The effect of network topology on credit network throughput

The Effect of Network Topology on Credit Network Throughput

VIBHAALAKSHMI SIVARAMAN, Massachusetts Institute of Technology, USA
WEIZHAO TANG, Carnegie Mellon University, USA
SHAILESHH BOJJA VENKATAKRISHNAN, The Ohio State University, USA
GIULIA FANTI, Carnegie Mellon University, USA
MOHAMMAD ALIZADEH, Massachusetts Institute of Technology, USA

Credit networks rely on decentralized, pairwise trust relationships (channels) to exchange money or goods. Credit networks arise naturally in many financial systems, including the recent construct of *payment channel networks* in blockchain systems. An important performance metric for these networks is their transaction throughput. However, predicting the throughput of a credit network is nontrivial. Unlike traditional communication channels, credit channels can become imbalanced; they are unable to support more transactions in a given direction once the credit limit has been reached. This potential for imbalance creates a complex dependency between a network's throughput and its topology, path choices, and the credit balances (state) on every channel. Even worse, certain combinations of these factors can lead the credit network to *deadlocked* states where no transactions can make progress. In this paper, we study the relationship between the throughput of a credit network and its topology and credit state. We show that the presence of deadlocks completely characterizes a network's throughput sensitivity to different credit states. Although we show that identifying deadlocks in an arbitrary topology is NP-hard, we propose a peeling algorithm inspired by decoding algorithms for erasure codes that upper bounds the severity of the deadlock. We use the peeling algorithm as a tool to compare the performance of different topologies as well as to aid in the synthesis of topologies robust to deadlocks.

Additional Key Words and Phrases: Performance Modeling, Payment Channel Networks

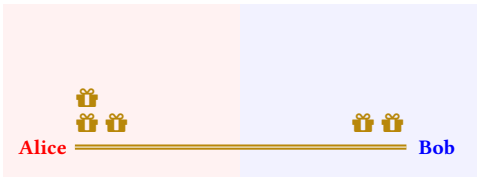
1 INTRODUCTION

The global economy relies on digital transactions between entities who do not trust one another. Today, such transactions are handled by intermediaries who extract fees (e.g., credit card providers). A natural question is how to build financial systems that limit the need for such middlemen.

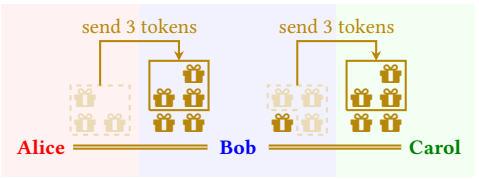
Credit networks (similarly, *debit networks*) are systems in which parties can bootstrap pairwise, distributed trust relations to enable transactions between parties who do not trust each other. The core idea is that even if Alice does not trust Charlie directly, if they both share a pairwise trust relationship with Bob, then Alice and Charlie can execute a credit- (or debit-) based transaction through Bob. The trust relationships that comprise such a network can be based on prior experience or observations[1] (e.g., credit scores in a credit network), or they can be based on escrowed funds that are managed either by a third party or an algorithm (debit networks). Recent debit/credit networks from the blockchain community establish pairwise trust relationships through cryptographically secured data structures stored on a blockchain. Prominent examples include *payment channel networks (PCNs)* [8, 17, 35] such as the Lightning Network [35] and the Ripple credit network [34].

Fig. 1 depicts the operation of a pairwise trust channel (or a *payment channel*) in PCNs. If Alice and Bob want to establish a payment channel, they each cryptographically escrow some number of tokens into a contract that is stored on the blockchain and ensures the money can only be used to transact between them for a predefined time period. In Fig. 1a, Alice commits 3 tokens and Bob commits 2 tokens to a payment channel for a week. While the channel is active, Alice and Bob can

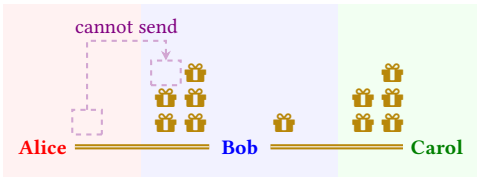
[1]One of the earliest examples is the *hawala* system [29], a credit network in existence since the 8th century, which relies on the past performance of a large network of money brokers.



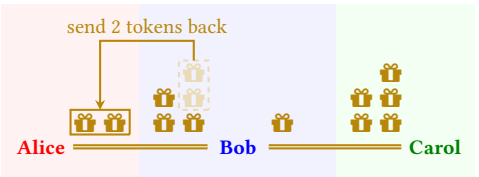
(a) Alice and Bob commit 3 and 2 tokens respectively to a channel for a week.



(b) Alice sends 3 tokens to Carol via Bob securely and privately.



(c) Alice's channel with Bob is *imbalanced* preventing Alice from making any more payments.



(d) Once Bob sends some tokens back to Alice, some of the balance is restored on the channel.

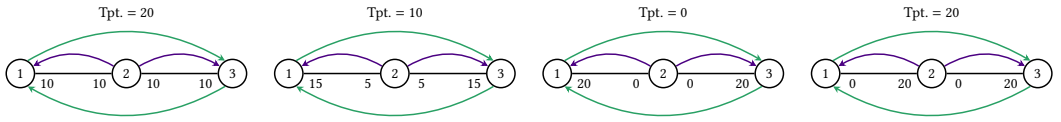
Fig. 1. PCN-style credit network between three parties consisting of two channels through a common intermediary. The network allows each party to transact with all other parties securely and privately without incurring consensus overhead.

exchange funds between themselves without committing to the public ledger. However, once the active period expires (or once either of the participants choose to close the channel), Alice and Bob commit the final state of the channel back to the blockchain. The cryptographic construction of these channels ensures that neither Alice nor Bob can default on their obligation.

PCNs are a network of these pairwise payment channels; if Alice wants to transact with Carol, she can use Bob as a relay (Fig. 1b). We call (Alice, Carol) a demand pair, and the flow of money over route (Alice, Bob, Carol) a *flow* (defined more precisely in Section 3). No party incurs risk of nonpayment because PCNs are implemented as debit networks, with settlement processed on the public blockchain. PCNs are viewed as an important class of techniques for improving the scalability of legacy blockchains with slow consensus mechanisms [50].

Like traditional communication networks, a central performance metric in credit and debit networks is **throughput**: the total number of transactions a credit network can process per unit time. However, reasoning about credit network throughput is more difficult than in traditional communication networks because of *imbalanced channels*. That is, because channels impose upper limits on credit (respectively debit) in either direction, transactions cannot flow indefinitely in one direction over a channel. For instance, in Fig. 1c, once Alice sends 3 tokens to Bob, she cannot send any more tokens to Bob (or Carol). If Bob sends some money back to Alice, she would then again have credit to send new payments to Bob or route payments via Bob (Fig. 1d).

Imbalanced channels can affect the throughput of credit networks in unusual ways that depend on topology, user transaction patterns, and transaction routes. An imbalanced channel can harm throughput in other parts of the network due to dependencies between flows (e.g., an imbalanced channel blocks a flow, which prevents that flow from balancing other channels, blocking more flows and so on). Certain configurations of imbalanced channels can even lead to *deadlocks* where no transactions can flow over certain edges or even the whole network. Recovering from degraded throughput caused by imbalanced channels requires settlement mechanisms outside of the credit network, such as performing "on-chain" transactions on the blockchain to add funds to a channel. These mechanisms incur higher cost and overhead compared to transactions within the credit network and should be avoided.



(a) Perfect Balance ($\boldsymbol{b}_1$) (b) Slight Imbalance ($\boldsymbol{b}_2$) (c) Deadlock ($\boldsymbol{b}_3$) (d) Good Boundary ($\boldsymbol{b}_4$)

Fig. 2. Example illustrating how throughput varies with balance state for the same total collateral

In this work, we study the role of network topology and channel imbalance on the throughput of credit networks. While system designers cannot directly control the topology of a decentralized network, existing PCNs (e.g., Lightning Network) indirectly influence network topology, for example, with "autopilot" systems that recommend new channels to participants based on a peer's channel degree, size, and longevity [9, 10]. These systems currently lack an understanding of how network topology and channel imbalance impacts the throughput in credit networks. Our goal is to bridge this gap, paving the way for future autopilot systems that encourage high-throughput topologies with minimal deadlocks. We make the following contributions:

(1) We present a synchronous round-by-round model to study the throughput of a credit network for a given demand pattern, routing, and channel balance state. We use this model to formulate the best- and worst-case throughput of a credit network as a function of its starting channel balance state as optimization problems. The best-case throughput is easy to compute using a linear program, but determining the worst-case throughput is non-trivial and requires identifying the worst configuration of channel balances. Since transaction patterns and the likelihood of different balance states in credit networks are unknown, we instead study the worst-case throughput in an effort to bound the throughput achievable by a topology.
(2) We introduce the notion of deadlocks — configurations of channel balances that irrevocably prevent transactions over a subset or all channels in the network. We show that deadlocks precisely capture scenarios in which a credit network's throughput is sensitive to the channel balances, i.e., if a topology is deadlock-free, then its throughput does not depend on the initial state of the channels.
(3) We show that the problem of determining whether a given network topology can be deadlocked given a fixed set of demand flows over the topology is NP-hard. Nevertheless, we propose a "peeling algorithm" inspired by decoding algorithms for erasure codes [26] that can be used to bound the number of deadlocked channels in the network. We show empirically that these bounds are very accurate for a variety of topologies, providing a computationally-efficient algorithm to estimate the worst-case throughput.
(4) We compare standard random graph topologies and a subset of the actual Lightning Network topology in terms of best- and worst-case throughput for randomly-sampled transaction demands. We find that different topologies have different benefits. For example, scale-free graphs have fewer deadlocks and achieve better worst-case throughput than random regular and Erdos-Renyi graphs when the network contains fewer flows, but achieve lower throughput when the network is heavily utilized.
(5) We take initial steps towards synthesizing deadlock-resilient topologies by exploiting the surprising connection with erasure codes. In particular, we build on prior approaches for designing efficient LT codes [26, 27] to synthesize topologies with better robustness to deadlocks than existing topologies for a randomized transaction demand model.

2 MOTIVATION

As a motivating example, consider the line topology in Fig. 2 where nodes 1 and 3 have balancing demands to each other, forming a circulation and node 2 has one-directional (DAG) demands going

out to 1 and 3. In steady state, regardless of the balances of the two channels, we only expect the green flows to contribute to throughput because tokens moved by one green flow is restored by the other, permitting more transactions. In Fig. 2a, both channels are perfectly balanced, and nodes 1 and 3 can send at most 10 tokens to each other without changing the balances on either channel. If we defined a *throughput* metric for the maximum transaction amount possible in a *round* when send as many tokens as available without changing channel balances at the end of the round, the throughput in the perfect balance state would be 20 (each green flow moves 10 tokens).

However, if the purple DAG flows were active for a short amount of time in Fig. 2a, causing 5 tokens on each channel to move to nodes 1 and 3, we would end up in Fig. 2b. Once in Fig. 2b, no amount of future flows between the shown sender-receiver pairs can ever restore the credit network to state Fig. 2a. In other words, Fig. 2a is *not reachable* from Fig. 2b. In particular, node 1 cannot send more than 5 tokens in a given round to node 3 since it needs funds on the $2 \rightarrow 3$ channel and vice-versa. Sending exactly 5 tokens in a round from each end (node 1 to node 3 and vice versa) leaves us back at Fig. 2b without any ability to get to Fig. 2a. Consequently, the steady-state throughput in Fig. 2b is 10 instead of 20 in Fig. 2a.

Worse still, once in Fig. 2b, if the DAG flows are active for longer, causing the remaining 5 tokens on node 2 on each channel to move outwards, we end up in Fig. 2c. Notice that Fig. 2c is a *deadlock*; none of the four flows can make any progress. Consequently, there is no way to get out of Fig. 2c to any of the other *balance states* for the credit network. In other words, the steady-state throughput for Fig. 2c is 0, and the only state reachable from Fig. 2c is itself.

The implications of Fig. 2 are concerning: for the same amount of escrowed funds in every channel in a credit network, the throughput can vary depending on the starting state. In other words, the throughput of the credit network in Fig. 2 is *sensitive* to the balance state. When every channel is perfectly balanced, the transaction throughput is highest. If the network stayed in in such a state and only routed circulation flows, the throughput would remain high [45]. However, transient DAG flows can alter the credit network state in ways that it cannot recover from, permanently harming its throughput. Better routing or rate control cannot eliminate this problem. While a scheme like Spider [45] may delay the onset of deadlocks by throttling flows experiencing congestion or more effective load-balancing across multiple paths, DAG demands like the purple flows in Fig. 2 would eventually move the topology to states with reduced or even zero throughput. We formalize the notions of throughput, balance states, and deadlocks in §3 and §4.

3 MODEL AND METRICS

We model the credit network as a graph $G(E, V)$. The vertices $V = \{v_1, v_2, \ldots, v_n\}$ of the graph denote the nodes, or users, of the credit network while the edges E denote pairwise channels. Each channel is an undirected edge and is denoted by the pair of nodes constituting the edge. Physically, these channels represent a credit relation between the endpoints. Each channel $\{u, v\}$ is associated with two *balances* $b_{(u,v)}$ and $b_{(v,u)}$ that denote the maximum number of tokens that u is willing to credit v ($b_{(u,v)}$) and vice versa ($b_{(v,u)}$) in the channel. These can be thought of as escrowed funds that are allocated to each channel when it is established. Each channel $\{u, v\} \in E$ has a *capacity* $c_{\{u,v\}}$ that denotes the total credit limit of the channel;[2] that is, $c_{\{u,v\}} = b_{(u,v)} + b_{(v,u)}$. We assume a static network, i.e., all channel capacities are fixed, and the channels remain open and funded with the same amount of tokens for the entire duration under consideration.

We capture the current *state* of the credit network using a balance vector $\boldsymbol{b} \in \mathbb{R}_{\geq 0}^{|E|}$, containing the balance information of all the $|E|$ channels. For every channel $\{u, v\} \in E$, the vector $\boldsymbol{b}$ contains only one entry corresponding to the balance at either u or v's end in the channel as follows.

[2]We use the notation $\{u, v\}$ to denote undirected edges, and (u, v) to denote directed edges for nodes $u, v \in V$.

Let $\overline{E} = ((v_{i_1}, v_{j_1}), (v_{i_2}, v_{j_2}), \ldots, (v_{i_{|E|}}, v_{j_{|E|}}))$ be an ordering of the channels in E, with the nodes within each channel also ordered such that $i_k < j_k$ for any channel $1 \leq k \leq |E|$. Similarly let $\underline{E} = ((v_{j_1}, v_{i_1}), (v_{j_2}, v_{i_2}), \ldots, (v_{j_{|E|}}, v_{i_{|E|}}))$ be an ordering containing edges in a direction reverse to that in $\overline{E}$. Now, the k-th entry of $\boldsymbol{b}$ for $1 \leq k \leq |E|$ contains the balance $b_{(v_{i_k}, v_{j_k})}$ of the k-th channel in $\overline{E}$. This allows us to visualize the entire network state space $\mathbb{B}$ as an $|E|$-dimensional polytope with each perpendicular axis representing balance on a distinct channel. Similar to the balance vector, let the vector $\boldsymbol{C} \in \mathbb{R}_{>0}^{|E|}$ contain the capacities of all the channels, with the k-th entry of $\boldsymbol{C}$ containing the total capacity $c_{\{v_{i_k}, v_{j_k}\}}$ of the k-th channel in $\overline{E}$. These definitions allow us to represent the balances on the reverse edges or those channels in $\underline{E}$ using the vector $\boldsymbol{C} - \boldsymbol{b}$.

Definition 1. *(Boundary, Corner, and Interior Balance States.) A channel $\{u, v\} \in E$ is imbalanced when all of its escrowed funds are on one end of the channel,* i.e., *either $b_{(u,v)} = 0$ or $b_{(v,u)} = 0$. A balance state $\boldsymbol{b}$ is located on the boundary of the $\mathbb{B}$ polytope when one or more of the channels is imbalanced. A corner of the $\mathbb{B}$ polytope is a point on the boundary where all $|E|$ channels are imbalanced in one of the two directions. In contrast, if none of the balances on either end of any of the channels is 0, such a balance state $\boldsymbol{b}$ is an* interior *balance state. The set of all interior balance states is denoted by $\mathbb{B}_+$. Similarly, $\mathbb{B}_{boundary}$ denotes the set of all boundary balance states, while $\mathbb{B}_{corner} \subset \mathbb{B}_{boundary}$ denotes the set of all corner states.*

Transactions attempted in the credit network follow a demand pattern captured through a binary matrix D where $d_{(i,j)} = 1$ if sender i wants to transact with receiver j. Such a sender-receiver pair can use one or more paths to transact. Let $\mathcal{P} = (p_1, p_2, \ldots, p_l)$ denote an ordering of all paths (across all sender-receiver pairs with demand) in the network through which tokens can be routed.

We assume that transactions in the credit network occur synchronously over *rounds* or epochs; every transaction starts and completes within the same round. We further assume that a transacting sender-receiver pair has unbounded demand; tokens can always flow from a sender to a receiver if feasible. We also assume that tokens are infinitely divisible. Suppose $\boldsymbol{b}$ is the balance state of the credit network at the beginning of a round. Let $\boldsymbol{f} \in \mathbb{R}_{\geq 0}^{|\mathcal{P}|}$ denote a flow vector, with the k-th entry of $\boldsymbol{f}$ being the number of tokens sent along path p_k during the round. For the flow vector $\boldsymbol{f}$ to be feasible, we require the total number of tokens sent on any channel $\{u, v\}$ from u towards v to not exceed $b_{(u,v)}$, since only $b_{(u,v)}$ tokens are available for use along channel $\{u, v\}$ (in u to v direction) during the round. To define feasibility of a flow precisely, let routing matrix $\overline{R} \in \{0, 1\}^{|E| \times |\mathcal{P}|}$ be such that $R_{((u,v),p)} = 1$ if the directed edge (u, v) is part of path p and $R_{((u,v),p)} = 0$ otherwise. The k-th column ($1 \leq k \leq |\mathcal{P}|$) of R corresponds to the k-th path p_k while the k-th row ($1 \leq k \leq |E|$) corresponds to the k-th directed edge in $\overline{E}$. We similarly define routing matrix $\underline{R} \in \{0, 1\}^{|E| \times |\mathcal{P}|}$ over the directed edges in $\underline{E}$. These routing matrices essentially capture which edges are involved in which paths. With this notation, for a flow $\boldsymbol{f}$ to be feasible we must have

$$\overline{R}\boldsymbol{f} \leq \boldsymbol{b} \text{ and } \underline{R}\boldsymbol{f} \leq \boldsymbol{C} - \boldsymbol{b}. \tag{1}$$

When such a feasible flow is sent, at the end of the round a node u's balance $b_{(u,v)}$ in channel $\{u, v\}$ is decremented by the total amount of tokens sent from u towards v and incremented by the tokens sent from v towards u during the round. To compute this balance change, we define $\Delta R \in \{1, -1, 0\}^{|E| \times |\mathcal{P}|}$ as $\Delta R := \overline{R} - \underline{R}$. For an edge $(u, v) \in \overline{E}$ and path p, we have

$$\Delta R_{(u,v),p} = \begin{cases} 1, & \text{if } (u, v) \in p \\ -1, & \text{if } (v, u) \in p \\ 0, & \text{otherwise.} \end{cases} \tag{2}$$

Letting $\boldsymbol{b}'$ be the balance state of the network at the end of the round, we then have

$$\boldsymbol{b}' = \boldsymbol{b} - \Delta R \boldsymbol{f}. \tag{3}$$

Tab. 1 summarizes the notation and provides an example for the credit network state described in Fig. 2b. The flow $\boldsymbol{f}$ sends 3 tokens from node 1 to 3, and 2 tokens from node 2 to 3, transferring all of the 5 available tokens on the $(2, 3)$ channel. Sending such a feasible flow results in the new balance state $\boldsymbol{b}'$ with no remaining tokens at 2's end of the $(2, 3)$ channel.

This process of token transfer allows us to define a set of *reachable states* from a starting state $\boldsymbol{b}$. We say a state $\boldsymbol{b}'$ is reachable from $\boldsymbol{b}$ in $k+1$ rounds if there exists a sequence of flows $\boldsymbol{f}^{(0)}, \boldsymbol{f}^{(1)}, \ldots, \boldsymbol{f}^{(k)}$ for $k \in \mathbb{N}$ such that

$$\boldsymbol{f}^{(i)} \geq 0 \quad \forall i \in [k], \tag{4}$$

$$\overline{R}\boldsymbol{f}^{(i)} \leq \boldsymbol{b}^{(i)} \quad \forall i \in [k], \tag{5}$$

$$\underline{R}\boldsymbol{f}^{(i)} \leq \boldsymbol{C} - \boldsymbol{b}^{(i)} \quad \forall i \in [k], \tag{6}$$

$$\boldsymbol{b}^{(i+1)} = \boldsymbol{b}^{(i)} - \Delta R \boldsymbol{f}^{(i)} \quad \forall i \in [k], \tag{7}$$

with $\boldsymbol{b}^{(0)} = \boldsymbol{b}$ and $\boldsymbol{b}^{(k+1)} = \boldsymbol{b}'$. A sequence of flows $\boldsymbol{f}^{(0)}, \boldsymbol{f}^{(1)}, \ldots, \boldsymbol{f}^{(k)}$, that satisfies equations (4)–(7) for a given initial state $\boldsymbol{b}^{(0)} = \boldsymbol{b}$ is called a feasible flow sequence for initial state $\boldsymbol{b}$.

Definition 2. *(Set of reachable states.) For a credit network with capacity $\boldsymbol{C}$ and initial balance state $\boldsymbol{b}$, the set of states $\mathbb{X}_{\boldsymbol{b}}$ reachable from $\boldsymbol{b}$ is given by*

$$\mathbb{X}_{\boldsymbol{b}} = \{\boldsymbol{b}' | \boldsymbol{b}' \text{ is reachable from } \boldsymbol{b}\}. \tag{8}$$

We say a flow $\boldsymbol{f}$ is *achievable* from state $\boldsymbol{b}$ if there exists a feasible flow sequence $\boldsymbol{f}^{(0)}, \boldsymbol{f}^{(1)}, \ldots, \boldsymbol{f}^{(k)}$ from $\boldsymbol{b}$ such that $\boldsymbol{f} = \sum_{i=0}^{k} \boldsymbol{f}^{(i)}$.

