Hook Payments and Network

Axel Ericsson (axel@evm.com)

November 15, 2020

1 Network

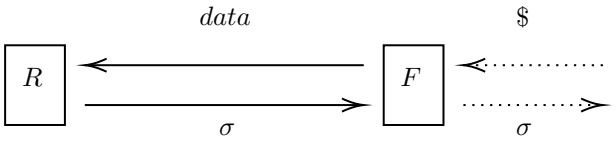
A Hook Payment is a mechanism for transferring data in exchange for payment across a network of non-trusted nodes. Each node is identified by a public key, and data is sent optimistically in the direction of the receiver on the network. The design guarantees that nodes that participate in routing get paid if and only if the data is delivered successfully. Furthermore, nodes do not need to trust each other to transfer data or receive payments, significantly increasing the potential size of the network. Payments can also be prepared ahead of time, so that heavily used paths through the network can be used to transfer data with minimal extra latency compared to a fully trusted network.

First, this paper will demonstrate a simplified design for paying a single node to transfer data in Section 2. In Section 3, the paper will generalize a type of zero-knowledge proof used in the simplified payment design. Lastly, the payment design will be generalized to work when multiple nodes are needed to reach a receiver in Section 4.

2 One-hop Hook Payments

2.1 Setup

A machine F wishes to forward a piece of data to a receiver R. In exchange, F expects a signature σ from R that the data has been received. σ will, in turn, unlock some out-of-band payment to F for successfully transferring the data.



2.2 Problem

If F transfers the data to R first, then R may choose to not produce and transfer σ. If R hands over σ first, then F already has what they need to get paid and may choose to not expend the effort of transferring the data.

A solution may not rely on simply revealing all pertinent information in an on-chain transaction, by forcing some payment to F to coincide with making the data itself or a decryption key to the data public. Rather, all information sharing must occur off-chain. Furthermore, a non-public mutually trusted third party may not be used for exchanging the data and σ.

Gregory Maxwell's work on Zero-Knowledge Contingent Payments describes a mechanism for revealing a file to a receiver if and only if a conditional payment to or from the receiver triggers. A key property of Hook Payments is that the receiver neither pays nor gets paid to receive the file. Instead, only nodes that aid in delivering the file to the receiver get paid. This is modelled after traditional mail, where a receiver never pays for incoming mail.

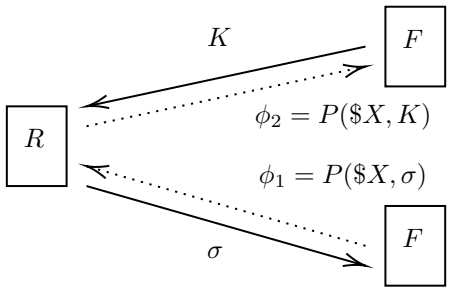
2.3 Details

F also has metadata from the original sender S of the transfer (which may differ from F) that contains the hash H_{data} of the data, a signature σ_S from the sender on the hash of the data, and the public key pk_S of the sender.

2.4 Solution: Hook Payments

Assume that F and R share two lightning-style payment channels that support conditional payments. Now consider the following setup:

1. F conditionally pays \$X to R when R reveals σ. Let this payment be called ϕ_1.
2. F encrypts the data to generate ciphertext C with a newly chosen symmetric key K.
3. F generates a zero-knowledge proof that he knows a symmetric key K that fulfills the following two criteria:
   - The hash of K matches a certain value H_K.
   - K can decrypt C to generate a piece of data that when hashed, matches H_{data}.
4. F transfers to R ciphertext C, the zero-knowledge proof, H_K, and the metadata (H_{data}, σ_S, pk_S).
5. R can now assume that a sender is trying to send them a piece of data whose hash is H_{data} by verifying σ_S. R then verifies that there exists a symmetric key that decrypts ciphertext C such that its hash matches H_{data}, and that the hash of this symmetric key matches H_K.
6. R now creates a conditional payment ϕ_2 using H_K of \$$X$ to F that unlocks when F reveals K.
7. F reveals K to R, claiming \$$X$ in ϕ_2 atomically.
8. R now has K and can obtain the data by decrypting C. R can then receive their \$$X$ back by revealing σ to F in ϕ_1.
9. The data has now been transferred to R, and σ has been given to F.



2.5 Analysis

Steps 1-5 occur only if the parties cooperate i.e if R wants to receive the incoming data. Note that if R is open to receiving incoming data, R has no reason to reject ϕ_1 in Step 1 since this is a conditional payment to R.

