AODV–Based Routing for Payment Channel Networks

Philipp Hoenisch and Ingo Weber(✉)

Data61, CSIRO, Sydney, Australia
philipp@hoenisch.at, ingo.weber@data61.csiro.au

Abstract. Payment Channel Networks such as the Lightning Network (LN), Raiden or COMIT were created to tackle the scalability problems of their underlying blockchains, by moving from expensive and slow on-chain transactions to inexpensive and fast off-chain ones. However, those networks are unregulated and decentralised, comprise point-to-point channels that may be opened or closed without coordination or warning, and fees may change at any time – making routing over these networks a hard problem. In addition, by connecting different blockchains using such off-chain networks, an immense network of channels will evolve which is under continues change. Routing needs to take into account the current network status, availability and distributions of channels' funding, fees for each node, and exchange rates between different currencies. In this work, we identify requirements for such a routing protocol and adapt the Ad-hoc On-Demand Distance Vector Routing (AODV) protocol to this end by enhancing the messages with information on fees and exchanges rates. This approach allows finding suitable routes through the network, while intermediate nodes can maintain their economic incentives. We simulate different network topologies and evaluate the adapted AODV protocol on 3 different networks of 500, 1,000 and 5,000 nodes.

1 Introduction

Ever since the first appearance of Bitcoin in 2008 [16] a multitude of blockchain derivatives and other implementations have emerged. The general purpose is to decentralise the management of a particular asset, such as a cryptocurrency, by removing the need of a trusted central entity and to create a network of untrusted nodes. However, common problems of blockchains include slow confirmation and commit times and high transaction fees. For Bitcoin, a transaction is often regarded as committed (irreversible) after 6 confirmation blocks, taking on average 60 min. Due to this commit delay and high transaction fees (recently between USD 0.50 and 50 on Bitcoin), small payments and particularly micropayments (i.e. payments of a few cents or even a fraction of a cent) are not very economical. In addition, many blockchains have a throughput (7 to 20 transactions per second, tps) that is many orders of magnitude below that of payment networks like VISA (up to 47,000 tps).

S. Chen et al. (Eds.): ICBC 2018, LNCS 10974, pp. 107–124, 2018.
https://doi.org/10.1007/978-3-319-94478-4_8

In order to tackle this problem, researchers proposed to not settle every transaction on the chain but rather move some transactions off-chain, allowing two parties to interact with each other directly. By tracking their payments between each other on their own, the two parties are able to avoid expensive and time-consuming interactions with the blockchain. Should there be a dispute regarding the balance or should one party become unresponsive, the most recent balance sheet provided by either of the two parties can be settled on the blockchain (on-chain). The Lightning Network (LN) is the most prominent of these off-chain solutions [20]. It proposes to create an overlay network of off-chain payment channels – i.e. Payment Channel Network (PCN) – where transaction between two parties are not recorded on the Bitcoin blockchain. For that, two parties create a pair of transactions first: a *funding transaction* and a *spending transaction*. The former one specifies the total amount held in the channel, i.e. either one or both of the participants pay an arbitrary amount into this channel. The other transaction specifies the output, i.e. it defines which party receives how much from the total amount. Only the funding transaction is settled directly on the blockchain whereas the output transaction can be delayed to a point of time in the future. If one of the party wants to pay the other party, it updates the output transaction mirroring the actual state. Note, the actual output transaction involves a complex reconciliation between the two parties, its process is described in detail in [20]. Decker et al. in [4] presented an alternative. While the LN was specifically designed for Bitcoin, PCNs can be realised on other blockchains as well if they provide a minimal scripting language that allows realising so called Hash-time Lock Contract (HTLC). Alternatives include the Raiden Network (Ethereum) [22], Sprites (Bitcoin) [12] or COMIT (cross-chain) [8].

Opening a channel only makes sense for recurring payments to (or through) the respective other party. To transact with parties where no direct channel exists, multiple payment channels can be chained together. The payment is then routed through the network with one or more intermediaries. This however gives rise to a major challenge: how to find an optimal (or acceptable) route through the network, i.e., from the sender of a payment to the recipient. The route needs to be acceptable (and ideally optimized) in terms of to specific criteria such as routing fees, exchange rates, and reliability. The routing within the LN applies a proactive routing protocol: each node broadcasts the information about its neighbours (e.g. the nodes which itself is connected to via a channel) through the network. Hence, the downside of this is that a lot of information needs to be send around prior a route can be established. Consequently, each node has complete knowledge about the network topology. So far, only information about the channel funding is included, but not the funding distribution, i.e. how much of the funding is currently on which side and the solution is limited to the Bitcoin blockchain. Obviously broadcasting global topology information is costly and introduces its own scalability limits, which are particularly severe when considering a cross-chain network.

We argue that, without a fully automated solution for payment routing, with localized routing and the ability to adapt to the ever-changing environment of

such a network, PCNs cannot realize their true potential or achieve meaningful coverage on a global scale. Hence, we make the following contributions:

- We present an adaptation of an Ad-hoc On-demand Distance Vector (AODV)-based routing algorithm for a network of off-chain payment channels.
- Our approach can cater for different currencies, hence allowing to route payments across multiple blockchains.
- We evaluate the applicability of our routing protocol experimentally and discuss advantages and disadvantages.

In the next section, we discuss related work in the field of (payment) routing. Afterwards we formulate the requirements of the routing protocol in Sect. 3 followed by the protocol in Sect. 4 and its evaluation in Sect. 5. We discuss the results in Sect. 6 and conclude in Sect. 7.

