds an additional scaling layer that aggregates multiple tickets and redeems them within one transaction.

Since probabilistic payments cannot scale arbitrarily and ethereum gas fees are only increasing it makes sense to aggregate multiple tickets and redeem them within one transaction. The preimage of a winning tickets stored in the node's database.

The node sends ticket or several winning tickets $ticket_A$ and $ticket_B$ with the

corresponding pre-images to the issuer who computes their aggregation $ticket_C$.

The issuer stores on chain $hash(ticket)$ of the redeemed tickets in a space-efficient data storage called Bloom Filters. The issuer computes the modified Bloom Filter by XORing the previous version with the one after inserting the ticket.

Once $ticket_C$ is sent to the smart contract, the intended version of the modified Bloom Filters is recovered by $bloom_{n+1} = bloom_n \lor diff$ and thereby invalidates $ticket_A$, $ticket_B$ as well as $ticket_C$.
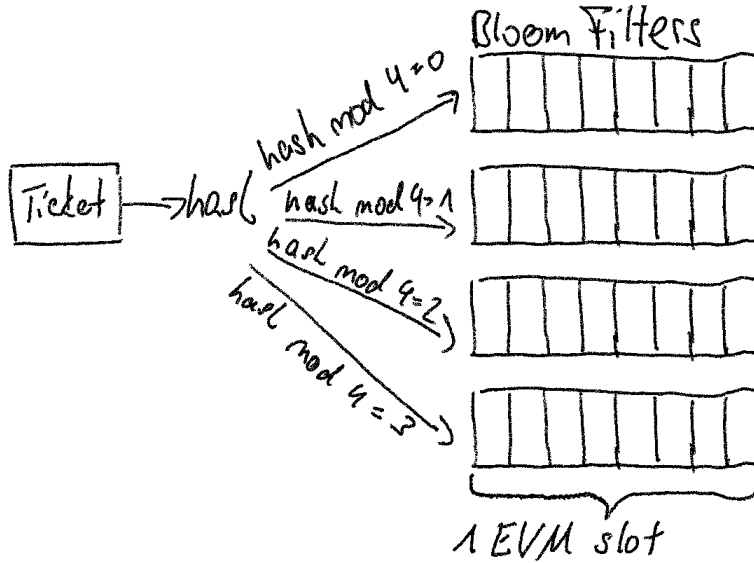

Figure 4: Ticket aggregation

7 Path Selection

HOPR uses a random selection algorithm to determine the identities of nodes participating in the network relaying service.

For each round, a subset k-of-n nodes will be chosen where n is the total number of nodes in the HOPR network. The selection is divided into two steps:

1. Pre-selection: During this phase, a subset $m << n$ nodes will be selected based on different factors:

    - Availability
    - Payment channel graph
    - Reputation
    - Stake

Each node gets a score that is proportional to the previously mentioned factors.

2. Random selection: Each edge (from node a to b) within the subset m is assigned a random number r_i. Edges are then sorted by $r_i * score(edge_i)$

Once an edge is selected, it is added to the current path. All paths are then sorted according to their weight and the path with the highest score is expanded next.

7.1   Availability

Availability is estimated using the heartbeat protocol. Each node maintains a list of neighbor nodes in the network and either ping or passively listen to them in order to determine whether they are online or offline. A node is considered online if the ping response ("PONG") comes back within a certain timeframe. Otherwise, the node is considered offline and its waiting time for the next PING attempt is doubled.

7.2   Payment channel graph

Every node that intends to send messages needs to have a basic understanding about the topology of the network which means whether the channel is open and funded with enough HOPR tokens for the relaying service. In case no existing payment channel is open, the sender creates a new channel and funds it with enough HOPR tokens

7.3   Reputation

The node's reputation either increases or decreases its probability of being chosen depending on its behavior. We assume the majority of nodes are honest but there are some who misbehave to attack the network. For that reason, HOPR plans to implement a reputation system (check future work for more information.

7.4   Stake

The HOPR tokens are used to create payment channels with other nodes in the network and thereby staked in the HOPR network. The node will then use the HOPR token to cover transaction costs when interacting with the blockchain. The more stake a node locks the higher probability that it would be chosen as a relayer.

8   On-chain Commitment

Definition

A commitment scheme is a protocol between usually two parties A and B and fulfills two properties (hiding and biding).

Hiding: The ability to commit a value only known by the sender.
Binding: The committed value must be the only one that the sender can compute and that validates during the reveal phase.

HOPR uses on-chain commitments to verify whether a ticket is a win or not in order to redeem it later on. The relayer doesn't know beforehand whether the ticket is a win or not until they receive an acknowledgment from the next downstream node eand can't change the outcome.

8.1   Setup phase

Once a node joins the HOPR network, it creates an iterated commitment and stores the opening key in the database. Iterated commitment scheme means $open_n$ opens cm_n whereas $open_{n-1}$ opens $cm_{n-1} = open_n$ and so on.
The final commitment is computed as $cm_n := hash^n(r)$ where hash is a preimage-resistant hash function and r is chosen uniformly at random by the node in order to prevent the issuer from knowing whether the ticket will be a win or not.

Furthermore, it will prevent it from tweaking the given challenge such that the ticket cannot be a win. The number of iterations n can be chosen as a constant and should reflect the number of tickets a node intends to redeem.