Step 5 convinces R that ciphertext C received from F can be decrypted using some symmetric key K whose hash is known by R. The use of a zero-knowledge proof here is critical since R must be convinced that ciphertext C is not just a piece of random data before agreeing to pay anything to F in ϕ_2 during Step 6 to get K.

After Step 7, R has received everything needed to recover the data, but at a cost of \$$X$. Therefore, R is incentivized to claim \$$X$ in ϕ_1 to recoup this cost by revealing σ to F. We can prevent R from being able to produce σ

prematurely and claim $\$X$ in ϕ_1 before ϕ_2 is set up by requiring σ to be a signature on a piece of data which can only be known by R after K has been shared with R. For example, σ can be a signature on $h^*(data)$ where h^* is a cryptographic hash function that differs from the one used to produce H_{data}. Note that the two payment channels used for ϕ_1 and ϕ_2 can simply be swapped after a successful transfer to allow any number of files with incentive $\$X$ to be transferred without ever requiring channel rebalancing.

In sum, the data and σ have not been exchanged atomically, rather, we have created a cost of $\$X$ for R to withhold σ. Assuming that the parties respond as intended to the added incentives, the net cost of the two conditional payments to the two parties is zero, disregarding the opportunity cost of locking up funds in the channels.

3 Zero-knowledge link proofs

Before we generalize the hook payment and signature reveal mechanism to support multiple forwarding parties F, we will first generalize the type of zero knowledge proof used in the One-hop Hook Payment construction. We will term the generalized version of this proof a "Zero-knowledge link proof" or ZKLP.

3.1 Recap

Recall that instead of sending over the data itself, the forwarding node F encrypted the data with a symmetric key K and produced a zero-knowledge proof that the resulting ciphertext C could be decrypted using some symmetric key K with a particular hash such that the hash of the plaintext matched a certain value. This was necessary since receiver R needs to be convinced that they have not received a blob of random data before they will agree to set up a conditional payment of $\$X$ to F in exchange for K. In this situation, F produces the proof and knows variables P (the plaintext, or data), the symmetric key K, and the ciphertext C.

The verifying party R, in turn, knows only C, $h(P)$, and $h(K)$. The zero-knowledge proof convinces R that F knows an input K that makes the following program return true:

```
def test(K):
    if h(K) is not <HASH_K>: return False
    P = decrypt(<C>,K)
    if h(P) is not <HASH_P>: return False
    return True
```

Note: $< ... >$ signifies a constant and h(...) refers to some cryptographic hash function

In essence, this proof links a ciphertext with the hash of its associated plaintext. A ZKLP will be a slight variation on this proof.

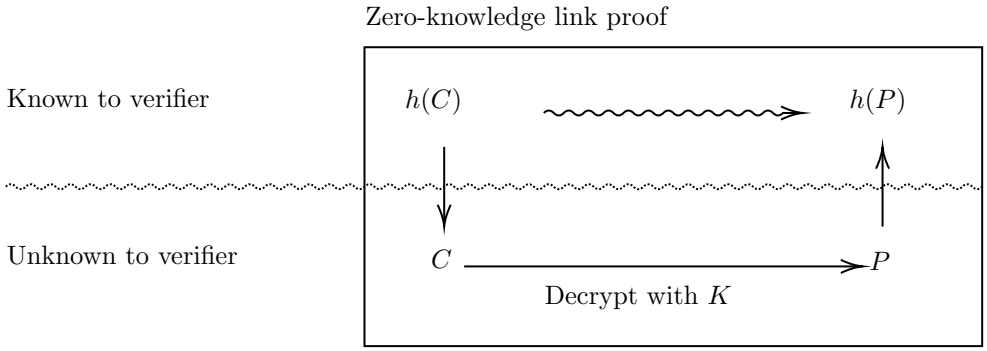
3.2 ZKLP

In a ZKLP, the proving party is in possession of a plaintext P, a symmetric key K, and a ciphertext C where $C = encrypt(P, K)$. However, the verifying party will only be in possession of $h(C)$, $h(P)$, $h(K)$ and the proof itself. Note that the verifying party is not in possession of ciphertext C. The proof will convince the verifying party that the proving party knows inputs C, K that makes the following program return true:

```
def test(C,K):
    if h(K) is not <HASH_K>: return False
```

```
    if h(C) is not <HASH_C>: return False
    P = decrypt(C,K)
    if h(P) is not <HASH_P>: return False
    return True
```

In essence, this proof links the hash of a ciphertext with the hash of its associated plaintext. Only the hashes of the plaintext, the ciphertext, and the symmetric key are known by the verifying party. We will let the notation $\pi(h(C), h(K), h(P))$ denote a ZKLP that convinces a verifying party that $C = encrypt(P, K)$ given some triplet $(h(C), h(K), h(P))$.



