CoinExpress: A Fast Payment Routing Mechanism in Blockchain-based Payment Channel Networks

Ruozhou Yu, Guoliang Xue, Vishnu Teja Kilari, Dejun Yang, Jian Tang

***Abstract*—Although cryptocurrencies have witnessed explosive growth in the past year, they have also raised many concerns, among which a crucial one is the scalability issue of blockchain-based cryptocurrencies. Suffering from the large overhead of global consensus and security assurance, even leading cryptocurrencies can only handle up to tens of transactions per second, which largely limits their applications in real-world scenarios. Among many proposals to improve cryptocurrency scalability, one of the most promising and mature solutions is the payment channel network (PCN), which offers off-chain settlement of transactions with minimal involvement of expensive blockchain operations. In this paper, we investigate the problem of payment routing in PCN. We suggest crucial design goals in PCN routing, and propose a novel distributed dynamic routing mechanism called *CoinExpress*. Through extensive simulations, we have shown that our proposed mechanism is able to achieve outstanding payment acceptance ratio with low routing overhead.**

***Keywords*—Cryptocurrency, payment channel network, routing, blockchain, Bitcoin Lightning Network**

I. Introduction

Ever since the invention of Bitcoin by Satoshi Nakamoto in 2008 [17], we have well witnessed the blooming of thousands of altcoins[1], which (along with Bitcoin) jointly support a digital payment market with over $800B of capitalization at its peak[2]. It is envisioned that, in addition to the digital world where digital payments already prevail, sectors such as banking, international trading, manufacturing, healthcare, taxation and many more will also enjoy extensive benefits from this trend. Furthermore, the underlying blockchain technology has inspired (and will continue to do so to) numerous innovations, fundamentally transforming the business models of internet-of-things, supply chain, auditing etc., and even exerts political and human right influences with applications in governance transparency, democratic voting and freedom of speech.

However, current mainstream cryptocurrencies such as Bitcoin and Ethereum, although achieving strong security through decentralization, bear some severe limitations that greatly hinder their applications in our daily life. A significant one is the *scalability issue* brought by the requirement of global consensus and the large-overhead of security assurance through expensive consensus algorithms. Both Bitcoin and Ethereum employs the Proof-of-Work consensus algorithm, which can only generate blocks in a pre-specified speed, and each block can only include a limited number of transactions to avoid centralization. As a consequence, the Bitcoin blockchain can only process up to 7 transactions per second (tps) [19], while the number for Ethereum is around 15 tps [7], compared to over 45000 peak tps handled by Visa [19].

Facing this issue, there have been extensive efforts on improving blockchain scalability in several orthogonal directions. Among them, a promising proposal is to construct off-chain *payment channels*, which carry out transactions with minimal involvement of the blockchain itself. Specifically, users build peer-to-peer (P2P) channels with pre-deposit funds, and transfer values by re-adjusting fund allocation on the channels for each on-going transaction. Each transaction is protected by a smart contract, such that any non-cooperative behavior will be punished by granting all fund on the channel to the counter-party. In such a scenario, all transactions via a channel are stacked, and will be jointly published to the public blockchain upon channel expiration. Therefore the involvement of expensive blockchain operations is limited to only channel establishment, close-out, and the rare events of dispute arbitration in case of non-cooperative behaviors.

It has been envisioned that a distributed network comprised of these payment channels, namely a *payment channel network* (PCN), can take most of the transactions off-chain, hence drastically reducing payment overhead and increasing scalability of the payment system [19]. Both leading cryptocurrencies are actively seeking the deployment of PCNs alongside their main blockchains; see Sec. VI. However, PCN has its own problems to be addressed. Maintaining a payment channel basically requires locking a certain amount of funds within the channel for an extended amount of time. Hence each normal user only has the capability to maintain a small number of channels with closely related parties. When paying to an arbitrary recipient in the world, most likely an indirect payment is needed, which spans multiple channels in the network. Such indirect payments can cause a number of problems. First, one or multiple payment paths from sender to recipient need to be provisioned before the payment starts. In network terminology, the payment must be *routed* before going through. Second, a contract is needed to guarantee non-repudiation at each intermediate node. Other issues include denial-of-service attacks, privacy, node availability, transaction fees, etc.

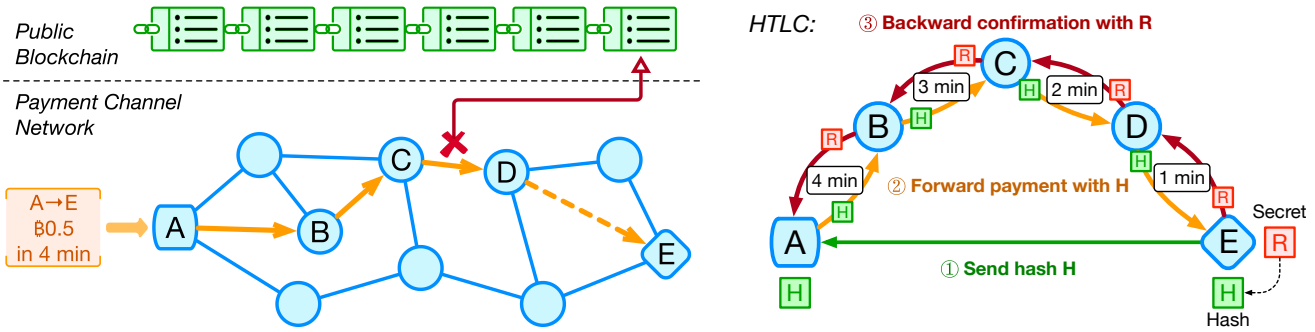
In this paper, we investigate routing for indirect payments in PCN. PCN routing has a number of distinct characteristics, which renders it intrinsically different from routing in traditional computer networks. First, PCN routing focuses on finding routes with *sufficient capacity (fund)* to serve a payment, rather than finding the shortest transmission paths as in traditional network routing. Second, PCN routing is fully distributed, as no central administrative operator exists in PCN; even if such an operator exists, it would not be trusted

Yu, Xue, and Kilari ({ruozhouy, xue, vkilari}@asu.edu) are with Arizona State University, Tempe, AZ 85287. Yang (djyang@mines.edu) is with Colorado School of Mines, Golden, CO 80401. Tang (jtang02@syr.edu) is with Syracuse University, Syracuse, NY 13244. This research was supported in part by NSF grants 1421685, 1525920, 1704092, 1717197, and 1717315.

[1]Altcoin refers to cryptocurrencies that are *alternative* to Bitcoin.

[2]Data is based on CoinMarketCap (https://coinmarketcap.com/).

