6-2018

A proposal for a decentralized liquidity savings mechanism with side payments

Adam FUGAL

Rodney GARRATT

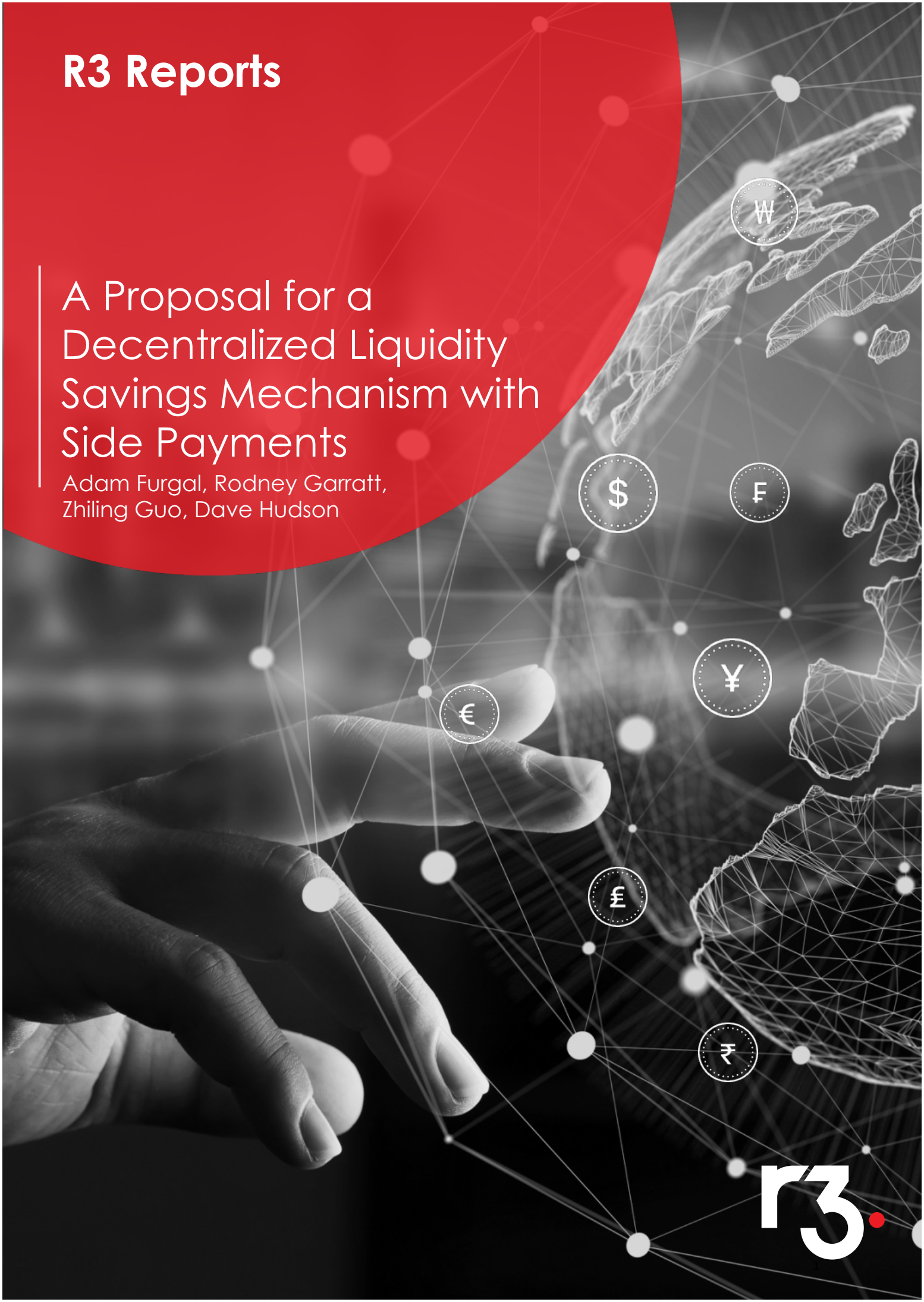
Zhiling GUO
Singapore Management University, ZHILINGGUO@smu.edu.sg

Dave HUDSON

R3 Reports

A Proposal for a Decentralized Liquidity Savings Mechanism with Side Payments

Adam Furgal, Rodney Garratt,
Zhiling Guo, Dave Hudson



Contents

R3 Research aims to deliver concise reports on DLT in business language for decision-makers and DLT hobbyists alike. The reports are written by experts in the space and are rooted in practical experience with the technology.

Visit R3's Wiki here.
Visit R3's Public Research here.

A Proposal for a Decentralized Liquidity Savings Mechanism with Side Payments*

Adam Furgal, R3
Rodney Garratt, University of California Santa Barbara
Zhiling Guo, Singapore Management University
Dave Hudson, R3

June 11, 2018

1 Introduction

In most countries, the central bank provides the medium to physically settle the smallest payments (cash) and the means to electronically settle the largest payments, which typically are wholesale payments between banks. For the latter purpose the central bank usually operates a system through which banks can settle payments in central bank money. Historically, interbank payments were settled via (end of day) netting systems, but as volumes and values increased central banks became worried about the risks inherent in deferred net settlement systems, so most central banks opted for the implementation of a Real Time Gross Settlement (RTGS) system. With RTGS, payments are processed individually, immediately and with finality during operational hours. This eliminates settlement risk and the potential unwinding of payments, at the cost of increased need for liquidity provision by participants.