2 Related Work

Routing can be described as the technique of "[...] sending a unit of information from point A to point B by determining a path through the network, and by doing so efficiently and quickly [...]" [11, Chap. 1, p. 3]. This topic has received a lot of attention in research and industry resulting in various different routing algorithms as routing is fundamental to power almost any small to large network efficiently. Examples range from Circular Switched Telephone Networks (PSTN) over Mobile Ad Hoc Network (MANET) to packet routing on the Internet. However, routing on payment channels has hardly been examined. Hence, we expand our search into the other fields. Especial the area of MANET networks is relevant, as these show similar characteristics as PCNs: Nodes may appear and disappear irregularly, be offline for a while (e.g. as do connections do in MANET) or the channel balances might change frequently. Routing protocols can be classified into five major types: reactive, proactive, hybrid (i.e. a combination out of both: reactive and proactive), hierarchical and coordinate-based.

Reactive protocols perform route discovery on-demand. They do so by *flooding* the network with route discovery requests. Two famous examples are Ad-hoc On-demand Distance Vector (AODV), Dynamic Source Routing (DSR) [9,19]. They work best in a highly dynamic environment (in cases where the network topology changes quickly). In contrast to that proactive routing protocols work well in static scenarios (whenever an update occurred, information is spread). In most cases, each single node maintains a routing table and can decide on the route on its own. Examples of such routing protocols are Destination-Sequenced Distance Vector routing (DSDV) and Wireless Routing Protocol (WRP) [15]. The offsite of re- and pro-active protocols is that excessive flooding can lead to network clogging. Hence, a combination of both achieves a better performance across a wide range of scenarios. Hybrid routing protocols such as Zone Routing Protocol (ZRP) or Enhanced Interior Gateway Routing Protocol (EIGRP) in which each node maintains a routing table on the routes inside its zone, for destinations outside the zones, a route discovery procedure is employed [1,7]. In

contrast to these kind of algorithms, hierarchical and coordinated-based protocols rather use location-based algorithms than flooding the network with messages. Two examples of hierarchical routing protocols are LANMAR [18], L+[14] and two coordination-based protocols are GPSR [10] and BVR [5]. These kinds of algorithms use location-dependent addresses to route information.

These routing protocols were mostly designed for MANET, however, PCN may differ, e.g., additional hops may increase the overall expanses, a *route response* changes the networks state as funds will need to be locked or a found route might be only usable up to a certain amount of times as the involved channels might run out of funding. Nevertheless, routing in off-chain channel networks can benefit from ideas of MANET routing protocols.

An example for hybrid routing is the protocol Flare which is meant to replace the current DSR-based routing protocol employed in the LN [21]. Nodes proactively gather information about the network topology from neighbour nodes (as in DSR) and from beacons which are close (in the sense of Bitcoin addresses). Hence, a sender can decide on the route and issue the payment. Each node broadcasts in a regular interval (or when a change occurred) its local information to their neighbours. If a node receives an update message it first updates its local routing table and enhances the information in the received message with its local information and forwards it to its neighbours. Messages can be encrypted using onion routing [23]. The Ripple Network integrates a path-finding algorithm called *ripple paths* [24]. Payments can be *rippled* through several nodes. This involves moving debt around. Cross currency payments are possible through so called *order books*. However, how exactly a path is found is not clearly defined.

Flare is closest to our work. However, the fundamental difference to our assumptions is that while Flare focuses primarily on security and censorship resistance for the sending node, we focus on the autonomy of each single intermediate node. Each node should not be forced to forward a payment into a certain direction. This decision is driven from the economical point of view, as we assume, nodes primarily focus on profit maximisation and hence are more likely to select routes which ensure high profit. Notably, we covered mostly abstract algorithm of each category within this section. There are various adaptations focusing on more concrete problems, e.g. an improved AODV protocol against 'black hole' attacks [13], reducing the message overhead of AODV using availability prediction [2], adaptive multipath source routing for DSR [28] or an anonymous DSR protocol or AODV routing [26,30]. We argue that if the most abstract protocol is suitable, the improved version might lead to even better results.

3 Requirements and Algorithm Selection

In order to find a suitable routing algorithm for transferring values (in the form of cryptocurrencies) from a sender to a receiver via one or more intermediate nodes in an inexpensive and reliable way we define the requirements for the routing protocol similar to the ones in Flare in [21]. These requirements were formed from community beliefs [17]. Hence, we derived the following requirements which are imposed directly in the LN:

1. **Autonomy and self-reliance:** In order to provide high availability and a failure resistant network, the nodes need to be self-configurable: each node should be able to act autonomously and independently. Hence, a node should be able to act as a sender or recipient at the same time and should be able to route payment requests in any direction. In addition, the functionality of the network must be preserved despite of random changes in the network's topology or due Byzantine behaviour of some nodes.
2. **Cost guaranties:** Each node in this network may charge a certain fee to forward a payment. In addition, when crossing different blockchains, the bridging node will ask for a specific exchange rate between two currencies. Hence, it is essential that the overall cost to issue a payment across the network from a sender to the final recipient is known prior to its execution. This is required, as the sender wants to ensure that the desired amount arrives at the final recipient and is not eaten up by fees or exchange rates.
3. **Time-lock guaranties:** It is required that each payment (channel update) is assigned with a time-lock (compare HTLC [20]). This serves two purposes, first, the receiver needs enough time to redeem the payment, and second, the sender needs to have enough time to rollback in case a failure occurred.
4. **Flexibility:** The routing protocol needs to be flexible enough to take frequent changes into account. Changes in a huge network are likely to happen in various aspects: channels may appear or disappear, the channel's funding distribution may change, nodes may update their fees or nodes between blockchains may change the rates in order to not lose money.
5. **Prevent network partitioning:** A single (or several) failing nodes should not forestall payment routing or split the network in sub-networks. A node *should* always be able to find a route to a desired opponent, i.e. no other node should be able to prevent a payment. In other words: if a route exists between two nodes, the routing protocol should be able to find it.
6. **Real-time:** A major goal of PCNs is to enable instant micropayments. Hence, it is natural that the routing protocol needs to be very fast. Hence, network traffic delays are the only timely constraints which are allowed, i.e. the routing should take less than a few seconds.
7. **Up-to-dateness:** Having up to date information available is crucial for finding the best route through the network. Hence, a requirement is that a found route contains up to date information about fees and exchange rates.
8. **Lightweight and scalable:** It is expected that the off-chain channel network will grow over time. Hence, routing should be able to adapt and scale with it. In addition, routing should only use a moderate amount of resources.
9. **Trustlessness:** Routing should withstand when nodes show Byzantine behaviour (i.e. are lying about fees or routes).
10. **Optimal solution for each node:** The routing protocol should allow each node to act within its own economic incentives. These will differ from node to node, e.g. a node issuing a payment may want to send the payment along the cheapest, fastest or the path with the highest success rate. Contrary, intermediate or forwarding nodes are driven by different incentives. For example, some may provide the cheapest route or aim for high reliability.