978-1-5386-5156-8/18/$31.00 ©2018 IEEE



(a) In a PCN, user A requests payment of 0.5 Bitcoin to E within 4 minutes. A finds payment path A→B→C→D→E which has sufficient balance and can settle within 4 minutes. A sends initiate payment to B for forwarding. Each hop forwards payment. If intermediate node D wants to steal the payment by not forwarding the payment, C will then publish this situation to the public blockchain, who will judge on D's dishonesty and grant all funds on channel C→D to C as a punishment to D. Assuming all parties are honest, no blockchain operation is involved.

(b) For request A→E, recipient E generates secret R and its hash H, and sends H to A. A encodes H and deadline 4 minutes into its contract with B, such that B cannot spend the payment without providing R to A within 4 minutes. Each node forwards payment similarly, and employs a decreasing deadline. Upon reception, E will provide R to D to spend D's payment, while D and so forth will do the same to previous hops, until A receives R.

Fig. 1: PCN overview and Hashed TimeLock Contract (HTLC)

by any user, since its existence already defeats the spirit of decentralization that has set the foundation of cryptocurrencies. Third, PCN routing is more sensitive to dynamics in the network, due to the unique property of payment channels that the fund in a channel is commonly consumptive and non-recoverable unless other events happen; see Sec. II. Last but not least, privacy plays an important role; minimizing privacy leaks of the payer/payee (and possibly the involved nodes) can be a priority task in certain scenarios.

In face of these challenges, we propose *CoinExpress*, a routing mechanism that efficiently finds "express lanes" for cryptocurrency-based digital payments in PCNs. As a first step in PCN routing, CoinExpress focuses on high-performance and efficient payment routing, while leaving the privacy issue as future enhancements. We make the following contributions:

- We investigate important design goals of routing in PCN, and propose a practical model for PCN routing based on network flow and concurrent flows.
- We propose a distributed approach for PCN routing, which, in addition to finding routes that fulfill the payment, providing guarantee to the timeliness and availability regarding user's payment deadline and the expiration times of the underlying channels, respectively.
- We have shown, through simulations, that CoinExpress not only achieves superior payment successful ratio, but also drastically reduces overhead over existing work.

The rest of this paper is organized as follows. In Sec. II, we introduce the background of PCN and routing, and state the design goals. In Sec. III, we present our system model. In Sec. IV, we propose our distributed PCN routing design, and describe the detailed algorithms for each involved party in routing. In Sec. V, we show performance evaluation results of our design compared to a state-of-the-art PCN routing scheme and other baselines. In Sec. VI, we review existing work in related areas. In Sec. VII, we conclude this paper.

II. Background and Overview

A. System Overview

A PCN consists of several components, as shown in Fig. 1(a). Their detailed functions are explained below.

In PCN, a user is identified by a unique account address, usually also the public key of the account. A payment channel can be viewed as a temporary joint account between two users, whose balance is divided by the two parties and the division can be adjusted based on agreement of both users. A channel is established by both parties each depositing a certain amount into the joint account. The total deposit amount is called the *channel capacity*, which defines the maximum amount of value that can be transferred via this channel. A *unidirectional channel* only allows monotone balance adjustments, while a *bidirectional channel* allows adjustments in both directions.

In PCN, a *transaction* is essentially a channel balance update agreed upon by both parties. A channel is protected by multi-signature smart contracts, which ensure validity, non-equivocality and non-repudiation of the on-going transactions. When one party publishes obsolete balance history to reverse settled transactions or to double-spend, the contract guarantees that the dishonest party is punished by granting all its remaining channel balance to the other party. This economically prevents an adversary from utility gain via dishonest behaviors.

A *payment* from sender to recipient is performed via a number of transactions in different channels, organized as either a *payment path* or a *payment flow*. A *direct payment* can be made between two parties who share a channel, and is settled immediately after the corresponding transaction is completed. If two parties do not share a channel, an *indirect payment* is needed, which requires balance updates on multiple channels. This, however, can lead to issues if an intermediate node denies performing a subsequent transfer after receiving a preceding one, or the recipient denies receiving the payment.

Hashed TimeLock Contract (HTLC) [19] is introduced to solve the multi-hop problem, as shown in Fig. 1(b). An HTLC consists of both a *hash lock* and a *time lock*. In the hash lock, recipient generates a random value R with hash value H, and sends H to the sender. Sender, as well as any intermediate node, includes H in the transaction contract, such that the transferred fund is spendable by the transferee only when the secret R used to generate H is provided to the transferor. This ensures non-repudiation of transfer reception, as no one can spend its received amount without acknowledging the reception. In the time lock, each transaction is restricted by

a completion deadline, such that if the transferor does not receive R by the deadline, the transferred fund will be refunded to the transferor. In an indirect payment, each forward hop employs a decreasing deadline, taking into account the time for completing its own balance update as well as the time the two parties wish to wait to tolerate delay fluctuations in other hops. Any non-cooperative behavior will cause the corresponding transaction to be published on the public blockchain, which will grant all funds in the channel to the counter-party as a punishment to the non-cooperative party. By employing a sequence of time-outs in the opposite direction of the payment path, it is guaranteed that no transferee or recipient can hold up a channel until any preceding channel expires (in which case it can repudiate the reception and steal the fund).

From the above discussion, the key in performing a successful indirect payment is to find a *route* with sufficient balance, in the form of a path or a set of paths (a flow), such that an end-to-end time lock can be established in each path that respects the expiration time of each channel in the path. Next, we elaborate a number of unique challenges of routing in PCN, and a set of desired design goals of any routing solution.

B. Challenges in Routing Design

At a first glance, routing in PCN is just a variant of the widest path problem or the maximum flow problem, for which many efficient algorithms exist. Unfortunately, the problem is not as simple as it seems due to many practical constraints.

First, due to the time lock in HTLC, each payment path needs to satisfy a sequence of time delay constraints. For example, the last hop's channel expiration time needs to be lower bounded by the cumulative update and transmission delay of all hops in the path; the second last hop's expiration time needs to be lower bounded by the same cumulative delay plus the backward delay of the last hop; so on and so forth. Moreover, a user may require fast payment settlement, by specifying a deadline before which the payment needs to be settled. For the payment flow problem, it is well-known that the Multi-Path routing with Bandwidth and Delay constraints problem (MPBD) is NP-hard, corresponding to our routing problem that only considers the last hop's expiration time.