Definition 3. *(Set of achievable flows.) The set of flows $\mathbb{F}_{\boldsymbol{b}}$ achievable from state $\boldsymbol{b}$ is given by*

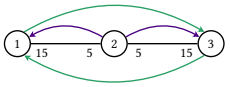
$$\mathbb{F}_{\boldsymbol{b}} = \{\boldsymbol{f} | \boldsymbol{f} \text{ is achievable from } \boldsymbol{b}\}. \tag{9}$$

3.1 Metrics: Best- and Worst-Case Throughput

We measure the performance of a credit network by its throughput. However, the throughput of a credit network depends on many factors, including the demand imposed by users. Prior work [45] has shown that a demand matrix can be separated into circulation and directed acyclic graph (DAG) components.[3] The circulation represents the portion of the demand that can be routed in a balanced manner and sustained in the long term. The circulation can be extracted from the demand matrix by computing the largest union of all cyclic demand pairs (a has demand to b, b has demand to c, who has demand back to a). The remaining demand pairs form the DAG component. With a circulation demand, transactions can be routed such that every token sent on channel $\{u, v\}$ from u to v is compensated by another token from v to u, thus maintaining channel balance. This allows a circulation demand to be sustained indefinitely. In contrast, a DAG demand sends one-way traffic on some channels, eventually leaving them imbalanced and unable to sustain more transactions in the same direction. Thus, the DAG portion of the demand cannot be sustained long-term without rebalancing. Rebalancing is a technique

[3]The demand matrix considered in prior work [45] is real-valued with the entries denoting how much demand sender-receiver pairs have. In contrast, the demand matrix we consider is binary-valued (§3) denoting only the presence or absence of demand between sender-receiver pairs. Nevertheless, the concept of decomposing a demand matrix into circulation and DAG components is general and applicable to our model also.

to arbitrarily modify the balance of a channel using a mechanism external to the credit network itself (e.g., "on-chain" transactions on a blockchain). Rebalancing usually comes at a substantial cost (high transaction fees, confirmation delay) and should be avoided to the extent possible.



Symbol	Meaning	Example
$\boldsymbol{b}$	Initial balance state	$\begin{bmatrix}15\\5\end{bmatrix}$
$\boldsymbol{C}$	Capacities	$\begin{bmatrix}20\\20\end{bmatrix}$
$\mathcal{P}$	Set of paths	$(1\to3, 3\to1,$ $2\to3, 2\to1)$
$\overline{R}$	Fwd. routing matrix	$\begin{bmatrix}1&0&0&0\\1&0&1&0\end{bmatrix}$
$\underline{R}$	Bwd. routing matrix	$\begin{bmatrix}0&1&0&1\\0&1&0&0\end{bmatrix}$
ΔR	$\overline{R}-\underline{R}$	$\begin{bmatrix}1&-1&0&-1\\1&-1&1&0\end{bmatrix}$
$\boldsymbol{f}$	Feasible flow	$\begin{bmatrix}3&0&2&0\end{bmatrix}^T$
$\boldsymbol{b}'$	Balance state after sending $\boldsymbol{f}$	$\begin{bmatrix}12\\0\end{bmatrix}$
$\psi(\boldsymbol{b})$	One-step tpt.	10
$\phi(\boldsymbol{b})$	Steady-state tpt.	10
$\Phi_{\max}$	Max. tpt.	20
$\Phi_{\min}$	Min. tpt.	0

Table 1. Summary of notations along with examples for above credit network.

Our goal is to study the throughput of a credit network in the absence of external rebalancing. Consequently our throughput metrics automatically attribute throughput only to the circulations (the long-term throughput of DAG demands is zero without rebalancing). However, we emphasize that our model is general and supports DAG demands. In particular, transient DAG demands transition the credit network to different states, possibly changing its long-term (circulation) throughput. In fact, even circulation demands can appear "DAG-like" over short time scales. We study the impact of such state transitions on the long-term throughput of the credit network.

We define the one-step throughput $\psi(\boldsymbol{b})$ of a state $\boldsymbol{b}$ to be the maximum flow that is feasible to send within an epoch starting from $\boldsymbol{b}$, while ensuring that the balances of all channels remain unchanged after the epoch. Since the balance state does not change, $\psi(\boldsymbol{b})$ can be achieved in every epoch by repeating the same set of flows.

Definition 4. *(One-step throughput.) The one-step throughput at state $\boldsymbol{b}$ is defined as*

$$\psi(\boldsymbol{b}) = \max\left\{1^\top \boldsymbol{f} \middle| \boldsymbol{f} \geq 0,\ \overline{R}\boldsymbol{f} \leq \boldsymbol{b},\ \underline{R}\boldsymbol{f} \leq \boldsymbol{C} - \boldsymbol{b},\ \Delta R\boldsymbol{f} = 0\right\}. \tag{10}$$

The one-step throughput is maximized when the channel capacity is equally divided between constituent nodes for all channels, *i.e.*, $\psi\left(\boldsymbol{C}/2\right) = \max_{\boldsymbol{b}\in\mathbb{B}} \psi(\boldsymbol{b})$.

We are interested in the *maximum steady-state* throughput of the network starting from a state $\boldsymbol{b}$. This depends not only on $\boldsymbol{b}$ but also on the states reachable from $\boldsymbol{b}$. In particular, a credit network starting from state $\boldsymbol{b}$ can undergo a series of state transitions facilitated by flows present in the demand to reach a state $\boldsymbol{b}'$ with higher $\psi(\boldsymbol{b}')$. It can then achieve throughput $\psi(\boldsymbol{b}')$ in every subsequent epoch. This motivates the following definition.

Definition 5. *(Steady-state throughput.) The steady-state throughput of a state $\boldsymbol{b}$ is defined as*

$$\phi(\boldsymbol{b}) = \sup_{\boldsymbol{a}\in\mathbb{X}_{\boldsymbol{b}}} \psi(\boldsymbol{a}). \tag{11}$$

In Fig. 2b, the balance on the middle node cannot be increased unilaterally because the green flows that contribute to steady-state throughput are tied to each other. As a result, the maximum throughput is constrained by the 5 available tokens on each side of node 2. This means that $\psi(\boldsymbol{b}) = \phi(\boldsymbol{b}) = 10$ because no state with higher throughput is reachable from $\boldsymbol{b}$ (Tab. 1) [4].

[4]By Definition 4, $\psi(\boldsymbol{b})$ is the optimal value of a linear programming problem with a compact feasible set. Because the feasible set is related to $\boldsymbol{b}$ in a continuous way, the resulting $\psi(\boldsymbol{b})$ is always a continuous function. In contrast, ϕ is typically

Without loss of generality, we use *throughput* of a state to refer to the steady-state throughput ϕ of a state in the rest of this paper. Note that the maximum throughput $\Phi_{\max}$ across all states is also attained at $\boldsymbol{C}/2$: $\Phi_{\max} \triangleq \max_{\boldsymbol{b}\in\mathbb{B}} \phi(\boldsymbol{b}) = \psi\left(\boldsymbol{C}/2\right)$.

To account for the sensitivity of a credit network's throughput to the balance state, we propose analyzing the lowest throughput it would yield across all possible balance states $\mathbb{B}$. We call this lowest throughput value **worst-case throughput** and denote it by $\Phi_{\min}$. Unlike average-case analysis, considering the worst case lets us compare the resilience of different topologies without making assumptions about the distribution of balance states. Since channel balances are private information, little is known about the balance states observed in credit networks in practice.

Definition 6. *(Worst-case Throughput) The worst-case throughput of a credit network with capacity $\boldsymbol{C}$ and a balance space $\mathbb{B}$ are defined as*

$$\Phi_{\min} = \min_{\boldsymbol{a}\in\mathbb{B}} \phi(\boldsymbol{a}) \tag{12}$$

We denote its counter-part $\Phi_{\max}$ as the **best-case throughput** achieved at $\boldsymbol{b} = \boldsymbol{C}/2$. The credit network in Fig. 2 achieves $\Phi_{\max} = 20$ at perfect balance (Fig. 2a) and $\Phi_{\min} = 0$ at Fig. 2c (Tab. 1).

4 THROUGHPUT SENSITIVITY AND DEADLOCKS

Best- and worst-case throughput depend on the transaction demand pattern. But since the transaction patterns are unknown a priori, our goal is to design credit network topologies such that $\Phi_{\min}$ is close to $\Phi_{\max}$ for most or all transaction patterns. It is important to ensure that such a design does not compromise on throughput. In other words, it is not desirable to have $\Phi_{\min} = \Phi_{\max}$ in exchange for a very low $\Phi_{\max}$ relative to the escrowed collateral.

Verifying conditions in which $\Phi_{\min} = \Phi_{\max}$ is difficult, because although we know that $\Phi_{\max}$ is attained at $\boldsymbol{b} = \boldsymbol{C}/2$, it is unclear *a priori* which states achieve $\Phi_{\min}$. In this section, we show that if a topology does not admit *deadlocks*, then it is always the case that $\Phi_{\min} = \Phi_{\max}$. We call such a topology *insensitive* to the starting balance.

A channel is said to be deadlocked at a particular state $\boldsymbol{b}$ if no feasible flow can alter its balance in either direction.[5] The collection of all such channels forms the deadlocked channel set $\mathbb{L}_{\boldsymbol{b}}$ for a state $\boldsymbol{b}$. A state $\boldsymbol{b}$ represents a deadlock if $\mathbb{L}_{\boldsymbol{b}} \neq \varnothing$.

Definition 7. *(Deadlocked Channel Set) Let $\mathbb{L}_{\boldsymbol{b}}$ denote the set of deadlocked channels under balance state $\boldsymbol{b}$. Formally,*

$$\mathbb{L}_{\boldsymbol{b}} = \left\{(u,v) \in \overline{E} \;\middle|\; [\Delta R\boldsymbol{f}]_{(u,v)} = 0,\ \forall \boldsymbol{f} \in \mathbb{F}_{\boldsymbol{b}}\right\},$$

where $[\Delta R\boldsymbol{f}]_{(u,v)}$ denotes the entry corresponding to channel (u,v) in the vector $\Delta R\boldsymbol{f}$.

Since no feasible flow at $\boldsymbol{b}$ can alter the balance values of the deadlocked channels $\mathbb{L}_{\boldsymbol{b}}$, all such channels have the same balance values across all reachable states from $\boldsymbol{b}$. Formally, if $(u,v) \in \mathbb{L}_{\boldsymbol{b}}$ and $\boldsymbol{a} \in \mathbb{X}_{\boldsymbol{b}}$ is a state reachable from $\boldsymbol{b}$, then we must have $a_{(u,v)} = b_{(u,v)}$, where $a_{(u,v)}$ and $b_{(u,v)}$ denote the balances of channel (u,v) in vectors $\boldsymbol{a}$ and $\boldsymbol{b}$ respectively. This effectively means that the deadlocked channels cannot sustain any flow and thus, make no contributions towards throughput.

discontinuous, because the sets of reachable states $\mathbb{X}_{\boldsymbol{b}}$ and $\mathbb{X}_{\boldsymbol{b}'}$ may differ drastically from each other, even if $\boldsymbol{b}$ and $\boldsymbol{b}'$ are arbitrarily close (particularly in the neighborhood of the boundaries of $\mathbb{B}$). Since we haven't established the closedness of the set of reachable states $\mathbb{X}_{\boldsymbol{b}}$, we use a supremum definition of steady-state throughput instead of a maximum.

[5]Notice a subtle difference between our definition of deadlocks and that in [31, 52]. They define a deadlock as a set of flows that starts executing during a round, but cannot complete because (a) different flows utilize the same channels, and (b) a poorly-chosen processing ordering for the path edges for different flows blocks all flows from making progress. Our model does not capture such deadlocks because we assume flows complete instantaneously within a round.

In other words, $f_p = 0$ for all $\boldsymbol{f} \in \mathbb{F}_{\boldsymbol{b}}$ if there exists an $(u, v) \in \mathbb{L}_{\boldsymbol{b}}$ such that $(u, v) \in p$ or $(v, u) \in p$. We formally state and prove these properties in App. A.2.

It is straightforward to verify that for every interior point $\boldsymbol{b} \in \mathbb{B}_+$, there exists $\epsilon > 0$ such that ϵ tokens can be transmitted over *some* flow in the network. Hence, for any state $\boldsymbol{b} \in \mathbb{B}_+$, $\mathbb{L}_{\boldsymbol{b}} = \varnothing$; in other words, deadlocks can *only* happen at the boundary points of $\mathbb{B}$. At every boundary point, a nonempty subset of channels is imbalanced in one of the two directions, so flows using those channels in the imbalanced direction(s) are blocked.

However, merely having imbalanced channels at a boundary point of polytope $\mathbb{B}$ does not imply the existence of deadlocks at that point. Consider the two corner balance states $\boldsymbol{b}_3$ and $\boldsymbol{b}_4$ shown in Fig. 2c and Fig. 2d respectively. $\boldsymbol{b}_3$ is a deadlock in that none of the four flows can make any progress, no other state in $\mathbb{B}$ is reachable from $\boldsymbol{b}_3$, and $\phi(\boldsymbol{b}_3) = 0$. In contrast, $\boldsymbol{b}_4$ is a good boundary point that can achieve the best-case throughput $\Phi_{\max}$. For example, node 3 can send 20 tokens to node 1 in one round, node 1 can then send the 20 tokens back to node 3 in the next round, and so on. The credit network returns to $\boldsymbol{b}_4$ every two rounds, achieving a throughput of 20 tokens per round. Further, at $\boldsymbol{b}_4$, sending 10 tokens from node 3 to 1 would return the credit network to the perfectly balanced state $\boldsymbol{b}_1$; thus $\phi(\boldsymbol{b}_4) = \psi(\boldsymbol{b}_1) = 20$. Effectively, the credit network can escape boundary $\boldsymbol{b}_4$ to reach any state in $\mathbb{B}$.

It turns out that to assess a topology's sensitivity to the starting balance state, it is enough to check if its boundary points can be deadlocked. Intuitively, if there are no deadlocks then every boundary point, and in particular every corner point of the polytope, can be made balanced by moving into an interior point. Since the set of directions along which the corner points can transition to interior points spans the entire state space, this implies one can move in every direction in the $\mathbb{B}$ polytope. As a result, the credit network must be able to transition from any balance state to any other balance state without being stuck. Such a credit network can always reach its throughput-maximizing state $\boldsymbol{C}/2$ from any other state. In other words, the throughput of such a topology should be *insensitive* to the starting balance state. In contrast, if a credit network has deadlocks, the *largest* deadlock (i.e., the deadlock involving the most channels) should lead to the maximum unused collateral and such states would have the lowest throughput. Therefore, the maximum throughput that can be extracted using only the remaining channels in the largest deadlock state constitutes the worst-case throughput of a credit network. The rest of this section formally describes these two results.

Theorem 1. *(Insensitive Throughput) If a credit network with balance state space $\mathbb{B}$ is deadlock-free, i.e., for any boundary state $\boldsymbol{c} \in \mathbb{B}_{boundary}$, there exists an interior point $\boldsymbol{p} \in \mathbb{B}_+$ such that $\boldsymbol{p} \in \mathbb{X}_{\boldsymbol{c}}$, then for any state $\boldsymbol{b}$, $\phi(\boldsymbol{b}) = \psi(\boldsymbol{C}/2)$.*

Proof Sketch (Full proof in App. A.3). We introduce the notion of **feasible directions** from a balance state. A unit vector $\boldsymbol{d}$ is a feasible direction from $\boldsymbol{b}$ if there exists $\epsilon > 0$ such that $\boldsymbol{b} + \epsilon\boldsymbol{d} \in \mathbb{X}_{\boldsymbol{b}}$. We list some useful properties about feasible directions below, and provide proofs in App. A.2.

(1) The union of feasible directions of any state is convex. (Prop. 5)
(2) Any interior state along a feasible direction of $\boldsymbol{b}$ is reachable. (Theorem 13)
(3) Feasible directions of a corner state are also feasible for an *arbitrary* interior state. (Lemma 15)

Let $\boldsymbol{a} \in \mathbb{B}_+$ be an arbitrary interior state. By assumption, every corner state has a corresponding reachable interior state. The transition from a given corner state to its reachable interior state produces a set of feasible directions which is located in an open orthant of the $\mathbb{R}^{|E|}$ space. Further, the feasible direction corresponding to a particular corner occupies a distinct orthant that is not shared by the feasible directions from any other corner state.

By property 3, the feasible directions of every corner are included in the the feasible directions of $\boldsymbol{a}$. By property 1, the set of $\boldsymbol{a}$'s feasible directions contains the convex hull of all of these $2^{|E|}$

sets of directions, which maps exactly to the full space $\mathbb{R}^{|E|}$. Furthermore, by property 2, any other interior state $\boldsymbol{a}'$ is reachable from $\boldsymbol{a}$, including the center state $\boldsymbol{C}/2$. For simplicity, we skip the extension to boundary states and refer it to App. A.3. Since $\boldsymbol{C}/2$ globally maximizes the one-step throughput ψ, we have for any $\boldsymbol{b} \in \mathbb{B}$, $\phi(\boldsymbol{b}) = \sup_{\boldsymbol{b}' \in \mathbb{X}_{\boldsymbol{b}}} \psi(\boldsymbol{b}') = \psi(\boldsymbol{C}/2)$. ∎

While a credit network might not be entirely deadlock-free, Theorem 1 can also be applied to any sub-network of the original credit network that does not have deadlocks. A sub-network is embedded within a subgraph of the original topology and only permits paths whose edges are entirely within the subgraph. If this sub-network is deadlock-free, then it enjoys throughput insensitivity within the balance states of the sub-network. In addition, its throughput lower bounds the maximum throughput achievable from an arbitrary initial state in the original credit network.

At the same time, every credit network with deadlocks has a set $\boldsymbol{B}_{\text{worst}}$ of balance states that maximizes the number of deadlocked channels. For any $\boldsymbol{b} \in \boldsymbol{B}_{\text{worst}}$, the set of channels deadlocked $\mathbb{L}_{\boldsymbol{b}}$ equals a fixed set L that contains all other sets of deadlocked channels across every balance state $\boldsymbol{b} \in \mathbb{B}$. The channels outside L form a deadlock-free sub-network. The throughput of the original credit network is at least the throughput achievable on this sub-network, regardless of the initial balance state of the original network. We state this formally in Theorem. 2 and prove it in App. A.4.

Theorem 2. *A state $\boldsymbol{b}$ has the worst throughput $\phi(\boldsymbol{b}) = \Phi_{\min}$, if the number of deadlocked channels in $\boldsymbol{b}$ is the largest across all states,* i.e., $|\mathbb{L}_{\boldsymbol{b}}| = \max_{\boldsymbol{b}' \in \mathbb{B}} |\mathbb{L}_{\boldsymbol{b}'}|$. *Furthermore, the throughput $\Phi_{\min}$ of a worst throughput state $\boldsymbol{b}$ can be computed by considering a state $\boldsymbol{b}'$ where*

$$b'_{(u,v)} = \begin{cases} b_{(u,v)}, & \text{if } (u,v) \in \mathbb{L}_{\boldsymbol{b}} \\ C_{(u,v)}/2, & \text{if } (u,v) \notin \mathbb{L}_{\boldsymbol{b}} \end{cases}.$$

Then, $\Phi_{\min} = \psi(\boldsymbol{b}')$.

5 DESIGNING DEADLOCK-FREE TOPOLOGIES

§4 suggests that to fully utilize the collateral in a credit network, the credit network needs to be deadlock-free. In this section, we first show that determining whether an arbitrary topology is deadlock-free is NP-hard (§5.1). Next, we propose and analyze a "peeling algorithm" that bounds the number of deadlock-free edges in a credit network (§5.2). The peeling algorithm provides a computationally-efficient way to estimate $\Phi_{\min}$, and reveals a surprising connection between designing deadlock-free topologies and LT Codes [26, 36], a well-known class of erasure codes.

5.1 Deadlock Detection Problem

Detecting a deadlock on a credit network is equivalent to finding a balance configuration $\boldsymbol{b}$ such that one or more channels is imbalanced and one or more of the imbalanced channels is deadlocked. A full deadlock involves all edges of the credit network while a partial deadlock can involve any non-empty *proper* subset of credit network edges.

Definition 8. *(Full and Partial Deadlock) A balance state $\boldsymbol{b}$ is a full deadlock on the credit network $G(E,V)$ if $|\mathbb{L}_{\boldsymbol{b}}| = |E|$, and a partial deadlock if $0 < |\mathbb{L}_{\boldsymbol{b}}| < |E|$.*

Theorem 3. *(Hardness Result) Given a credit network $G(E,V)$ with channel capacity $c_{\{u,v\}} > 0$ for each $\{u,v\} \in E$, demand matrix D and set of paths $\mathcal{P}$ for satisfying the demand, finding a balance state $\boldsymbol{b}$ such that $\mathbb{L}_{\boldsymbol{b}} \neq \emptyset$ is NP-hard.*

Proof Sketch (Full proof in App. A.5). When a channel $\{u,v\} \in E$ is imbalanced with $b_{(u,v)} = 0$, the channel cannot support flow in the (u,v) direction unless the balance state is altered. As a result, all flows using $\{u,v\}$ from u to v are blocked. If all flows using a channel in both directions

are blocked, then the channel is deadlocked. We use this intuition to present a reduction from the Boolean Satisfiability Problem (SAT) to the full deadlock-detection problem for credit networks. We consider a boolean expression in Conjunctive Normal Form (CNF) and construct a credit network such that, each variable is mapped to a unique channel and each clause is mapped to a unique path in the constructed network. The truth value of a literal is associated with whether or not the balance at a particular end of a channel is zero, while the truth value of a clause relates to whether or not a particular path is blocked by some channel. The boolean expression is therefore satisfiable if and only if there exists a full deadlock in the credit network, proving the *NP*-hardness claim.

Note that a straightforward mapping of variables to channels, and clauses to paths results in a disconnected network with invalid paths. To make our constructing valid, we introduce a polynomial number of auxiliary variables (and accompanying clauses), in a way that preserves the connection between the satisfiability of the initial CNF and the existence of deadlocks in the constructed network. We defer more details about our reduction technique to App. A.5. Our reduction can be easily extended to establish that detecting a partial deadlock is also *NP*-hard.

5.2 Peeling Algorithm

Although it is NP-hard to check if a topology is deadlock-free, it is possible to identify subsets of edges that are provably deadlock-free in polynomial time. We design such an algorithm inspired by the peeling algorithm used to decode LT codes [26]. The key insight of our peeling algorithm is the following: if an edge has a dedicated flow in either direction, tokens can move freely in the direction of the flow, as long as there are tokens available at the origin end. We call such flows "length 1" flows, since they traverse a single edge. If an edge has *two* such flows in both directions, it can never be deadlocked. The peeling algorithms progressively finds edges with length 1 flows (which cannot be deadlocked), and eliminates them as potential bottlenecks for all flows that traverse them; this can be viewed as "peeling" these edges from these flows. Once it is established that none of the edges in a flow can be deadlocked, the flow can be removed from consideration. This process continues until all edges in the topology can move in either direction, or none of remaining flows traverses edges that can be peeled, causing the process to terminate unsuccessfully. We describe the terminology borrowed from LT Codes in §5.2.1 and define the deadlock peeling algorithm in §5.2.2.

5.2.1 LT Process. LT codes are a *rateless* code, designed for channel coding under changing or unknown channel conditions [26]. LT codes map a sequence of input bits $\boldsymbol{x} = (x_1, \ldots, x_n)$ to a sequence of encoded bits $\boldsymbol{y} = (y_1, \ldots, y_m)$. Each encoded bit y_j is the XOR of a subset of input bits: $y_j = \oplus_{k \in S_j} x_k$, where set $S_j \subseteq [n]$ indexes a subset of input symbols, chosen at random. The LT encoding procedure can be represented as a bipartite graph, where each input symbol x_i, $i \in [n]$ corresponds to a node on the right, and each encoded symbol y_j, $j \in [m]$ to a node on the left. An edge exists between input node x_i and encoded node y_j iff $x_i \in S_j$. The degree of an encoded node y_j denotes the number of XORed input symbols, $|S_j|$.