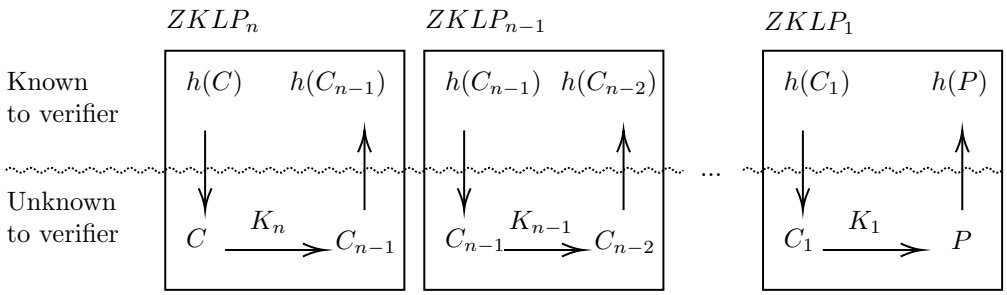
3.3 Recursive encryption

If a plaintext P is encrypted n times using different keys $K_1, K_2, ..., K_n$, to yield a ciphertext C_n where $C_i = encrypt(C_{i-1}, K_{i-1})$, it is possible to use a set of ZKLPs to prove a connection between the hash of the final ciphertext C_n and the hash of the initial plaintext P. This is because verifying proofs $\pi(h(C_i), h(K_{i-1}), h(C_{i-1}))$ and $\pi(h(C_{i-1}), h(K_{i-2}), h(C_{i-2}))$ will prove a link between $h(C_i)$ and $h(C_{i-1})$, and a link between $h(C_{i-1})$ and $h(C_{i-2})$, implying a transitive link between $h(C_i)$ and $h(C_{i-2})$.

We let the notation $\Pi(h(C_n), h(C_1))$ denote the set of ZKLP proofs that proves a transitive link between $h(C_n)$ and $h(C_1)$:

$$\Pi(h(C_n), h(C_1)) = \left\{ \begin{array}{l} \pi(h(C_n), h(K_{n-1}), h(C_{n-1})), \\ \pi(h(C_{n-1}), h(K_{n-2}), h(C_{n-2})), \\ ... \\ \pi(h(C_2), h(K_1), h(C_1)) \end{array} \right\}$$

In order to verify a proof Π, a verifier needs only to verify each constituent proof π. This can be done in parallel since validity of a proof π does not depend on the validity of other proofs. The only requirement is that hashes $(h(C_i), h(K_{i-1}), h(C_{i-1}))\ \forall i$ are known to the verifier.



4 Multi-hop Hook Payments

4.1 Setup

A sender S wishes to send a piece of data to a receiver R. Both S and R are identified by their respective public keys, and S may not know how to contact R directly on a network they are both connected to. However, S is directly connected to a set of forwarding nodes $\mathbf{F}$ which may be connected to R. Therefore, S will attempt to send the data via $\mathbf{F}$, paying the nodes that take part in successfully routing the data to R. S pays nothing if the data doesn't arrive, and R never pays for receiving incoming data.

4.2 Problem

There are a series of problems that a workable design needs to address:

- S does not know the path through the network that reaches R. Therefore, a conditional payment that gets triggered upon successful delivery will need to be built up sequentially.
- A forwarding node masquerading as several should not get paid more.
- It should not be possible to use the network for free or steal money from a sender, receiver, or forwarding node.
- Since R doesn't get paid, a design with similar properties to a One-hop Hook payment will be required.

4.3 Solution

4.3.1 Metadata

S first generates the following metadata: the double hash of the data, a signature on the double hash of the data, and the public key of S $(H^2_{data}, \sigma_S, pk_S)$.

4.3.2 Routing

Nodes (which can act as senders, receivers, or a forwarding nodes) maintain a map of their peers that maps public keys to multiaddrs. A multiaddr is an network address format that contains the protocol over which a machine may be contacted, in addition to its address. This map is used to forward messages between nodes that are connected. The actual transfers of data between nodes may occur over any network or direct line of communication.