Second, the remaining balance on each directional payment channel is *non-interfering monotone*. Here, a directional payment channel refers to either a unidirectional channel, or one direction of a bidirectional channel. For each directional payment channel, its remaining balance is *consumed* after each transaction. *Non-interfering monotonicity* means that unless the channel is explicitly interfered with, *e.g.*, is recharged via external funding or opposite-direction transactions, the consumed balance cannot be used by other transactions on the same direction. We use the term *monotonicity* for abbreviation.

One consequence of channel monotonicity is the unpredictability of balances in the network, which is due to the highly dynamic nature of the network. The available balance of each channel is constantly changing with every transaction. Thus it is impractical or even impossible for every node to keep track of the real-time balances of all channels in the network, especially when modern payment networks typically scale to billions of nodes [3], [19]. To find a route with sufficient balance, the sender could either estimate channel balances based on empirical data, but with a high risk of payment failure due to estimation inaccuracy, or actively probe for available balances, which incurs probing overhead.

Moreover, high network dynamics and channel monotonicity can cause concurrency issues. When multiple concurrent payments compete on one or more channels, they may block each other from progressing. In the worst case, deadlocks could happen, locking up funds on all involved channels [15].

C. Design Goals

Addressing these challenges require routing mechanisms that satisfy a set of design goals, which we elaborate below.

- **Timelock-compatibility:** In an HTLC-guarded PCN, a major requirement of payment routing is to ensure that a feasible end-to-end time lock can be successfully established along each path, which guarantees the commitment of honest processing at any involved node.
- **Distributedness:** The routing mechanism must not rely on a centralized trusted party. Centralized routing is subject to single point of failure upon external attacks, and hence cannot be trusted by users. Instead, nodes need to communicate with each other and conduct local computations to find routes for payments.
- **Concurrency:** A routing mechanism should be non-blocking in that at any time, at least one payment can progress without waiting for concurrent payments.
- **Goodput:** The mechanism should maximize system goodput, measured by the number or total value of successful payments in a given time window. This is not equivalent to maximizing system throughput, which is the total value that can be delivered in the period. A partially fulfilled payment is viewed as failed if the recipient does not receive the full amount within the deadline.
- **Efficiency:** First the mechanism should minimize the routing and payment latency incurred by users. In more time-sensitive scenarios, the mechanism should guarantee payment settlement within a user-specified deadline. Second, the mechanism should only incur limited overhead on both the end-points and the intermediate nodes.
- **Privacy:** The mechanism should preserve secrecy of various information in the network, which is distinguished into the following types:
  - *Sender/Recipient Privacy:* An adversary should not be able to determine the sender/recipient of any payment between non-compromised parties.
  - *Value Privacy:* An adversary should not be able to learn the exact value of any payment. Moreover, the adversary should learn as little information as possible about the range of value of any payment.
  - *Path Privacy:* An adversary should not be able to learn the path(s) of any payment, other than the nodes it has already compromised.
  - *Channel Balance Privacy:* An adversary should not be able to learn the exact balance of a payment channel at any given time, unless the channel connects to a node it has already compromised.
  - *Channel Load Privacy:* An adversary should not be able infer the load on a payment channel connecting non-compromised nodes, from sources such as the settlement delay of the channel.

As a first step, in this paper we aim to achieve the first five goals: *timelock-compatibility*, *distributedness*, *concurrency*, *goodput* and *efficiency*. Jointly achieving *privacy* with these goals is a non-trivial task due to the large overhead of most privacy-preservation techniques. Hence we leave efficient privacy-preserving routing to our future work.

III. System Model

A. Network Model

A distributed PCN is modeled as a weighted directed graph $G = (V, E)$. V is the set of nodes, *i.e.*, users each of whom has established at least one payment channel with a peer user. E is the set of links. A link denotes either a unidirectional channel from one user (the transferor) to another (the transferee), or one direction of a bi-directional channel between two users.

Each link has several attributes. First, each link $e \in E$ has a *channel capacity* c_e, denoting the total amount of value deposit into the underlying payment channel by both parties. Second, each link $e \in E$ also has a current *balance* b_e. For a unidirectional channel (represented by a single link in G), the capacity c_e is the maximum amount of value that the transferor can send to the transferee before the channel expires, while the balance b_e is the remaining amount of value that the transferor can send. For a bidirectional channel (two collateral links in opposite directions between two users), the two links have the same capacity, equal to the sum of their balances, *i.e.*, $b_{(u,v)} + b_{(v,u)} = c_{(u,v)} = c_{(v,u)}$. A link e always has $b_e \leq c_e$. For simplicity, we assume the set E only contains links with positive balances at any time. A link with zero balance is temporarily removed from the graph (*i.e.*, the views of the nodes it connects), until its balance gets recharged by new deposits or payment transactions on the opposite direction.

Payment through a channel is not instant. First, each channel needs to complete arriving payments sequentially. Second, it requires time for the two parties to agree on the balance update, which is the forward processing time of the current hop. Third, due to the time lock in HTLC, the channel further needs to release the locked transferred value after receiving acknowledgement from the next hop, which requires a certain amount of waiting plus backward processing time. For simplicity, we omit the transmission delay between parties, which can be incorporated into the forward and backward processing times. We use d_e^1 to denote the forward wait time plus the forward processing time of a link e, and d_e^2 to denote the backward wait time plus the backward processing time, at any given time. We let $d_e = d_e^1 + d_e^2$. Each link further has an expiration time η_e, denoting the time when the underlying channel becomes unavailable. For a payment via e to be successful, it must settle before the channel expires.

We assume that each user only has local knowledge on all its in-coming and out-going links, including their capacities, expiration times, as well as instantaneous balances and delays. Each sender/recipient additionally knows the other party's address, but does not know the other party's location in the network. In general, each node may have rough estimation of the overall network status based on local information, but cannot know the instantaneous balance or delay of any remote link, due to network asynchrony and dynamics.

B. Payment Model

A *payment request* is denoted by a quintuple $R = (s, t, a, st, dl)$, where s and t are the sender and recipient respectively, a is the amount to be paid, and st and dl are the start time and deadline respectively. Let $\mathcal{P}$ be the set of all paths in the PCN. A payment request R is realized by a *payment plan*, *i.e.*, a set of paths $P_R \in \mathcal{P}$, where each $p \in P_R$ is an (s, t)-path in the network. Each path $p \in P_R$ is associated with a payment value, denoted by v_p. For a payment plan P_R to be successful, it needs to satisfy the following conditions:

- P_R is **feasible**, iff for any link $e \in E$ where $P_R(e) \subseteq P_R$ is the set of paths through e, we have
$$\sum\nolimits_{p \in P_R(e)} v_p \leq b_e. \tag{1}$$
- P_R is **available**, iff for any $p \in P_R$ and $e \in p$, let $p_e^+ \subseteq p$ be links including and after e in p, then we have
$$st + \sum\nolimits_{e \in p} d_e^1 + \sum\nolimits_{e \in p_e^+} d_e^2 \leq \eta_e. \tag{2}$$
- P_R is **timely**, iff for any path $p \in P_R$, we have
$$st + \sum\nolimits_{e \in p} d_e \leq dl. \tag{3}$$
- P_R is **fulfilling**, iff
$$\sum\nolimits_{p \in P_R} v_p \geq a. \tag{4}$$

Based on the above definition, a payment plan that is *feasible, available, timely and fulfilling* is able to transfer a amount of value from sender s to recipient t within the deadline dl, and we call it a *realizing* payment plan for request R. In this paper, we are interested in finding a realizing payment plan for each payment request, while satisfying as many design goals as possible.