Decoding the encoded symbols proceeds iteratively. A degree 1 encoded symbol can be immediately decoded to identify the associated input symbol since the encoded symbol's value matches the input symbol. Such a degree 1 encoded symbol is *released* to *cover* or decode its associated input symbol. The input symbol is then added (if not already present) to the *ripple* $\mathcal{R}$, a set of covered input symbols that are yet to be *processed*. At every time step, an (unprocessed) input symbol x_i is chosen at random from the ripple $\mathcal{R}$ and processed: its value x_i is XORed with every encoded symbol y_j that it is a neighbor of (i.e., for which $i \in S_j$). This reduces the degree of every such encoded symbol, subsequently releasing more encoded symbols that become degree 1. However, the newly covered neighbors of degree 1 flows increase the size of the ripple only if they were previously *uncovered*. This routine (*LT process*) of releasing encoded symbols, covering input symbols and

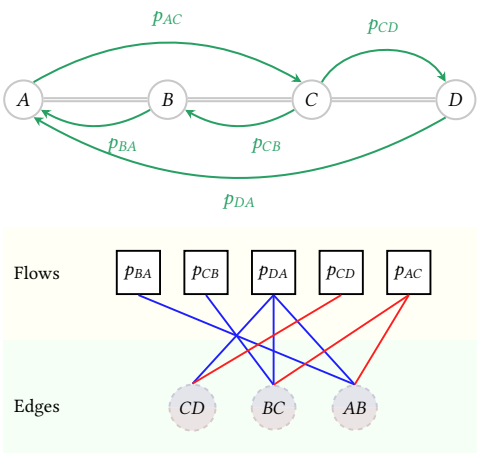


Fig. 3. Bipartite graph that is peelable and identifies the lack of deadlock in the topology

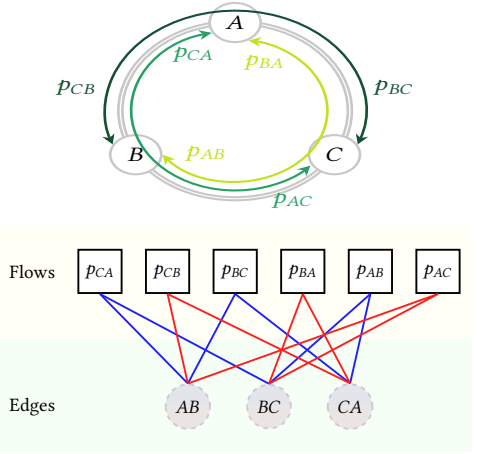


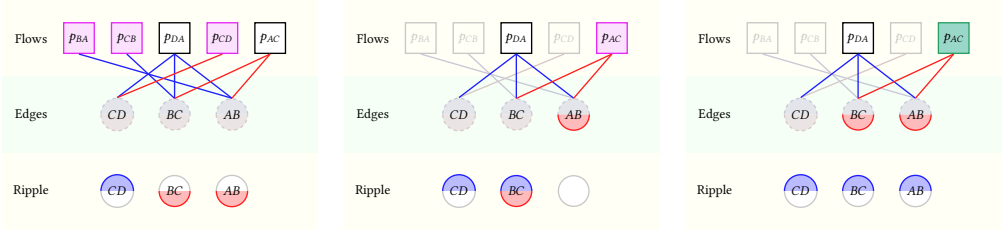
Fig. 4. Bipartite graph that cannot be peeled even though there is no deadlock in the topology

adding them to the ripple, and processing input symbols from the ripple, continues until either the ripple runs empty or all input symbols are successfully decoded.

Since encoded symbols incur communication overhead, it is desirable to minimize their quantity. A key factor affecting the number of encoded symbols is the choice of the degree distribution for the encoded symbols. However, if minimizing the number of encoded symbols leads to a ripple that vanishes before the algorithm terminates, the decoding process fails altogether. Prior work [26, 36] has focused extensively on the design of degree distributions that achieve a ripple size that is neither redundant nor so small that the ripple disappears before the LT process completes. For example, under a *robust soliton distribution* [26], LT codes recover n input symbols with probability at least $1 - \delta$ from $n + O(\sqrt{n}\log(n/\delta))$ encoded symbols.

5.2.2 Deadlock Peeling. The idea of peeling in LT codes can be extended to detecting deadlocks with a few modifications. Here, channels correspond to input symbols. A channel can be used (and covered) in one of two directions. We will consider the two directions *blue* and *red* with the *red* direction corresponding to usage of the edges in the same direction as $\overline{E}$ and blue denoting the directions specified by $\underline{E}$. A flow (an encoded symbol) uses one or more channels (input symbols), each in one of the two directions; the degree of the flow is the path length, or the number of channels used by the flow. The deadlock peeling process can be represented as a bipartite graph, where each flow p_i, $i \in [|\mathcal{P}|]$ corresponds to a node in the top partition, and each channel $e_j\{u, v\}$, $j \in [|E|]$ to a node in the bottom partition. An edge exists between channel $e_j\{u, v\}$ and flow p_i iff either (u, v) or (v, u) is part of p_i; the edge is red if $(u, v) \in p_i$ and blue if $(v, u) \in p_i$. Figures 3 and 4 show the the bipartite graph construction for their associated credit network topologies and paths.

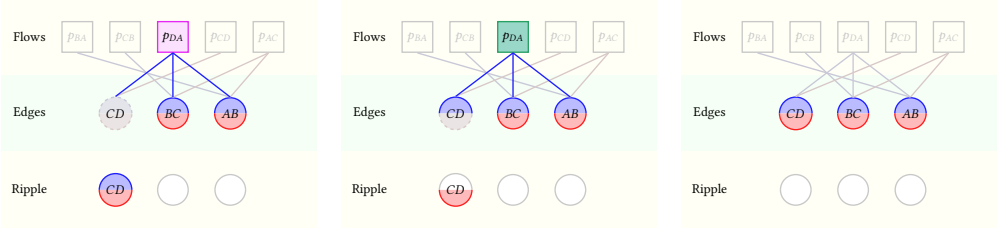
Fig. 5 shows the deadlock peeling process for the example credit network in Fig. 3. A flow of degree 1 can be released to cover and add to the ripple its last channel in the direction opposite to the direction in use. In other words, a flow p_1 of degree 1 using the edge $e\{u, v\}$ $u < v$ in the blue direction (v, u), can be released to cover e in the red direction (u, v). Newly covered channels are added to the ripple $\mathcal{R}$. Covering e in the (u, v) direction signifies that the channel e can always support flows in the (u, v) direction because flow p_1 would restore tokens back to the u end regardless of e's current balance state. Fig. 5a shows the impact of releasing the initial degree 1 flows. At every subsequent time step, a randomly chosen channel (and its associated color) is processed from the ripple. Processing a channel is similar to the LT process; the degree



(a) At the start, releasing the three degree 1 flows adds three symbols to the ripple.

(b) After processing red AB, flow p_{AC} becomes degree 1 to cover blue BC.

(c) Processing red BC makes p_{AC} degree 0, adding blue AB (and blue BC) to the ripple.



(d) Once blue AB and BC are processed, p_{DA} becomes degree 1 and covers red CD.

(e) Processing blue CD makes p_{DA} degree 0, but its release adds no new symbols to the ripple.

(f) Processing red CD ends the peeling algorithm with all channels deadlock-free.

Fig. 5. Evolution of the bipartite graph as channels are processed in the peeling algorithm.

of every flow that uses the channel in the processed color is reduced by 1. This may lead to new flows being released to cover more channels (Fig. 5b). However, only the previously uncovered channels increase the ripple size (e.g. blue AB in Fig. 5c). This peeling process continues as long as the ripple is non-empty and at least one remaining flow is of degree 1. As is the case with the LT process, to establish that the entire topology is deadlock free, the ripple should not vanish before the entire peeling process is complete. Consequently, prior analyses [26, 36] that design good degree distributions become relevant for the deadlock peeling process also.

The deadlock peeling process differs from the LT process in one key aspect. Unlike the LT process, flows of degree 1 cannot be removed immediately from the bipartite graph. A flow is removed only after it reaches degree 0 when every channel that it uses is covered in the color opposite to the direction of use. The intuition behind this is that a flow of degree 0 is unconstrained and can move tokens on all the channels it uses (restoring balance if the opposite direction was imbalanced). Note the important distinction here that a flow of degree 1 covers (or frees) only the last remaining channel in the opposite direction while a flow of degree 0 covers *all* of its channels in the opposite direction (Figures 5c, 5d, 5e). Algorithm 1 in App. B.2 summarizes the deadlock peeling process.

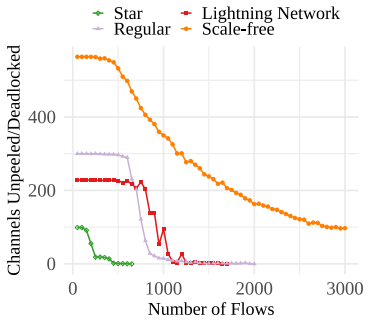


Fig. 6. Comparison between channels unpeeled at the end of the peeling algorithm (line) and channels deadlocked (points on the same lines) based on an ILP. Across all four topologies, the points fall on the line suggesting that the peeling algorithm's unpeeled channels matches the number of deadlocked channels exactly.

The deadlock peeling process cannot always detect all deadlock-free edges: the number of edges it successfully peels is a *lower bound* on the actual number of deadlock-free edges in a topology. This is because it identifies patterns of bidirectional dedicated flows on the channels in a topology. Such flows, while guaranteed to render the associated channels deadlock-free, are not required for deadlock-freeness. For example, the topology in Fig. 4 is deadlock-free but it has no length 1 flows. Hence, the peeling algorithm terminates on this topology without peeling any channels.

Empirical evaluation of the deadlock peeling process. While we know that the deadlock peeling process is a lower-bound on the true number of deadlock-free edges in a topology (and consequently its associated $\Phi_{\min}$), it is useful to know how good of a lower-bound it is. Since the deadlock detection problem is NP-hard, we formulate an Integer Linear Program (ILP) to identify the largest deadlock on a given topology. We describe the ILP in App. B and compare it to the output of the deadlock peeling process. Since the ILP is slow, we evaluate it on small samples of four different topologies (described in §6.1) consisting of only 100 nodes.

In Fig. 6, the number of channels unpeeled (line) matches the maximum deadlock reported by the ILP (points on the same lines) in all cases, suggesting that the deadlock peeling process provides not just a lower-bound, but rather an accurate approximation. Hence, in our evaluation (§6.2), we use the deadlock peeling process to peel as many channels as it can and compute the worst-case throughput as the one-step throughput of the sub-network consisting of all the peeled edges.

6 EVALUATION

We empirically evaluate a number of standard topologies for their throughput and deadlock characteristics to understand their performance. We describe our evaluation setup in §6.1, define metrics in §6.2, and compare existing topologies in §6.3. We use the peeling algorithm to understand the behavior of different topologies and propose a preliminary synthesis approach in §6.4.

6.1 Setup

Baseline Topologies. We compare the performance of credit networks based on random graphs and a graph based on the Lightning Network for their throughput and deadlock behaviors. We use five random graph types: a Watts Strogatz *small-world* graph [11] with 10% rewiring probability, a Barabási-Albert *scale-free* graph [2], a *power-law* graph whose degree distribution follows a power-law, an *Erdős-Rényi* graph [4], and a *random regular* graph.

We sample 5 different random graphs with 500 nodes and about 2000 edges for each graph type. To simulate a topology with properties similar to the Lightning Network, a PCN currently in use, we retrieved a snapshot of the topology on Oct. 5, 2020 using a c-lightning [1] node running on the Bitcoin Mainnet. The original full topology has over 5000 nodes and 29000 edges. Similar to prior work [45], we snowball sample [22] the full topology to generate a PCN with 452 nodes and 2051 edges. We also evaluate stars with 500 nodes as an example of a topology with good throughput and deadlock properties at the cost of being highly centralized. The throughput of all credit networks is normalized to a constant collateral C distributed equally amongst all their channels.

Demand Matrix and Path Choice. We generate demand matrices by sampling a fixed number of unique source-destination pairs from the set of nodes in the graph. The results in this section use demand pairs sampled uniformly at random, but we include results with demand pairs sampled with a skew towards "heavy-hitting" nodes in App. B.3. Every non-zero entry in the demand matrix adds one sender-receiver pair and allows the shortest path between them to move tokens in the credit network across rounds [6]. We refer to such a path as a flow in this section since it maps

[6]Our throughput metrics only depend on the total number of permissible paths. Adding more edge-disjoint paths for a given demand pair and sampling more demand pairs both increase the total paths; we use the latter approach in our evaluations.

to a single flow node in the bipartite graph associated with the peeling algorithm. Unlike prior work [45], we do not explicitly control for a circulation demand since we are interested in the effect of DAG demands on circulation throughput. As the demand matrix becomes more dense, the amount of circulation demand naturally grows. Unless otherwise mentioned, we sample 4 different demand matrices per random instance of the random graphs, to generate 20 unique points over which we average the throughput and deadlock behavior. Since there is only one instance of the star, Lightning Network and our synthesized topology, we use 20 different demand matrices instead. We only present results for demand density ranges that shows variation between the topologies. If the demand matrix is too sparse, there is not enough demand for any topology to perform well; if it is too dense, just routing one-hop demands (between end-points of every edge in the network) uses all the available collateral in the credit network ($\Phi_{\max} = \Phi_{\min} = 1$).

6.2 Metrics

Maximum Throughput. We first compute the maximum *per-epoch* throughput $\Phi_{\max}$ that a topology achieves when none of its channels are imbalanced or constrained. Recall that the throughput of a credit network is maximized at its perfect balance state $C/2$. Further, $\Phi_{\max} = \psi(C/2)$. We use an LP solver to compute ψ based on the constrained optimization problem in Eq. 10.

Worst-case Throughput. The worst-case throughput of a credit network $\Phi_{\min}$ is the minimum steady-state throughput achievable in an epoch from any state in its balance polytope [7]. We know from Theorem. 2 that $\Phi_{\min}$ is achieved at the state with the largest deadlock . Though detecting deadlocked states on an arbitrary topology is NP-hard, we use our approximate deadlock peeling process (§5.2) to identify the deadlock-free channels and compute the worst-case throughput $\Phi_{\min}$ as $\psi(C'/2)$ where $C'_i = C_i$ if i is reported as a deadlock-free channel, and $C'_i = 0$ otherwise.

Fraction of Channels Unpeeled. In addition to the above throughput metrics, we also report the fraction of the channels in the topology that the deadlock peeling process fails to peel. This acts as an upper-bound for the true number of deadlocked channels in a given topology.

6.3 Performance of Random Topologies

Fig. 7 shows the best-case throughput $\Phi_{\max}$ and the worst-case throughput $\Phi_{\min}$ achievable across all starting balance states for a set of random topologies with 500 nodes and an LN topology with 452 nodes for a fixed collateral budget for 2500–25000 demand pairs sampled uniformly at random. Stars outperform all the other topologies by over 50% even at the midpoint of the range. However, stars are highly centralized topologies (i.e., the hub has degree n); consequently, they are undesirable for decentralized use cases of credit networks. The $\Phi_{\max}$ value is comparable across most topologies; the small-world topology alone stands as an outlier because it has 50% longer paths than other topologies which results in lower throughput.

Fig. 7 also shows the variation in the $\Phi_{\min}$ values and the fraction of unpeeled channels across topologies. For instance, we notice that the Lightning Network, power-law and scale-free topologies have much better $\Phi_{\min}$ with fewer flows when compared to the other random graphs suggesting that they exhibit less throughput sensitivity to the channel balance state. At 7500 demand pairs, both Lightning Network and scale-free topologies peel 20-70% more channels than the small-world, Erdős-Rényi, and random regular topologies and correspondingly have 7-9x higher $\Phi_{\min}$. However, the hardship that the scale-free graph faces in peeling the last 10% of its channels manifests as a 12% hit to its $\Phi_{\min}$ when compared to the Lightning Network, particularly with a denser demand

[7] Our definitions of $\Phi_{\max}$ and $\Phi_{\min}$ assume the ability to send the maximum feasible amount between a sender-receiver pair using an ideal routing algorithm. This allows us to reason about throughput without concerning ourselves with the precise dynamics of any routing algorithm such as transaction sizes or splitting.

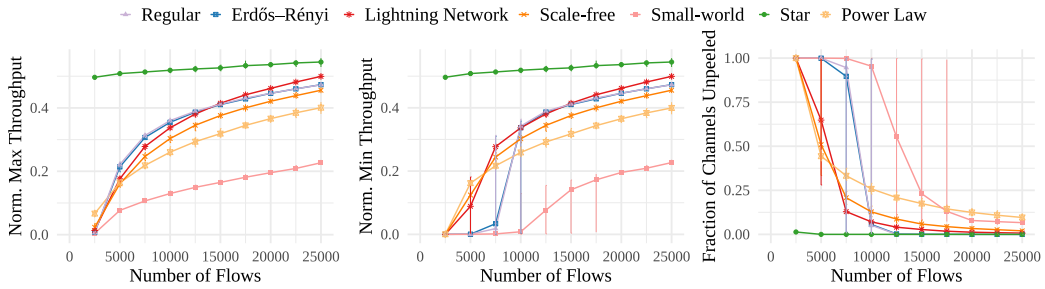


Fig. 7. Maximum ($\Phi_{\max}$) and minimum throughput ($\Phi_{\min}$) achieved by different topologies and channels left unpeeled as the number of flows is varied. Stars outperform all other topologies but are undesirable due to their centralization. The Lightning Network, power-law and scale-free graphs are less sensitive than the other topologies: they peel earlier and have higher $\Phi_{\min}$. The fraction of channels peeled corresponds well with $\Phi_{\min}$: topologies that peel well have a higher $\Phi_{\min}$. Whiskers denote max and min data point.

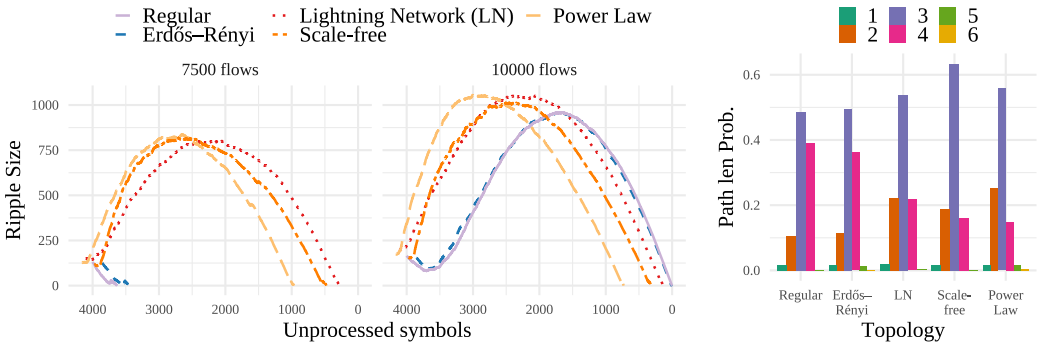


Fig. 8. Evolution of ripple size during the deadlock peeling process on different topologies. At 7500 flows, the ripples of Erdős-Rényi and random regular vanish too quickly preventing them from progressing at all in contrast to Lightning Network, power-law and scale-free topologies. However, while Lightning Network, power-law and scale-free topologies experience good growth at the beginning, they are unable to sustain a good ripple size resulting in the last 6-7% of channels left unprocessed even at 10000 flows.

matrix. This effect is even more pronounced in the power-law graph. In contrast, the Erdős-Rényi and random regular topologies do not peel as well with fewer flows, but quickly improve to peel all channels, on average, with 12500 demands. Beyond this point, their $\Phi_{\min}$ is comparable with the Lightning Network. The small-world topology peels only 50% of the flows even with 12500 demands which when compounded with its long paths leads to very low $\Phi_{\min}$ over the entire range. Fig. 13 in App. B.3 shows similar trends across topologies even when demand matrices are skewed in that some "heavy-hitter" nodes are more likely to both send and receive transactions.

Explaining the relative behavior of topologies. Given the apparent correlation between $\Phi_{\min}$ and the fraction of channels peeled, we now consider the effect of topology on the evolution of the deadlock peeling process. Fig. 8 shows the evolution of the ripple size at 7500 and 10000 demand pairs, along with the path length distributions for the random regular, Erdős-Rényi, Lightning Network, power-law and scale-free topologies. We consider the total number of symbols at the start to be twice the number of channels in the topology, one for each direction of a channel. Each peeling step involves processing one channel in one of the two directions. A processed symbol may lead to the release of some flow nodes and consequently, add more directed channels to the ripple.

In Fig. 8, all topologies start with similar initial ripple sizes because they have the same sparsity. In other words, a randomly sampled demand is equally likely to be of length 1 (span only one

edge) across all topologies. However, their ripple evolution patterns quickly diverge. The Lightning Network, power-law and scale-free topologies experience fast initial ripple growth, attributed to the 20% degree 2 flows (with path length 2) and up to 60% flows of degree 3. These short flows are likely to be released early, helping the deadlock peeling process pick up a robust ripple size. In contrast, at 7500 demands, the ripple vanishes quickly for the random regular and Erdős-Rényi topologies due to 10% fewer degree 2 flows. Yet, the ripples in Lightning Network, power-law and scale-free topologies vanish before the entire topology is peeled, with 250 and 500 directed channels respectively unpeeled. This behavior happens with 10000 demands too, albeit to a lesser extent. However, at 10000 flows, Erdős-Rényi and random regular topologies offset the initial dip in the ripple size compared to the remaining topologies; they experience later peaks, but peel the entire topology. The poor tail behavior of the scale-free topology can be attributed to its 6% less channel coverage for the same number of demand pairs: the presence of hubs means edges far away from the hub tend to be less used. Such channels are difficult to peel without a large increase in the number of flows that ensures the relevant edges see enough token movement. A similar analysis of the ripple evolution for flows sampled in a skewed manner is shown in Fig. 14 in App. B.3.

Predicting Performance using LT Codes. Since the deadlock peeling process was inspired by the design on LT Codes, we evaluate whether the analysis of LT Codes predicts the performance of the deadlock peeling process. We use the same 5 random graphs from Fig. 8 at 7500 flows and view their predicted ripple evolution in an LT code with the same degree distribution. The predicted ripple evolution computes the probability that a flow of degree d is released and adds a symbol to the ripple when there are L unprocessed symbols remaining. Extending this to the expected ripple addition at every step helps build a ripple evolution curve prediction (Eq. 6 of [36]) [8].