4.3.3 First hop

Assume that each connected pair of nodes share two lightning-style payment channels. When S wishes to send the data to R, he generates two symmetric keys K_0 and K_1. S encrypts the data with K_0 to get ciphertext C_0. S then generates a zero knowledge proof ZK_0 that he knows a symmetric key K_0 that fulfills the following two criteria:

- The hash of K_0 matches a certain value $h(K_0)$.
- K_0 can decrypt the encrypted blob C_0 to generate a piece of data that when double hashed, matches H^2_{data}.

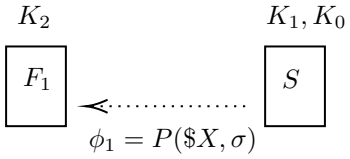
S then encrypts K_0 using K_1 as the symmetric key to generate ciphertext C_1. S generates a zero-knowledge link proof $ZKLP_1 = \pi(h(C_1), h(K_1), h(K_0))$. $ZKLP_1$ will prove that S knows a symmetric key K_1 that fulfills the following two criteria:

- The hash of K_1 matches a certain value $h(K_1)$
- K_1 can decrypt some piece of data whose hash is $h(C_1)$ to generate a piece of data that when hashed, matches the hash of K_0.

S then transmits a packet containing the metadata, ciphertexts C_0 and C_1, hashes $h(K_0)$ and $h(K_1)$, and proofs ZK_0 and $ZKLP_1$ to at least one of their peers. S may use any heuristic for deciding which peer $\in \mathbf{F}$ to send the packet to. Let the forwarding node that receives the packet be called F_1.

Note that S doesn't reveal K_0 or K_1 to F_1 and that (K_0, K_1) is all the additional information F_1 needs from S to recover the data. This holds since K_1 decrypts C_1 to yield K_0, and K_0 decrypts C_0 to yield the data. S also sets up a conditional lightning-style payment ϕ_1 of $\$X$ to F_1 which unlocks with a signature from R on the pre-image to H^2_{data} i.e. $h(data)$. The value of $\$X$ will equal the total amount paid to all forwarding nodes along the path to R. This conditional payment can be set up either before or after the packet is transferred.

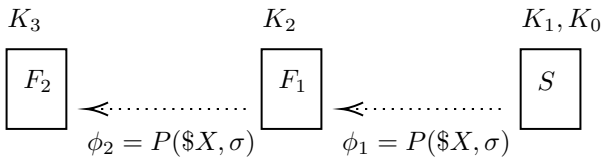
Though not mandatory, F_1 can at this point verify ZK_0 and $ZKLP_1$ using C_0, C_1, $h(K_0)$, and $h(K_1)$ to convince themselves that a symmetric key K_1 with hash $h(K_1)$ is known by S which decrypts C_1 to yield K_0, and that some symmetric key K_0 with hash $h(K_0)$ is known by S which decrypts C_0 to yield the data.



4.3.4 Second hop

At this stage, F_1 will choose their own symmetric key K_2. F_1 will then encrypt C_1 with K_2 to yield C_2. F_1 will then generate their own zero-knowledge link proof $ZKLP_2 = \pi(h(C_2), h(K_2), h(C_1))$. F_1 then transmits a packet containing the metadata, ciphertexts C_0 and C_2, hashes $h(C_1), h(K_0), h(K_1), h(K_2)$, and proofs $ZK_0, \Pi(h(C_2), h(C_1)) = \{ZKLP_2, ZKLP_1\}$ to at least one of their peers. S may use any heuristic for deciding which peers $\in \mathbf{F}$ to send the packet to. Let the forwarding node that receives the packet be called F_2.

Note that F_1 does not know K_0 or K_1, and does not reveal K_2 to F_2. Additionally, further note that (K_0, K_1, K_2) is all the additional information F_2 needs from S and F_1 to recover the data. This holds since K_2 decrypts C_2 to yield C_1, K_1 decrypts C_1 to yield K_0, and K_0 decrypts C_0 to yield the data. F_1 also sets up a conditional lightning-style payment ϕ_2 of $\$X$ to F_2 which unlocks with a signature σ from R on the pre-image to H^2_{data} i.e. $h(data)$. Like before, this conditional payment can be set up either before or after the packet is transferred.



Though not mandatory, F_2 can at this point verify ZK_0, $\Pi(h(C_2), h(C_1))$ along with C_0, C_2, $h(C_1)$ and $h(K_0), h(K_1), h(K_2)$ to convince themselves that C_2 is the result of encrypting K_0 with K_1 and then K_2, and that C_0 decrypted with K_0 yields a piece of data whose double hash has been signed by S.