Liquidity demands in RTGS systems can be enormous. Fedwire Funds, the large value payment system in the United States, averages around 590,209 payments a day with a total value close to $3 trillion in 2016.[1] CHAPS, the large value payment system in the United Kingdom has an average daily volume around 165,285 worth close to £ 334 billion in 2017.[2] Target2, the European Union's large value payment system, has average daily volume of 342,008 payments worth up to €1.7 trillion Euro in 2016.[3] To put these numbers in perspective, in most countries every 5 to 6 business days, payment values equate to annual GDP.

Over the last 20 years or so, central banks around the world have implemented Liquidity Savings Mechanisms (LSMs) in their large-value payment systems in order to reduce liquidity demands of these systems. LSMs work by either encouraging greater liquidity recycling or by allowing banks to settle net obligations intraday. Liquidity recycling arises from the fact that banks can use incoming liquidity to make outgoing payments and generally, more liquidity will be recycled if banks make payments in a timely fashion.[4] Policies that incentivize timely payment processing

*Authors are listed alphabetically. We thank conference participants at the 2017 Payments Canada Payments Workshop for helpful feedback. Author contact information: garratt@ucsb.edu; zhilingguo@smu.edu.sg.

[1] Fedwire Funds Service https://www.federalreserve.gov/paymentsystems/fedfunds_ann.htm

[2] Bank of England CHAPS Statistics https://www.bankofengland.co.uk/payment-and-settlement/chaps

[3] TARGET Annual Report 2016 https://www.ecb.europa.eu/pub/pdf/other/ecb.targetar2016.en.pdf?45a0984578894290ca5228a5465fe81e

[4] As a simple illustration, consider a situation where bank A has to make a one million dollar payment to bank B, bank B has to make a one million dollar payment to bank C, and bank C has to make a one million dollar payment to bank A. If these payments are made in sequence they can all be completed using one million dollars in liquidity.

(e.g. throughput guidelines) or punish delayers (e.g. time-varying tariffs) make it more likely that liquidity will be recycled and hence reduce the overall liquidity needs of the system.

The most effective LSMs, however, are the ones that economize on liquidity needs by matching offsetting payments that have been submitted to a central queue and settling these payments using only the liquidity needed to cover the net obligations. Suppose bank A needs to make a payment to bank B for a value of 100, and bank B needs to make a payment to bank A for a value of 80. Then the amount of liquidity requirement to settle these two payments, if they were entered into a queue operating an offsetting algorithm, would be 20, the value of the net obligation of bank A to bank B. In contrast, in a pure RTGS system without an LSM, the liquidity requirement to settle these two payments would be 100.[5] Potentially greater gains can be obtained by netting payments between more than two banks on a multilateral basis.

The potential liquidity savings of offsetting algorithms are decreasing in the frequency with which net obligations are calculated. The largest potential liquidity savings would occur if all payments for the entire payments day were netted at the end of the day. This is, in fact, what is done in systems like Bacs, Cheque and Credit, or FPS in the United Kingdom, ACH in the United States, or ACSS in Canada. However, most wholesale payments cannot be delayed for the entire day. Typical payment requests that banks receive from their customers require same day processing, but banks often have some flexibility in terms of when, during the day, they execute them. Ball et al. (2011) estimates that 4% of payments are urgent, meaning they must be processed immediately otherwise there will be significant consequences to the customer. But this means 96% of payments are somewhat flexible. Many economists have suggested, however, that delaying payments results in some customer dissatisfaction.[6] Hence, there is a trade-off between liquidity savings resulting from delaying payments to increase netting opportunities and conserve liquidity and the cost of delay.

Netting algorithms run by central banks operate a centralized queue which typically clears at predetermined time intervals. Operators try to balance the trade-off between the increased potential for finding offsetting payments that results from longer netting cycles and the cost of longer average settlement times. This is a difficult task which the central bank must try to resolve with imperfect information. The central bank does not know what payment obligations individual banks are holding outside of the RTGS system or if they will choose to enter them into the queue. In addition, netting opportunities would be quite limited if centralized systems only considered perfectly offsetting payments. Instead, algorithms run by centralized systems look for sets of imperfectly offsetting payments that can be settled with the available liquidity. The liquidity available to the queue is a choice variable of the participants and again, this is something that a central operator may have to estimate when designing its system.

Even if liquidity and payments are known, the problem of solving for the optimal netting solution can be very complex. If there are n payments in a payment file the number of netting combinations to consider is 2^n-1, which can become NP hard for large n. Another issue is that centralized netting algorithms seek to maximize the value or volume of payments they can settle, but they generally have no way of weighting the importance of payments or assessing the cost of liquidity provision, which can vary across banks and throughout the day. There is an incomplete information problem that prevents the central operator from maximizing the social welfare of the system. Finally, centralized queues operate within a given jurisdiction or banking system. Netting payments across systems can be done, but it involves providing liquidity to yet another centralized system (e.g. CLS pay-ins).