We notice in Fig. 9 that the prediction for the Lightning Network, power-law and scale-free topologies are closer to each other but different from the Erdős-Rényi and random regular graphs. Like the real ripple evolution (Fig. 8), the initial growth rate for the Lightning Network, power-law and scale-free topologies is faster than the other two graphs. Interestingly, the prediction suggests that Erdős-Rényi and random regular topologies will have difficulty peeling at 7500 flows: the predicted ripple sizes approach 0 with around 3000 unprocessed symbols remaining. Such a ripple evolution is not robust to the variance encountered during a typical peeling process as we observe in Fig. 8 where the ripple decreases drastically early on and vanishes. The prediction is more optimistic than the real evolution: the peak ripple sizes are higher and all topologies peel all their edges. In reality, the deadlock peeling process does not peel all the edges even in the Lightning Network, power-law and scale-free topologies. This difference is anticipated since the LT Codes analyses [26, 36] rely on an *i.i.d. bipartite encoding graph* where flows (encoded symbols) choose channels (input symbols) uniformly at random. On real graphs, the channels traversed by

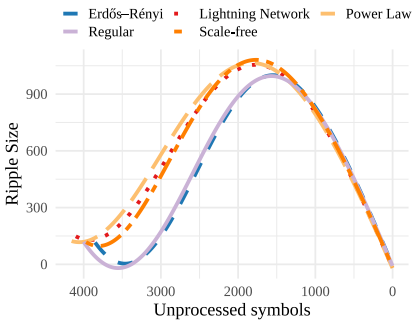


Fig. 9. Ripple size predicted by the LT codes analysis on different random topologies at 7500 flows. The prediction suggests that Erdős-Rényi and random regular ripple sizes dip to zero early on while Lightning Network, power-law and scale-free experience big growths. However, it departs from the real ripple size significantly towards the end of the peeling process.

[8]The trajectories use a slightly different expression (Eq. 22) for the expected symbols added that accounts for overlaps between symbols covered by the release of different flows at the same step.

flows are correlated; in fact, correlation between edges of flows makes it very hard to peel the last 6% of channels in the scale-free topology. Further, we found that with a skewed demand matrix, the correlations become stronger and the predictions made using the i.i.d. bipartite graph model deviate from reality more significantly (Fig. 14). Hence, a full analysis of the deadlock peeling process should take into account correlations induced by the topology structure and demand pattern (we leave this to future work). As a first step to understanding the value of such an analysis, we next investigate whether the LT code analysis can be used to synthesize good topologies for uniform random demand matrices, where fortuitously LT code predictions are reasonably accurate.

6.4 Topology Synthesis

Generating a path length distribution. Our goal is to find topologies that require fewer demand pairs to render a topology deadlock-free. We start with an approach for LT Codes [36], which fixes a desired ripple evolution and numerically computes a degree distribution that closely approximates it. In our setting, the degree distribution corresponds to a distribution over path lengths in the credit network. Like [36], we choose a ripple evolution of the form $R(L) = 1.7c^{1/2.5}$ where L is the number of unprocessed symbols and $R(L)$ is the ripple size when L symbols remain unprocessed. For a target graph with 300 nodes and 1500 edges, this ensures a ripple of 30 at the start of the peeling algorithm, decaying slowly to 25 halfway and eventually to 2 towards the end of the algorithm.

Next, we choose a path length distribution for the topology that achieves the desired ripple evolution. Prior work shows that the expected number of symbols added to the ripple at each LT process step is a linear function of the path length distribution [36]. The linear transformations and constants correspond to the release probabilities and the desired ripple addition at each step of the algorithm respectively. For completeness, we outline the equations in App. B.4.3. We minimize the ℓ_2-norm of the difference between the expected and desired ripple size. Since our degree distributions map to paths on a graph, we constrain the maximum path length and bound path length probabilities based on the maximum node degree (to enforce decentralization). We also ensure that the number of length 1 paths does not exceed the number of edges and that the path length probabilities decrease from length 2 onwards. Solving this least-squares optimization problem with linear constraints yields a path length distribution.

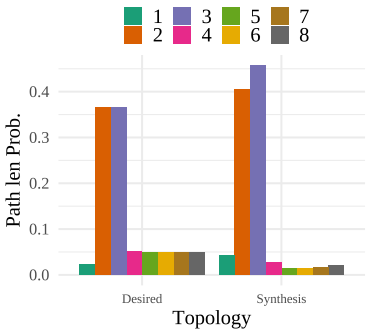


Fig. 10. Desired path length distribution vs. synthesized topology's distribution.

Synthesizing a matching topology. There are potentially many ways to synthesize a topology with a given shortest-path length distribution. Our approach exploits a known link between the distribution of shortest path lengths for a random graph and its pairwise joint degree distribution [33, 46].[9] We first set up an optimization using MATLAB to generate a joint degree distribution whose path length distribution minimizes the ℓ_2-norm of the distance from the target path length distribution (output from the previous subsection). Given a joint degree distribution, we use established methods to generate a random graph that matches the joint degree distribution [5, 20]. This approach is more expressive than the well-known configuration model [18], and can (approximately) recover Erdős-Rényi, scale-free, small-world and random regular graphs.

[9]The pairwise joint degree distribution of a graph evaluated at j and ℓ specifies the probability that a randomly sampled edge connects nodes of degree j and ℓ.

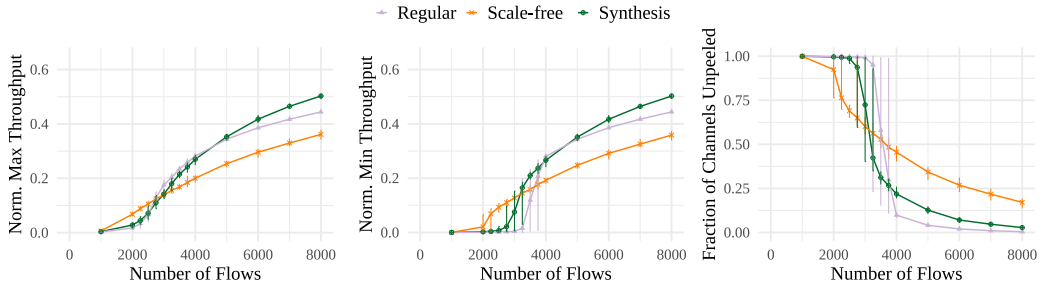


Fig. 11. Throughput comparison between a synthesized topology with 271 nodes and its random counterparts at 300 nodes and 1500 edges. The synthesized topology has lower throughput sensitivity than the random regular graph and better overall throughput than the scale-free graph. Whiskers denote max and min points.

Results. Our target topology has 300 nodes and 1500 edges. We first use the numerical optimizer to find a path length distribution that peels well. We observe that permitting a large maximum node degree generates path length distributions that the downstream MATLAB optimization routine fails to match well, generating graphs with only 60% of the desired edges. Thus, we ensure that the maximum node degree is 10 and no path is longer than 10 edges. The output distribution from the numerical optimizer is shown in Fig. 10. When using the MATLAB optimization routine to generate a joint degree distribution, we observe that setting the maximum node degree to 20, average degree to 10, and maximum path length to 10 ensures a close match to the desired path length distribution while avoiding extremely long paths during the synthesis step. We then synthesize a graph with the desired node distribution and take its largest connected component. The resulting topology has 271 nodes and 1513 channels, and achieves a path length distribution close to the one desired (Fig. 10). To evaluate how well the synthesized topology performs, we compare it to 5 instances of random regular and scale-free topologies with 300 nodes and about 1500 channels in Fig. 11. The maximum throughput $\Phi_{\max}$ achieved by the synthesized topology is comparable to the random regular graph and up to 15% better than the scale-free graph, particularly with more demand pairs. The $\Phi_{\min}$ and the fraction of channels unpeeled of the synthesized topology strikes a balance between the random regular and scale-free graphs. It is less sensitive than the random regular topology, notably with fewer demand pairs, achieving better 10–20% better $\Phi_{\min}$ and peeling 25–50% more channels. While it is more sensitive than the scale-free graph with sparser demand, it compensates with a 15% larger $\Phi_{\min}$ at denser demands. This approach shows promise in generating topologies that show good peeling properties and throughput insensitivity. We leave it to future work to explore generalizing this to different demand models to generate even better topologies.

Remark. While the above synthesis shows promise, most credit networks are formed by individual nodes' connectivity decisions, rather than by a centralized authority. However, router nodes in credit networks are incentivized to support high throughput via routing fees, and some PCNs already include peer recommendation systems that suggest peers to connect to [9, 10]. While we leave the details to future work, we envision building on these systems to encourage desirable topologies. For example, a credit network software client could "score", or suggest, peers such that the overall topology obeys a given joint degree distribution.

7 RELATED WORK

Network Topology Design. The peer-to-peer networking literature [25] has studied the design of structured [32, 38, 47] and unstructured topologies [3, 24], particularly for efficient content-retrieval. Meanwhile, fat tree [12], Clos [21], small world [41], and random graph [43] topologies have been designed to maximize throughput in datacenters. Adapting these ideas to credit networks is hard

due to their centralization—e.g., in a fat tree [12] with n top-of-rack (ToR) switches, the aggregation and core switches require a degree of $O(n^{1/3})$.

Credit Network Performance. There has been substantial recent interest in quantifying credit network performance, broadly defined as transaction throughput or success rate; prior work has studied the impact of several categories of influencing factors, including routing and scheduling protocols [31, 44, 45, 51, 52], privacy constraints [30, 40, 48, 49], defaulting agents [37], and network topology [13, 15, 16, 23]. Prior work exploring the effects of credit network topology on transaction success rate [15] assumed sequential transactions and a full demand matrix. In this setting, maximum throughput can be trivially achieved with length-1 flows, so no deadlocks are observed.

Demand-aware design of payment channel network topologies [16] poses the problem as an ILP: given a channel budget, a set of nodes, and a demand matrix, the ILP finds an adjacency matrix that either maximizes the number of connected demand pairs [13] or minimizes the number of channels added and the average path length [23]. This approach ignores the effects of channel imbalance, which causes the deadlock-related problems explored in this work.

Erasure Codes. Sparse graph codes have been studied for decades in channel coding [14, 19, 26–28, 39, 42]. We identify a parallel between such codes and detecting deadlocked edges in a topology. However, despite a rich literature on sparse graph codes, our setting cannot fully utilize most existing constructions for two reasons. First, we need to be able to change the number of flows (encoding symbols) flexibly, without requiring a totally new encoding. We therefore use *rateless* codes [26, 27, 42]; specifically, our peeling algorithm builds closely on Luby's LT codes [26]. Second, our encoding procedure must respect the topology constraints of the underlying graph; we cannot assign channels uniformly at random to flows, as in classical sparse graph codes. To our knowledge, this constraint has not been previously studied. This is why our theoretical predictions for the number of flows needed to peel a graph (from [26]) are lower than the number needed in practice (§6). More broadly, this suggests an interesting direction of further study on error-correcting codes that obey encoding constraints imposed by a graph topology.

8 CONCLUSION

In this paper, we studied the effects of topology and channel imbalance on the throughput of credit networks. We demonstrated a close relation between worst-case throughput and the presence of deadlocks, and proposed a heuristic peeling algorithm for identifying the presence of deadlocks. Important directions for future work include synthesizing graphs that are: (a) easy to peel, (b) exhibit high throughput, and (c) limit the maximum degree of any node. While we have made progress on that front, we were not able to find topologies that perform substantially better than random graphs. Hence, an interesting question is whether this is even possible. Another interesting direction is to analyze the peeling algorithm given the correlations induced (in the bipartite graph) by an arbitrary topology and demand pattern. Further, this work assumes perfect routing; in practice it is unclear whether one can actually achieve $\phi(\boldsymbol{b})$. Designing decentralized routing protocols that are tailored to good credit network topologies and understanding their performance with respect to these upper bounds will be important in practice. Finally, designing suitable incentive/recommendation mechanisms to achieve a desired topology in a decentralized manner is an important problem.

ACKNOWLEDGMENTS

We thank our anonymous reviewers for their detailed feedback. This work was supported in part by an AFOSR grant FA9550-21-1-0090; NSF grants CNS-1751009, CNS-1910676, and CIF-1705007; a Microsoft Faculty Fellowship; awards from the Cisco Research Center and the Fintech@CSAIL program; and gifts from Chainlink, the Sloan Foundation, and the Ripple Foundation.

REFERENCES

[1] [n.d.]. Amount-independent payment routing in Lightning Networks. https://medium.com/coinmonks/amount-independent-payment-routing-in-lightning-networks-6409201ff5ed.

[2] [n.d.]. Barabási-Albert Graph. https://networkx.github.io/documentation/networkx-1.9.1/reference/generated/networkx.generators.random_graphs.barabasi_albert_graph.html.

[3] [n.d.]. BitTorrent. https://www.bittorrent.com.

[4] [n.d.]. Erdős-Rényi Graph. https://networkx.org/documentation/stable//reference/generated/networkx.generators.random_graphs.erdos_renyi_graph.html.

[5] [n.d.]. Joint-Degree Graph. https://networkx.org/documentation/stable/reference/generated/networkx.generators.joint_degree_seq.joint_degree_graph.html.

[6] [n.d.]. Joint-Degree Graph Validation. https://networkx.org/documentation/stable/reference/generated/networkx.generators.joint_degree_seq.is_valid_joint_degree.html.

[7] [n.d.]. MATLAB optimizer. https://www.mathworks.com/help/optim/ug/fmincon.html.

[8] [n.d.]. Raiden Network. https://raiden.network/.

[9] [n.d.]. The Node Operator's Guide to the Lightning Galaxy, Part 1. https://lightning.engineering/posts/2019-07-30-routing-quide-1/.

[10] [n.d.]. The Node Operator's Guide to the Lightning Galaxy, Part 2: Node Scoring and Pathfinding. https://lightning.engineering/posts/2019-11-07-routing-guide-2/.

[11] [n.d.]. Watts-Strogatz Graph. https://networkx.github.io/documentation/networkx-1.9/reference/generated/networkx.generators.random_graphs.watts_strogatz_graph.html.

[12] Mohammad Al-Fares, Alexander Loukissas, and Amin Vahdat. 2008. A scalable, commodity data center network architecture. *ACM SIGCOMM computer communication review* 38, 4 (2008), 63–74.

[13] Lukas Aumayr, Esra Ceylan, Matteo Maffei, Pedro Moreno-Sanchez, Iosif Salem, and Stefan Schmid. 2020. Demand-Aware Payment Channel Networks. *arXiv preprint arXiv:2011.14341* (2020).

[14] John W Byers, Michael Luby, Michael Mitzenmacher, and Ashutosh Rege. 1998. A digital fountain approach to reliable distribution of bulk data. *ACM SIGCOMM Computer Communication Review* 28, 4 (1998), 56–67.

[15] Pranav Dandekar, Ashish Goel, Ramesh Govindan, and Ian Post. 2011. Liquidity in credit networks: A little trust goes a long way. In *Proceedings of the 12th ACM conference on Electronic commerce*. 147–156.

[16] Pranav Dandekar, Ashish Goel, Michael P Wellman, and Bryce Wiedenbeck. 2015. Strategic formation of credit networks. *ACM Transactions on Internet Technology (TOIT)* 15, 1 (2015), 1–41.

[17] Christian Decker and Roger Wattenhofer. 2015. A fast and scalable payment network with bitcoin duplex micropayment channels. In *Symposium on Self-Stabilizing Systems*. Springer, 3–18.

[18] Bailey K Fosdick, Daniel B Larremore, Joel Nishimura, and Johan Ugander. 2018. Configuring random graph models with fixed degree sequences. *Siam Review* 60, 2 (2018), 315–355.

[19] Robert Gallager. 1962. Low-density parity-check codes. *IRE Transactions on information theory* 8, 1 (1962), 21–28.

[20] Minas Gjoka, Bálint Tillman, and Athina Markopoulou. 2015. Construction of simple graphs with a target joint degree matrix and beyond. In *2015 IEEE Conference on Computer Communications (INFOCOM)*. IEEE, 1553–1561.

[21] Albert Greenberg, James R Hamilton, Navendu Jain, Srikanth Kandula, Changhoon Kim, Parantap Lahiri, David A Maltz, Parveen Patel, and Sudipta Sengupta. 2009. VL2: A scalable and flexible data center network. In *Proceedings of the ACM SIGCOMM 2009 conference on Data communication*. 51–62.

[22] Pili Hu and Wing Cheong Lau. 2013. A survey and taxonomy of graph sampling. *arXiv preprint arXiv:1308.5865* (2013).

[23] Julia Khamis and Ori Rottenstreich. [n.d.]. Demand-aware Channel Topologies for Off-chain Blockchain Payments. ([n. d.]).

[24] Joohwan Kim and Rayadurgam Srikant. 2013. Real-time peer-to-peer streaming over multiple random hamiltonian cycles. *IEEE transactions on information theory* 59, 9 (2013), 5763–5778.

[25] Eng Keong Lua, Jon Crowcroft, Marcelo Pias, Ravi Sharma, and Steven Lim. 2005. A survey and comparison of peer-to-peer overlay network schemes. *IEEE Communications Surveys & Tutorials* 7, 2 (2005), 72–93.

[26] Michael Luby. 2002. LT codes. In *The 43rd Annual IEEE Symposium on Foundations of Computer Science, 2002. Proceedings.* IEEE, 271–280.

[27] Michael G Luby, Michael Mitzenmacher, Mohammad Amin Shokrollahi, and Daniel A Spielman. 2001. Efficient erasure correcting codes. *IEEE Transactions on Information Theory* 47, 2 (2001), 569–584.

[28] David JC MacKay. 1999. Good error-correcting codes based on very sparse matrices. *IEEE transactions on Information Theory* 45, 2 (1999), 399–431.

[29] Mr Samuel Munzele Maimbo, Mr Mohammed El Qorchi, and Mr John F Wilson. 2003. *Informal Funds Transfer Systems: An analysis of the informal hawala system.* International Monetary Fund.

[30] Giulio Malavolta, Pedro Moreno-Sanchez, Aniket Kate, and Matteo Maffei. 2017. SilentWhispers: Enforcing Security and Privacy in Decentralized Credit Networks.. In *NDSS*.

[31] Giulio Malavolta, Pedro Moreno-Sanchez, Aniket Kate, Matteo Maffei, and Srivatsan Ravi. 2017. Concurrency and privacy with payment-channel networks. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. 455–471.

[32] Petar Maymounkov and David Mazieres. 2002. Kademlia: A peer-to-peer information system based on the xor metric. In *International Workshop on Peer-to-Peer Systems*. Springer, 53–65.

[33] Sergey Melnik and James P Gleeson. 2016. Simple and accurate analytical calculation of shortest path lengths. *arXiv preprint arXiv:1604.05521* (2016).

[34] Pedro Moreno-Sanchez, Navin Modi, Raghuvir Songhela, Aniket Kate, and Sonia Fahmy. 2018. Mind your credit: Assessing the health of the ripple credit network. In *Proceedings of the 2018 World Wide Web Conference*. 329–338.

[35] Joseph Poon and Thaddeus Dryja. 2016. The Bitcoin Lightning Network: Scalable Off-chain Instant Payments. *draft version 0.5* 9 (2016), 14.

[36] Petar Popovski, Jan Ostergaard, et al. 2012. Design and analysis of LT codes with decreasing ripple size. *IEEE Transactions on Communications* 60, 11 (2012), 3191–3197.

[37] Geoffrey Ramseyer, Ashish Goel, and David Mazières. 2020. Liquidity in credit networks with constrained agents. In *Proceedings of The Web Conference 2020*. 2099–2108.

[38] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. 2001. A scalable content-addressable network. In *Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*. 161–172.

[39] Thomas J Richardson, Mohammad Amin Shokrollahi, and Rüdiger L Urbanke. 2001. Design of capacity-approaching irregular low-density parity-check codes. *IEEE transactions on information theory* 47, 2 (2001), 619–637.

[40] Stefanie Roos, Pedro Moreno-Sanchez, Aniket Kate, and Ian Goldberg. 2017. Settling payments fast and private: Efficient decentralized routing for path-based transactions. *arXiv preprint arXiv:1709.05748* (2017).

[41] Ji-Yong Shin, Bernard Wong, and Emin Gün Sirer. 2011. Small-world datacenters. In *Proceedings of the 2nd ACM Symposium on Cloud Computing*. 1–13.

[42] Amin Shokrollahi. 2006. Raptor codes. *IEEE transactions on information theory* 52, 6 (2006), 2551–2567.

[43] Ankit Singla, Chi-Yao Hong, Lucian Popa, and P Brighten Godfrey. 2012. Jellyfish: Networking data centers randomly. In *9th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 12)*. 225–238.

[44] Vibhaalakshmi Sivaraman, Shaileshh Bojja Venkatakrishnan, Mohammad Alizadeh, Giulia Fanti, and Pramod Viswanath. 2018. Routing Cryptocurrency with the Spider Network. In *Proceedings of the 17th ACM Workshop on Hot Topics in Networks*. ACM, 29–35.

[45] Vibhaalakshmi Sivaraman, Shaileshh Bojja Venkatakrishnan, Kathleen Ruan, Parimarjan Negi, Lei Yang, Radhika Mittal, Giulia Fanti, and Mohammad Alizadeh. 2020. High Throughput Cryptocurrency Routing in Payment Channel Networks. In *17th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 20)*. 777–796.

[46] Isabelle Stanton and Ali Pinar. 2012. Constructing and sampling graphs with a prescribed joint degree distribution. *Journal of Experimental Algorithmics (JEA)* 17 (2012), 3–1.

[47] Ion Stoica, Robert Morris, David Karger, M Frans Kaashoek, and Hari Balakrishnan. 2001. Chord: A scalable peer-to-peer lookup service for internet applications. *ACM SIGCOMM Computer Communication Review* 31, 4 (2001), 149–160.

[48] Weizhao Tang, Weina Wang, Giulia Fanti, and Sewoong Oh. 2020. Privacy-utility tradeoffs in routing cryptocurrency over payment channel networks. *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 4, 2 (2020), 1–39.

[49] Sergei Tikhomirov, Pedro Moreno-Sanchez, and Matteo Maffei. 2020. A quantitative analysis of security, anonymity and scalability for the lightning network. In *2020 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*. IEEE, 387–396.

[50] Kyle Torpey. 2016. Greg Maxwell: Lightning Network Better Than Sidechains for Scaling Bitcoin. https://bitcoinmagazine.com/articles/greg-maxwell-lightning-network-better-than-sidechains-for-scaling-bitcoin-1461077424/.

[51] Peng Wang, Hong Xu, Xin Jin, and Tao Wang. 2019. Flash: efficient dynamic routing for offchain networks. In *Proceedings of the 15th International Conference on Emerging Networking Experiments And Technologies*. 370–381.

[52] Shira Werman and Aviv Zohar. 2018. Avoiding deadlocks in payment channel networks. In *Data Privacy Management, Cryptocurrencies and Blockchain Technology*. Springer, 175–187.

A THEORETICAL RESULTS

A.1 Basic Properties

Proposition 4. *For any state $\boldsymbol{b}$, $\mathbb{X}_{\boldsymbol{b}}$ is a convex set and $\boldsymbol{b} \in \mathbb{X}_{\boldsymbol{b}}$.*

Proposition 5. *For a balance state $\boldsymbol{b} \in \mathbb{R}_{\geq 0}^{|E|}$, if $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{D}_{\boldsymbol{b}}$, then $\alpha\boldsymbol{x} + \beta\boldsymbol{y} \in \mathbb{D}_{\boldsymbol{b}}$ for any $\alpha, \beta \in \mathbb{R}_{\geq 0}$.*

Proposition 6. *For any $\boldsymbol{f} : \boldsymbol{f} \geq 0$ and $R\boldsymbol{f} \leq \begin{bmatrix} \boldsymbol{b} \\ C - \boldsymbol{b} \end{bmatrix}$, we have $\boldsymbol{b} - \Delta R\boldsymbol{f} \in \mathbb{X}_{\boldsymbol{b}}$ and $\boldsymbol{f} \in \mathbb{F}_{\boldsymbol{b}}$.*

Proposition 7. $\boldsymbol{b} \in \mathbb{X}_{\boldsymbol{a}} \iff \mathbb{X}_{\boldsymbol{b}} \subseteq \mathbb{X}_{\boldsymbol{a}}$.