Hence, intermediate nodes may not forward routes if this compromises their reputation.

Algorithm Selection – AODV. We have chosen the above mentioned 10 requirements in order to have a reliable and usable network. It should be noted that especially the last point is from highest priority as we want to guarantee each participant's economic incentives. The point lead to the routing protocol selection of AODV. AODV is a routing protocol originally designed for MANET and other wireless ad hoc networks. Key features are the ability to quickly adapt to dynamic condition changes, a low processing and memory overhead and loop freedom at all times [19]. As the name implies, a route is established only on demand on a hop-by-hop basis. Each node should only have one option to send the message through the network (notable, a sending node which is connected to multiple neighbours might end up with multiple routes) as each intermediate node can decide how to forward the message to the final receiver.

One of our main assumption about the network is that it is under continue change. Even more, we assume that network changes are more likely to happen than nodes are sending payments around. Examples for frequent changes are (I) Channel balances change frequently, i.e. a route which was valid before a payment must not necessarily be valid after that payment. (II) Fees may change frequently. Each intermediate node can charge an arbitrary amount of fee when forwarding a payment. This node may regularly update that fee in order to keep its channels balanced. For example, if one channel is at risk of running out of liquidity, the node could charge a higher fee when forwarding payments through that channel. (III) Exchange rates may change quickly. Nodes between two blockchains will need to adapt the exchange rate frequently. Otherwise this node may be at risk of losing money. (IV) Nodes may be offline (or not reachable) for some time. While it is fundamental that nodes are online at all time during a payment as this node needs to accept and forward payment request, if an intermediate node is offline (even for a short amount of time), the network should adapt accordingly.

An alternative would have been a proactive routing algorithm similar to Flare which is based on source routing [21]. In contrast to AODV, source routing can be easily combined with Onion Routing and brings a higher level of security to the network participants. It allows to encrypt the payment message in different layers so that no intermediate node in between is aware of what the message's final destination will be or who the original sender was. However, the downside of this is that a sending node can indirectly attack an arbitrary node in the middle, e.g. by draining its liquidity (locking it up) for some time so that it cannot forward any more payments. Even worse, if the message is encrypted, the attacked node will not even know who it gets attacked from in the first place. A more detailed discussion of this attack can be found in Sect. 6. The other main difference between a reactive and a proactive approach is that reactive routing eliminates the need of periodically flooding the network with table update messages. However, the disadvantage of this approach is that a route request messages may flood the network, i.e. a large amount of messages may be sent

around until a suitable route has been found. We have decided for a reactive routing algorithm as it has the abilities of obtaining an up to date route, being loop free and a quick adaptation to the ever-changing network conditions. In addition, as each node can decide on its own how and where to route a payment to, it is easier to maintain its economic incentives.

4 AODV-Based Routing in PCN

System Model. Before defining to the routing protocol in pseudo code in Sects. 4.1 and 4.2 we define our PCN as an undirected graph $G = (N, C)$ whereas N is the set of all nodes and C is the set of channels between the nodes $C \subseteq \{(n_1, n_2) \mid n_1, n_2 \in N\}$. Each node n represents an independent party who wants to participate in the network either by playing an active role as payer or payee, or passively by earning money as an intermediate node which forwards payments. Each node is assigned with a globally unique id. Further, for $\forall c \in C$ we define $bal(n, c)$ as a the current balance of node n in channel c as a binary function $Bal : N \times C \rightarrow [0, +\infty)$. If a node n is not part of the channel the balance function is not defined, i.e. $\forall n \in N,\ c = (n_1, n_2) \in C,\ n \notin \{n_1, n_2\} \iff \uparrow bal(n, c)$ (read as $bal(n, c)$ is not defined). When forwarding a payment over a channel c an intermediate node n may charge a fee $f \in (-\infty, +\infty)$ and provide a rate $r \in [0, +\infty)$. This means, the cost to send an amount x via a node n_i is $p_i = r_i \times x + f_i$, i.e. the sending node has to pay p_i so that x arrives at the next node while the intermediate node will deduct f_i for itself. Hence, the total cost p (for payment) to send a payment from a starting node n_1 over the intermediate nodes n_2 and n_3 to node n_4 can be expressed as $p = p_3(p_2(x))$ or $p = r_3 \times (r_2 \times x + f_2) + f_3$. Notably, if an intermediate node is in between two blockchains, the arriving amount x is in a different currency. Note that nodes are deliberately enabled to offer a negative fee for forwarding a request. By providing this incentive, i.e. making routes through them cheaper compared to potential other available routes, the forwarding node is able to rebalance its channels in a specific way.

The payment process consists of two phases: first, the *route discovery* phase (Sect. 4.1) and second, the *route selection* phase (Sect. 4.2).