This paper considers an alternative to a centralized netting queue that addresses problems mentioned above while at the same time adding additional benefits associated with distributed ledger technology. We refer to the approach described in this paper as a *decentralized liquidity savings mechanism*. The decentralized approach allows individual participants to make netting proposals

However, if bank B delays in making the payment to C, either because it is not processing payments in a timely fashion or because it uses the liquidity it received from bank A for another payment and is not willing to provide additional liquidity, then bank C may have to provide one million in liquidity to complete its payment to bank A.

[5] One possibility would be that bank A provides liquidity of 100 to send its payment to bank B, and then bank B uses 80 units of that liquidity to make its payment to bank A. Another possibility would be that bank B provides liquidity of 80 to send its payment to bank A, and then bank A adds a further 20 units of liquidity to send its payment to bank B. Either way, the total liquidity required is 100.

[6] See for example, Angelini (1998) and Bech and Garratt (2003, 2012).

that reflect, in real time, their current conditions. This allows for competitive proposals that reflect market conditions and increase the welfare of participants. Distributed ledger systems are likely to span multiple jurisdictions and participants in these systems would be able to propose liquidity saving opportunities within and across jurisdictions without pre-paying liquidity to multiple operators.

There are two parts to the proposed solution. One part is the method by which individual nodes collect information necessary to determine netting opportunities and make a netting proposal. The other part is the proposal for how liquidity is provided to settle net obligations and the actual settlement of obligations across the distributed system. The liquidity proposal may be as simple as having each participant contribute liquidity equal to their own net debit positions, but we also entertain the possibility of side payments. Properly selected side payments facilitate outcomes that, under certain conditions, replicate bargaining outcomes and meet accounting objectives of fairness, equity and neutrality (Roth and Verrecchia 1979).

There are two key technological innovations involved the proposed solution. The first is the *recursive graph scanning algorithm* that allows any node on the network to identify all of the netting opportunities that it could potentially contribute to. The second is the use of atomic transactions that bundle together multiple changes to the ledger into one single transaction that either succeeds or fails as a whole. These innovations require certain functionalities that may not exist in all distributed ledger platforms. The current version of Corda, a blockchain-inspired open source distributed ledger platform developed by R3 and a consortium of banks and financial institutions, meets our requirements. Corda has the ability to create atomic transactions that contain multiple payments, which either settle all-at-once or not-at-all. A complex netting resolution involving multiple parties, obligations, and resolution payments may be created by any participant, and the whole set is settled or none at all. This atomicity is crucial, as a partial resolution of a netting cycle would leave the system in a complex state to unwind.

Section 2 reviews the relevant literature. Section 3 discusses centralized queues and gives examples of different approaches used by operators to settle obligations in the queue. Section 4 discusses problems of imperfect and incomplete information that limit the ability of central operators to settle obligations in the centralized queue in a welfare maximizing way. Section 5 formally introduces our approach to decentralized multilateral netting and introduces the of notion side payments. Section 6 concludes.

2 Related Literature and Ongoing Projects

There are a few theoretical papers that explore the potential for liquidity savings mechanism to enhance welfare in priced credit (e.g. Fedwire) and collateralized credit (e.g. CHAPS) systems. See Martin and McAndrews (2008) for an example of the former and Jurgilas and Martin (2013) for an example of the latter. These papers use specialized models and provide general insight. The conclusions are very straightforward. In the collateralized credit world, the only equilibrium without an LSM is “bad" (all participants delay payment processing) and the LSM always helps. In the priced credit world, there is a “good" (no participants delay payment processing) and a “bad" equilibrium without the LSM and it is possible that the LSM makes things worse. In reality the welfare benefits of an LSM are likely to depend upon several factors, such as the cost of providing liquidity, the cost of delay, individual bank payment characteristics (size, arrival time, time criticality), joint payment characteristics and behavioural factors, (e.g. which payments are put into the queue), and these models cannot fully account for these things.

More relevant, perhaps, are the papers that test the performance of actual LSMs. Norman (2010) provides an overview of studies that examine benefits of specific applications of queues. Norman (2010) reports savings of 20% for Bank of Korea’€™s BOK-Wire+ payment system and 15% for the Japanese BOJ-Net system. More recently, Davey and Grays (2014) and the follow up by David Seaward (2016) estimate the liquidity savings resulting from the LSM introduced to CHAPS in April of 2013. They found initial savings around 20%, however this number fell to near zero in recent years as banks have been flush with liquidity due to quantitative easing policies.

Atalay et al. (2010) quantify the hypothetical benefits of adding a LSM to Fedwire. They find

that an LSM can be welfare decreasing, but that potential gains are more than 500,000 USD per day under realistc scenarios. Using a similar approach, Diehl and Schollmeyer (2011) conduct an assessment of welfare benefits of the LSM in TARGET2. They evaluate both the fee-based and collateral-based liquidity provision and demonstrate the welfare effects can reach the order of 170,000 to 300,000 Euro per day.