IV. Dynamic Routing Design

Most existing bandwidth-aware routing algorithms cannot be applied in our scenario, due to the frequently changing link balances and delays. A naive approach is to empirically estimate the channel statuses and route accordingly, yet such an approach is subject to poor accuracy and can lead to low goodput. We propose *CoinExpress*, a novel probing-based dynamic routing mechanism. In CoinExpress, the sender probes for channel statuses before payment, and reserves balances in advance. CoinExpress is fully distributed, adaptive to network dynamics and concurrent, and achieves high goodput. Unfortunately it does not provide privacy guarantee, which we aim to address in future work.

It should be noted that due to the availability and timeliness constraints, the problem of finding a realizing payment plan is NP-hard, even when the sender has full knowledge of the network. With only the timeliness constraint, the problem is equivalent to the MultiPath routing with Bandwidth and Delay constraints problem (MPBD), which has been proved to be NP-hard in [16], and the best centralized algorithm one can expect is an expensive *fully polynomial-time approximation scheme* (FPTAS) [16]. Instead, our approach is a distributed algorithm derived from the Ford-Fulkerson algorithm for maximum flow [9], and applies a locking technique to resolve concurrency among multiple simultaneous requests. Before diving into our algorithm, we first define some basic concepts on network flow.

A. Network Flow Preliminaries

To avoid ambiguity, we use *balance* to replace the term *capacity* that is commonly used in flow networks. For simplicity, for $u, v \in V$ such that $(u, v) \notin E$, we assume $b_{(u,v)} = 0$.

Definition 1 (Network flow). *Given network G with balances $\{b_e\}$, source s and destination t, an (s,t)-**flow** is defined as a mapping $f : V \times V \mapsto \mathbb{R}^*$, with the following properties:*
1) Flow conservation: for $\forall v \in V, v \neq s, t$,

$$\sum_{(u,v)\in E} f(u,v) = \sum_{(v,u)\in E} f(v,u);$$

2) Balance constraint: $f(u,v) \leq b_{(u,v)}$.
*We define $b(f) = \sum_{(s,v)\in E} f(s,v) - \sum_{(v,s)\in E} f(v,s)$ as the **flow value** of f.* □

Definition 2 (Concurrent network flows). *Given network G and a set of flows $F = \{f\}$, we say that the flows are **concurrent** iff they satisfy the joint balance constraint: $\sum_{f\in F} f(u,v) \leq b_{(u,v)}$.* □

Definition 3 (Residual network). *Given network G with balances $\{b_{(u,v)}\}$, and a flow f, the **residual balance** $b^f_{(u,v)}$ of each pair of $u, v \in V$ is defined as follows:*

$$b^f_{(u,v)} = \begin{cases} b_{(u,v)} - f(u,v) + f(v,u), & (u,v) \text{ or } (v,u) \in E; \\ 0, & \text{otherwise.} \end{cases}$$

*The **residual network** G^f regarding f is the network with the residual balances.* □

Note that the residual network may contain more links than the original one, due to the addition of backward links (v, u) when only the forward link (u, v) exists in the original network. For simplicity, we assume that each flow contains no loop with positive flow value, hence $f(u, v)$ and $f(v, u)$ cannot both be positive for $\forall u, v \in V$. For completeness, we show the Ford-Fulkerson algorithm in Algorithm 1.

Algorithm 1: Ford-Fulkerson max-flow algorithm [9]

Input: network $G = (V, E)$, source s, destination t
Initialize: start with an empty flow f and $G^f = G$
repeat
  Find (s, t)-path p in G^f with positive balance f_p;
  Add p to f, and update G^f;
until *no augmenting (s, t)-path can be found*;
return f.

B. Dynamic Routing Design

To apply the Ford-Fulkerson algorithm, we need to address several challenges. The first one is to transform Algorithm 1 into a distributed algorithm where each node only has local knowledge. Second, the timeliness and availability constraints need to be taken into account. Third, the concurrency issues needs to be tackled to ensure no harmful racing between concurrent requests. We address these in the following.

Our CoinExpress algorithm is jointly shown in Algorithms 2, 3 and 4. Note that when we say "send a message along a link e", we essentially mean *one party sending message to the other through a secure communication channel*, rather than actually sending fund through the payment channel.

Algorithm 2: Sender algorithm

Input: local links of G, request $R = (s, t, a, st, dl)$
Initialize: empty flow f, residual balances $\{b^f_e\}$
while $b(f) < a$ **do**
  for *any out-going link e of s such that $b^f_e > 0$* **do**
    Construct probe $\rho = (R, \min\{a - b(f), b^f_e\}, d^1_e, [(e, \eta_e, d^2_e)])$;
    Send ρ along e;
  Wait for current-round confirmation for time T_{conf};
  if *confirmation $\gamma = (R, p, \beta)$ is received from e* **then**
    if $b^f_e \geq \beta$ **then**
      Add path p to f with value β;
      Update residual b^f_e;
    else
      Send cancellation $\kappa = (R, p, \beta, \text{cancel})$ via p;
  else
    Send cancellation along all paths;
    Retry within time T;
return *flow f with $b(f) = a$.*