Proposition 8. *The center state has maximized ψ. Precisely, $\psi\left(\frac{C}{2}\right) = \max_{\boldsymbol{a} \in \mathbb{B}} \psi(\boldsymbol{a})$.*

Proposition 9.

$$\mathbb{L}_{\boldsymbol{b}} = \left\{ i \in [|E|] \;\middle|\; a_i = b_i,\ \forall \boldsymbol{a} \in \mathbb{X}_{\boldsymbol{b}} \right\}$$

Proposition 10. *If $i \in \mathbb{L}_{\boldsymbol{b}}$, then for all $\boldsymbol{f} \in \mathbb{F}_{\boldsymbol{b}}$ whose decomposition is a sequence of flows $\left(\boldsymbol{f}^{(j)}\right)_{j=0}^{k-1}$, we have*

(1) *For all $j \in [k]$, $\overline{R}_i \boldsymbol{f}^{(j)} = \underline{R}_i \boldsymbol{f}^{(j)} = 0$;*
(2) *For all $j \in [k]$ and $\pi \in \Pi$ (index of path) where $\overline{R}_{i,\pi} = 1 \vee \underline{R}_{i,\pi} = 1$, $\boldsymbol{f}_{\pi}^{(j)} = 0$.*

A.2 Supporting Theorems

Lemma 11. *If for some $\epsilon \geq 0$, $\epsilon \begin{bmatrix} \boldsymbol{a} \\ C - \boldsymbol{a} \end{bmatrix} \leq \begin{bmatrix} \boldsymbol{b} \\ C - \boldsymbol{b} \end{bmatrix}$, then for all $\boldsymbol{p} \in \mathbb{X}_{\boldsymbol{a}}$, we have $\boldsymbol{b} + \epsilon(\boldsymbol{p} - \boldsymbol{a}) \in \mathbb{X}_{\boldsymbol{b}}$.*

Proof. Since $\boldsymbol{p} \in \mathbb{X}_{\boldsymbol{a}}$, $\exists k \in \mathbb{N}$, a sequence of flows $\boldsymbol{f}^{(0:k-1)}$ and a sequence of transient states $\boldsymbol{a}^{(0:k)}$, where $\boldsymbol{a}^{(0)} = \boldsymbol{a}$, $\boldsymbol{a}^{(k)} = \boldsymbol{p}$, and for all $j \in [k]$,

$$\boldsymbol{a}^{(j+1)} = \boldsymbol{a}^{(j)} - \Delta R\boldsymbol{f}^{(j)}, \quad \boldsymbol{f}^{(j)} \geq 0, \quad R\boldsymbol{f}^{(j)} \leq \begin{bmatrix} \boldsymbol{a}^{(j)} \\ C - \boldsymbol{a}^{(j)} \end{bmatrix}.$$

Consider new flows $\boldsymbol{g}^{(j)} = \epsilon \boldsymbol{f}^{(j)}$ for $\epsilon \geq 0$ and $j \in [k]$. Starting from state $\boldsymbol{b}^{(0)} = \boldsymbol{b}$, we send flows $\left(\boldsymbol{g}^{(j)}\right)_{j=0}^{k-1}$ and obtain transient states $\left(\boldsymbol{b}^{(j)}\right)_{j=0}^{k}$, where for all $j \in [k]$,

$$\boldsymbol{b}^{(j+1)} = \boldsymbol{b}^{(j)} - \Delta R\boldsymbol{g}^{(j)} = \boldsymbol{b}^{(j)} - \epsilon \Delta R\boldsymbol{f}^{(j)} = \boldsymbol{b} - \epsilon \Delta R \sum_{i=0}^{j} \boldsymbol{f}^{(i)} = \boldsymbol{b} + \epsilon \left(\boldsymbol{a}^{(j+1)} - \boldsymbol{a}\right).$$

Note that $\boldsymbol{b}^{(0)} = \boldsymbol{b} + \epsilon 0 = \boldsymbol{b} + \epsilon\left(\boldsymbol{a}^{(0)} - \boldsymbol{a}\right)$. Staring with $\epsilon \begin{bmatrix} \boldsymbol{a} \\ C - \boldsymbol{a} \end{bmatrix} \leq \begin{bmatrix} \boldsymbol{b} \\ C - \boldsymbol{b} \end{bmatrix}$, we have

$$\begin{aligned}
& \epsilon \begin{bmatrix} \boldsymbol{a} \\ C - \boldsymbol{a} \end{bmatrix} \leq \begin{bmatrix} \boldsymbol{b} \\ C - \boldsymbol{b} \end{bmatrix} \\
\implies & \epsilon \begin{bmatrix} \boldsymbol{a}^{(j)} \\ C - \boldsymbol{a}^{(j)} \end{bmatrix} \leq \begin{bmatrix} \boldsymbol{b} \\ C - \boldsymbol{b} \end{bmatrix} + \epsilon \left(\begin{bmatrix} \boldsymbol{a}^{(j)} \\ C - \boldsymbol{a}^{(j)} \end{bmatrix} - \begin{bmatrix} \boldsymbol{a} \\ C - \boldsymbol{a} \end{bmatrix} \right) = \begin{bmatrix} \boldsymbol{b}^{(j)} \\ C - \boldsymbol{b}^{(j)} \end{bmatrix}, \quad \forall j \in [k] \\
\implies & R\boldsymbol{g}^{(j)} = \epsilon R\boldsymbol{f}^{(j)} \leq \epsilon \begin{bmatrix} \boldsymbol{a}^{(j)} \\ C - \boldsymbol{a}^{(j)} \end{bmatrix} \leq \begin{bmatrix} \boldsymbol{b}^{(j)} \\ C - \boldsymbol{b}^{(j)} \end{bmatrix}, \quad \forall j \in [k].
\end{aligned}$$

Therefore, combining with $\boldsymbol{g}^{(j)} = \epsilon \boldsymbol{f}^{(j)} \geq 0$, we know $\left(\boldsymbol{g}^{(j)}\right)_{j=0}^{k-1} = \epsilon \left(\boldsymbol{f}^{(j)}\right)_{j=0}^{k-1}$ is a valid sequence of flows that leads to a reachable state from $\boldsymbol{b}$. In other words,

$$\mathbb{X}_{\boldsymbol{b}} \ni \boldsymbol{b}^{(k)} = \boldsymbol{b} - \Delta R \sum_{j=0}^{k-1} \boldsymbol{g}^{(j)} = \boldsymbol{b} - \epsilon \Delta R \sum_{j=0}^{k-1} \boldsymbol{f}^{(j)} = \boldsymbol{b} + \epsilon(\boldsymbol{p} - \boldsymbol{a}). \qquad \blacksquare$$

Lemma 12. *Let $\boldsymbol{b} \in \mathbb{X}_{\boldsymbol{a}} - \{\boldsymbol{a}\}$ and $\boldsymbol{p} \triangleq \arg\max_{\boldsymbol{p}' \in \mathbb{B}} \{\|\boldsymbol{p}' - \boldsymbol{a}\|_2 | \exists t \geq 0 : \boldsymbol{p}' - \boldsymbol{a} = t(\boldsymbol{b} - \boldsymbol{a})\}$ be the intersection with the boundary of state space with the direction along $\boldsymbol{a}$ to $\boldsymbol{b}$. Let $r = \dfrac{\|\boldsymbol{p} - \boldsymbol{b}\|_2}{\|\boldsymbol{p} - \boldsymbol{a}\|_2} \in [0, 1)$. For all $t \in [0, 1 - r^2]$, $\boldsymbol{a} + t(\boldsymbol{p} - \boldsymbol{a}) \in \mathbb{X}_{\boldsymbol{a}}$ and for all $t \in [1 - r, 1 - r^2]$, $\boldsymbol{a} + t(\boldsymbol{p} - \boldsymbol{a}) \in \mathbb{X}_{\boldsymbol{b}}$.*

Proof. By $\boldsymbol{b} \in \mathbb{X}_{\boldsymbol{a}}$, there exists a $k \in \mathbb{N}$, a sequence of flows $\boldsymbol{f}^{(0:k-1)}$ and a sequence of transient states $\boldsymbol{a}^{(0:k)}$, where $\boldsymbol{a}^{(0)} = \boldsymbol{a}$, $\boldsymbol{a}^{(k)} = \boldsymbol{b}$, and for all $j \in [k]$,

$$\boldsymbol{a}^{(j+1)} = \boldsymbol{a}^{(j)} - \Delta R \boldsymbol{f}^{(j)}, \quad \boldsymbol{f}^{(j)} \geq 0, \quad R\boldsymbol{f}^{(j)} \leq \begin{bmatrix} \boldsymbol{a}^{(j)} \\ C - \boldsymbol{a}^{(j)} \end{bmatrix}.$$

States on segment $[\boldsymbol{a}, \boldsymbol{b}]$. These states correspond to $t \in [0, 1 - r]$.

For any constant $\epsilon \in [0, 1]$, we have $\epsilon \begin{bmatrix} \boldsymbol{a} \\ C - \boldsymbol{a} \end{bmatrix} \leq \begin{bmatrix} \boldsymbol{a} \\ C - \boldsymbol{a} \end{bmatrix}$. Citing Lemma 11 and using the fact that $\boldsymbol{b} \in \mathbb{X}_{\boldsymbol{a}}$, we obtain $\boldsymbol{a} + \epsilon(\boldsymbol{b} - \boldsymbol{a}) \in \mathbb{X}_{\boldsymbol{a}}$, for all $\epsilon \in [0, 1]$. From $\boldsymbol{a} + \epsilon(\boldsymbol{b} - \boldsymbol{a}) = \boldsymbol{a} + \epsilon(1 - r)(\boldsymbol{p} - \boldsymbol{a})$, we know the statement is true for all $t \in [0, 1 - r]$.

States on segment $[\boldsymbol{b}, \boldsymbol{c}]$. These states correspond to $t \in [1 - r, 1 - r^2]$, or a part of the segment $[\boldsymbol{b}, \boldsymbol{p}]$.

Starting with $0 \leq \epsilon \leq r < 1$ and $-\begin{bmatrix} \boldsymbol{a} \\ C - \boldsymbol{a} \end{bmatrix} \leq 0 \leq \begin{bmatrix} \boldsymbol{p} \\ C - \boldsymbol{p} \end{bmatrix}$, we have the following deductions.

$$\begin{aligned} & (0 \leq \epsilon \leq r < 1) \wedge \left(-\begin{bmatrix} \boldsymbol{a} \\ C - \boldsymbol{a} \end{bmatrix} \leq 0 \leq \begin{bmatrix} \boldsymbol{p} \\ C - \boldsymbol{p} \end{bmatrix} \right) \\ \implies & (\epsilon - r) \begin{bmatrix} \boldsymbol{a} \\ C - \boldsymbol{a} \end{bmatrix} \leq 0 \leq (1 - r) \begin{bmatrix} \boldsymbol{p} \\ C - \boldsymbol{p} \end{bmatrix} \\ \implies & \epsilon \begin{bmatrix} \boldsymbol{a} \\ C - \boldsymbol{a} \end{bmatrix} \leq \begin{bmatrix} \boldsymbol{p} \\ C - \boldsymbol{p} \end{bmatrix} - r \left(\begin{bmatrix} \boldsymbol{p} \\ C - \boldsymbol{p} \end{bmatrix} - \begin{bmatrix} \boldsymbol{a} \\ C - \boldsymbol{a} \end{bmatrix} \right) = \begin{bmatrix} \boldsymbol{b} \\ C - \boldsymbol{b} \end{bmatrix}. \end{aligned}$$

Citing Lemma 11, we have $\boldsymbol{b} + \epsilon(\boldsymbol{b} - \boldsymbol{a}) \in \mathbb{X}_{\boldsymbol{b}}$. By transitivity of reachable states (Prop. 7), we have for all $\epsilon \in [0, r]$,

$$\mathbb{X}_{\boldsymbol{a}} \supseteq \mathbb{X}_{\boldsymbol{b}} \ni \boldsymbol{b} + \epsilon(\boldsymbol{b} - \boldsymbol{a}) = \boldsymbol{a} + (1 - r)(1 + \epsilon)(\boldsymbol{p} - \boldsymbol{a}).$$

In other words, for all $t \in [1 - r, 1 - r^2]$, $\boldsymbol{a} + t(\boldsymbol{p} - \boldsymbol{a}) \in \mathbb{X}_{\boldsymbol{a}} \cap \mathbb{X}_{\boldsymbol{b}}$. The proof is now complete. $\blacksquare$

Theorem 13. *Assume $\boldsymbol{\xi} \in \mathbb{D}_{\boldsymbol{a}} - \{0\}$ is a feasible direction and $\boldsymbol{p} \triangleq \arg\max_{\boldsymbol{p}' \in \mathbb{B}} \{\|\boldsymbol{p}' - \boldsymbol{a}\|_2 | \exists t \geq 0 : \boldsymbol{p}' - \boldsymbol{a} = t\boldsymbol{\xi}\}$ is the ray's last intersection with the state space. For all $t \in [0, 1)$, $\boldsymbol{a} + t(\boldsymbol{p} - \boldsymbol{a}) \in \mathbb{X}_{\boldsymbol{a}}$.*

Proof. By definition of $\mathbb{D}_{\boldsymbol{a}}$, $\exists \boldsymbol{b} \in \mathbb{X}_{\boldsymbol{a}}$, $r \in [0, 1)$, such that $\boldsymbol{p} - \boldsymbol{b} = r(\boldsymbol{p} - \boldsymbol{a})$. If $\boldsymbol{b}$ itself is the end point on the boundary, then $r = 0$ and $\boldsymbol{p} = \boldsymbol{b}$. Then, using Lemma 12, we can trivially prove the claim.

If $0 < r < 1$, we can take $k = \lceil \log_r(1 - t) \rceil \in \mathbb{N}$ which satisfies $1 - r^k \geq t$. Then, we construct a sequence of transient states $\left(\boldsymbol{a}^{(j)}\right)_{j=0}^{k}$, where $\boldsymbol{a}^{(j)} = \boldsymbol{a} + (1 - r^j)(\boldsymbol{p} - \boldsymbol{a})$ for all $j \in [k] \cup \{k\}$. By

this definition,

$$\boldsymbol{a}^{(0)} = \boldsymbol{a}, \quad \boldsymbol{a}^{(1)} = \boldsymbol{b}, \quad \forall j \in [k] : \boldsymbol{p} - \boldsymbol{a}^{(j+1)} = r\left(\boldsymbol{p} - \boldsymbol{a}^{(j)}\right).$$

We recursively show $\boldsymbol{a}^{(j+1)} \in \mathbb{X}_{\boldsymbol{a}^{(j)}}$ for all $j \in [k]$. This is true for $j = 0$ given $\boldsymbol{b} \in \mathbb{X}_{\boldsymbol{a}}$.
Suppose $\boldsymbol{a}^{(j+1)} \in \mathbb{X}_{\boldsymbol{a}^{(j)}}$ is true for $j = J - 1 < k - 1$. Then for $j = J$, we have

$$\boldsymbol{a}^{(J+1)} = \boldsymbol{p} - r\left(\boldsymbol{p} - \boldsymbol{a}^{(J)}\right) = \boldsymbol{a}^{(J-1)} + (1 - r^2)\left(\boldsymbol{p} - \boldsymbol{a}^{(J-1)}\right).$$

By Lemma 12, combined with $\boldsymbol{p} - \boldsymbol{a}^{(J+1)} = r\left(\boldsymbol{p} - \boldsymbol{a}^{(J)}\right)$ and $\boldsymbol{a}^{(J)} \in \mathbb{X}_{\boldsymbol{a}^{(J-1)}}$, we have $\boldsymbol{a}^{(J+1)} \in \mathbb{X}_{\boldsymbol{a}^{(J)}}$.
Therefore, $\boldsymbol{a}^{(j+1)} \in \mathbb{X}_{\boldsymbol{a}^{(j)}}$ for all $j \in [k]$. By Proposition 7 (transitive reachability), this further implies $\boldsymbol{a}^{(k)} \in \mathbb{X}_{\boldsymbol{a}}$. Since

$$\boldsymbol{c} \triangleq \boldsymbol{a} + t(\boldsymbol{p} - \boldsymbol{a}) = \boldsymbol{a} + \frac{t}{1 - r^k}\left(\boldsymbol{a}^{(k)} - \boldsymbol{a}\right),$$

we know by $\dfrac{t}{1 - r^k} \leq 1$ that $\boldsymbol{c}$ is on segment $\left[\boldsymbol{a},\ \boldsymbol{a}^{(k)}\right]$. By Lemma 12, we have $\boldsymbol{a} + t(\boldsymbol{p} - \boldsymbol{a}) = \boldsymbol{c} \in \mathbb{X}_{\boldsymbol{a}}$. ∎

Corollary 14. *If $\boldsymbol{\xi} \in \mathbb{D}_{\boldsymbol{a}}$ is a feasible direction, $\{\boldsymbol{b} \mid \boldsymbol{b} \in \mathbb{B}_+,\ \exists t \geq 0(\boldsymbol{b} = \boldsymbol{a} + t\boldsymbol{\xi})\} \subseteq \mathbb{X}_{\boldsymbol{a}}$.*

Lemma 15. *For balance states $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{B}$, if $Z_{\boldsymbol{b}} \subseteq Z_{\boldsymbol{a}}$, then $\mathbb{D}_{\boldsymbol{a}} \subseteq \mathbb{D}_{\boldsymbol{b}}$.*

Proof. For $\boldsymbol{\xi} = 0 \in \mathbb{D}_{\boldsymbol{a}}$, we trivially obtain that $\boldsymbol{\xi} = 0 \in \mathbb{D}_{\boldsymbol{b}}$.
For any $\boldsymbol{\xi} \in \mathbb{D}_{\boldsymbol{a}} - \{0\}$, there exists $t > 0$ and $\boldsymbol{p} \in \mathbb{X}_{\boldsymbol{a}}$, where $\boldsymbol{p} = \boldsymbol{a} + t\boldsymbol{\xi}$. Take

$$\epsilon = \min_{i \notin Z_{\boldsymbol{a}}} \frac{\begin{bmatrix} \boldsymbol{b} \\ C - \boldsymbol{b} \end{bmatrix}_i}{\begin{bmatrix} \boldsymbol{a} \\ C - \boldsymbol{a} \end{bmatrix}_i}.$$

By $Z_{\boldsymbol{b}} \subseteq Z_{\boldsymbol{a}}$, we have $\epsilon > 0$ and $\epsilon \begin{bmatrix} \boldsymbol{a} \\ C - \boldsymbol{a} \end{bmatrix} \leq \begin{bmatrix} \boldsymbol{b} \\ C - \boldsymbol{b} \end{bmatrix}$. By Lemma 11, we have for all $\boldsymbol{p} \in \mathbb{X}_{\boldsymbol{a}}$, $\boldsymbol{b} + \epsilon t\boldsymbol{\xi} = \boldsymbol{b} + \epsilon(\boldsymbol{p} - \boldsymbol{a}) \in \mathbb{X}_{\boldsymbol{b}}$. Given $\epsilon t > 0$, we have $\boldsymbol{\xi} \in \mathbb{D}_{\boldsymbol{b}}$. Therefore, $\mathbb{D}_{\boldsymbol{a}} \subseteq \mathbb{D}_{\boldsymbol{b}}$. ∎

Lemma 16. *Let $\phi(\boldsymbol{b}; \Pi)$ denote the throughput starting from initial state $\boldsymbol{b}$ under set of feasible **paths** Π. For any state $\boldsymbol{b}$, if $\Pi' \subseteq \Pi$,*

$$\phi(\boldsymbol{b}; \Pi) \geq \phi(\boldsymbol{b}; \Pi'). \tag{13}$$

Let $\mathbb{F}_{\boldsymbol{b}} \subseteq \mathbb{R}^{|\Pi|}, \mathbb{F}'_{\boldsymbol{b}} \subseteq \mathbb{R}^{|\Pi'|}$ denote the set of feasible flows from $\boldsymbol{b}$ under Π, Π', respectively. Throughput equality is attained when all paths in $\Pi - \Pi'$ are unused or in other words, if $\boldsymbol{f} \in \mathbb{F}_{\boldsymbol{b}}$, $f_\pi = 0$ holds for any $\pi \notin \Pi'$.

Proof. Let $\mathbb{X}_{\boldsymbol{b}}, \mathbb{X}'_{\boldsymbol{b}}$ denote the set of reachable states from $\boldsymbol{b}$, under feasible paths Π, Π', with routing matrices R, R' respectively.
Since, $\Pi' \subseteq \Pi$ implies there exists a matrix $P \in \{0,1\}^{|\Pi| \times |\Pi'|}$, such that $R' = RP$. Note that every row of P can have *utmost* one non-zero entry and every column of P has *exactly* one non-zero entry. In particular, $P_{ij} = 1$ if $\Pi_i = \Pi'_j$ and 0 otherwise. Consequently, for all $\boldsymbol{g} \in \mathbb{R}^{|\Pi'|}$, there exists an $\boldsymbol{f} = P\boldsymbol{g} \in \mathbb{R}^{|\Pi|}$ satisfying $R\boldsymbol{f} = RP\boldsymbol{g} = R'\boldsymbol{g}$ and $\Delta R\boldsymbol{f} = \Delta R'\boldsymbol{g}$.

Mapping the reachable states.
For any $\boldsymbol{p} \in \mathbb{X}'_{\boldsymbol{b}}$, there exists a $k \in \mathbb{N}$, a sequence of flows $\boldsymbol{g}^{(0:k-1)}$ and a sequence of transient states

$\boldsymbol{b}^{(0:k)}$, where $\boldsymbol{b}^{(0)} = \boldsymbol{b}$, $\boldsymbol{b}^{(k)} = \boldsymbol{p}$, and for all $j \in [k]$,

$$\boldsymbol{b}^{(j+1)} = \boldsymbol{b}^{(j)} - \Delta R' \boldsymbol{g}^{(j)}, \quad \boldsymbol{g}^{(j)} \geq 0, \quad R'\boldsymbol{g}^{(j)} \leq \begin{bmatrix} \boldsymbol{b}^{(j)} \\ \boldsymbol{C} - \boldsymbol{b}^{(j)} \end{bmatrix}.$$

Now let $\boldsymbol{f}^{(j)} = P\boldsymbol{g}^{(j)}$ for each $j \in [k]$. We may directly obtain

$$\boldsymbol{b}^{(j+1)} = \boldsymbol{b}^{(j)} - \Delta R \boldsymbol{f}^{(j)}, \quad \boldsymbol{f}^{(j)} \geq 0, \quad R\boldsymbol{f}^{(j)} \leq \begin{bmatrix} \boldsymbol{b}^{(j)} \\ \boldsymbol{C} - \boldsymbol{b}^{(j)} \end{bmatrix}.$$

This implies $\boldsymbol{f} = \sum_{j=0}^{k-1} P\boldsymbol{g}^{(j)} = P\boldsymbol{g}$ is also feasible that leads to a reachable state $\boldsymbol{p} = \boldsymbol{b} - \Delta R'\boldsymbol{g} = \boldsymbol{b} - \Delta R\boldsymbol{f}$. Hence, $\boldsymbol{p} \in \mathbb{X}_{\boldsymbol{b}}$, and since $\boldsymbol{p}$ is an arbitrary element in $\mathbb{X}'_{\boldsymbol{b}}$, we have $\mathbb{X}_{\boldsymbol{b}'} \subseteq \mathbb{X}_{\boldsymbol{b}}$.