In 2016 some central banks began experimenting with the possibility of adding LSMs to their proofs of concept for distributed ledger technology based on wholesale payment platforms. The first to report their findings was Bank of Canada's Jasper Project (Chapman et al. 2017, Project Jasper White Paper 2017). This group implemented a centralized LSM in the form of a centralized queue with periodic multilateral payment netting.

The centralized queue embedded in the Jasper phase 2 design is intended to be used for payments for which the sender is willing to accept delay before settlement in exchange for possible liquidity savings. Rather than choosing atomic settlement, users submit payment obligations to the queue. The queue collects all the payment obligations submitted by users over a short period of time (configured by the Notary node) known as a "matching cycle". At the end of the matching cycle a multilateral netting algorithm is run which offsets as many payments in the queue as it can subject to liquidity constraints.

In order for the matching algorithm to run it needs to know exactly how much liquidity it has available to settle net obligations. One way to do this would be to freeze the whole system while the algorithm runs. However, the Jasper phase 2 architects had a better idea. Instead of freezing the whole system, participants make payments to the notary just before the algorithm runs. This is called the inhale phase. With these funds under the control of the notary, the algorithm runs, draws on this liquidity as needed, and returns unused liquidity and any additional net credits from the matching solution to the participants when the algorithm had finished. This is called the exhale phase. The inhale/exhale architecture allowed a centralized queue to operate within an otherwise decentralized system.

A recent report on the joint venture by the ECB and Bank of Japan (2017) describes a decentralized approach for bilateral netting in the Stella Project. The LSM is implemented via smart contracts that allow banks to conduct payment transfers with LSMs in the distributed ledger environment.

In Singapore, Project Ubin explores the use of distributed ledger technology for interbank payment and settlements with LSMs introduced for the purpose of gridlock resolution (Monetary Authority of Singapore 2017). Gridlock refers to a situation where banks are unable to settle any payments because the liquidity available to each participant in the payment system is less than any of their outstanding payment obligations. The standard approach to gridlock resolution is to look for netting cycles within the set of gridlocked payments that deliver net amounts that can be settled with available liquidity. Phase 2 of the project was completed in October 2017. Three software prototypes were developed on three leading DLT platforms: Corda, Hyperledger Fabric and Quorum. The three workstream designs leverage different capabilities with their distinct methods. For example, fund transfer functionality and privacy of transactions is achieved by Corda with its Unspent Transaction Output (UTXO) model and confidential identities, Hyperledger Fabric with its Channels design, and Quorum using Constellation and zero knowledge proofs (ZKP). In addition, the queuing mechanism and LSM are implemented through different solution designs. The EAF2 algorithm[7], originally developed to support bilateral and multilateral net settlements in centralized queues, was used for gridlock resolution in both Hyperledger Fabric and Quorum prototypes. The former splits the gridlock resolution into two main stages: initiation/participation and settlement, while the latter uses a cycle of four system states: normal, line up, resolving, and settling. Since the Hyperledger Fabric relies on a specialized MAS node representing a central authority to facilitate the settlement, and Quorum relies on a global gridlock queue to track queued payments in the system in order to trigger gridlock resolution, these two platforms haven't achieved a full decentralized deign. In contrast, the Corda workstream achieved the highest degree of decentralization in implementing the decentralized gridlock resolution. It proposed a graph-based queue-scan approach to facilitate the discovery of queued payments in the decentralized environment. It further developed a new cycle-based algorithm that consists of three stages: detect, plan, and execute. We will provide more technical details of this innovative design in Section 5.

[7]The EAF2 algorithm is the earliest FIFO-based algorithm used in Germany. Please refer to Section 3 for more details of the offsetting algorithms in central queues.

The mechanism described here is an extension of the Corda gridlock resolution mechanism described in the Project Ubin Phase 2 report. Gridlock mechanisms are emergency features of payment systems and while they conserve on liquidity they are not specifically invoked for this purpose. Our proposed mechanism differs from a gridlock resolution mechanism, in three important ways. First, we do not invoke our mechanism only as a gridlock solution. Rather it is envisioned that banks voluntarily put payments into the queue for netting in order to save liquidity, reflecting on the trade-offs between delay and liquidity savings discussed above. Second, the gridlock resolution mechanisms described in the Ubin report are programmed to select a particular netting cycle subject to certain pre-specified criteria (e.g under the Corda approach the gridlock mechanism executes the netting cycle that clears the largest obligation sum). In contrast, we allow banks to make propose any netting cycle. Third, in the gridlock solutions described in the Ubin report banks pay their net obligations under the selected matching cycle. In contrast, we allow banks to propose side-payments.

3 Centralized Queue Design

Queuing arrangements used in the interbank settlement systems economize on liquidity needs by matching (partially) offsetting payments that have been submitted to a queue and settling these payments using only the liquidity needed to cover the net obligations. There are many design issues that have to be considered when setting up a centralized queuing mechanism. One of these issues is the decision of how often to clear the queue. Potential liquidity savings are decreasing in the frequency with which net obligations are calculated and yet delay is costly as it results in customer dissatisfaction.[8] Hence there is a trade-off between liquidity savings resulting from delaying payments to increase netting opportunities and the cost of delay.