4.1 Route Discovery

As designed in the RFC of AODV a route discovery is only issued when a node decides to send a payment over the network: a route request is broadcast to each connected node, i.e. nodes which are connected via a payment channel: $REQ = <n_o, n_d, n_l, nr_d, nr_o, d_{req}, d_{route}, \#_{max}, \#_{rate}, \#_{fee}, \#_{hops}>$. The route request is defined as REQ whereas n_o defines the route request originator. This is needed so that each node can update its local routing table with a path towards n_o. n_d is the desired recipient and n_l is the last hop, i.e. the node this request was send from. nr_d and nr_o define a sequence number. The former one is the latest sequence number received in the past by the originator n_o for any route towards

the destination n_d. The latter one is the sequence number to be used in the route towards the originator n_o of the route request. d_{req} defines the lifetime of the request, i.e. how long this request is valid. These fields can be found in the original AODV as well. $\#_{hops}$ defines the amount of intermediate nodes (or hops) to the request originator n_o. However, in order to have a more sophisticated algorithm we enhanced the REQ with additional fields. Whenever a node sends out a route request, we use this chance to establish a route backwards (i.e. towards the originator) on each node which receives the request. Hence, we added 3 more fields: d_{route} defines how long the route towards n_o is valid, $\#_{rate}$ is the rate to reach this node and $\#_{fee}$ represents the cost to reach this node.

As by definition of the channel network, each channel has a certain capacity, hence each node has a certain maximum it can pay over a specific channel ($bal(n, c)$). While this is the overall maximum, a specific node may only be willing to forward a fraction of it, i.e. $\#_{max}$. Notably, $\#_{max} \leq bal(n_i, c)$ where i is the current node forwarding this request over channel c. This value varies depending on the request. Further, if a node receives a REQ and is not the destination node, it first updates the REQ before forwarding it. The fields n_o and n_d remain unchanged, n_l is set to the current node. The rate $\#_{rate}$ is updated with its local rate times the former rate and so is the new fee the sum of the local fee and the former fee. In order to compute the new $\#_{max}$ the current node checks how much it can send over the channel to n_l ($\#_{maxOld}$) and takes the minimum of those two values, i.e. $\#_{maxNew} = Min(\#_{maxOld}, \#_{maxCur})$. The same applies for the expiry time d_{route}, i.e. the current node updates this field with the minimum between the last d_{route} and the time it is willing to offer this route to n_o. Hence, using REQ each receiving node receives the information of how much it can send towards the originator n_o ($\#_{maxNew}$), for how long the route is valid d_{route}. In terms of units, the expiry date is expressed using the unix timestamp format. The rate and the fee are floating-point numbers in order to be able to represent the smallest unit of an arbitrary currency, e.g. 1 Satoshi.

Algorithm 1 shows a pseudo code of how a REQ is handled if received by a node. Lines 1 and 2 perform validation steps. First, the local node checks the req is still valid, i.e. if $req.d_{req}$ has not been expired and the max amount of hops has not been reached ($\#_{hops} \leq MAX\_HOPS$), i.e. if the REQ is dropped if it has traversed more than MAX_HOPS intermediate nodes. Afterwards, it checks if a valid channel exists to the last hop and if this channel has enough balance. Thereupon, the local node checks in its routing table if a route towards n_o is already present (line 3). If this is the case, the node checks whether the current req is *better* than the routing table entry. Within this check, the local node can follow its own economic incentives, e.g. take a lower rate and fee, a longer time expiry date, etc. The new routing table entry is created in line 5. In the next line 7, it is checked if this req has already been handled before, i.e. for this case the $req.nr_d$ is taken. If so, no further actions are performed. In line 8 it is checked whether the local node is already the final recipient. If this is the case, a REP is returned to the last hop n_l (see Sect. 4.2). Similar, in line 11 the local node checks if a route is already known to the destination node n_d.

Algorithm 1. handleRouteReq(req)

```
   Input: req: Route Request data
 1 if !isValid(req) then return;
 2 if !channelToExists(req.n_l) OR bal(req.n_l,c) ≤ 0 then return;
 3 t_o = getRoutingTableTo(req.n_o);
 4 if t_o != null AND req is better than routing table entry then
 5 |   t_o = updateTable(req.n_o, req);
 6 end
 7 if reqAlreadyHandled(req.nr_d) then return;
 8 if this == req.n_d then
 9 |   sendRouteRep(req.n_l); //node is destination, send rep to requester;
10 end
11 t_d = getRoutingTableTo(req.n_d);
12 if t_d != null then
13 |   sendRouteRep(t_o.n_n); //take from routing table and send response;
14 end
15 req.#_rate *= this.#_rate; req.#_fee += this.#_fee; req.#_max = Min(req.#_max,
     this.#_max); req.n_l = this; req.#_hops++;
16 lockFunding(req.n_l, req.#_max, req.d_route);
17 for c in outgoingChannels do
18 |   sendReq(req, c.counterNode);
19 end
```

If so, a REP is returned to the next hop in the routing table entry. If the local node is an intermediate node the REQ will be forwarded to all nodes to which the local node has a channel to. Hence, starting from line 15, the REQ will be updated, i.e. the new rate and fee to the next node is calculated and the last hop n_l is set to the current node. Also, the hop count ($req.\#_{hops}$) is updated and incremented by 1. Since within the REQ the node also promises a route towards n_o, the local node has to lock up some funds for some time (line 16). This is needed, in order to ensure enough funding is available for a payment over this route. Notably, as it is required to lock funds up, a node can decide on its own whether it will forward the REQ or not. So, at any time, a local node can decide not to forward a REQ at all, this is however not depicted in the algorithm.

4.2 Route Selection

As soon the REQ has reached its destination node n_d, or if a route towards the destination was already known by an intermediate node, the route responds message (REP) is returned towards the originator. REP has a similar format as REQ: $REP = <n_o, n_d, n_l, nr_d, nr_o, d_{route}, \#_{max}, \#_{rate}, \#_{fee}>$.

The field n_o defines the originator of the message, i.e. it is the destination node n_d of the REQ message. Similar to that, n_d is the destination of REP (or the originator n_o of REQ). n_l follows the same principle, i.e. it is always set to the last hop the message was send from. nr_d and nr_o define sequence numbers.