4.3.5 Until R is reached

As we can see, each forwarding node will choose their own "unlocking" symmetric key to add another layer of encryption to K_0, which is the actual symmetric key which decrypts C_0 to yield the data. Only the most encrypted ciphertext C_i is forwarded to the next forwarding node.

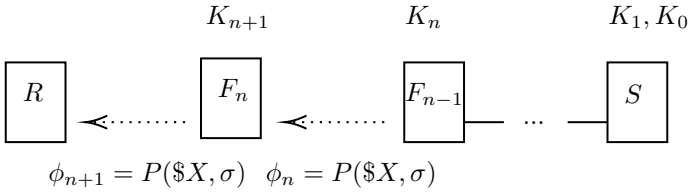
Using only the hashes of the previous ciphertexts $C_1, C_2, C_3, ..., C_i$ and the hashes of the unlocking symmetric keys $K_0, K_1, K_2, ..., K_i$ it is possible for each node to convince themselves by verifying all zero knowledge proofs in

their possession that if they obtained $K_0, K_1, K_2, ...K_i$, they could sequentially decrypt C_i to yield K_0 and therefore also the data. It follows that for each node there is a set of keys $K_0, K_1, K_2, ..., K_i$ that is required to access the data, and the hashes of the required keys are always known to each respective node. Furthermore, the size of this set grows by one key for each hop. A forwarding node may, of course, choose several keys and perform the work of several forwarding nodes, but as we shall see, this does not increase the amount that they can get paid, meaning that time would be wasted on useless extra work.

4.3.6 R is reached

The procedure of encrypting C_i and generating a zero knowledge proof linking it back to earlier keys is repeated until a machine claiming to have access to R's keypair is encountered. If this occurs, the forwarding nodes have been successful at finding a path between S and R in the network. Let the last forwarding node before R is reached be called F_n.

Note that while it is possible for F_n to verify that they're talking to R by simply asking R to sign some message, it is not possible for F_n to verify that they are talking to a node which is not R. Therefore, F_n will communicate with R just like any other node. Thus, F_n transfers a complete packet to R. Similarly, a conditional payment ϕ_{n+1} of $\$X$ will be made from F_n to R which unlocks when R shares with F_n signature σ on the preimage of H^2_{data}.

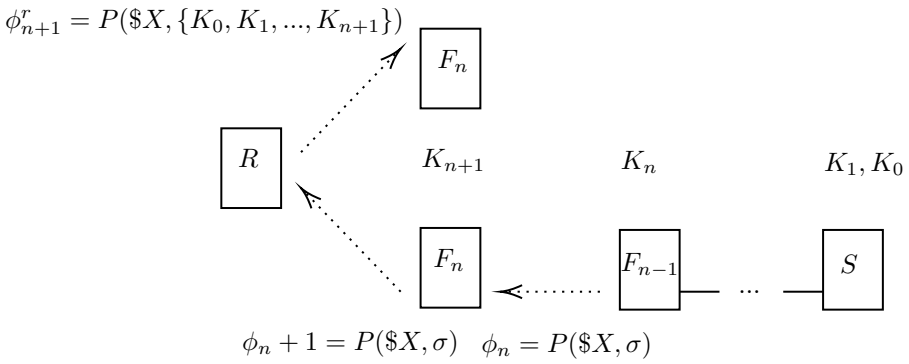


Note that at this stage, R doesn't have access to the data since R doesn't have access to the set of keys $K_0, K_1, K_2, ..., K_n$ required to get the data. Therefore, R cannot simply produce σ to trigger the conditional payment, reaping $X.

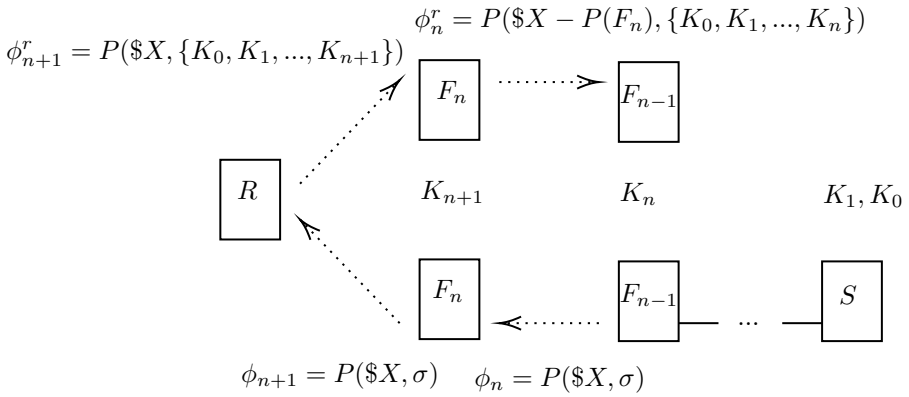
R can now verify $\Pi(h(C_n), h(C_1))$ and ZK_0 along with all hashes, C_0, and C_n to convince themselves that C_0 can be decrypted with knowledge of all keys $K_0, K_1, K_2, ..., K_n$ to yield some data whose double hash has been signed by S. After this verification, R can assume that S is attempting to send a message to R.