Another important decision central operators have to make is which payments to settle. Given a set of payments in a queue the central operator can compute each bank's net obligation. The entire queue can be settled if all banks in a negative net debit position provide liquidity equal to their negative positions. However, central queues are typically prefunded, with banks specifying amounts they are willing to contribute to clear the queue, and these amounts may not be sufficient to cover all negative debit positions. When this situation arises, it is necessary to determine a subset of queued payments that can be settled on a net basis with the available liquidity.

The problem of solving for the optimal netting solution can be very complex. If there are n payments in a payment file the number of netting combinations to consider is $2^n - 1$, which for large n is extraordinarily large. More formally, the decision problem is NP-complete and the optimization problem is NP-hard, which means that there is no known algorithm that is guaranteed to find an optimal solution in polynomial time.[9] This problem belongs to the general class of combinatorial optimization problems. The traveling salesman problem and the knapsack problem in operations research and the winner determination problem in combinatorial auctions are other well-known examples of combinatorial optimization problems. As the size of the problem increases, the first-best solution is computationally challenging to be obtained.

To obtain a second-best solution, banks utilize routines that discard payments until liquidity constraints are met. Here, multiple objectives may be at play. On the one hand, the goal of the operator may be to maximize the value of payments settled or in some instances the number of payments. But at the same time, it is recognized that not all payments in the queue have equal priority from the point of view of the bank that inserted it into the queue. Centrally located queuing facilities in currently operating payment systems (see Table 1) mostly utilize the First-in-first-out (FIFO) principle because of its simplicity and fairness, or a variation on the FIFO rule to improve efficiency. Under FIFO, in the event that a liquidity shortfall prevents all payments in the queue from being settled, payments that were added first are excluded first. This is not always adhered to strictly. Some systems allow banks to designate priority levels to the payments that they place in the queue.[10] Payments are then released for settlement on a FIFO basis within each

[8] Different payments will have different delay costs depending on the underlying customer need. Some payments cannot be delayed at all and must be processed through a pure RTGS stream. However, within the group of payments that do not require immediate processing there are still varying degrees of urgency.

[9] See Güntzer, Jungnickel and Leclerc (1998).

[10] Payment priority can be automatically defined by the payment type or assigned by the sender.

priority level. The use of priorities helps banks to achieve greater flexibility in FIFO processing. Other variations on FIFO include combining simple FIFO with facilities such as reordering or bypass. FIFO-Reorder involves moving a payment to the end of the queue or changing its priority code. FIFO-Bypass allows the operator to bypass a large payment if it cannot be settled due to a lack of funds, without the need to reorder payments.

In addition, some algorithms utilize sorted queue method to improve the offsetting efficiency. This allows the operator to abandon FIFO altogether. The algorithm first sorts payments in ascending or descending order based on value. It then gradually adds or removes payments to identify the largest subset of payments which can be settled simultaneously without any bank incurring an overdraft and without deviating from the banks' designated priority order. To achieve higher efficiency in the offsetting algorithms, some other forms of optimization have also been proposed. These include optimization routines that simulate the net balance of each participant under a variety of subsets of queued incoming payments and queued outgoing payments in order to find the arrangement that settles as many of the queued payments as possible subject to the liquidity constraints.

Flexible and sophisticated queue arrangements enable systems to achieve higher offsetting efficiency. The most commonly used FIFO and sorted-queue offsetting algorithms are provided below:

FIFO-based Algorithm

1. Activate all pending queued payments and calculate the net balance position of each participant.
2. Identify the participant with the largest uncovered debit position (i.e. the largest difference between outgoing and incoming payments and available liquidity).
3. Remove the latest payments of the participant identified in step 2 until its balance is no longer negative.
4. Repeat steps 2 and 3 until all participants' balances become non-negative.
5. Settle all remaining payments in the queue using available liquidity.

Sorted-Queue Algorithm

1. Activate all pending queued payments and calculate the net balance position of each participant.
2. Identify the participant with the largest uncovered debit position (i.e. the largest difference between outgoing and incoming payments and available liquidity).
3. Sort the payments of the participant identified in step 2 in ascending (or descending) order based on amount and remove payments, progressing backwards from the end of the list, until its balance is no longer negative.
4. Repeat steps 2 and 3 until all participants' balances become non-negative.
5. Settle all remaining payments in the queue using available liquidity.

The above basic forms of FIFO and sorted-queue algorithms cannot guarantee the optimal settlement of queued payments. Some systems have developed more sophisticated algorithms to improve the offsetting performance. For example, the Electronic Payment System (MEPS+) of the Monetary Authority of Singapore has taken iterative approaches to reorder payments in Step 3 of the FIFO algorithm, and chooses the iteration that results in the highest total transaction amount for each participating bank. MEPS+ also runs through a series of inactivation/activation checks in combination with the sorted-queue algorithm. The algorithms first try to identify harmless removals that have no adverse effect on receiving banks. If no such opportunity is found, then the algorithms revert to a sorted-queue algorithm.