Mapping Throughput Functions.

For any $\boldsymbol{g} \in \mathbb{R}^{|\Pi'|}$, there exists $\boldsymbol{f} = P\boldsymbol{g} \in \mathbb{R}^{|\Pi|}$, where $1^\top \boldsymbol{f} = 1^\top P\boldsymbol{g} = 1^\top \boldsymbol{g}$ and $R\boldsymbol{f} = RP\boldsymbol{g} = R'\boldsymbol{g}$. Hence, $\left(\boldsymbol{g} \geq 0,\ R'\boldsymbol{g} \leq \begin{bmatrix} \boldsymbol{b} \\ \boldsymbol{C} - \boldsymbol{b} \end{bmatrix},\ \Delta R'\boldsymbol{g} = 0\right) \Longrightarrow \left(\boldsymbol{f} \geq 0,\ R\boldsymbol{f} \leq \begin{bmatrix} \boldsymbol{b} \\ \boldsymbol{C} - \boldsymbol{b} \end{bmatrix},\ \Delta R\boldsymbol{f} = 0\right)$, which further indicates

$$\Psi' \triangleq \left\{ 1^\top \boldsymbol{g} \middle| \boldsymbol{g} \geq 0,\ R'\boldsymbol{g} \leq \begin{bmatrix} \boldsymbol{b} \\ \boldsymbol{C} - \boldsymbol{b} \end{bmatrix},\ \Delta R'\boldsymbol{g} = 0 \right\} \subseteq \left\{ 1^\top \boldsymbol{f} \middle| \boldsymbol{f} \geq 0,\ R\boldsymbol{f} \leq \begin{bmatrix} \boldsymbol{b} \\ \boldsymbol{C} - \boldsymbol{b} \end{bmatrix},\ \Delta R\boldsymbol{f} = 0 \right\} \triangleq \Psi.$$

Let $\psi(\boldsymbol{a}; \widetilde{\Pi})$ denote the one-step maximum throughput at state $\boldsymbol{a}$ under arbitrary set of feasible flows $\widetilde{\Pi}$. By definition of ψ, we have

$$\psi(\boldsymbol{a}; \Pi) = \sup \Psi \geq \sup \Psi' = \psi(\boldsymbol{a}; \Pi').$$

Equality Case. Assume for any feasible flow $\boldsymbol{f} \in \mathbb{F}_{\boldsymbol{b}}$, $f_\pi = 0$ holds for any $\pi \notin \Pi'$. Based on the construction of P, $(PP^\top)_{\pi,\pi} = 0$ if $\pi \notin \Pi'$ and $(PP^\top)_{\pi,\pi} = 1$ if $\pi \in \Pi'$. So, for this special case, we can argue $PP^\top \boldsymbol{f} = \boldsymbol{f}$.

Let $\boldsymbol{p}$ be an arbitrary state in $\mathbb{X}_{\boldsymbol{b}}$. Then, there exists $\boldsymbol{f} \in \mathbb{F}_{\boldsymbol{b}}$ such that $\boldsymbol{p} = \boldsymbol{b} - \Delta R\boldsymbol{f}$. Construct $\boldsymbol{g} = P^\top \boldsymbol{f}$, such that $-\Delta R'\boldsymbol{g} = -\Delta RPP^\top \boldsymbol{f} = -\Delta R\boldsymbol{f}$. $\boldsymbol{g} \in \mathbb{F}'_{\boldsymbol{b}}$. Similar to the previous argument, $\boldsymbol{p} = \boldsymbol{b} - \Delta R\boldsymbol{f} = \boldsymbol{b} - \Delta R'\boldsymbol{g} \in \mathbb{X}'_{\boldsymbol{b}}$. Thus, $\mathbb{X}_{\boldsymbol{b}} \subseteq \mathbb{X}'_{\boldsymbol{b}}$. This, with the previously proved $\mathbb{X}'_{\boldsymbol{b}} \subseteq \mathbb{X}_{\boldsymbol{b}}$ implies $\mathbb{X}_{\boldsymbol{b}} = \mathbb{X}'_{\boldsymbol{b}}$.

Further, $P\boldsymbol{g} = PP^\top \boldsymbol{f} = \boldsymbol{f}$. Thus, for any $\boldsymbol{f}$ with $\left(\boldsymbol{f} \geq 0,\ R\boldsymbol{f} \leq \begin{bmatrix} \boldsymbol{b} \\ \boldsymbol{C} - \boldsymbol{b} \end{bmatrix},\ \Delta R\boldsymbol{f} = 0\right)$ we have $1^\top \boldsymbol{g} = 1^\top P^\top \boldsymbol{f} = 1^\top \boldsymbol{f}$, $R'\boldsymbol{g} = RP\boldsymbol{g} = R\boldsymbol{f}$, and $\left(\boldsymbol{g} \geq 0,\ R'\boldsymbol{g} \leq \begin{bmatrix} \boldsymbol{b} \\ \boldsymbol{C} - \boldsymbol{b} \end{bmatrix},\ \Delta R'\boldsymbol{g} = 0\right)$. This indicates

$$\Psi = \left\{ 1^\top \boldsymbol{f} \middle| \boldsymbol{f} \geq 0,\ R\boldsymbol{f} \leq \begin{bmatrix} \boldsymbol{b} \\ \boldsymbol{C} - \boldsymbol{b} \end{bmatrix},\ \Delta R\boldsymbol{f} = 0 \right\} \subseteq \left\{ 1^\top \boldsymbol{g} \middle| \boldsymbol{g} \geq 0,\ R'\boldsymbol{g} \leq \begin{bmatrix} \boldsymbol{b} \\ \boldsymbol{C} - \boldsymbol{b} \end{bmatrix},\ \Delta R'\boldsymbol{g} = 0 \right\} = \Psi'.$$

Given $\Psi' \subseteq \Psi$, we have $\Psi = \Psi'$ and

$$\psi(\boldsymbol{a}; \Pi) = \sup \Psi = \sup \Psi' = \psi(\boldsymbol{a}; \Pi').$$

In summary,

$$\phi(\boldsymbol{b}; \Pi) = \sup_{\boldsymbol{a} \in \mathbb{X}_{\boldsymbol{b}}} \psi(\boldsymbol{a}; \Pi) \overset{(*)}{\geq} \sup_{\boldsymbol{a} \in \mathbb{X}_{\boldsymbol{b}}} \psi(\boldsymbol{a}; \Pi') \overset{(*)}{\geq} \sup_{\boldsymbol{a} \in \mathbb{X}^{\text{free}}_{\boldsymbol{b}}} \psi(\boldsymbol{a}; \Pi') = \phi(\boldsymbol{b}; \Pi').$$

When equality condition holds, both inequalities pointed by $(*)$ are tight. ∎

Lemma 17. *Assume the set of feasible paths Π is constant. Let $\mathbb{L}_{\boldsymbol{b}}$ denote the set of deadlocked channels under balance state $\boldsymbol{b}$. If state $\boldsymbol{a}$ satisfies $a_i = b_i, \forall i \in \mathbb{L}_{\boldsymbol{b}}$, then $\mathbb{L}_{\boldsymbol{b}} \subseteq \mathbb{L}_{\boldsymbol{a}}$.*

Proof. We prove the statement by contradiction. Assume the set $\mathbb{L}_\Delta \triangleq \mathbb{L}_{\boldsymbol{b}} - \mathbb{L}_{\boldsymbol{a}} \neq \varnothing$. By definition, there exists $\boldsymbol{a}' \in \mathbb{X}_{\boldsymbol{a}}$, such that $\forall j \in \mathbb{L}_\Delta : a'_j \neq a_j = b_j$. This also means $\exists \boldsymbol{f} \in \mathbb{F}_{\boldsymbol{a}}$, such that $-\Delta R_j \boldsymbol{f} \neq 0$ for all $j \in \mathbb{L}_\Delta$.

Since deadlocked channels' balances are constant across all reachable states (Prop. 9) and we can move to any arbitrary interior state along feasible directions (Thm. 13), $\exists \boldsymbol{b}' \in \mathbb{X}_{\boldsymbol{b}}$, such that $\forall i \in \mathbb{L}_{\boldsymbol{b}} : b'_i = b_i$ and $\forall j \notin \mathbb{L}_{\boldsymbol{b}} : b'_j \neq b_j, 0 < b'_j < C_j$.

Since $a_i = b_i = b'_i$ for all $i \in \mathbb{L}_{\boldsymbol{b}}$, $Z_{\boldsymbol{b}'} \subseteq Z_{\boldsymbol{a}}$, which leads to $\mathbb{D}_{\boldsymbol{a}} \subseteq \mathbb{D}_{\boldsymbol{b}'}$ by Lemma 15.

Note that the direction $\boldsymbol{a}' - \boldsymbol{a}$ corresponds to changing the balances of the channels in $\mathbb{L}_\Delta$. Since $\boldsymbol{a}' \in \mathbb{X}_{\boldsymbol{a}}$, $\boldsymbol{a}' - \boldsymbol{a} \in \mathbb{D}_a \subseteq \mathbb{D}_{\boldsymbol{b}'}$. This means we can construct a new state $\boldsymbol{b}'' \in \mathbb{X}_{\boldsymbol{b}'}$ also with changed balances for the channels in $\mathbb{L}_\Delta$. In other words, $t > 0$, $b''_j = b'_j - t\Delta R_j \boldsymbol{f} \neq b'_j = b_j$ for all $j \in \mathbb{L}_\Delta$. Further, transitivity of reachable states (Prop. 7) implies $\boldsymbol{b}'' \in \mathbb{X}_{\boldsymbol{b}}$.

We have constructed $\boldsymbol{b}''$, a state reachable from $\boldsymbol{b}$ but with changed balances for all channels in $\mathbb{L}_\Delta \subseteq \mathbb{L}_{\boldsymbol{b}}$. This contradicts the definition of deadlocked states. Therefore, $\mathbb{L}_{\boldsymbol{b}} - \mathbb{L}_{\boldsymbol{a}} = \varnothing$, which implies $\mathbb{L}_{\boldsymbol{b}} \subseteq \mathbb{L}_{\boldsymbol{a}}$. ∎

Lemma 18. *For any state $\boldsymbol{a}$ and $\boldsymbol{b}$, there exists a state $\boldsymbol{c}$, where $\mathbb{L}_{\boldsymbol{c}} \supseteq \mathbb{L}_{\boldsymbol{a}} \cup \mathbb{L}_{\boldsymbol{b}}$.*

Proof. We construct $\boldsymbol{c}$ where $\forall i \in \mathbb{L}_{\boldsymbol{a}} : c_i = a_i$ and $\forall i \notin \mathbb{L}_{\boldsymbol{a}} : c_i = b_i$. By Lemma 17, we immediately obtain $\mathbb{L}_{\boldsymbol{a}} \subseteq \mathbb{L}_{\boldsymbol{c}}$.

Consider any $\boldsymbol{f} \in \mathbb{F}_{\boldsymbol{c}}$. There exists $k \in \mathbb{N}$, a sequence of flows $\left(\boldsymbol{f}^{(j)}\right)_{j=0}^{k-1}$ and a sequence of transient states $\left(\boldsymbol{c}^{(j)}\right)_{j=0}^{k}$, where $\boldsymbol{f} = \sum_{j=0}^{k-1} \boldsymbol{f}^{(j)}$, $\boldsymbol{c}^{(0)} = \boldsymbol{c}$, and for each $j \in [k]$,

$$\boldsymbol{c}^{(j+1)} = \boldsymbol{c}^{(j)} - \Delta R \boldsymbol{f}^{(j)}, \quad \boldsymbol{f}^{(j)} \geq 0, \quad R\boldsymbol{f}^{(j)} \leq \begin{bmatrix} \boldsymbol{c}^{(j)} \\ \boldsymbol{C} - \boldsymbol{c}^{(j)} \end{bmatrix}.$$

Let $\boldsymbol{b}^{(0)} = \boldsymbol{b}$ and for each $j \in [k]$, if $\boldsymbol{f}^{(j)}$ is feasible, $\boldsymbol{b}^{(j+1)} = \boldsymbol{b}^{(j)} - \Delta R \boldsymbol{f}^{(j)}$. We show that $\boldsymbol{f}^{(j)}$ is indeed feasible by considering an arbitrary channel $i \in [|E|]$.

- $i \in \mathbb{L}_{\boldsymbol{a}}$: By construction and Lemma 17, $i \in \mathbb{L}_{\boldsymbol{c}}$. Since a deadlocked channel can never sustain any flow in either direction in any feasible flow (Prop. 10.1), $R_i \boldsymbol{f}^{(j)} = R_{i+|E|} \boldsymbol{f}^{(j)} = 0$ for each $j \in [k]$. Therefore, $R_i \boldsymbol{f}^{(j)} = 0 \leq \begin{bmatrix} \boldsymbol{b}^{(j)} \\ \boldsymbol{C} - \boldsymbol{b}^{(j)} \end{bmatrix}_i = b_i^{(j)}$ and $R_{i+|E|} \boldsymbol{f}^{(j)} = 0 \leq \begin{bmatrix} \boldsymbol{b}^{(j)} \\ \boldsymbol{C} - \boldsymbol{b}^{(j)} \end{bmatrix}_{i+|E|} = C_i - b_i^{(j)}$ for each $j \in [k]$.
- $i \notin \mathbb{L}_{\boldsymbol{a}}$: Starting with $c_i^{(0)} = c_i = b_i = b_i^{(0)}$, we have for all $j \in [k] \cup \{k\}$,

$$c_i^{(j)} = c_i - \Delta R_i \sum_{j'=0}^{j-1} \boldsymbol{f}^{(j')} = b_i - \Delta R_i \sum_{j'=0}^{j-1} \boldsymbol{f}^{(j')} = b_i^{(j)}.$$

  Since $\boldsymbol{f}^{(0)}$ is feasible from $\boldsymbol{b}$ trivially, by extension, for $j \in [k]$, $R_i \boldsymbol{f}^{(j)} \leq \begin{bmatrix} \boldsymbol{c}^{(j)} \\ \boldsymbol{C} - \boldsymbol{c}^{(j)} \end{bmatrix}_i = b_i^{(j)}$ and $R_{i+|E|} \boldsymbol{f}^{(j)} \leq \begin{bmatrix} \boldsymbol{c}^{(j)} \\ \boldsymbol{C} - \boldsymbol{c}^{(j)} \end{bmatrix}_{i+|E|} = C_i - b_i^{(j)}$.

To sum up, for all $\boldsymbol{f} \in \mathbb{F}_{\boldsymbol{c}}$, we have $\boldsymbol{f} \in \mathbb{F}_{\boldsymbol{b}}$. Since for all $\boldsymbol{f} \in \mathbb{F}_{\boldsymbol{b}}$, channels in $\mathbb{L}_{\boldsymbol{b}}$ experience no change in balance (Def. 7), we have $\mathbb{L}_{\boldsymbol{b}} \subseteq \mathbb{L}_{\boldsymbol{c}}$. Combining it with $\mathbb{L}_{\boldsymbol{a}} \subseteq \mathbb{L}_{\boldsymbol{c}}$, we justify the lemma. ∎

A.3 Formal Proof of Theorem 1

Proof. For every corner state $\boldsymbol{c}^{(\pi)}$, let $\boldsymbol{p}^{(\pi)}$ be the corresponding interior point such that $\boldsymbol{p}^{(\pi)} \in \mathbb{X}_{\boldsymbol{c}^{(\pi)}}$ and $\pi \in \left[2^{|E|}\right]$. Let $\boldsymbol{\xi}^{(\pi)} = \boldsymbol{p}^{(\pi)} - \boldsymbol{c}^{(\pi)}$ be the state offset. By definition of set of directions

$\mathbb{D}_{\boldsymbol{c}^{(\pi)}}$, we have offset $\boldsymbol{\xi}^{(\pi)} \in \mathbb{D}_{\boldsymbol{c}^{(\pi)}}$. For each channel i, we have either $i \in Z_{\boldsymbol{c}^{(\pi)}}$ or $i + |E| \in Z_{\boldsymbol{c}^{(\pi)}}$ since one of the two ends of every channel has 0 balance.

- If $i \in Z_{\boldsymbol{c}^{(\pi)}}$, $c_i^{(\pi)} = 0$. By $0 < p_i^{(\pi)} < C_i$, we have $\xi_i^{(\pi)} > 0$;
- If $i + |E| \in Z_{\boldsymbol{c}^{(\pi)}}$, $c_i^{(\pi)} = C_i$. By $0 < p_i^{(\pi)} < C_i$, we have $\xi_i^{(\pi)} < 0$.

State of one channel on the corner determines the $\pm$ sign of its corresponding δ entry. Each open orthant in $\mathbb{R}^{|E|}$ contains exactly one $\boldsymbol{\xi}^{(\pi)}$ vector. Across all the corners, all of the $2^{|E|}$ (open) orthants in $\mathbb{R}^{|E|}$ are covered.

Interior Points. Consider an arbitrary interior state $\boldsymbol{p} \in \mathbb{B}_+$. By Lemma 15, $\boldsymbol{\xi}^{(\pi)} \in \mathbb{D}_{\boldsymbol{c}^{(\pi)}} \subseteq \mathbb{D}_{\boldsymbol{p}}$ for all $\pi \in \left[2^{|E|}\right]$, because $Z_{\boldsymbol{c}^{(\pi)}} \supseteq \varnothing = Z_{\boldsymbol{p}}$. By putting all the $2^{|E|}$ vectors $\left\{\boldsymbol{\xi}^{(\pi)}\right\}$ in its columns, we construct a matrix Ξ.

We claim that for any $\boldsymbol{\xi} \in \mathbb{R}^{|E|}$, there exists $\boldsymbol{f} \geq 0$, such that $\Xi \boldsymbol{f} = \boldsymbol{\xi}$. Let us assume otherwise. By Farkas' Lemma, there must exist a $\boldsymbol{\gamma} \in \mathbb{R}^{|E|}$, such that $\Xi^\top \boldsymbol{\gamma} \geq 0$ and $\boldsymbol{\xi}^\top \boldsymbol{\gamma} < 0$. If $\boldsymbol{\gamma} \neq 0$, because columns of Ξ covers all the orthants, there must exist a column $\boldsymbol{\xi}'$ in Ξ, where for all i, if $\gamma_i \neq 0$, then $\xi_i' \gamma_i < 0$. This gives us a negative entry in $\Xi^\top \boldsymbol{\gamma}$, contradicting $\Xi^\top \boldsymbol{\gamma} \geq 0$, so $\boldsymbol{\gamma}$ must equal 0. However, this contradicts $\boldsymbol{\xi}^\top \boldsymbol{\gamma} < 0$.

Therefore, there exists $\boldsymbol{f} \geq 0$ where $\Xi \boldsymbol{f} = \boldsymbol{\xi}$ for any $\boldsymbol{\xi} \in \mathbb{R}^{|E|}$. Since $\boldsymbol{\xi} \in \mathbb{D}_{\boldsymbol{p}} \forall \boldsymbol{\xi}$, by convexity of the feasible direction set (Prop. 5), $\mathbb{D}_{\boldsymbol{p}} = \mathbb{R}^{|E|}$.

Since all interior states along a feasible direction are reachable (Corollary 14) and $\mathbb{D}_{\boldsymbol{p}} = \mathbb{R}^{|E|}$, $\mathbb{B}_+ \subseteq \mathbb{X}_{\boldsymbol{p}}$. Thus, $\boldsymbol{C}/2$ is reachable from any interior point $\boldsymbol{p}$.

Boundary Points. We consider an arbitrary boundary state $\boldsymbol{q}$ with a corner $\boldsymbol{c}^{(\pi)}$ where $Z_{\boldsymbol{c}^{(\pi)}} \supseteq Z_{\boldsymbol{q}}$. Let $\underline{q}$ denote the minimum positive entry in vector $\begin{bmatrix} \boldsymbol{q} \\ \boldsymbol{C} - \boldsymbol{q} \end{bmatrix}$. We construct

$$\epsilon = 0.9 \min \left\{ 1, \ \frac{\underline{q}}{\|\boldsymbol{\xi}^{(\pi)}\|_\infty} \right\}.$$

- For $i \notin Z_{\boldsymbol{q}}$, we have

$$0 \leq q_i - \frac{q_i \left|\xi_i^{(\pi)}\right|}{\|\boldsymbol{\xi}^{(\pi)}\|_\infty} < q_i + \epsilon \xi_i^{(\pi)} < q_i + \frac{(C_i - q_i)\left|\xi_i^{(\pi)}\right|}{\|\boldsymbol{\xi}^{(\pi)}\|_\infty} \leq C_i;$$

- For $i \in Z_{\boldsymbol{q}}$ and $i \leq |E|$, we have $q_i = c_i^{(\pi)} = 0$, $\xi_i^{(\pi)} > 0$, and

$$0 < \epsilon \xi_i^{(\pi)} = q_i + \epsilon \xi_i^{(\pi)} < q_i + \xi_i^{(\pi)} = c_i^{(\pi)} + \xi_i^{(\pi)} = p_i^{(\pi)} < C_i;$$

- For $i \in Z_{\boldsymbol{q}}$ and $i > |E|$, we have $0 < q_i + \epsilon \xi_i^{(\pi)} < C_i$ analogously.

Effectively, $\boldsymbol{q} + \epsilon \boldsymbol{\xi}^{(\pi)}$ is an interior state which lies on the direction $\boldsymbol{\xi}^{(\pi)}$ from $\boldsymbol{q}$. Since $\boldsymbol{q}$ is less constrained than $\boldsymbol{c}^{(\pi)}$, we know $\boldsymbol{\xi}^{(\pi)} \in \mathbb{D}_{\boldsymbol{q}}$ (Lemma 15). By Corollary 14, the interior state $\boldsymbol{q} + \epsilon \boldsymbol{\xi}^{(\pi)} \in \mathbb{X}_{\boldsymbol{q}}$ since it is along a feasible direction.

The previous argument shows that $\boldsymbol{C}/2$ is reachable from any interior state. Thus, $\boldsymbol{C}/2 \in \mathbb{X}_{\boldsymbol{q}+\epsilon\boldsymbol{\xi}^{(\pi)}}$. By transitivity of reachable states, (Proposition 7), $\mathbb{X}_{\boldsymbol{q}+\epsilon\boldsymbol{\xi}^{(\pi)}} \subseteq \mathbb{X}_{\boldsymbol{q}}$, which implies $\boldsymbol{C}/2 \in \mathbb{X}_{\boldsymbol{q}}$, i.e., $\boldsymbol{C}/2$ is also reachable from the boundary state $\boldsymbol{q}$.