Algorithm 2. handleResponse(rep)

Input: rep: Route Response

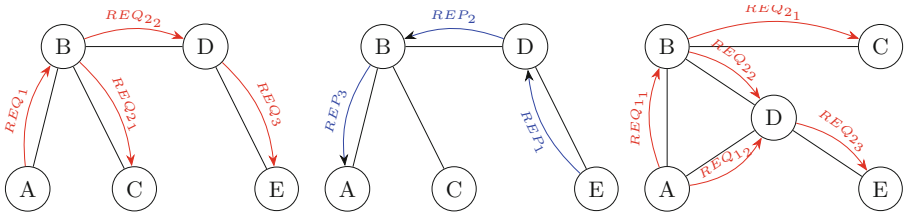
```
1  if !isValid(rep) then return;
2  if !channelToExists(rep.n_l) || !channelIsFunded(rep.n_l) then return;
3  t_o = getRoutingTableTo(rep.n_o);
4  if t_o == null || rep is better than t_o then
5  |  updateTable(rep.n_o, rep); //new rep is better, update table;
6  else
7  |  return; //known route is better;
8  end
9  if this == rep.n_d then
10 |  return; //local node was original requester
11 end
12 t_d = getRoutingTableTo(rep.n_d);
13 if t_d == null then
14 |  return;//error, no route to origin found
15 end
16 rep.#_rate *= this.#_rate; rep.#_fee += this.#_fee; rep.#_max = Min(rep.#_max,
     this.#_max); rep.#_hops++; rep.n_l = this;
17 lockFunding(t_o.nextHop, rep.#_max, rep.d_route);
18 sendRep(rep, t_d.nextHop);
```

The former one is the latest sequence number of the route's destination node and the latter one is the sequence number to be used in the route towards the originator n_o of the route request. d_{route} defines how long the route towards n_o is valid, $\#_{rate}$ is the rate to reach this node and $\#_{fee}$ represents the fee.

Algorithm 2 shows the pseudo code of how a REP message is handled: similar to handling the route request message, in line 1 and 2 the REP is verified to be valid. In this case, the field d_{route} is checked, i.e. if the route promised in REP has not yet expired. If this is the case it is checked if enough funding is available in the channel towards the last hop. In line 3 the local routing table is checked whether a route is already present to the node $rep.n_o$ and if the known route is *better* than the new route in REP. Again, the local node can follow its own economic incentives and accept only routes which are suitable. If the new route is better, or no routing table entry exists, the routing table is updated with a new entry towards the destination node. On the contrary, if a route already exists, the process ends here (line 7). In line 9 it is verified if the local node is the destination, i.e. the REP has reached the original requester of this route request. If this is the case, the process ends here and the local node can issue a payment along this route. Alternatively, the local node is an intermediate node and is meant to forward the REP. For that, it checks in the routing table if a route is known towards n_d (see line 12). If no route is found, the process ends here as it is not possible to forward the REP. Consequently, if the route towards n_d is known, the REP message is updated. Starting from line 16 to 16 the rate ($rep.\#_{rate}$), fees ($rep.\#_{fee}$), the max amount for this route ($rep.\#_{max}$), the last

hop ($rep.n_l$) and the hop count ($rep.\#_{hops}$) is updated. Afterwards, the node locks the max amount of the promised route ($rep.\#_{max}$) in line 17 and forwards the route to the next node according to the routing table entry ($t_d.nextHop$) in line 18. Similar to the route discovery phase, the most important part is the locking of the funds in a specific channel for the time the route is valid ($rep.d_{route}$) as can be seen in line 17. Since this information is only kept offline in a local node, it only represent a *promise* that the funds are available but it does not give a 100% guarantee that the route is valid until $rep.d_{route}$ expired.

Figure 1a show the process of how a route from node A to node E is established. For that, node A sends a route request (REQ_1) to its connected neighbours, i.e. B. In turn, B forwards this request to C and D (REQ_{2_1}, REQ_{2_2}) and so on. Eventually, the REQ arrives at node E which returns a REP message towards the originator A. Notable, since each REQ contains the information of how to reach the originator node, every node in this example has now the information of how to route towards A. The path the REP message follows is comparably simpler. As it can be seen in Fig. 1b, node E issues the first REP_1 message along the path towards A. Hence, the message first passes node D (REP_1), then node B (REP_2) and eventually arrives back at node A (REP_3). If node C now wants to pay A it can do so immediately as it has already all the information. If C wants to pay E, it will only need to issue one REQ towards B. B knows already the required information and returns REP immediately. Hence, the more active the overall network is, the less messages will be needed to find a route. After having explained the algorithm, we will evaluate it in Sect. 5.



(a) REQ phase (b) REP phase (c) Ig. redundant messages

Fig. 1. REQ and REP phase, and ignoring redundant messages

5 Evaluation

In order to understand if the adapted AODV routing protocol (i.e. by enhancing the messages with information on fees and exchanges rates) is applicable for payment routing in PCNs we evaluated it in a simulated environment. As at the time of writing the LN just went live on the Bitcoin main net[1], no real data about how the network will look like in big scale was available. Hence, we have to come up with some assumptions in regard to the network topology in Sect. 5. The results of the evaluation are discussed in Sect. 6.

[1] https://lnmainnet.gaben.win/.