Some centralized queues allow participants to specify conditions under which payments can be released from the queue. The trigger to run the offsetting algorithms can be based on time or the occurrence of some payment event. Time-driven algorithms run at predefined time intervals, some designated times, or upon decision of the system operator. Event-driven algorithms run whenever a criterion is met, such as the arrival of a new payment, or accumulated volume or value of payments, or if certain netting opportunities appear. In addition, a balance-reactive LSM allows banks to select a balance threshold below which payments are not released from the queue (Martin and McAndrews (2010)). Systems may also allow participants to set additional restrictions on the amount of their own liquidity that can be used. Bilateral limits restrict the maximum net amount a participant is willing to pay to another participant. Multilateral limits determine the maximum net outflow of funds a participant is willing to allow to all other participants or groups of participants in the system. And finally, a total limit restricts the total amount of liquidity available for both RTGS and LSM operations.

Note that there is a distinction between the usage of offsetting algorithms as a gridlock-preventing feature in a pure-RTGS system versus as a separate settlement mode alongside the RTGS mode. In the former case, the offsetting mechanism runs only occasionally in order to clear the outstanding payment queues at a certain time or when necessary (e.g. when a certain volume of payments remains unsettled for a certain period). In the latter case (as shown in Table 1), the system is called an "RTGS system with LSM," which normally has two settlement modes: RTGS mode and LSM mode. In general, banks can use the settlement account's full liquidity in the RTGS mode and only the assigned liquidity for LSM mode. The assigned liquidity can take the form of dedicated LSM accounts or reservations created for certain type of payments. In the case of dedicated LSM account, the LSM account liquidity can be managed with different design features. Participants normally need to prefund their account balance at the beginning of the day. Supplemental funding is also possible throughout the day. Two common forms of credit extension by central bank are repurchase agreements (repo) and overdraft on central bank accounts (pledge). Settlement institutions may also provide intraday credit up to a predefined limit (a net debit cap).

Table 1 summarizes the existing centralized queuing mechanisms in major countries. It shows that there are large differences among the countries and systems in terms of the design of central queuing functions and LSM account features. First, almost all systems have adopted a FIFO centralized queue with priority design. Normally there are two or three categories of priority representing (highly) urgent and normal payments with the exception that MEPS+ has 5 levels and RIX allows up to 9 levels of payment priority. Second, existing systems all adopted the FIFO principle and its variations including reordering and bypass. Those systems that specifically mentioned FIFO-bypass (RITS, TARGET2, BOK-Wire+, BOJ-NET and RIX) and sorted queue (MEPS+ and CHAPS) in their offsetting algorithms are identified in Table 1. Third, even though bilateral offsetting and multilateral offsetting are used in combination, most systems run bilateral offsetting algorithms continuously throughout the day and the multilateral offsetting algorithms less frequently (eg BOK Wire+ every 30 minutes and BOJ-NET four times a day). Finally, BOJ-NET and RIX have dedicated LSM accounts, while a majority of the systems do not have separate LSM accounts but allow liquidity reservations for LSM queue processing.

The most recent RTGS system with an LSM is CHAPS in the U.K., which introduced its LSM in 2013. In this system, the RTGS mode and LSM mode are operated in a single stream of processing, where RTGS mode is switched to LSM mode every two minutes in a matching cycle (Denbee and McLafferty (2013), Dent and Dison (2012)). The matching cycle lasts around 20 seconds, where either bilateral or multilateral offsetting algorithms run to match and settle non-urgent payments. The offsetting algorithms may choose between different queue release methods: FIFO or sorted queue by value. Any payments not settled by a matching cycle remain in the queue and the system will attempt to match and settle them in the next cycle.

The Bank of Japan operates the RTGS and LSM separately as two independent mechanisms. Participants are required to transfer funds to the LSM account at the beginning of each settlement day. Additional transfer of funds to the LSM account is possible at any time during the day. At the end of the day, the balance in the LSM account is automatically transferred back to the RTGS account. In BOJ-NET, the bilateral offsetting is the primary LSM mechanism, which runs continuously during the day, while multilateral offsetting is supplemental, which runs at four fixed times (10:30, 13:30, 14:30, and 15:30) each day. The offsetting algorithms follow the basic FIFO

Table 1: Properties of selected centralized queuing mechanisms.

Country	System	Year of LSM	Centralized Queueing Functions				Dedicated LSM Account
			Priority of payments	Queue release method	Bilateral offsetting	Multilateral offsetting	
Australia	RITS	2009	Priority Active Deferred	FIFO Bypass	Continuous	NA	N
Eurosystem	TARGET T2	2007	Highly urgent Urgent Normal	FIFO Bypass	Continuous	Continuous	N
Korea	BOK-Wire+	2009	Urgent Normal	FIFO Bypass	Continuous	Runs every 30 minutes	N
Japan	BOJ-NET	2008	Urgent Non-urgent	FIFO Bypass	Continuous	Runs 4 times a day	Y
Mexico	SPEI	2004	High priority Normal	FIFO	Continuous	Every few seconds	N
Singapore	MEPS+	2006	5 levels	FIFO Sorted Queue	Continuous	Continuous	N
Sweden	RIX	2009	9 levels	FIFO Bypass	At certain specified intervals	At certain specified intervals	Y
Switzerland	SIC	2008	High Norm	FIFO	Every few seconds	NA	N
U.K.	CHAPS	2013	High priority Non-urgent	FIFO Sorted Queue	Every 2 minutes	Every 2 minutes	N

release order, but participants are allowed to change the order of payments (Nakajima 2015).