Our algorithm employs a distributed *breadth-first-search* (BFS) to search for augmenting paths. For the request, each node maintains a local view of the residual network, with residual balances initialized to the initial link balances. In each round, the sender sends a forward probe $\rho = (R, \beta, \delta, \Pi)$ to each neighbor who has a link e with positive residual balance from the sender, with starting balance $\beta = \min\{a - b(f), b^f_e\}$ and delay $\delta = d^1_e$; the parameter Π is an ordered list of all links, their expiration times and their backward delays along the path, and is initialized as $\Pi = [(e, \eta_e, d^2_e)]$. Each node, upon reception of the probe, also sends out a probe along each out-going link e with positive residual balance (except to the receiving neighbor), with updated probe $\rho' = (R, \beta', \delta', \Pi')$ where $\beta' = \min\{\beta, b^f_e\}$, $\delta' = \delta' + d^1_e$ and $\Pi' = \Pi \,\|\, (e, \eta_e, d^2_e)$ ($\|$ means appending). When the recipient receives a probe, first it checks for the timeliness constraint, *i.e.*, whether $st + \delta + \sum_{e\in p} d^2_e \leq dl$; second, it checks for availability of each link, *i.e.*, whether $st + \delta + \sum_{e\in p^+_e} d^2_e \leq \eta_e$. If either check fails, the recipient drops the probe and waits for the next one. Otherwise, the recipient returns a confirmation $\gamma = (R, p, \beta)$ backward along p, to confirm the new augmenting path p and flow value β. Each node then updates its residual balances, and waits for the next round. The algorithm stops when the sender has collected sufficient flow value for the request. If the recipient cannot find an augmenting path in the current round, it informs the sender, who cancels all reserved flows and retry later. If the sender does not receive confirmation within T_{conf} of sending the probe, it also cancels all flows and retry later.

There are two things worth mentioning. First, the asynchrony of the network can affect the algorithm's performance. Ideally, we want each node to forward the probe that has the lowest cumulative processing time, in order to better meet the timeliness and availability constraints. However, such a probe may not be the first to reach an intermediate node or the recipient; probes from paths with longer processing times but shorter transmission delays may arrive first. To account for this, each intermediate node can wait for a certain period τ after receiving the first probe of the current round, before forwarding the probe with the lowest cumulative delay ever

Algorithm 3: Intermediate node algorithm

– `Forward direction operation:`
Input: probe $\rho = (R, \beta, \delta, \Pi)$, incoming link e_{in}
Initialize: request list $\mathbf{R}$, flow f_R, residuals $\{b_e^f(R)\}$, current BFS round last hop $e_{\text{last}}(R)$
if $R \notin \mathbf{R}$ **then**
  Add R to $\mathbf{R}$, and create empty flow f_R and $b_e^f(R)$;
for *any out-going link* $e \neq e_{in}$ *such that* $b_e^f(R) > 0$ **do**
  Update probe $\rho = (R, \beta, \delta, \Pi)$ such that
    $\beta = \min\{\beta, b_e^f(R)\}$,
    $\delta = \delta + d_e^1$, and
    $\Pi = \Pi \,\|\, (e, \eta_e, d_e^2)$ ($\|$ means appending);
  Send ρ to neighbor along e;
  Store last hop: $e_{\text{last}}(R) \leftarrow e_{\text{in}}$;

– `Backward direction operation:`
Input: confirmation $\gamma = (R, p, \beta)$, incoming link e
if *residual balance* $b_e^f(R) \geq \beta$ **then**
  Add path p to f_R with value β;
  Update residual $b_e^f(R)$ based on Eq. (5);
  Send γ backward along link $e_{\text{last}}(R)$;
else
  Send cancellation $\kappa = (R, p, \beta, \text{cancel})$ to t along p;

– `Cancellation operation:`
Input: cancellation $\kappa = (R, p, \beta, \text{cancel})$
Cancel previous update of β of residual balance;
Send κ to next hop along p;

Algorithm 4: Recipient algorithm

– `Upon receiving probe:`
Input: probe $\rho = (R, \beta, \delta, \Pi)$
Form path p from list Π;
if $\delta + \sum_{e \in p} d_e^2 > dl - st$ **then** drop ρ and wait;
for $e \in p$ **do**
  if $\delta + \sum_{e \in p_e^+} d_e^2 > \eta_e - st$ **then** drop ρ and wait;
Construct confirmation $\gamma = (R, p, \beta)$;
Send γ backward along p;
For subsequent probes of the same round who also pass Lines 2–4, save them into $\boldsymbol{\rho}_{\text{list}}$ until a next-round probe is received;

– `Upon receiving cancellation:`
if *there is subsequent probe in* $\boldsymbol{\rho}_{list}$ **then**
  Pop the next probe ρ with path p and value β;
  Construct confirmation $\gamma = (R, p, \beta)$;
  Send γ backward along p;
else
  Inform sender s of the failure.

seen; the recipient can also wait before making a decision on path selection. The choice of τ is subject to each node's estimation of the network status, as well as the urgency of the request. On the other hand, probes may arrive out-of-order (w.r.t. rounds) in asynchronous networks. The sender can attach a round number rnd in each probe. Each node should discard probes in earlier rounds after a probe with higher rnd is received, because the recipient has already made a decision on the augmenting paths in the previous rounds.

The second thing is about concurrency. When multiple requests are jointly probing in the residual network, they may steal each other's flow by pushing flow through backward links that have reversed flow of other requests, a problem defined as *capacity stealing* by Rohrer *et al.* [22]. To address this, we apply their locking technique. Specifically, each node maintains a residual network for each request R, with flow f_R and residual balances $b_{(u,v)}^f(R)$ for $u, v \in V$. For each (u, v), the amount $f_R(v, u)$ is locked for request R only, while the rest $b_{(u,v)}^f(R) - f_R(v, u)$ can be shared by other requests. Therefore the residual balance in the concurrent case is defined as

$$b_{(u,v)}^f(R) = \begin{cases} b_{(u,v)} - \sum_{R' \in \mathbf{R}} f_{R'}(u, v) + f_R(v, u), \\ \qquad\qquad \text{if } (u, v) \text{ or } (v, u) \in E; \\ 0, \qquad\qquad \text{otherwise.} \end{cases} \quad (5)$$

When a new augmenting path goes through a link with locked balance, it first consumes the locked balance before consuming any remaining link balance, thus allowing other requests to come through. Another race condition that may happen is when two or more requests find augmenting paths that share the same link simultaneously, in which case their joint augmented flow may exceed the link balance. In this case, each node will serve confirmations in a *first-come first-serve* (FCFS) manner; when a subsequent confirmation arrives that cannot be served, the node will send a cancellation to reverse all updated residual capacities at intermediate nodes and inform the recipient. The recipient can then select another path to augment, or if no path can be found, inform the sender to either initiate a new round of BFS or abort and retry later.

Hence, the only race condition that may happen is when multiple requests are simultaneously confirming paths that block each other, in which case each path's confirmation is cancelled. The algorithm resolves this by two methods. First, the recipient can select another path to confirm, until all received paths are simultaneously blocked by others, which is very rare under normal network conditions. Second, in the rare event of all paths being blocked, the recipient will inform the sender, who will cancel its reserved balances on all confirmed paths to let pass other requests, meanwhile employing *random back-off* to retry in a later time (before its start time). In summary, through flow locking, each request will enjoy dedicated balance once enough confirmations are received. Therefore no single request will block the network from progressing at any stage of the actual payments.