Setup. We evaluate our routing protocol on 3 different network topologies, with 500 nodes, 1,000 nodes, 5,000 nodes. In regards of the network, we divide the nodes among 3 different blockchains, e.g. BTC, ETH and LTC. Each node is placed at random in one of these 3 blockchains with a uniform probability of $p = 0.3$. In the next step we take the smallest blockchain (i.e. the one with the least amount of nodes $n_{\#}$) and select randomly between 1 and $|n_{\#}|$ nodes. These nodes are Liquidity Providers between two blockchains, i.e. they have a wallet on both blockchains enabling routing between them. We repeat the same procedure and connect the smallest chain with the second chain. The same is repeated between the remaining two blockchains. In the next step we randomly connect the nodes within each chain according the *dynamics of small-world* networks as presented by Watts and Strogatz [29]. We chose $k = 4$ as the average connection between each node within each chain and a probability of rewiring of $p = 0.3$. To generate the graph, we used the WattsStrogatzGenerator (http://graphstream-project.org). We decided to have the funding on each channel side randomly generated with a value $f = [1, 100]$. We ignored the different currencies, meaning that the overall maximum a channel can have is $MAX(bal(c)) = 200$ BTC (100 BTC on each side of the channel) and the minimum is $MIN(bal(c)) = 2$ LTC (1 LTC on each side). As exchange rates between BTC:LTC, LTC:ETH and BTC:ETH we took fix values as monitored from http://coinmarketcap.com (2017/09/25, 3:25 pm UTC+10): 1 ETH = 0.07508560 BTC, 1 LTC = 0.01262120 BTC and 1 LTC = 0.16506884 ETH. Each Liquidity Provider offers the same exchange rate; however, each node randomly charges a fee when forwarding payments ($rep.\#_{fee}$ or $req.\#_{fee}$). This fee is randomly generated only once per node and is between $*.\#_{fee} = [0, 1] * 10^{-9}$. The details of generated topologies can be found in Table 1. Notably, since channels are bidirectional a connection between two nodes counts as one channel.

Table 1. Evaluation settings

Nodes	500			1000			5000		
Channels	1098			2367			10689		
	BTC	ETH	LTC	BTC	ETH	LTC	BTC	ETH	LTC
Nodes	149	147	204	285	306	409	1519	1496	1985
Channels	298	294	408	570	612	818	3038	2992	397
BTC-ETH channels	13			257			408		
BTC-LTC channels	76			103			193		
ETH-LTC channels	9			7			88		

Scenario & Evaluation Criteria. In order to verify the quality of our approach we run 1,000 randomly generated transactions on each network topology, i.e. we randomly select one node which pays another random node. The payment amount is randomly generated with $\# = [0, 1] * 10^{-9}$. Notable, while in theory

it is possible to have even smaller values as payments are not recorded on the blockchain immediately and hence are not limited to it, this would require a different setup. This means, if a payment is 1/10 of a satoshi, we would need 10 of these payments in order to have a noticeable change, hence, we ignored this factor. The limiting factor of the AODV routing protocol is the hop count (MAX_HOPS), i.e. if the REQ has traversed more than MAX_HOPS intermediate nodes, the request is dropped. Hence, we run each transaction with different MAX_HOP, i.e. with an $MAX\_HOP = [0, N]$ where N is the cheapest possible path but capped with 10. Differently expressed, if two nodes are connected directly the hop count is 0 and there is one channel between them. If there is an additional node in between, the hop count 1 and there are two channels between them. Hence, we have a maximum of 10 hops or a maximum of 11 channels. The cheapest possible path was computed manually using the Floyd-Warshall all pair shortest path algorithm [3, pp. 558–565]. As a single route request may change the networks topology we reset the nodes and the channel states before each new request. We compare the found routes with the optimal path, i.e. the cheapest overall path for each hop count. This means that a lower hop count could lead to a more expensive route than the optimal. In addition, we count the overall messages which were send around and measure the reachability, i.e. how many transactions where successfully depending on the hop count.

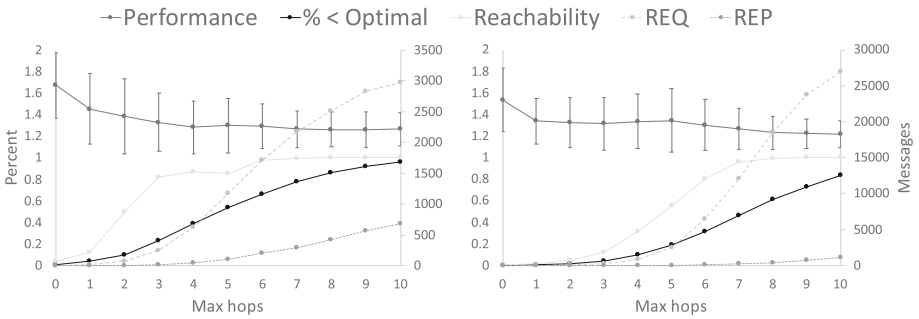
6 Discussion

The results of our evaluation can be found in Fig. 2 (not showing the 1,000 nodes scenario due to space constraints). As mentioned above in Sect. 5 we have 3 different network topologies with 500, 1,000 and 5,000 nodes. On each topology we ran 1,000 randomly selected transactions, i.e. the payer and payee were selected randomly. Each transaction was run 0 to N times where N is capped with 10 or with the shortest optimal route. Hence, the higher the HOPS get, the less transactions are executed. Two nodes are separated by 1 hop if there is a single intermediate node in between or by 0 hops if they are connected directly. $\overline{P}$ is the arithmetic mean of the performance P. It indicates how much more expensive the found paths were on average compared to the optimal shortest path (The optimal shortest path was calculated using Floyd-Warshall algorithm.), e.g. an $\overline{P}$ of 1.53 means that on average the found path was 1.53 times more expensive than the optimal path. In addition, $\sigma_{\overline{P}}$ states its standard deviation. For simplicity reasons we normalised the number of send REQ and REP messages by dividing it through the amount of transaction ($\#_{TX}$).