4.3.7 Hook Payment

Now, R will continue in setting up a Hook Payment similar in design to a One-hop Hook Payment. Since R knows that it needs $K_0, K_1, K_2, ..., K_{n+1}$ to get the data, and R already knows $h(K_0), h(K_1), h(K_2), ..., h(K_{n+1})$, R will create a conditional payment ϕ^r_{n+1} to F_n of $\$X$ inside their second channel payment that unlocks when preimages $K_0, K_1, K_2, ..., K_{n+1}$ are revealed.



After setting up ϕ^r_{n+1}, F_n will choose an amount $P(F_n)$ that it wants to get paid for successfully forwarding the data from F_{n-1} to R. Since F_n already knows K_{n+1}, they will set up a conditional payment ϕ^r_n to F_{n-1} for $\$X - P(F_n)$ that unlocks when F_{n-1} reveals all the keys that F_n doesn't have, namely $K_0, K_1, K_2, ..., K_n$.



Similarly, F_{n-1} will then choose an amount $P(F_{n-1})$ that they want to get paid for successfully forwarding the data from F_{n-2} to F_n. Since F_{n-1} already knows K_n per definition, they will set up a conditional payment ϕ^r_{n-1} (not pictured) to F_{n-2} of $\$X - P(F_n) - P(F_{n-1})$ that unlocks when F_{n-2} reveals all the keys that F_{n-1} and F_n don't have, namely $K_0, K_1, K_2, ..., K_{n-1}$.

Note that the difference in the amount paid from F_{i+1} to F_i and the amount paid from F_i to F_{i-1} for some i is always $P(F_i)$, i.e. the amount that F_i chooses to get paid. A self-interested node will always choose the largest payment they can such that there are sufficient funds left for the other nodes in the path back to S. Choosing this amount is the responsibility of each node.

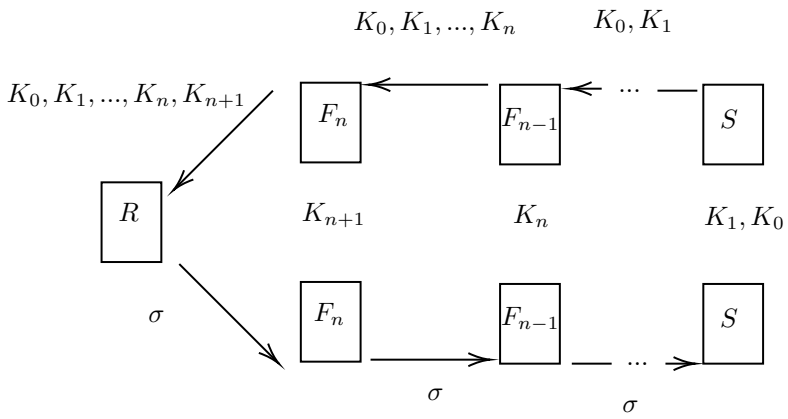
This process of setting up conditional payments of decreasing size continues backwards in the path between S and R until S is reached. Here F_1 will set up a conditional payment ϕ^r_1 of \$0 to S that unlocks when S reveals K_0 and K_1. S will agree even though he is not getting paid as this is the indication that the packet can be successfully delivered at a price of $\$X$ to S, which was the original intention of S.

4.3.8 There and back again

After ϕ^r_1 has been created, S will reveal K_0 and K_1 to F_1, triggering the conditional payment of \$0 to S. At this point, F_1 can reveal K_0, K_1, K_2 to F_2 to immediately get paid $P(F_1)$ by F_2. F_2, now $P(F_1)$ out of pocket will be incentivized to get paid some larger amount of money from F_3 to cover this payment to F_1. F_2 can trigger this payment from F_3 by revealing K_0, K_1, K_2, K_3 to F_3. Note that since these conditional payments are lightning-style payments, revealing the keys will force the payment i.e. a node cannot go offline to halt the payment to a node

earlier in the path after the conditional payment has been created.