C. Discussions

Here we highlight a few things that should be considered when implementing the above mechanism.

Criterion for path selection: In the proposed mechanism, the sender initiates each probing round, while the recipient is responsible for selecting paths to confirm in each round. One thing that may affect the algorithm's performance is the criterion of selecting paths to confirm. In general, selecting short paths over long paths can reduce payment settlement time. On the other hand, selecting paths with larger balance can reduce the number of paths per payment, hence reducing the number of routing rounds, as well as the overhead and transaction fees during actual payment. Choosing the right

criterion depends on the specific quality-of-service requirement of the actual use cases [25]. For example, requests with larger amounts may prefer widest paths to avoid long waiting time for routing and large fees, while smaller and more time-sensitive requests may prefer shortest paths to reduce settlement time.

Flooding avoidance in large networks: In large-scale networks, using BFS can lead to large flooding overhead. A common alleviating technique is to limit the hop count of each probe, *i.e.*, encoding a time-to-live (TTL) field in the message. The sender specifics the maximum TTL in the initial probe. Each node, upon reception of a forward probe, deducts the TTL by 1. If the TTL of a probe becomes 0, it will be dropped by the node that receives it. Advanced techniques can be employed to find the best TTL value to use in practice [6].

V. Performance Evaluation

A. Experiment Settings

To realistically evaluate our distributed mechanism, we developed a simulation tool based on network simulator `ns-3` [1]. The tool was developed as an application module in `ns-3`, which agrees with the actual standing of PCN protocol in real-world networks.

We used mesh topologies between users to model a PCN overlay network, where each node is connected to its PCN neighbors via direct communication links. Users were deployed in randomly generated Watts-Strogatz graphs [26]. The number of nodes varied from 50 to 250. Each node had a degree of 10, and each link had a re-wiring probability of 0.2 in the Watts-Strogatz model. We generated bi-directional payment channels with a uniform capacity of 100. However, each channel's initial balances on both directions were uniformly divided. Each channel's settlement times were uniformly generated in $[10, 50]$ seconds. Since our approach guarantees path availability, we assumed an arbitrarily large expiration time for all channels, to simulate the PCN in a static period of time; we focused on the timeliness constraint in our evaluation. We considered the transmission delays between nodes, which were uniformly generated in $[50, 200]$ ms. The data rate of the communication links were 100 Mbps.

In each simulation, we generated 1000 Poisson arriving requests between random nodes with mean arrival of 30 seconds. Requests had amounts uniformly generated in $[25, 75]$. We allowed 5 minutes for routing before the payment start time, and also a 5 minute payment deadline after start. Requests not routed before start time were cancelled and aborted.

B. Comparison Algorithms

CoinExpress has two versions: *CnExp-W* with widest path selection, and *CnExp-S* with shortest path selection.

We compared our algorithm to a state-of-the-art routing algorithm proposed by Rohrer *et al.* [22], which is based on the push-relabel algorithm for network flow. Before heading to the results, we first elaborate on a few issues of their algorithm. First, their algorithm is *delay-agnostic*. Unlike our augmenting path algorithm where path lengths can be easily bounded, the push-relabel algorithm cannot encode delay information during flow updates. Second, their algorithm requires an additional step of *flow decomposition* after obtaining a flow, which incurs extra overhead and routing time. Our algorithm directly derives payment paths during probing. Later on, we also show that their algorithm results in an excessive number of paths using a standard flow decomposition (Edmonds-Karp algorithm). This greatly increases system overhead during the payment process, and leads to high transaction fees in pay-for-use PCNs.

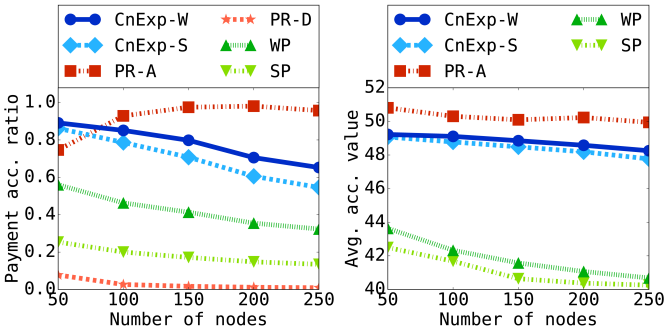
We denote their algorithm as *PR-A*, which stands for *Push-Relabel in delay-Agnostic mode*. For reference, we also implemented *PR-D* (*Push-Relabel in Delay-aware mode*), where we enforced strict delay bound to paths generated by the standard flow decomposition. In addition, two more baselines were implemented: *WP* for one-round widest path routing, and *SP* for one-round shortest path routing. Both baselines used confirmation after probing to assure non-blockingness.

C. Performance Metrics

All algorithms achieve *timelock-compatibility* (except *PR-A*), *distributedness* and *concurrency*. We therefore mainly evaluated the *goodput* and *efficiency* of the algorithms. The following metrics were used:

- **Acceptance ratio:** number of accepted payments over all submitted requests.
- **Average accepted value:** the average amount of values of each accepted payment.
- **Payment delay:** the average payment delay of each accepted payment.
- **Routing time:** the average time consumed for routing for each accepted payment.
- **Number of messages:** the average number of network-wide messages for successfully routing a payment.
- **Number of paths:** the average number of payment paths for each accepted payment.

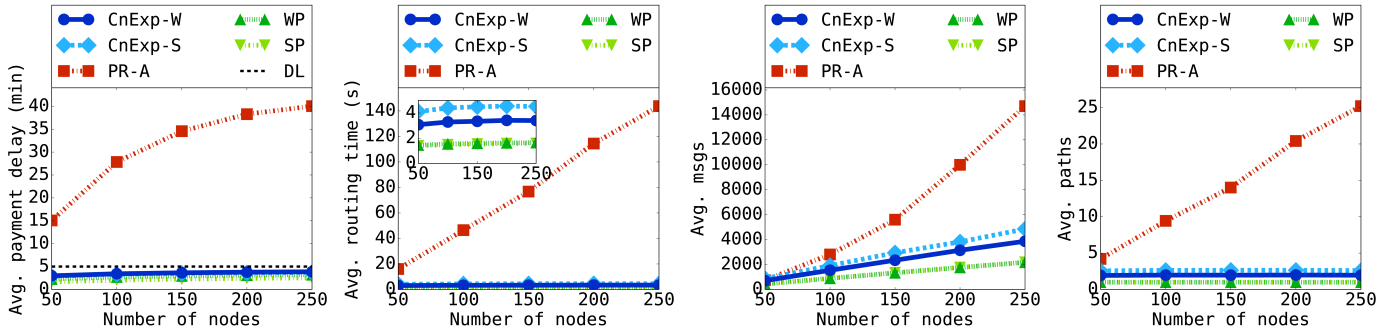
D. Evaluation Results



(a) Acceptance ratio (b) Average accepted value

Fig. 2: *Goodput (higher the better)*: acceptance ratio and average accepted value against number of nodes.