Increasing computing power makes it possible to implement complex settlement mechanisms and offsetting algorithms can now be applied on a continuous basis. Technological progress will broaden the set of feasible designs and support the development of innovative new solutions. However, technological progress alone may not allow operators of centralized queues to achieve outcomes that maximize participant welfare.

4 Problems with Centralized Queues

Central operators have no way of assessing the importance of payments (beyond the coarse priority categorizations the systems allow) or assessing individual bank's cost of liquidity provision, which can vary across banks and throughout the day. As such they cannot make accurate welfare assessments or set system parameters to maximize social welfare.[11]

Using the terminology of formal models of dynamic games (Fudenberg and Tirole (1991)) the problems faced by the central operator are ones of imperfect and incomplete information. In the case of imperfect information, a player is unable to observe the earlier or simultaneous moves of some other players. In the case of incomplete information, the player is unsure about some of the underlying characteristics of the game, such as another player's payoff.

4.1 Imperfect Information

Imperfect information arises because the central operator does not see all the moves of all the participants in the system. Actions that are hidden from the central operator include the arrivals of payment requests to participants from their customers. This has always been a challenging aspect of RTGS systems with or without LSMs. The smooth functioning of any payment system requires

[11] There is a literature that predates the consideration of decentralized LSMs that evaluates the welfare impacts of introducing LSMs to RTGS systems: eg Atalay, Martin and McAndrews (2010), Diehl and Schollmeyer (2011) and Diehl (2013). These studies use the stylized models of Martin and McAndrews (2008) and Jurgilas and Martin (2013) which assume there is a single matching cycle in the morning period and all payments entered into the queue perfectly offset. This work attempts to measure welfare benefits based on aggregate statistics of the underlying system. This work abstracts from the queue implementation and management features that are the focus of this proposal.

that participants process payments in a timely fashion. Since liquidity provision is costly, banks have an incentive to free-ride on the other system participants by waiting for incoming payments in order to obtain liquidity to make outgoing ones. Detecting free-riding behavior is difficult, however, because of imperfect information (See Denbee et al (2012, 2015) and Diehl (2011)). The central bank does not know when payment requests by customers arrive at banks. As such RTGS systems usually employ relatively weak guidelines to ensure timely payment processing, such as through put guidelines that must only be met on average over periods of time. For instance, CHAPS participants in the UK are required to complete 50% by noon and 75% by 2:30 pm, but this restriction is only assessed as an average over a monthly period and if a bank is in violation the "punishment" is that they must appear before the Star Chamber (a collection of CHAPS members) and explain their behavior. In short, it is difficult to enforce good behavior when it is not possible to definitively detect bad behavior. This is true in RTGS systems with or without a central queue.

4.2 Incomplete Information

Even if the central operator had perfect information about the payment obligations of all participants in the system (i.e. it either can see the requests come in from customers or see the decisions of participants to process or not process these payments), it would still have difficulty setting up a welfare maximizing system because it does not know all of the costs and benefits of participants. One the benefit side, we have already discussed that the most urgent payments have to be settled immediately and will not be entered into a queue by the bank, but rather will be settled immediately using the pure RTGS stream[12]. However, within the set of payment requests that banks receive that do not require immediate settlement there is variation in the degree of urgency. Existing systems sometimes allow banks to specify a priority level when they enter payments into a queue. Systems that allow this typically only permit two categories: urgent (high priority) and non-urgent (normal). TARGET2 allows three levels and MEPS+ allows five levels of priority (see Table 1).

There are good reasons for limiting the number of categories as partitioning the payments set too finely can limit netting opportunities. Our point here, however, is that any degree of coarseness in information limits the ability of the central operator to distinguish between any two payments within a category and hence payments may be selected for processing in a suboptimal order.

The other important piece of private information that participants possess is their cost of providing liquidity. The opportunity cost of funds that participants provide to a central queue for settling net obligations depends on their own cost of liquidity and their predictions on future liquidity needs. Banks often set limits on funds they are willing to make available to the centralized queue at the end of any matching cycle, and they use their private information to inform this decision. However, given specified limits the central operator does not have this information available when deciding whether to settle one set of payments using particular sources of liquidity versus another set using different sources.