8.2   Opening phase

In order to redeem a ticket, a node has to reveal the opening to the current commitment cm_n that is stored in the smart contract.
The opening shouldn't be revealed otherwise since it discloses whether a ticket is a win or not. The ticket is a win if:

$$H(ticketHash, response, open) < P_w$$

where

$$ticketHash := H(T_D)$$

The opening is computed as $open_{n-1} = hash^{n-1}(r)$ such that $cm_n = hash(open_{n-1})$.

The on-chain logic verifies whether the latest opening $open_n$ indeed opens the current on-chain commitment cm_n. If this is the case, the current on-chain commitment is replaced by the given opening.
The on-chain keeps track of updates to the on-chain commitments to prevent

double-spending. So whenever a node resets a commitment, a counter, namely *account.counter* is increased and a call to *updateCommitment* invalidates all previously unredeemed tickets.

9   Event propagation

HOPR uses a decentralized event propagation method (blockchain indexer) to allow information to be aggregated at many points in the network and shared with other nodes. This method prevents single point of failure vulnerabilities that service providers like infura or Alchemy could create. It also allows HOPR end users to use the HOPR network without needing any additional computational and bandwidth resources.

HOPR decentralized trustless event propagation

HOPR uses Ethereum full nodes to fetch events and forward them to their HOPR instance which improves network latency. HOPR then aggregates the events and store them chronologically:

- $Open_i := (channelId, open, balance, balance_a)$

- $Close_i := (channelId, close)$

The nodes then publish the update independently to the DHT and allow HOPR end users to download it and verify the validity of the data.
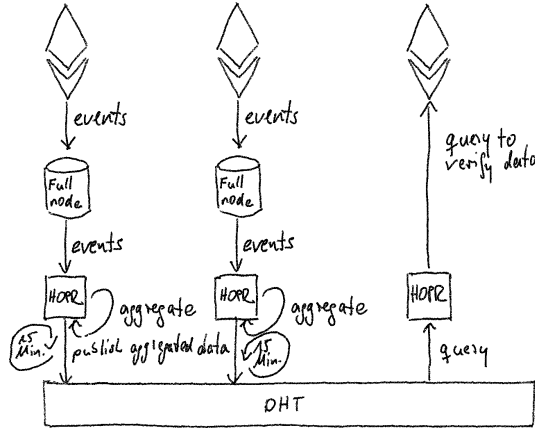

Figure 5: Event propagation

HOPR is a privacy preserving protocol that guarantees sender-receiver unlinkability while still providing high availability.
In order to achieve the aforementioned properties, nodes use DHT to place information in the network and ask selected other nodes to replicate it and update

that list every 5 minutes. Since the indexer retrieves on-chain information, there is no leakage about HOPR nodes' knowledge of the network.

Event logs are set to:

$$log_i := hash(log_{i-1}||Open) \quad or \quad log_i := hash(log_{i-1}||Close)$$

where $log_0 := hash(0)$.

When a node receives log_i from another node, the validity can be checked by recreating the hash and comparing the value with the on-chain value.

10   Conclusion

10.1   Future work

10.1.1   Path position leak

In HOPR, payments are performed hop-by-hop along a packet's route. The incentives break the unlinkability guarantees inherited from the SPHINX packet format as they reveal the identity of the packet origin who transfers those incentives in the channel using their signature.

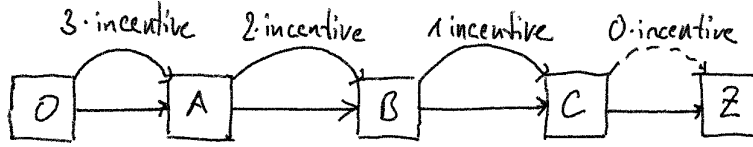To solve this problem, HOPR forward incentives next to the packet.

Figure 6: Incentive flow

This however leaks the relayer's position within the selected path since the value of the ticket is set according to the current relay fee and the number of intermediate hops, more precisely

$$amount := \frac{(hops - 1) * relayFee}{winProb}$$

This leakage is considered to have a low severity but further research will be conducted on the subject.

10.1.2   Reputation (aggregated trust matrix)

In HOPR, we assume the majority of nodes are honest and act properly. Nevertheless, there might be nodes who actively try to attract the network by:

- Dropping packets or acknowledgements
- Sending falsy packets, tickets or acknowledgements

17

Since nodes need to monitor the network to select paths, they need to filter nodes that behave inappropriately. In order to do so, HOPR plans to implement a transitive reputation system which gives a score to each node that acts as a relayer.

The node's reputation either increases or decreases its probability of being chosen depending on its behavior.

Transitive trust evaluation

The reputation can be defined as: "a peer's belief in another peer's capabilities, honesty and reliability based on the other peers recommendations." Trust is represented by a triplet (trust, distrust, uncertainty) where:

- Trust: $td^t(d, e, x, k) = \frac{n}{m}$ where m is the number of all experiences and n are the positive ones

- Distrust: $tdd^t(d, e, x, k) = \frac{l}{m}$ where l stands for the number of the trustor's negative experience.

- Uncertainty = 1 - trust - distrust.

11   References