1) Goodput: Fig. 2 shows the acceptance ratios and average accepted values of the algorithms. From Fig. 2(a), CoinExpress algorithms achieve almost the highest goodput, except when compared to PR-A. This is because PR-A does not consider the timeliness constraints, hence although it achieves higher acceptance ratio, most accepted payments will fail due to deadline violation, as shown later. CnExp-W achieves slightly better goodput than CnExp-S, because the latter in general needs more paths due to each path carrying less value, where there may not be sufficient number of paths in some cases.



(a) Average payment delay (b) Average routing time (c) Average number of messages (d) Average number of paths

Fig. 3: *Timeliness and efficiency (lower the better)*: average payment delay, routing time, number of messages, and number of paths. In (a), dotted line DL shows the uniform 5-minute deadline for all payment requests.

All algorithms (except PR-A) has decreasing acceptance with increasing nodes. This is because although the nodes increase, the rewiring probability does not, which means longer paths between arbitrary node pairs. In other words, less paths that satisfy the timeliness constraints exist. PR-A has increasing acceptance because it neglects the deadlines. When enforcing timeliness on PR-A, we get the PR-D algorithm, which has extremely low acceptance. Less than 1% of the payments can be successfully settled using PR-D. Hence we neglect PR-D in the rest of our analysis due to insufficient samples for average analysis. The overhead of PR-D is very similar to PR-A, as their only difference is in the flow decomposition, which has a low overhead compared to the push-relabeling process.

In Fig. 2(b), our algorithms not only accept more payments, but also accept payments with larger amounts, compared to WP and SP. CoinExpress achieve near-optimal average accepted values (the average is 50 based on our experiment setting), very close to the delay-agnostic PR-A algorithm.

2) Timeliness and Efficiency: Fig. 3 further shows the timeliness status and efficiency metrics of the compared algorithms. First, in Fig. 3(a), we can observe the important timeliness measure of the algorithms. We can see that all algorithms except PR-A respect timeliness constraints (below DL). PR-A can result in payment delays of $8\times$ the deadline, severely violating users' fast payment requirements.

Fig. 3(b) shows the routing time of the algorithms. PR-A has extremely long routing times compared to the other algorithms. This may be a little counter-intuitive: Push-Relabel is regarded as a faster algorithm for maximum flow than Ford-Fulkerson. The reason is that here we are interested in finding a fulfilling flow rather than a max-flow, in which case Push-Relabel is slower due to its non-greedy pushing of flow in each step. Meanwhile, all others algorithms employed the flooding avoidance technique by setting a fixed TTL, hence their efficiency grows very slowly with increasing network size, demonstrating the great scalability of our algorithms.

Fig. 3(c) further shows the routing overhead in terms of the total number of routing messages per request. We can see that PR-A has a much larger overhead than CoinExpress, not to mention WP and SP. One reason is the flooding avoidance technique, which greatly restricts the overhead of our algorithms; however, Push-Relabel is hard to employ such techniques, resulting in large overhead for flow operations.

Fig. 3(d) shows the average number of payment paths output by each algorithm. As a baseline, both WP and SP have only one path. We can see that both CnExp-W and CnExp-S result in very few payment paths, typically around 2–3. Meanwhile, PR-A can result in as many as 25 different payment paths for a single payment. This reflects two things. First, this further explains why the routing overhead of CoinExpress is much lower than PR-A: our algorithms require much fewer paths to be probed. Second, our algorithms have much lower payment overhead by employing only a few paths, which can result in much lower transaction fees in practice.

To summarize, our algorithm achieves very impressive acceptance goodput performance compared to all algorithms that consider user timeliness constraints, while achieving much lower overhead than the state-of-the-art Push-Relabel routing.

VI. Related Work

A. Blockchain Scalability

Since the invention of blockchain in Bitcoin [17], extensive efforts have been devoted to improving the scalability of blockchain-based cryptocurrencies. Existing efforts can be divided into *on-chain solutions* and *off-chain solutions*. On-chain solutions focus on improving scalability by modifying existing blockchain design. A few promising techniques include increasing block size, using lightweight consensus algorithms, sharding [13], using Directed Acyclic Graph (DAG) instead of chain to store blocks [20], etc. Increasing block size directly increases a blockchain's capability to store and process more transactions, yet its direct threat is the fear of centralization. Lightweight consensus, such as Delegated Proof-of-Stake (DPoS) [12] or Practical Byzantine Fault-Tolerance (PBFT) [5], can greatly reduce overhead and increase scalability over the original Proof-of-Work (PoW) algorithm in Bitcoin and Ethereum. However, they either sacrifice decentralization (*e.g.*, DPoS) or require trust relationship between users (*e.g.*, PBFT). Sharding alleviates the scalability issue by dividing transactions into shards that are stored and processed at different nodes [13]. Block DAGs use weaker consensus where each transaction is only confirmed by a few instead of all up-coming blocks, which lowers block security.

Off-chain solutions seem more promising in solving blockchain scalability with limited compromise to its decentralization and security. One approach is to run multiple parallel blockchains that support cross-chain communications. Currently, the difficulty in this direction lies in the design of cross-chain communication protocols. Exchange-based protocols are the most popular at present, which uses one or multiple

chains as cryptocurrency exchanges that bridge between all other chains; for example, see [24]. The problem is that the exchanges can be more vulnerable to attacks, which may endanger the entire exchanging system. Another proposal is a hierarchy of blockchains organized as a *blockchain tree*, where child chains are supervised and secured by parent chains [18]. It does not solve the attacks on chains, but instead constrains the loss due to attacks to the local chain only.

PCN is possibly the only mechanism that are totally *off-chain* for now. Here, most transactions are carried in the off-chain payment network, and does not involve the blockchain at all. The only involvement of the blockchain is either when opening or closing the channels, or when parties are non-cooperative in channel updates, when the blockchain is used as arbitration. Through protocols such as HTLC, PCN guarantees almost the same security as the original blockchain, while dramatically increasing its scalability. Moreover, PCN technology is among the most mature over all the above, since the leading two cryptocurrencies are already on the edge of deploying PCN for their global chains: the Lightning Network for Bitcoin [19], and the Raiden Network for Ethereum [2].