As the figures clearly show, the performance $\overline{P}$ goes towards 1 slowly the higher the max allowed of hops get (MAX_HOP). The reason why we do not hit 1 at 10 already is twofold. First, we allowed only a max amount of 10 hops although there were a few transactions with an optimal route with more than 10 hops, i.e. for 500 nodes 3.9%, for 1,000 nodes 1.6% and for 5,000 nodes 16.3% needed more than 10 hops to find an optimal solution. Besides, the standard

deviation of the $\overline{P}$ decreases with the number of hops. Hence, we can say that our routing finds more results closer to the actual optimal route. The chance of being able to find the optimal route depending on the hops is expressed by $\% < Optimal$. As it can be seen, the higher the hop count is the higher the chance to finding the optimal solution the closer we get to it. The second reason why this number approximates slowly to 1, is that there is a chance that the optimal route is never found, although the maximum allowed hops would allow for it. This can be reduced to the fact that we try to keep the amount of unnecessary messages low. For example, each node handles each REQ and REP only once unless its information is fresher or *better*. Figure 1c shows a route request (transaction) from A to E. In the first phase, A sends a request to B and D (REQ_{1_1} and REQ_{1_2}). As soon as received, these nodes forward their messages REQ_{2_1} and REQ_{2_2} from B to C and D and REQ_{2_3} from D to E. As D has received a REQ for a payment from A to E prior from A, it ignores the message from B. This could be a limiting factor if the route from A to D via B is cheaper than A to D. However, we made this decision knowingly in order to reduce the amount of REQ which are send around, as otherwise the network would get flooded completely. In addition, in order to have a higher probability of finding a better route, we allowed for the following. If the REQ from B arrives at D when D has already found a route to E it returns indeed a REP message. Hence, in this example, A would receive two REP messages from which it can chose the more suitable one. This fact explains why our results show a relatively high amount of REP. The figures show the total amount of REQ and REP messages in the network normalised by the amount of sent transactions. It is obvious that the amount increases almost exponential with the amount of allowed hops. The problem lays in the information each node is acting on, i.e. while nodes to ignore already handled REQ messages it cannot know whether a node has already received this message from a different node or not. A simple solution would be to add information to the REQ/REP message *where* it has already been. However, while this would reduce the amount of messages, it would increase the size of it. Interestingly, while for 500 nodes and 1,000 nodes we were able to achieve a reachability of over 99% for 5,000 nodes we were able to achieve a similar number only with 7 hops. This lead to a dramatic increase of REQ messages which were send around, i.e. while we had a reachability of 80% and 6,405 REQ messages for 6 hops we had a reachability of 98% and 12,097 REQ messages for 7 hops. Hence, we would recommend to have a dynamic value for the maximum allowed hops, i.e. if the network grows the maximum hop count should increase. Notably, the amount of REQ is decreasing over time heavily as nodes cache route information for some time. In the example in Fig. 1c at least 5 REQ messages are broadcast as A wants to pay E initially. If B wants to pay E in a later phase as well, it might have the information already available and does not need to send a REQ message but can execute a payment straight away.

While the numbers show that the adaptation of AODV for payment routing in off-chain channel networks is quite feasible for networks of nodes up to a few thousand participants we doubt it is scalable for millions of users or more. As



(a) 500 Nodes (b) 5000 Nodes

Fig. 2. Evaluation results in % (left scale) and number of messages (right scale)

in AODV the message size is comparably small (∼80 bytes) a network should be able to handle easily several thousand requests simultaneously as this would end up in only a few megabytes. Although established routes may expire over time, using a route maintenance message, crucial information can be updated. For that, the original route request issuer sends out a maintenance message along the desired route. Each intermediate node updates the information with its current fees and the distribution of funds in its channels and forwards the message along the path. Hence, once a route is established between two nodes, it may be reused infinitely if updated regularly.

Compared to Flare, our AODV-based protocol does not consider security protection of the message or its content. Hence, the sender and receiver may be publicly known along the payment route. However, significant attempts have been done to secure the AODV routing protocol [6]. Different extensions to AODV to increase the security have been proposed in the past: e.g. SAODV or ARAN which authenticates non-mutable fields and mutable information (hop count) of the message, using digital signature and hash chain [25,27,30]. These extensions can prevent tampering of control messages and data dropping attacks. We argue that a protection of data tempering in a *REP* or *REQ* message is not necessarily required as a route execution follows the atomic principle, i.e. either all transaction are successful or all fail. This means, if an intermediate node lies about the fees it would take (it promises a lower fee but would take a higher fee by falsifying the *REP* message), the payment will fail later on as the sender will only attach enough money so that the desired amount arrives at the receiver who in turn will reject the payment. Privacy and anonymity has been a direct focus of Flare which integrates Onion Routing in a way that only the last node actually knows who has been sending something to whom, all the nodes in the middle just forward it to the next hop. There is more privacy in such a system, however it has been shown that in cases where every node broadcasts the transaction onto the blockchain in similar time frames, clues of the routing can be deducted by the entire network. Even more, in source routing, edge nodes

can misuse this anonymity and attack an intermediate node in a way that it is unable to forward future payments. To do so, the attacker will need to have a higher funding (either concentrated on one node or several nodes) than the victim and it will need to be able to controller the payee. The attacker issues a payment request via an intermediate node, which will need to lock up funding for some time. This lock will be released automatically in case the payment was not successfully. However, in the meantime this node cannot use this funding for other purposes such as participating in other payment processes and will not earn money through additional fees. The same attack can be done using AODV. However, the difference is that in AODV each sender is known to intermediate nodes while this is not the case if using Onion Routing where the sender (and the final receiver) remains anonymously. Hence, using AODV each node can protect itself by either accepting or rejecting route requests by specific nodes.

Last but not least, a general problem of decentralised routing requires each participating node to be online as offline nodes are not able to forward any requests. This is why, the LN (or other PCNs) incentives nodes to stay online as they can earn transaction fees by routing payments through them.

7 Conclusion

In this paper we presented an adaptation of AODV for payment routing in payment channel networks such as Lightning, Raiden, or COMIT. We enhanced the messages with information on fees and exchanges rates in order to find a economical route through the network. AODV is a reactive routing protocol that only establishes a route when needed, thus avoiding the overhead of superfluous messages sent in a proactive routing protocol. However, AODV carries the risk of flooding the network if the maximal amount of hops is not set correctly. Our experiments reveal that the adapted AODV can easily be used in a network up to a few thousand nodes. Hence, AODV-based routing can be integrated into PCN. In future work we will focus on routing protocols that scalable further and evaluate how fee management of nodes impact the liquidity flow. Among respective developers, limited liquidity is a well-known problem for off-chain channel networks which remains to be solved.