Throughput Insensitivity. Since $\boldsymbol{C}/2$, the global ψ maximizer by Proposition 8, is reachable from any state $\boldsymbol{p} \in \mathbb{B}$,

$$\phi(\boldsymbol{p}) = \sup_{\boldsymbol{q} \in \mathbb{X}_{\boldsymbol{p}}} \psi(\boldsymbol{q}) = \psi\left(\frac{\boldsymbol{C}}{2}\right). \qquad \blacksquare$$

A.4 Proof of Theorem 2

Proof. By Lemma 18, we may assert the existence of a grand deadlocking state $\boldsymbol{b}$, where $\mathbb{L}_{\boldsymbol{a}} \subseteq \mathbb{L}_{\boldsymbol{b}}$ for any other state $\boldsymbol{a}$. In such cases, $|\mathbb{L}_{\boldsymbol{b}}|$ is maximized.

Filtering out unused paths and deadlocked edges. Let $E_{\text{free}} = E - \mathbb{L}_{\boldsymbol{b}}$ and $\Pi_{\text{free}} = \{\pi \in [\Pi] \mid \forall j \in \mathbb{L}_{\boldsymbol{b}} : R_{j,\pi} = R_{j+|E|,\pi} = 0\}$. Based on these subsets, we construct matrices $P \in \{0,1\}^{|E|\times|E_{\text{free}}|}$ and $Q \in \{0,1\}^{|\Pi|\times|\Pi_{\text{free}}|}$ that extracts the rows in E_{free} and columns in Π_{free}, respectively. Every row of P, Q can have *utmost* one non-zero entry and every column of P, Q has *exactly* one non-zero entry. In particular,

$$P_{ij} = \begin{cases} 1, & E_i = (E_{\text{free}})_j; \\ 0, & \text{otherwise.} \end{cases}, \qquad Q_{ij} = \begin{cases} 1, & \Pi_i = (\Pi_{\text{free}})_j; \\ 0, & \text{otherwise.} \end{cases}.$$

In addition, we define $\hat{P} = \begin{bmatrix} P & P \end{bmatrix}$ and take submatrices $\widetilde{R} = \hat{P}^\top RQ$ and $\Delta\widetilde{R} = P^\top \Delta RQ$.

We point out that for all $i \in \mathbb{L}_{\boldsymbol{b}}$ and $\boldsymbol{g} \in \mathbb{R}^{|\Pi_{\text{free}}|}$, we have the π-th column $Q_\pi = 0$ for all $\pi : R_{i,\pi} = 1$. Therefore $R_i Q\boldsymbol{g} = 0$ and analogously, $R_{i+|E|}Q\boldsymbol{g} = 0$. An important implication is that any vector $RQ\boldsymbol{g}$ has 0 entries on the positions that the selecting matrix $\hat{P}$ throws away. Similarly, any vector $\Delta RQ\boldsymbol{g}$ has 0 entries on the positions that P throws away. Thus, for any $\boldsymbol{x} \in \mathbb{R}^{2|E|}$ and $\boldsymbol{y} \in \mathbb{R}^{|E|}$, we have the following bijection.

$$\boldsymbol{x} = RQ\boldsymbol{g} \iff \hat{P}^\top \boldsymbol{x} = \hat{P}^\top RQ\boldsymbol{g} = \widetilde{R}\boldsymbol{g}, \qquad \boldsymbol{y} = \Delta RQ\boldsymbol{g} \iff P^\top \boldsymbol{y} = P^\top \Delta RQ\boldsymbol{g} = \Delta\widetilde{R}\boldsymbol{g}.$$

These equalities support (15).

Throughput Mapping. Since some of the edges used by flows outside of Π_{free} are deadlocked, by Proposition 10.2, we have $\forall \pi \notin \Pi_{\text{free}} : f_\pi = 0$, as long as $\boldsymbol{f} \in \mathbb{F}_{\boldsymbol{b}}$. Since these 0 flows cannot contribute throughput, by Lemma 16,

$$\phi(\boldsymbol{b}; \Pi) = \phi(\boldsymbol{b}; \Pi_{\text{free}}) \tag{14}$$

Let $\mathbb{X}_{\boldsymbol{a}}^{\text{free}}$ denote the set of reachable states in $\mathbb{R}^{|E|}$ (without removing channels outside E_{free}) using only paths in Π_{free}. For any state $\boldsymbol{a}$ (not necessarily equal to $\boldsymbol{b}$),

$$\begin{aligned}
\phi(\boldsymbol{a}; \Pi) &\geq \phi(\boldsymbol{a}; \Pi_{\text{free}}) \\
&= \sup_{\boldsymbol{a}' \in \mathbb{X}_{\boldsymbol{a}}^{\text{free}}} \psi(\boldsymbol{a}; \Pi_{\text{free}}) \\
&= \sup_{\boldsymbol{a}' \in \mathbb{X}_{\boldsymbol{a}}^{\text{free}}} \left\{ 1^\top \boldsymbol{g} \;\middle|\; \boldsymbol{g} \geq 0,\ RQ\boldsymbol{g} \leq \begin{bmatrix} \boldsymbol{a}' \\ \boldsymbol{C} - \boldsymbol{a}' \end{bmatrix},\ \Delta RQ\boldsymbol{g} = 0 \right\} \\
&= \sup_{\boldsymbol{a}' \in \mathbb{X}_{\boldsymbol{a}}^{\text{free}}} \left\{ 1^\top \boldsymbol{g} \;\middle|\; \boldsymbol{g} \geq 0,\ \widetilde{R}\boldsymbol{g} \leq \hat{P} \begin{bmatrix} \boldsymbol{a}' \\ \boldsymbol{C} - \boldsymbol{a}' \end{bmatrix},\ \Delta\widetilde{R}\boldsymbol{g} = 0 \right\} \qquad (15) \\
&= \widetilde{\phi}(P^\top \boldsymbol{a}; \Pi_{\text{free}}), \qquad (16)
\end{aligned}$$

where $\widetilde{\phi}(P^\top \boldsymbol{a}; \Pi_{\text{free}})$ describes the throughput at initial state $P^\top \boldsymbol{a}$ on a credit network characterized by routing matrix $\widetilde{R}$.

Reachable States Equivalence. Let $\widetilde{\mathbb{B}} = \left\{ \boldsymbol{x} \in \mathbb{R}^{|E_{\text{free}}|} \middle| 0 \leq \boldsymbol{x} \leq P^\top \boldsymbol{C} \right\}$ and $\widetilde{\mathbb{B}}_+ = \left\{ \boldsymbol{x} \in \mathbb{R}^{|E_{\text{free}}|} \middle| 0 \prec \boldsymbol{x} \prec P^\top \boldsymbol{C} \right\}$. We also let $\widetilde{\mathbb{X}}_{\boldsymbol{q}} \subseteq \mathbb{R}^{|E_{\text{free}}|}$ denote the set of reachable states in the credit network (after removing channels outside E_{free}) from state $\boldsymbol{q}$.

Now we prove that for any state $\boldsymbol{a}, \boldsymbol{c} \in \mathbb{B}$, if $\boldsymbol{a} \in \mathbb{X}_{\boldsymbol{c}}^{\text{free}}$, then $P^\top \boldsymbol{a} \in \widetilde{\mathbb{X}}_{P^\top \boldsymbol{c}}$.

Let $k \in \mathbb{N}$ and $\left(\boldsymbol{f}^{(j)}\right)_{j=0}^{k-1}$ be the sequence of flows where $\boldsymbol{c}^{(0)} = \boldsymbol{c},\ \boldsymbol{c}^{(k)} = \boldsymbol{a}$, and for all $j \in [k]$,

$$\boldsymbol{c}^{(j+1)} = \boldsymbol{c}^{(j)} - \Delta R \boldsymbol{f}^{(j)}, \quad \boldsymbol{f}^{(j)} \geq 0, \quad R\boldsymbol{f}^{(j)} \leq \begin{bmatrix} \boldsymbol{c}^{(j)} \\ \boldsymbol{C} - \boldsymbol{c}^{(j)} \end{bmatrix}.$$

Since $\boldsymbol{a} \in \mathbb{X}'_{\boldsymbol{c}}$, no flow outside Π_{free} is used, and thus, $f_\pi^{(j)} = 0$ for all $\pi \notin \Pi_{\text{free}}$. As a result, $QQ^\top \boldsymbol{f}^{(j)} = \boldsymbol{f}^{(j)}$, and furthermore,

$$P^\top \Delta R \boldsymbol{f}^{(j)} = P^\top \Delta R Q Q^\top \boldsymbol{f}^{(j)} = \Delta \widetilde{R} Q^\top \boldsymbol{f}^{(j)}, \qquad \hat{P}^\top R \boldsymbol{f}^{(j)} = \hat{P}^\top R Q Q^\top \boldsymbol{f}^{(j)} = \widetilde{R} Q^\top \boldsymbol{f}^{(j)}.$$

By left-multiplying $P^\top$ (or $\hat{P}^\top$), we directly obtain $P^\top \boldsymbol{c}^{(0)} = P^\top \boldsymbol{c}$, $P^\top \boldsymbol{c}^{(k)} = P^\top \boldsymbol{a}$, and for all $j \in [k]$,

$$P^\top \boldsymbol{c}^{(j+1)} = P^\top \boldsymbol{c}^{(j)} - \Delta \widetilde{R} Q^\top \boldsymbol{f}^{(j)}, \quad Q^\top \boldsymbol{f}^{(j)} \geq 0, \quad \widetilde{R} Q^\top \boldsymbol{f}^{(j)} \leq \hat{P} \begin{bmatrix} \boldsymbol{c}^{(j)} \\ \boldsymbol{C} - \boldsymbol{c}^{(j)} \end{bmatrix}.$$

Hence, $\sum_{j=0}^{k-1} Q^\top \boldsymbol{f}^{(j)}$ is a feasible flow in a credit network with routing matrix $\widetilde{R}$ and initial state $P^\top \boldsymbol{c}$. The destination $P^\top \boldsymbol{a} \in \widetilde{\mathbb{X}}_{P^\top \boldsymbol{c}}$.

Throughput insensitivity of filtered credit network. We consider all the corners $\left\{\boldsymbol{c}^{(\pi)}\right\}$ where $\forall i \in \mathbb{L}_{\boldsymbol{b}}$ and $\pi \in 2^{|E_{\text{free}}|} : c_i^{(\pi)} = b_i$. From any $\boldsymbol{c}^{(\pi)}$, there exists $\boldsymbol{q}^{(\pi)} \in \mathbb{X}_{\boldsymbol{c}^{(\pi)}}$ such that $\forall i \in E_{\text{free}} : 0 < q_i^{(\pi)} < C_i$. Since $|\mathbb{L}_{\boldsymbol{b}}|$ is maximized and its deadlocked channels continue to be imbalanced at $\boldsymbol{c}^{(\pi)}$, by Lemma 17, we have $\mathbb{L}_{\boldsymbol{c}^{(\pi)}} = \mathbb{L}_{\boldsymbol{b}} = E - E_{\text{free}}$. Thus, all paths outside Π_{free} are not used, and we have $\boldsymbol{q}^{(\pi)} \in \mathbb{X}_{\boldsymbol{c}}^{\text{free}}$, which indicates $P\boldsymbol{q}^{(\pi)} \in \widetilde{\mathbb{X}}_{P^\top \boldsymbol{c}^{(\pi)}}$.

By definition, the states $\left\{P^\top \boldsymbol{c}^{(\pi)}\right\}$ are corners in $\widetilde{\mathbb{B}}$, while $\left\{P\boldsymbol{q}^{(\pi)}\right\}$ are interior states in $\widetilde{\mathbb{B}}_+$. So this sub-credit network is deadlock-free with balance state space $\widetilde{\mathbb{B}}$ and routing matrix $\widetilde{R}$. By Theorem 1, a deadlock-free credit network is insensitive to throughput. Thus, we know for all states $\widetilde{\boldsymbol{q}} \in \widetilde{\mathbb{B}}$, $\phi(\widetilde{\boldsymbol{q}}; \Pi_{\text{free}}) \equiv C$ for some constant $C \geq 0$.

Summary. Combine (14) with (16), and we get for any state $\boldsymbol{a} \in \mathbb{B}$,

$$\phi(\boldsymbol{a}; \Pi) \geq C = \phi(\boldsymbol{b}; \Pi_{\text{free}}) = \phi(\boldsymbol{b}; \Pi).$$

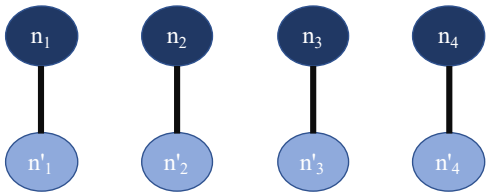
Hence, $\boldsymbol{b}$ is the worst corner (more generally, state) with lowest throughput. ∎

A.5 Formal Proof of Theorem 3

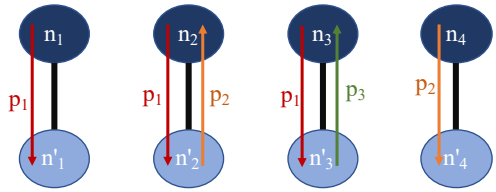
Proof. We first prove that detecting a full deadlock on a credit network $G(E, V)$ is NP-hard by reducing the boolean satisfiability (SAT) problem, a decision problem known to be NP-complete, to deadlock-detection. We then extend the reduction to establish that partial deadlock detection is also NP-hard.

We start with an arbitrary instance of the SAT problem, which determines satisfiability of a boolean expression in conjunctive normal form (CNF). The expression consists of ℓ literals ($x_1 \cdots x_\ell$) and C clauses ($c_1 \cdots c_C$).

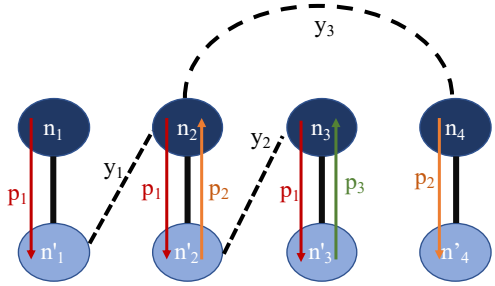
Preprocessing the CNF expression. We preprocess the expression to ensure that literals always appear in the same order in every clause. We then map it to an equivalent expression s' by inserting fresh variables (y_i's) between every pair of adjacent literals in any clause of the original CNF expression. For each ordered pair i, j ($i < j$), there are 4 cases in total where x_i (or $\neg x_i$) is adjacent to x_j (or $\neg x_j$). We create a new literal $y_{i,j,*}$ for each case. Notice that the total # of literals (ℓ') in s' is polynomial in ℓ, the original problem size. To ensure that the new literals have no effect on the satisfiability of the original expression, we add new clauses (exactly $\ell' - \ell$ new clauses). Let the total number of clauses in s' be C' respectively. Tab. 2 shows an example conversion for a



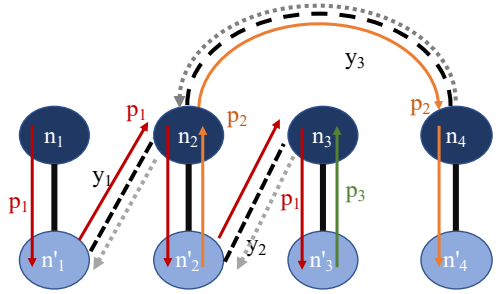
(a) 4 edges with their own endpoints are introduced for each of the 4 literals $x_1 \cdots x_4$.



(b) 3 flows added based on the directions given by negation of each literal in each clause.



(c) Dashed lines denote edges y_k added to connect the topology. Each y_k edge maps to one of 4 orientations for a pair of literals.



(d) Completed credit network by connecting flows corresponding to clauses the left to right and using dedicated (dotted) flows on the y_k edges from right to left.

Fig. 12. credit network conversion for the sample pre-processed CNF expression in Tab. 2. Deadlock detection on this credit network is equivalent to deciding whether the boolean expression is satisfiable.

particular CNF expression. It is fairly straightforward to see that the original CNF s and the altered expression s' are equisatisfiable: s' is satisfiable if and only if s is satisfiable.

Original Expr. s	$(x_1 \vee x_2 \vee x_3) \wedge (x_4 \vee \neg x_2) \wedge (\neg x_3)$
Ordered Expr.	$(x_1 \vee x_2 \vee x_3) \wedge (\neg x_2 \vee x_4) \wedge (\neg x_3)$
Fresh Variables	$(x_1 \vee y_1 \vee x_2 \vee y_2 \vee x_3) \wedge (\neg x_2 \vee y_3 \vee x_4) \wedge (\neg x_3)$
Final Expr. s'	$(x_1 \vee y_1 \vee x_2 \vee y_2 \vee x_3) \wedge (\neg x_2 \vee y_3 \vee x_4) \wedge (\neg x_3) \wedge (\neg y_1) \wedge (\neg y_2) \wedge (\neg y_3)$

Table 2. Example showing pre-processing for a CNF expression whose satisfiability is to be determined before mapping it to credit network deadlock detection.

Mapping the expression to a credit network. To construct the credit network on which deadlocks are detected, every x_i literal in the original expression is mapped to an edge e_i that is oriented in the vertical direction with two endpoints n_i and n_i'. Fig. 12a shows this for the example expression in Tab. 2. The clauses c_i dictate the paths p_i on the credit network and the direction of use on each edge. If x_i appears in negated form in any clause c_i, the path p_i will send flow on the edge x_i from n_i' towards n_i and vice-versa. As shown in Fig. 12b, based on clauses c_1 and c_2, p_1 in red sends flow in the downward direction on both edges e_1 and e_2 while p_2 (in orange) will send flow in the upward direction on x_2 and downward on the x_4.

The edges corresponding to the fresh variables y_k's act as intermediary edges that help us complete paths. If y_k is a fresh variable corresponding to $x_i \vee x_j$, y_k connects n_i' and n_j enabling a

path that runs downward on both x_i and x_j. We also add a *dedicated flow* in the opposite direction from n_j to n_i' to denote the clause $\neg y_k$. Such dedicated flows ensure that the intermediary edges can never be deadlocked unless the original edges themselves are deadlocked. Similarly, if y_m is a fresh variable for $x_i \vee x_j'$, y_m connects n_i' to n_j', enabling a path that runs downward on x_i and upward on x_j. A dedicated flow mapped to the clause $\neg y_m$ is also added from n_j' to n_i'. This can be extended to the other two orientations of e_i and e_j to give four possible ways of drawing y_k. Fig. 12c denotes the credit network after adding the fresh variables. Once we add the dedicated flows and connect the original paths or clauses, the final credit network looks like Fig. 12d.

Reduction. Effectively, we have a credit network instance $G(E, V)$ based on the original expression that we want to detect a full-deadlock on. In a fully deadlocked configuration for this credit network, the vertical edge $e_i(n_i, n_i')$ is unable to support downward flow if all the tokens are on the n_i' end (on the node that is below). Since downward flow maps to the non-negated x_i, $x_i = 1$ if all the tokens are at n_i' and $x_i = 0$ if the tokens are at n_i. The y_k edges cannot sustain left-right flow if all the tokens are all on its right end. Thus, $y_k = 1$ puts all its tokens on its right end.

If there exists a configuration with all edges of the above described credit network deadlocked, there must be a satisfying assignment to the original boolean expression. If all paths are deadlocked, each of the dedicated flows (gray dashes) must be bottlenecked because tokens are on the left end of the respective y_k edges ($y_k = 0$). This ensures that all added clauses on the fresh variables are true. However, if the tokens are on the left end of each y_k, the p_i paths associated with the original clauses cannot be bottlenecked on any of the y_k edges. Thus, for each path, one of the e_i's must be preventing flow in the corresponding direction. Such an e_i corresponds to a true literal in every clause ensuring that the original boolean expression is satisfiable.

If there exists a satisfying assignment to the original boolean expression s, then there exists a satisfying assignment to s' also by just setting all $y_k = 0$. But, this implies a full deadlock exists on the equivalent credit network. This is because one or more literals in every original clause must be true. The true literal in every original clause maps to a direction of its edge e_i that bottlenecks the associated p_i flow. The independent gray flows are trivially bottlenecked because a satisfying assignment sets $y_k = 0$ which puts all tokens on the left end of each of the y_k edges. Thus, in aggregate all paths in the credit network are deadlocked, thereby leaving us with a full deadlock on the edges.

To sum up, a solution to a full-deadlock problem of polynomial size constructed based on a SAT problem instance can be converted in polynomial time to a solution to the SAT problem. This yields the NP-hardness of the full-deadlock problem.

Partial Deadlocks While the above construction is for a full deadlock on the credit network, but adding extraneous edges with dedicated flows that cannot be deadlocked allows us to extend the reduction to partial deadlocks. In particular, after the above credit network construction with edges for an arbitrary boolean expression, consider adding k' edges with dedicated flows in both directions on each edge that make them all deadlock-free. The identification of a $k - k'$ deadlock on the k-edged final credit network is now equivalent to finding a satisfying assignment on the original boolean expression. A deadlock involving the $k - k'$ edges map to the original literals from the expression. We can use this to establish finding an $k - k'$ deadlock is NP-hard too for any k. In other words, finding a partial deadlock is also NP-hard. ■

B ADDITIONAL DETAILS AND RESULTS

B.1 Deadlock Detection Integer Linear Program

We set up the following Integer Linear Program to define the dependencies between imbalanced channels, the paths using them and the resulting set of deadlocked channels. We define $x_{(u,v)} \in \{0,1\}$ to denote if a channel is imbalanced with $x_{(u,v)} = 0 \iff b_{(u,v)} = 0$. Since an edge can't be imbalanced on both ends,

$$x_{(u,v)} + x_{(v,u)} \geq 1 \quad \forall \ (u,v) \in E \tag{17}$$

We next define $y_p \in \{0,1\}$ such that $y_p = 0$ if path p is bottlenecked on one or more of its edges and hence can't make progress. $y_p = 1$ if no edge is imbalanced along the path. Consequently, we have

$$y_p \leq x_{(u,v)} \quad \forall \ (u,v) \in p \quad \forall p \in \mathcal{P} \tag{18}$$

$$y_p \geq \sum_{(u,v)\in p} x_{(u,v)} - |p| + 1 \quad \forall p \in \mathcal{P} \tag{19}$$

Lastly, we capture whether an edge is deadlocked by defining $z_{\{u,v\}} \in \{0,1\}$. $z_{\{u,v\}} = 0$ if the edge is deadlocked. An edge is deadlocked if and only if it is imbalanced and no path using it in either direction can make progress in either direction. The former prevents a temporarily imbalanced but deadlock-free channel from blocking a flow, and ensures that a channel that is not at one of its imbalanced states is deadlock-free. To enforce this, we have

$$z_{\{u,v\}} = x_{(u,v)} + x_{(v,u)} - 1 \quad \forall \ \{u,v\} \in E \tag{20}$$

$$z_{\{u,v\}} \geq y_p \quad \forall p : (u,v) \in p \text{ or } (v,u) \in p \tag{21}$$

Given the above constraints, detecting largest deadlock is equivalent to

$$\min \sum_{\{u,v\}\in E} z_{\{u,v\}}$$

We use an ILP solver like Gurobi to solve the above minimization problem to detect the largest deadlock and compare it to the unpeeled channels at the end of the deadlock peeling process.