B. PCN and Routing

The PCN concept originates from the *credit/payment networks* in economics and finance [8]. Early credit networks do not have blockchains, hence they commonly rely on the trust relationship between peers to establish and maintain channel states [8]. Ripple [3] and Stellar [4] are among the first to employ blockchain technology in credit networks. Much like in PCN, routing in credit networks can only be done in a distributed manner due to decentralization. Malavolta *et al.* [14] studied privacy-preserving routing in credit networks, where they designed a landmark-based routing scheme for privacy-preserving distributed routing. This idea is extended by Roos *et al.* [23] to provide enhanced routing capability. However, landmark routing assumes a small set of *trusted* landmark users who controls the entire routing process, an assumption that is commonly not true, and if true can lead to centralization of the P2P network. For PCN, Prehodko *et al.* [21] first proposed a beacon-based routing scheme in the Lightning Network, borrowing from existing ideas in mobile ad hoc networks. Their proposal is a path-based routing scheme, and does not guarantee the fulfillment of the payment. Rohrer *et al.* [22] proposed a distributed push-relabel algorithm for PCN routing with guaranteed concurrency.

Aside from the routing problem, some related efforts in PCN include automatic channel re-balancing [11], privacy-preserving contracts [10], [15], etc. In general, PCN is a promising area of research, where extensive efforts are in need to address its performance, security and privacy issues.

VII. Conclusions

In this paper, we studied the routing problem in PCN. We distinguished a number of important design goals for PCN routing, and proposed a mathematical model to capture these goals. As a first step, we designed a distributed routing mechanism, which achieved all but the privacy goals. We showed through extensive simulations that the proposed mechanism achieves outstanding goodput performance and very small overhead compared to the state-of-the-art routing design. In our future work, we will further address the privacy issue in PCN routing, as well as other possible issues.

References

[1] "NS-3 Network Simulator." URL: https://www.nsnam.org/

[2] "Raiden Network." URL: https://raiden.network/

[3] "Ripple." URL: https://www.ripple.com/

[4] "Stellar." URL: https://www.stellar.org/

[5] M. Castro and B. Liskov, "Practical Byzantine Fault Tolerance and Proactive Recovery," *ACM Trans. Comput. Syst.*, vol. 20, no. 4, pp. 398–461, nov 2002.

[6] N. Chang and M. Liu, "Revisiting the TTL-based Controlled Flooding Search," in *Proc. ACM MobiCom*, 2004, pp. 85–99.

[7] CoinDesk, "How Will Ethereum Scale?" URL: https://www.coindesk.com/information/will-ethereum-scale/

[8] D. Delli Gatti, M. Gallegati, B. Greenwald, A. Russo, and J. E. Stiglitz, "The Financial Accelerator in an Evolving Credit Network," *J. Econ. Dyn. Control*, vol. 34, no. 9, pp. 1627–1650, sep 2010.

[9] L. R. Ford and D. R. Fulkerson, "Maximal Flow Through a Network," *Can. J. Math.*, vol. 8, pp. 399–404, jan 1956.

[10] M. Green and I. Miers, "Bolt: Anonymous Payment Channels for Decentralized Currencies," in *Proc. ACM CCS*, 2017, pp. 473–489.

[11] R. Khalil and A. Gervais, "Revive: Rebalancing Off-Blockchain Payment Networks," in *Proc. ACM CCS*, 2017, pp. 439–453.

[12] A. Kiayias, A. Russell, B. David, and R. Oliynykov, "Ouroboros: A Provably Secure Proof-of-Stake Blockchain Protocol," in *Proc. CRYPTO*, 2017, pp. 357–388.

[13] L. Luu, V. Narayanan, C. Zheng, K. Baweja, S. Gilbert, and P. Saxena, "A Secure Sharding Protocol For Open Blockchains," in *Proc. ACM CCS*, 2016, pp. 17–30.

[14] G. Malavolta, P. Moreno-Sanchez, A. Kate, and M. Maffei, "SilentWhispers: Enforcing Security and Privacy in Decentralized Credit Networks," in *Proc. ISOC NDSS*, 2017.

[15] G. Malavolta, P. Moreno-Sanchez, A. Kate, M. Maffei, and S. Ravi, "Concurrency and Privacy with Payment-Channel Networks," in *Proc. ACM CCS*, 2017, pp. 455–471.

[16] S. Misra, G. Xue, and D. Yang, "Polynomial Time Approximations for Multi-Path Routing with Bandwidth and Delay Constraints," in *Proc. IEEE INFOCOM*, 2009, pp. 558–566.

[17] S. Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System," 2008.

[18] J. Poon and V. Buterin, "Plasma: Scalable Autonomous Smart Contracts Scalable Multi-Party Computation," Whitepaper, 2017. URL: http://plasma.io/plasma.pdf

[19] J. Poon and T. Dryja, "The Bitcoin Lightning Network: Scalable Off-Chain Instant Payments," Whitepaper, 2016.

[20] S. Popov, "The Tangle," Whitepaper, 2017. URL: https://www.iota.org/IOTA_Whitepaper.pdf

[21] P. Prihodko, S. Zhigulin, M. Sahno, and A. Ostrovskiy, "Flare: An Approach to Routing in Lightning Network," Whitepaper, 2016. URL: http://bitfury.com/content/5-white-papers-research/whitepaper_flare_an_approach_to_routing_in_lightning_network_7_7_2016.pdf

[22] E. Rohrer, J.-F. Laß, and F. Tschorsch, "Towards a Concurrent and Distributed Route Selection for Payment Channel Networks," in *Proc. ESORICS Workshop–CBT*, 2017, pp. 411–419.

[23] S. Roos, P. Moreno-Sanchez, A. Kate, and I. Goldberg, "Settling Payments Fast and Private: Efficient Decentralized Routing for Path-Based Transactions," in *Proc. ISOC NDSS*, 2018.

[24] M. Spoke and Nuco Engineering Team, "Aion: The Third-Generation Blockchain Network," Whitepaper, 2017. URL: https://aion.network/downloads/aion.network_technical-introduction_en.pdf

[25] Z. Wang and J. Crowcroft, "Quality-of-service routing for supporting multimedia applications," *IEEE J. Sel. Areas Commun.*, vol. 14, no. 7, pp. 1228–1234, 1996.

[26] D. J. Watts and S. H. Strogatz, "Collective Dynamics of Small-World' Networks," *Nature*, vol. 393, no. 6684, pp. 440–442, jun 1998.