This continues until F_n is reached. F_n will be forced to pay $\$X - P(F_n)$ to F_{n-1} upon learning $K_0, K_1, K_2, ..., K_n$. Therefore, F_n will be incentivized to reveal $K_0, K_1, K_2, \ldots, K_n, K_{n+1}$ to R to recoup $\$X$. When this conditional payment is triggered, F_n's net change in balance is $\$X - (\$X - P(F_n)) = P(F_n)$, the amount F_n chose earlier.



At this stage, R has all the keys needed to retrieve the data. However, R has effectively paid $\$X$ in conditional payment ϕ^r_{n+1} to F_n to receive all the decryption keys. Therefore, upon retrieving the data, R will sign its hash to produce σ. σ can be used by R to claim back $\$X$ in conditional payment ϕ_{n+1} with F_n. This is the same incentivized signature reveal as in a One-hop Hook payment. Now, R has retrieved the data and handed σ to F_n. Moreover, R has not changed their net balance in their two payment channels with F_n since $\$X - \$X = 0$. F_n, now $\$X$ out of pocket will recoup this payment by claiming $\$X$ in their payment channel with F_{n-1} by revealing σ to F_{n-1}. This will continue until S is reached again since no node wants to carry the extra cost of $\$X$. Finally, F_1 will claim $\$X$ from S, causing S to have paid for the whole transfer, and causing S to learn σ. S receiving σ indicates that R received the data, was able to decrypt it, and then sign it.

5 Routing heuristics

5.1 Bloom filters

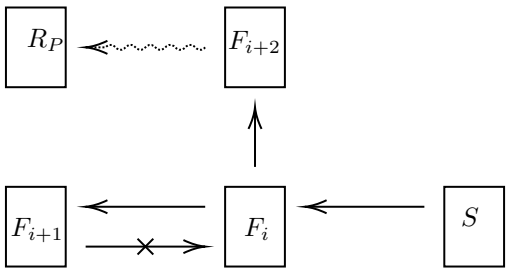
As noted in Section 4.3, a node may use any heuristic for choosing which node to forward a packet to. If all forwarding nodes behave honestly, a node may gain a full view of the network if each node receives a list of reachable peers from each of its peers. Furthermore, list size constraints can be overcome by having each node share a bloom filter containing each reachable peer instead of a list. Since bloom filters are additive, a node will simply share the sum of all the bloom filters of its peers.

However, while there is no direct benefit for a forwarding node to pretend to be several nodes in the Multi-hop Hook Payment design, there is a benefit to receive a packet since this can potentially result in a payment if the receiver is found. Therefore, we cannot expect forwarding nodes to honestly broadcast the bloom-filter containing only the public keys of their reachable peers. Instead, a self-interested forwarding node will simply broadcast a full bloom-filter containing all public keys, in order to get the chance to route every packet.

To mitigate this, we know that the false-positive rate of a bloom-filter is a monotonically increasing function of how full it is (i.e. how many of its bits are activated). If a forwarding node broadcasts a full or nearly full bloom-filter, the high false-positive rate can be taken into account when its peers decide whether to forward a packet to it. However, there is nothing preventing a forwarding node from adding a small number of known popular receivers to

their bloom filter.

Fortunately, this attack is limited in its effectiveness as a receiver still needs to be reachable by a forwarding node in order for the forwarding node to get paid. Let a packet addressed to a popular receiver R_P be sent from F_i to F_{i+1}. If a path already exists between R_P and F_{i+1} that does not go through F_i, then F_{i+1} should always include the public key of R_P in their bloom-filter. If no path exists between R_P and F_{i+1} that does not go through F_i, and F_{i+1} includes the public key of R_P in their bloom filter anyway, then the packet will eventually reach F_i again. Now, F_i can see if they have already attempted to forward the ciphertext in the packet. If this is the case, F_i can simply disregard the incoming packet and forward the original packet to another peer that also claims to be connected to R_P.



6 Optimizations

6.1 Precomputing zero-knowledge proofs

For popular paths through the network, it is possible to significantly reduce latency by computing and verifying ZKLPs before the data is tranferred.

6.1.1 The bottleneck

When a new package is received by a forwarding node, the node must perform one encryption and generate one ZKLP linking the resulting ciphertext C_{i+1} to the received ciphertext C_i. Optionally, a forwarding node may want to verify all previous ZKLPs Π in order to guarantee that they can get paid by forwarding the packet. All of these operations may be performed before the packet is received.