References

1. Albrightson, B., Garcia-Luna-Aceves, J., Boyle, J.: EIGRP - a fast routing protocol based on distance vectors (1998)
2. Chao, L., Aiqun, H.: Reducing the message overhead of AODV by using link availability prediction. In: Zhang, H., Olariu, S., Cao, J., Johnson, D.B. (eds.) MSN 2007. LNCS, vol. 4864, pp. 113–122. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-77024-4_12
3. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms, 3rd edn. MIT Press, Cambridge (2009)

4. Decker, C., Wattenhofer, R.: A fast and scalable payment network with Bitcoin duplex micropayment channels. In: Pelc, A., Schwarzmann, A.A. (eds.) SSS 2015. LNCS, vol. 9212, pp. 3–18. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-21741-3_1
5. Fonseca, R., Ratnasamy, S., Zhao, J., Ee, C.T., Culler, D., Shenker, S., Stoica, I.: Beacon vector routing: scalable point-to-point routing in wireless sensornets. In: Proceedings of Symposium on Networked Systems Design and Implementation (2005)
6. Gharehkoolchian, M., Hemmatyar, A.M.A., Izadi, M.: Improving security issues in MANET AODV routing protocol. In: Mitton, N., Kantarci, M.E., Gallais, A., Papavassiliou, S. (eds.) ADHOCNETS 2015. LNICST, vol. 155, pp. 237–250. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-25067-0_19
7. Haas, Z.J., Pearlman, M.R., Samar, P.: The Zone Routing Protocol (ZRP) for Ad Hoc Networks. IETF Internet Draft (2002)
8. Hosp, J., Hoenisch, T., Kittiwongsunthorn, P.: COMIT - cryptographically-secure off-chain multi-asset instant transaction network (2017). http://www.comit.network/doc/COMIT%20white%20paper%20v1.0.2.pdf
9. Johnson, D.B., Maltz, D.A.: Dynamic source routing in ad hoc wireless networks. In: Imielinski, T., Korth, H.F. (eds.) Mobile Computing. Springer, Boston (1996). https://doi.org/10.1007/978-0-585-29603-6_5
10. Karp, B., Kung, H.T.: GPSR: greedy perimeter stateless routing for wireless networks. In: International Conference on on Mobile Computing and Networking. ACM (2000)
11. Medhi, D.: Network routing: algorithms, protocols, and architectures (2010)
12. Miller, A., Bentov, I., Kumaresan, R., McCorry, P.: Sprites: payment channels that go faster than lightning (2017)
13. Mistry, N., Jinwala, D.C., Zaveri, M., et al.: Improving AODV protocol against blackhole attacks. In: International Multi Conference of Engineers and Computer Scientists (2010)
14. Mitton, N., Fleury, E.: Distributed node location in clustered multi-hop wireless networks. In: Cho, K., Jacquet, P. (eds.) AINTEC 2005. LNCS, vol. 3837, pp. 112–127. Springer, Heidelberg (2005). https://doi.org/10.1007/11599593_9
15. Murthy, S., Garcia-Luna-Aceves, J.J.: An efficient routing protocol for wireless networks. Mob. Netw. Appl. **1**, 183–197 (1996)
16. Nakamoto, S.: Bitcoin: a peer-to-peer electronic cash system (2008). https://bitcoin.org/bitcoin.pdf. Accessed 17 Apr 2017
17. Pacia, C.: Lightning network skepticism (2015). https://chrispacia.wordpress.com/2015/12/23/lightning-network-skepticism/. Accessed 22 Mar 2018
18. Pei, G., Gerla, M., Hong, X.: LANMAR: landmark routing for large scale wireless ad hoc networks with group mobility. In: ACM International Symposium on Mobile Ad Hoc Networking and Computing (2000)
19. Perkins, C.E., Royer, E.M.: Ad-hoc on-demand distance vector routing. In: Second IEEE Workshop on Mobile Computing Systems and Applications (1999)
20. Poon, J., Dryja, T.: The Bitcoin lightning network: scalable off-chain instant payments (2015)
21. Prihodko, P., Zhigulin, S., Sahno, M., Ostrovskiy, A., Osuntokun, O.: Flare: an approach to routing in lightning network (2016)
22. Raiden: Raiden network (2016). http://raiden.network/. Accessed 07 Aug 2017
23. Reed, M.G., Syverson, P.F., Goldschlag, D.M.: Anonymous connections and onion routing. IEEE J. Sel. Areas Commun. **16**, 482–494 (1998)

24. Ripple: Ripple paths. https://ripple.com/build/paths. Accessed 12 Sept 2017
25. Sanzgiri, K., Dahill, B., Levine, B.N., Shields, C., Belding-Royer, E.M.: A secure routing protocol for ad hoc networks. In: IEEE International Conference on Network Protocols (2002)
26. Song, R., Korba, L., Yee, G.: AnonDSR: efficient anonymous dynamic source routing for mobile ad-hoc networks. In: ACM Workshop on Security of Ad Hoc and Sensor Networks (2005)
27. Wadbude, D., Richariya, V.: An efficient secure AODV routing protocol in MANET. Int. J. Eng. Innov. Technol. **1**, 274–279 (2012)
28. Wang, L., Shu, Y., Dong, M., Zhang, L., Yang, O.W.: Adaptive multipath source routing in ad hoc networks. In: IEEE International Conference on Communications (2001)
29. Watts, D.J., Strogatz, S.H.: Collective dynamics of 'small-world' networks. Nature **393**, 440–442 (1998)
30. Zapata, M.G., Asokan, N.: Securing ad hoc routing protocols. In: Proceedings of the 1st ACM workshop on Wireless Security (2002)