B.2 Pseudo-code of Peeling Algorithm

Algorithm 1 is a description of our graph-embedded peeling algorithm in pseudo-code form. It takes as input a graph topology $G = (V, E)$ and a set of demand paths $\mathcal{P}_{\text{init}}$. Each demand path is a collection of directed channels, where each directed channel has form (channel, color). The first entry is the identifier of an undirected channel, and the second entry "color" represents the direction, which can only be either red or blue. For simplicity, we use $\neg\text{color}_0$ to represent the opposite color of color_0.

B.3 Non-uniform Random Demand Matrix Results

In addition to the results with demand matrices were senders and receivers are sampled uniformly at random (§6), we investigate the throughput and deadlock behavior of different topologies when the senders and receivers are sampled in a skewed manner. Specifically, we designate 10% of the 500 nodes as "heavy-hitters" and sample sources and destinations from them 70% of the time when sampling a new source-destination pair for the demand matrix. 30% of the senders and receivers (independently) are sampled from the remaining 450 infrequent nodes. We vary the number of demand pairs and measure the $\Phi_{\max}$ and $\Phi_{\min}$ values, repeating the procedure for Fig. 7.

Comparing topologies. Fig. 13 shows the variation in the $\Phi_{\min}$ values and the fraction of unpeeled channels across topologies with the skewed demand matrix. Compared to Fig. 7, we notice that the throughput values are lower. This is expected because the skewed sampling results in more flows

Algorithm 1: The Peeling Algorithm

input : Graph $G = (V, E)$, set of demand paths $\mathcal{P}_{\text{init}}$
output: **Success** in peeling every channel in both directions, or **Failure**

Initialize $E_{\text{processed}} \leftarrow \emptyset$, $R \leftarrow \emptyset$, $\mathcal{P} \leftarrow \mathcal{P}_{\text{init}}$; // R is the ripple
for $p : [p \in \mathcal{P}] \wedge [\texttt{length}(p) = 1]$ **do** // Add all channels with dedicated flows to ripple
    $\mathcal{P}$.remove(p);
    (channel, color) $\leftarrow p[0]$;
    R.add((channel, $\neg$color));
while $R \neq \emptyset$ **do**
    (channel, color) $\leftarrow R$.pop();
    $E_{\text{processed}}$.add((channel, color)) ; // Process an arbitrary directed channel in ripple
    for $p : [p \in \mathcal{P}] \wedge [(\text{channel}, \text{color}) \in p]$ **do**
        p.remove((channel, color));
        if length(p) = 1 **then** // Release of a degree 1 flow frees the only remaining channel in the opposite direction
            (channel′, color′) $\leftarrow p[0]$;
            R.add((channel′, $\neg$color′));
        else if length(p) = 0 **then** // Release of a degree 0 flow frees all channels it originally contained in the opposite direction
            $p_{\text{init}} \leftarrow$ look up for initial p in $\mathcal{P}_{\text{init}}$;
            $\mathcal{P}$.remove(p);
            for (channel′, color′) $\in p_{\text{init}}$ **do**
                if (channel′, $\neg$color′) $\notin E_{\text{processed}}$ **then**
                    R.add((channel′, $\neg$color′));
if $|E_{\text{processed}}| = 2|E|$ **then**
    return Success;
else
    return Failure;

that are close to each other, resulting in lower throughput due to the shared tokens across channels close to the frequent nodes. However, we still notice that the Lightning Network, power-law and scale-free topologies have much better $\Phi_{\min}$ with fewer flows when compared to the other random graphs. At 5000 demand pairs, both Lightning Network and scale-free topologies peel 25% more channels than the small-world, Erdős-Rényi, and random regular topologies and correspondingly have higher $\Phi_{\min}$. But, the trends start flipping once we have a few more demand pairs. The scale-free and power-law graphs once again struggle to peel the last 10-25% of their channels compared to other random topologies. In contrast, the Erdős-Rényi and random regular topologies do not peel as well with fewer flows, but quickly improve to peel all channels, on average, with 15000 demands. Beyond this point, their $\Phi_{\min}$ is comparable with the Lightning Network, similar to trends in the uniform random demand matrix case. It is worth noting that the star's throughput shows a slight dip with an increase in flows. This behavior depends on whether the "heavy-hitter" nodes includes the hub or not, which in turn affects the average path length across all flows and throughput values.

Explaining the relative behavior of topologies. Like the uniform demand matrix case, we seek to understand if the differences in the $\Phi_{\min}$ of different topologies is explained by the evolution of

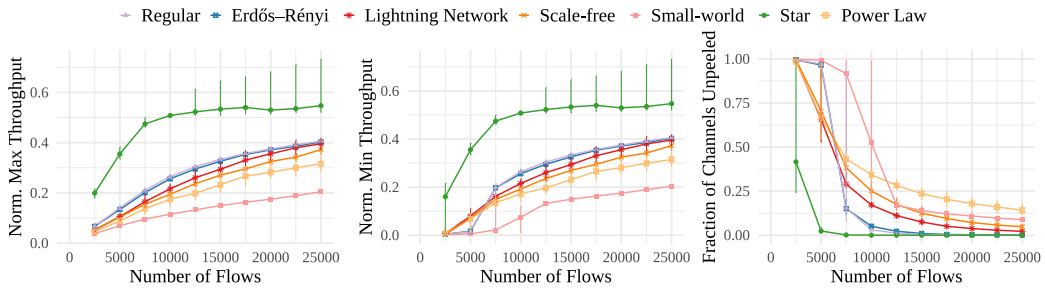


Fig. 13. Maximum ($\Phi_{\max}$) and minimum throughput ($\Phi_{\min}$) achieved by different states and the corresponding channels left unpeeled as the number of flows is varied, when flows' senders and receivers are sampled in a skewed manner. While individual throughput values are lower than the uniform random demand matrices, stars still outperform other random topologies. The Lightning Network, power-law and scale-free graphs peel earlier and have higher $\Phi_{\min}$, but over a smaller range of demand matrix density than with a uniform random demand. However, the fraction of channels peeled continues to correlate well with the $\Phi_{\min}$ achieved by the topologies. Whiskers denote max and min data point.

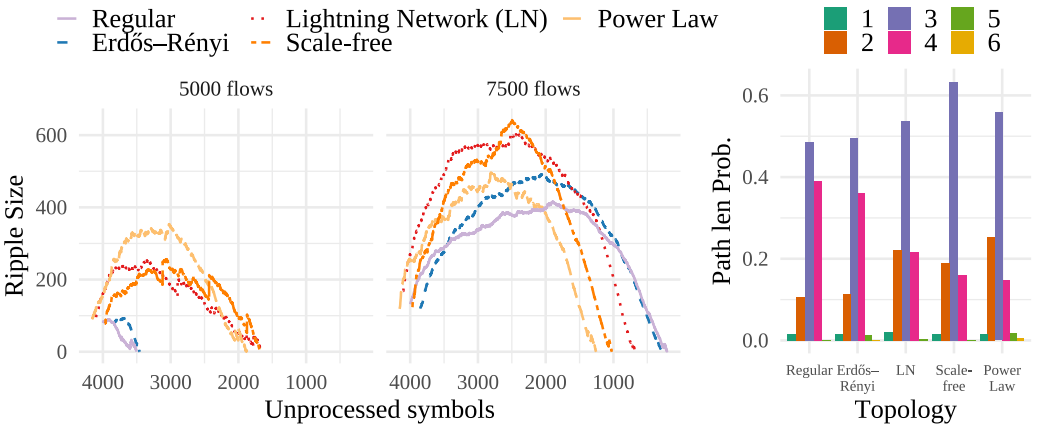


Fig. 14. Evolution of ripple size during the deadlock peeling process on different topologies with a skewed demand matrix. The tendency of the ripple to vanish at 5000 flows explains the poor performance of Erdős-Rényi and random regular graphs compared to the other random graphs. However, at 7500 flows, despite a lower peak value, the ripples of Erdős-Rényi and random regular take longer to vanish than their Lightning Network, power-law, and scale-free counterparts. This allows for a larger number of peeled or deadlock-free channels, and consequently slightly better $\Phi_{\min}$. This trend is markedly different from the ripple evolution at the same 7500 flows with a uniform random demand matrix, despite very similar path length distributions.

the deadlock peeling process. Fig. 14 shows the evolution of the ripple size as the deadlock peeling process progresses at 5000 and 7500 demand pairs, along with the path length distributions for the random regular, Erdős-Rényi, Lightning Network, power-law and scale-free topologies for the skewed demand matrix case. Similar to Fig. 8, we consider the total number of symbols at the start to be twice the number of channels in the topology, one for each direction of a channel. Every step of the deadlock peeling process involves processing one channel in one of the two directions. Each processed symbol may lead to the release of some flow nodes and consequently, add more directed channels to the ripple.

Firstly, notice that the trajectory of the ripple explains the relative performance of the topologies at 5000 and 7500 flows. At 5000 flows, the ripples associated with random regular and Erdős-Rényi topologies fails to grow after the initial release of degree 1 flows resulting in large number of unpeeled edges, and consequently low $\Phi_{\min}$. In contrast, at 7500 flows, the Erdős-Rényi and random regular topologies' ripples last longer than its counterparts resulting in higher $\Phi_{\min}$ values. However, this is a marked difference from the trajectory with a uniform demand matrix at the same 7500 flows (Fig. 8) where the Erdős-Rényi and random regular topologies' ripples disappear soon into the peeling process. This is despite nearly identical path length distributions, which suggests an identical prediction based on the LT code analysis (Fig. 9). This difference is due to the presence of correlations between the randomly sampled flows with a skewed demand matrix. Particularly, since a small set of nodes are more likely to be senders or receivers, certain channels close to these nodes are more likely to be part of flows than if demand pairs were sampled randomly. This means two contradicting trends for the ripple evolution: first, a processed symbol is likely to reduce the degree of more nearby flows, which could potentially lead to their release and add more unprocessed channels to the ripple; second, flows are likely to contain many of the same channels, and thus fewer overall symbols are covered through the peeling process. We see both of these trends in Fig. 14; the maximum ripple size is smaller across all topologies than the uniform random demand case but, the Erdős-Rényi and random regular topologies' ripples grow early on because the few initially processed symbols are able to release more flows and add more symbols.
Discussion. Figs. 13 and 14 suggests that the the deadlock peeling process and ripple trends help compare the best and worst-case throughput behaviors of different topologies regardless of the nature of the demand matrix. However, the prediction and analysis of the deadlock peeling process needs to be revised in the presence of correlated bipartite graphs that deviate significantly from the i.i.d. bipartite encoding graph where channels are sampled randomly to be part of flows. This is not unexpected; in fact, it is rather fortunate that with a uniform random demand matrix, these correlations are minimized to a point that we can assume an i.i.d. bipartite graph and extend the LT codes analysis. However, with a skewed demand matrix, these correlations that affect how likely a channel is to be part of a flow need to be accounted for more carefully. One way to approach this is to consider a channel or input symbol degree distribution in addition to the encoded symbol degree distribution. Such a revised analysis can then be used in the rest of the synthesis pipeline to generate good degree distributions and topologies that are robust to deadlocks across more demand patterns.

B.4 Topology Synthesis Using LT Codes Analysis

B.4.1 Preliminaries. Just as LT codes' degree distributions are designed to minimize the number of encoded symbols, with the deadlock peeling process, it is desirable to minimize the number of paths that need to sustain active payments in a credit network with a fixed number of channels. A natural question to ask towards this goal, is what path length distribution $\Omega(\cdot)$, akin to the degree distribution in the LT code design, is optimal.

However, despite the similarities, the deadlock peeling process is markedly different from the LT process. First, every channel is processed twice: once in the blue direction and once in the red direction. This can be considered akin to having twice the number of input symbols as channels unprocessed at the start of the peeling process. Second, while the release of flows of degree 1 covers one unprocessed symbol (as is with the LT process) the release of flows of degree 0 does not have an LT counterpart. In particular, when all the channels that a flow of length d uses have been covered in the direction of use, such a flow can freely move tokens on all of the associated channels covering up to $d-1$ new channels in the opposite direction. In addition, with LT codes, an encoded symbol of degree d chooses d input symbols *at random* to XOR. However, a flow of length d can

never choose d channels at random: the flows belong to an underlying topology with the proximity of two channels dictating their presence in any given flow.

To account for these differences, we make two assumptions. First, we analyze the deadlock peeling process under an *i.i.d. bipartite construction.* In other words, we assume that a flow of degree d is allowed to choose d channels at random regardless of their location in the underlying graph. This is rather optimistic but captures some important trends in the ripple evolution of a topology, as has been already demonstrated in Fig. 9. Next, we assume that when a channel is processed in one direction, it is immediately also processed in the opposite direction. We approximate this in practice by ensuring that whenever a channel is processed from the ripple, if the channel's opposite direction is also in the ripple, it gets processed immediately thereafter. Our evaluations suggest that $80 - 95\%$ of the channels in a random graph can be processed sequentially. This implies that we can treat a channel and its two directions as a single unit, akin to a single unprocessed symbol in the LT codes terminology. During the deadlock peeling process, an unprocessed channel from the ripple is chosen and processed (in both directions) reducing the degree of an aggregate set of flows (that use it in either direction), leading to more channels being added to the ripple. An added advantage of this is that we can entirely ignore the effect of flows reaching degree 0 and covering $d-1$ channels since *both directions* of earlier encountered symbols have already been processed. Under these assumptions, we can apply the analyses of LT codes [26, 36] exactly to the deadlock peeling process.

B.4.2 Analysis. Consequently, we are able to reuse the following three results from the LT codes analysis [26, 36], restated here for clarity in the context of a credit network.

Theorem 19. *(Flow release probability) Given a credit network $G(E,V)$, the probability that a flow of degree d is released L out of $|E|$ channels remain unprocessed, is given by*

$$q(d,L) = \begin{cases} 1, & \text{if } L = |E|, R = 0 \\ \frac{d(d-1)L}{(|E|-d+1)(|E|-d+2)} \prod_{j=0}^{d-3} \frac{|E|-(L+1)-j}{|E|-j}, & \text{if } d = 2,\ldots,|E|, 1 \le L \le |E|-d+1 \\ 0, & \text{otherwise.} \end{cases}$$

Theorem 20. *(Ripple addition probability) Given a credit network $G(E,V)$, the probability that a flow of degree d is released and adds a new channel to the ripple when the ripple size is R and L out of $|E|$ channels remain unprocessed, is given by*

$$q(d,L,R) = \begin{cases} 1, & \text{if } L = |E|, R = 0 \\ \frac{d(d-1)(L-R+1)}{(|E|-d+1)(|E|-d+2)} \prod_{j=0}^{d-3} \frac{|E|-(L+1)-j}{|E|-j}, & \text{if } d = 2,\ldots,|E|, 1 \le R \le L \le |E|-d+1 \\ 0, & \text{otherwise.} \end{cases}$$

Theorem 21. *(Expected channels added) Given a credit network $G(E,V)$ with $|\mathcal{P}|$ paths whose path lengths d are drawn from a distribution $\Omega(d)$, the expected number of channels added to the ripple due to the most recently processed symbol, when the ripple size is $R(L+1)$ and L out of $|E|$ channels remain unprocessed, is given by*

$$Q(L) = \sum_{d=1}^{|E|} |\mathcal{P}| \Omega(d) q(d, L, R(L+1))$$

Accounting for Overlaps. Theorem. 21 assumes that there is no overlap across the channels that are covered by different flows that are released simultaneously after the most recent symbol has

been processed (when the current ripple size is R and there are L unprocessed symbols). We find that this isn't the case in practice. To account for this, consider viewing the release of encoded symbols as a series of n balls that are dropped into some bins corresponding to the input symbol. A given ball (or flow) is released with probability $r(L) = \sum_{d=1}^{k} \Omega(d)q(d, L)$. All the bins they fall into corresponding to the covered symbols must be unprocessed; there are L bins in total. $L - R$ of these are not in the ripple.

The expected number of symbols added newly to the ripple, can therefore be computed as the expected number of non-empty non-ripple bins at the end of the ball drops. Consider a random variable X_i per bin where $i = 1, \ldots, L$. $X_i = 1$ only if the i^{th} bin isn't already in the ripple and it is non-empty after all releasable flows are released. For the $L - R$ bins that correspond to input symbols not in the ripple, this is determined by the chance that a bin is non-empty after all n flows are released.

$$E[X_i] = 1 \cdot Pr(X_i = 1) + 0 \cdot \Pr(X_i = 0) = \Pr(i^{th}\text{bin is not empty}) = \left(1 - \left(1 - \frac{r(L)}{L}\right)^n\right)$$

For the remaining R bins, $E[X_i] = 0$ because those input symbols are already in the ripple. Thus,

$$Q(L) = \sum_{i=0}^{L} E[X_i] = (L - R)\left[1 - \left(1 - \frac{r(L)}{L}\right)^n\right] = (L - R)\left[1 - \left(1 - \frac{\sum_{d=1}^{k} \Omega(d)q(d, L)}{L}\right)^n\right] \quad (22)$$

We use this expression when evaluating the predicted ripple size as a function of L in Fig. 9.

B.4.3 Numerical Optimization. Since we have established that a number of the results translate from the LT codes analysis to the deadlock peeling process, we next try to optimize for a good path length distribution using a numerical approach [36]. This section describes the steps involved in the numerical optimization which ultimately outputs a good path length distribution.

The optimization aims to find a good degree distribution by matching a desired ripple evolution that is not only efficient, but also robust enough to ensure success of encoding. Prior work [36] has proposed the use of an exponential decaying function of the following form where L denotes the number of unprocessed symbols and $R(L)$ denotes the ripple size when L symbols remain unprocessed:

$$R(L) = \begin{cases} c_1 L^{1/c_2}, & \text{if } c_1 L^{1/c_2} \leq L; \\ L, & \text{otherwise.} \end{cases} \quad (23)$$

We use the same ripple evolution curve with $c_1 = 1.7$ and $c_2 = 2.5$. Once, we have the exact values for the sizes of the ripple at every step of the deadlock peeling process, we can compute the desired number of symbols added to the ripple $Q(L)$ when L symbols remain unprocessed as follows:

$$Q(L) = \begin{cases} R(L), & \text{for } L = k; \\ R(L) - R(L + 1) + 1, & \text{for } k > L \geq 0. \end{cases} \quad (24)$$

Let $\Omega(\cdot)$ represent the path length distribution of the flows (encoded symbols). For a credit network with k channels, the path length can be utmost k. Ideally, we want to solve for $\Omega(\cdot)$ that matches the desired ripple evolution as closely as possible. The key insight that enables searching for such an $\Omega(\cdot)$ is that Theorem. 21 allows us to establish a linear relationship between the expected symbols added and the unknown degree distribution. The desired degree distribution should ideally satisfy Eq.(25) where m denotes the total number of flows. Notice that the variables here are $m_1 \cdots m_k$ rather than $\Omega(1) \cdots \Omega(k)$ directly, so the total number of flows in the satisfying solution can be computed by summing all the variables.

For simplicity, we denote the matrix below by A, vector $m \cdot \begin{bmatrix} \Omega(1) & \cdots & \Omega(k) \end{bmatrix}^\top$ by x and the vector on the right hand side by b.

$$\begin{bmatrix} q(1,k,R(k+1)) & 0 & 0 \\ \vdots & \ddots & 0 \\ q(1,1,R(2)) & \cdots & q(k,1,R(2)) \end{bmatrix} \begin{bmatrix} m\Omega(1) \\ \vdots \\ m\Omega(k) \end{bmatrix} = \begin{bmatrix} Q(k) \\ \vdots \\ Q(1) \end{bmatrix} \tag{25}$$

However, as already mentioned this equation may not be satisfiable, particularly for non-negative $\begin{bmatrix}\Omega(1) & \cdots & \Omega(k)\end{bmatrix}$. The standard optimization procedure used for LT-code degree distribution synthesis seeks to minimize the ℓ_2-norm of $Ax - b$ for $x \geq 0$.

While this procedure could be directly useful for generating path length distributions, it is particularly convenient that this structure lends itself to adding more constraints that are specific to finding path lengths on a topology. Assume we want to generate a topology with k channels and n nodes. We impose constraints on a maximum path length $p_{\max}$ (Eq.(31)) since long paths are undesirable both from a throughput and deadlock perspective. We impose a maximum node degree in the graph $d_{\max}$ (Eq.(32)) to enforce limits of centralization. We also ensure that the number of paths of length 1 does not exceed the number of edges in the graph (Eq.(29)) and that the probability of finding longer paths decreases as path length increases (Eq.(30)). Further, since $\Omega(\cdot)$ is a probability distribution, we enforce constraints Eq.(27) and Eq.(28).

As a consequence of these constraints, instead of solving $Ax = b$ for $x \geq 0$, we find a good path length distribution $\hat{x}$ by solving the following optimization.

$$\begin{aligned}
&\text{minimize} && \|Ax - b\|_2 && (26)\\
&\text{subject to} && x \geq 0 && (27)\\
& && 1^\top x = m && (28)\\
& && x_1 \leq \frac{2km}{n(n-1)} && (29)\\
& && x_{i+1} \leq x_i,\ \forall i \geq 2 && (30)\\
& && x_i = 0,\ \forall x > p_{\max} && (31)\\
& && x_i \leq \frac{(d_{\max})^i m}{n} && (32)
\end{aligned}$$

The solution to this optimization problem gives us the path length distribution that satisfies all the constraints and mimics the desired ripple evolution as closely as possible.

B.4.4 Generating a matching topology. Once we have a path length distribution from the constrained optimization outlined above, we aim to synthesize a graph topology that achieves the desired path length distribution. We approach this by searching over the space of random graphs captured using their joint degree distribution. The joint degree distribution captures the probability that a randomly sampled edge in the graph connects nodes of degree j and k. We choose to focus on the joint degree distribution in the hope that its expressiveness will lend itself to a higher likelihood of matching our desired path length distribution. There are still two steps to the process of using a joint degree distribution:a) finding the right distribution and b) synthesizing a graph to match it. The latter problem has been well-studied, with popular graph packages like networkx implementing generators [5]. For the former portion, there exist recurrent analytical and exact expressions that capture the probability that nodes are located within distance d of each based on the likelihood that an edge connects nodes of degree j and k. (Eq.(5-10) in [33]). We use this relationship to express the path length distribution as a function of the joint degree distribution and iteratively search for a joint degree distribution whose corresponding path length distribution

moves closer and closer to the desired path length distribution. We perform this using MATLAB's black-box optimizer ([7]). We then verify that the output joint degree distribution is valid (Theorem 2 in [46]) since not all joint degree distributions can be mapped to a simple graph. Once we produce a valid degree distribution, we sample from it to generate a joint degree sequence that denotes how many edges need to be connected between a nodes of degree j and k. Such a sequence also needs to be validated ([6]) after which it generates a graph. In the final output graph, we take the largest connected component and use it as our synthesized graph.

In the context of the experiments in §6.4, we sought to generate a topology with 300 nodes and 1500 edges. We first obtained a desirable path length distribution from the numerical optimization with a maximum path length constraint of 10 and a maximum degree constraint of 10. We then feed it into the MATLAB optimizer to generate a valid joint degree distribution. Direct sampling from the output of the MATLAB optimizer did not give us a valid joint degree sequence; we improved it by adding some edges until the conditions for a valid joint degree sequence were met. This resulted in a sequence for 301 nodes and 1540 edges. The synthesized graph was disconnected with the largest component containing 271 nodes and 1513 edges. We use this largest component as our synthesized topology in Fig. 11.