6.1.2 Fast transfers

S wishes to do a low-latency transfer of data to a receiver R. To prepare, S transmits a "probe" packet which is a full packet minus ciphertext C_0. All forwarding nodes can choose their own symmetric keys, generate their own ZKLPs, and verify existing ZKLPs without knowledge of C_0 itself. When the receiver R has received and verified Π, C_0 can be transferred with no additional latency on the path between S and R.

After C_0 has been transferred, the next set of symmetric keys and ZKLPs can be chosen, generated, and shared between the forwarding nodes for paths that are used often, or for paths which require low latency when used. As we will see in Section 6.2, nodes will be heavily incentivized to do this for high-usage and high-paying paths.

6.2 Competitive delivery

In the Multi-hop Hook Payments design, a forwarding node F_i can increase their probability of reaching R by forwarding a packet to several peers. However, this carries additional risk because signature σ from R on the delivered data can be used to unlock all outgoing conditional payments ϕ_{i+1} of $\$X$ from F_i. Therefore, peers of F_i may collude by sharing σ to increase their combined payout. Phrased differently, F_i cannot be sure that all of its peers are not the same machine. To solve this, we must require that conditional payments ϕ_i are dependent on the path of nodes

between S and R. This is non-trivial, since the path is not fully known while conditional payments $\phi_1, \phi_2, ..., \phi_{n+1}$ are being set up.

We can think of the conditional payment ϕ_i as containing a "challenge" which, when solved, transfers $\$X$ to F_i. In the Multi-hop Hook Payment, the challenge is simply to produce σ. To allow F_{i-1} to forward the packet risk-free to several of their peers, including F_i, the challenge given to each of the peers needs to differ so that the solution to one cannot be used to trigger multiple payments. This can be achieved by having F_{i-1} choose different symmetric keys for each of its peer, and letting each conditional payment require σ and the symmetric key chosen for that peer. Since F_{i-1} will only share one of the symmetric keys while claiming ϕ_i^r, F_{i-1} can be guaranteed that only one of the outgoing conditional payments ϕ_i can be triggered.

Now, forwarding nodes are heavily incentivized to forward a packet in a timely manner, as they can assume that several competing nodes also have the packet. The downside is that the work of choosing a symmetric key, producing ciphertext C_i, and generating a ZKLP now needs to be performed for each peer that a forwarding node wishes to forward a packet to. However, with precomputed and preshared symmetric key hashes and ZKLPs, this overhead can be decoupled from the transfer of the data itself.

7 Forwarding node pools

A forwarding node operator may want to operate several forwarding nodes in different locations on the network to maximize the number of peers and potential packets that they can forward. Likewise, forwarding nodes which trust each other may share keys and peer-lists to act as a single forwarding node. Such a pool of forwarding nodes may then share earnings equally or based on some metric of contributions to the throughput of the pool. In this way, the variance of the distribution of node earnings may decrease for each participating node in the pool.

8 Future research

8.1 Proof of delivery

While a sender S should receive σ once a transfer is complete if all forwarding nodes and the receiver react as expected to the incentives in the Multi-hop Hook Payment design, it is not guaranteed that S receives σ. Specifically, any participating forwarding node or the receiver may withhold σ from S at a cost of $\$X$. Furthermore, S cannot be sure that the data has been successfully delivered after sharing keys K_0 and K_1 with F_1 as any participating forwarding node may withhold keys from the rest at a cost $\leq \$X$, halting the transfer. Simultaneously, it would be highly useful for S to receive some sort of proof that the data has been delivered to R, similar to the "proof" that a user of a blockchain can have that an address has received a transfer of funds. It is possible to produce such a proof if successful conditional payments are periodically committed to on-chain. How to do this efficiently while keeping all conditional payments completely off-chain for scalability purposes will remain an area of research.

8.2 Zero-trust forwarding node pools

The trivial implementation of a forwarding node pool as described above requires nodes to trust each other because all valuable data (symmetric keys, peer-lists, etc) are shared between all participants. It would be highly useful to be able to run a zero-trust forwarding node pool that arbitrary and anonymous forwarding nodes may join to contribute throughput to. How to implement this will remain an area of research.

9 Summary

This paper describes a mechanism for transferring data in exchange for payment across a network of non-trusted nodes. Data is transferred optimistically in the direction of the receiver and no forwarding nodes get paid if the data

cannot be successfully delivered. If the data is successfully delivered, each forwarding node is guaranteed to get paid. The system can be optimized so that heavily used paths can transfer data with minimal extra latency compared to a fully trusted network.