4.3 Prefunding and Hard Limits

A constraining aspect of centralized queues, which is not a direct consequence of imperfect or incomplete information, is that banks make decisions on how much liquidity to provide to centralized queues before knowing what the benefits are the central queue operator has to look for netting solutions that are subject to these given liquidity amounts. In other words, netting algorithms look for a solution subject to hard limits. It is not difficult to construct examples where a large number of high-value payments could be processed from a central queue if a small amount of additional liquidity was provided, however, we know of no centralized queue systems with functionality to request and receive additional liquidity during a matching cycle.[13]

[12] Regardless of the liquidity savings opportunities that systems provide, all systems must and do allow a RTGS stream

[13] In principle, the additional liquidity needed to net large sets of payments could of provided, when deemed worthwhile, by the central operator. However, this would expose the operator to credit risk.

5 Decentralized Multilateral Netting

The basic premise of the decentralized approach is that netting proposals are made by and agreed upon by the participants rather than the central operator. In order for this to happen, participants must be able to collect information about what the netting opportunities are, across system participants, at any point in time. This requires banks with existing (unsettled) payment obligations to be willing to announce these obligations to their respective obligees. A payment obligation is a payment request that has been received from a system participant's (i.e. bank's) customer, or an obligation arising from a bank's own proprietary operations (e.g. interbank trading). The bank knows the details of the payment, such as the amount, the payee, and also the urgency of the payment, namely some measure of the cost of delay to the customer, and indirectly itself through the impact on the bank-customer relationship if the payment is not made within a certain time frame. Often, obligations are not visible to any other party unless the obilgor makes them visible by revealing them. Banks may or may not want to inform obligees of every payment.[14]

An obligation is ultimately either settled via a cash payment, via one or more new obligations, or via a combination of the two. An obligation may also, optionally, be cancelled.

Given a series of bilateral obligations, liquidity saving can be reduced to a 3-stage process:

1. Detect obligations between various participating nodes.

2. Plan a strategy to successfully meet payment obligations on a net basis.

3. Execute the plan through a single atomic netting transaction.

The detect stage operates independently of the other two stages. In theory, the plan and execute stage, making use of results from an earlier detect phase, can operate in parallel with another detect stage.

5.1 Stage 1: Detect

The detect operation is based around a concept of a *recursive graph scanning algorithm.*[15] Once banks have broadcast their (selected) payments obligations to the obligees, any individual bank can obtain a complete picture of the netting opportunities that involve that bank using this algorithm.

Consider, for example, the graph displayed in Figure 1. In this graph, we see a series of vertices, labelled A through Q, representing nodes in a network. The edges and directions represent the payment obligations that have been broadcast by the obligors to the obligees. In reality there would be payment values (weights) associated with these edges, and in fact there could be multiple payments (a vector of weights) associated with an edge, however we suppress this information in the diagram. During the detect phase any node with payment obligations may initiate a scan. We can identify each scan with a unique scan ID, which allows each specific scan to be tracked.

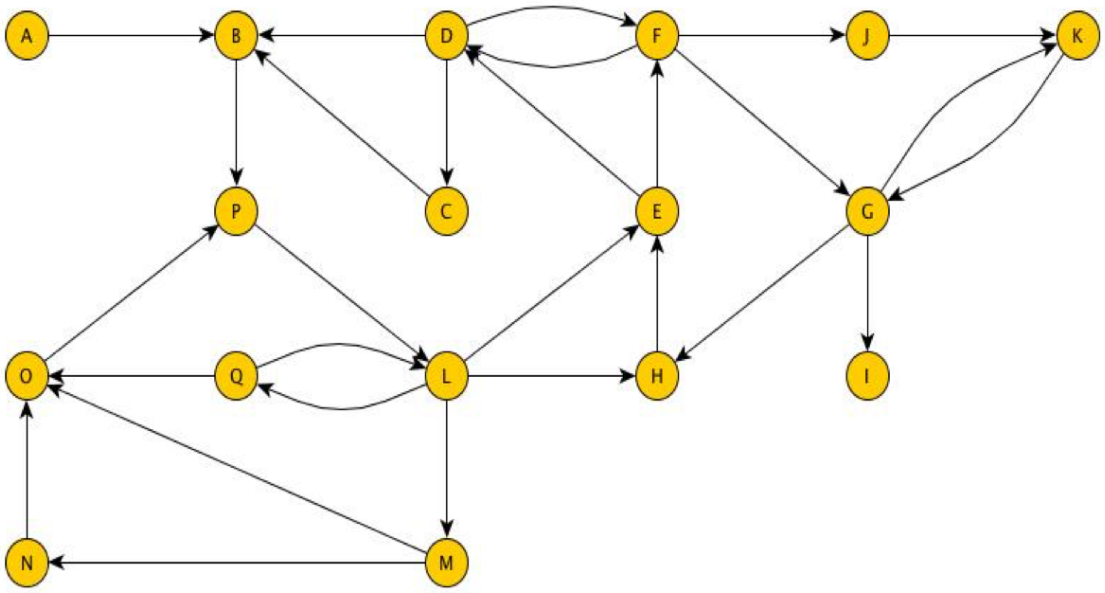
As a node starts to participate in a scan (either as the initiator, or after having been contacted as a result of the scan being initiated by another node) the following actions occur:

- The node checks all payment obligations that it has made to other nodes. For all obligations that it would like to see involved in netting it adds these to a list of graph edges. This does not *need* to include all (or even any) obligations - only those that it would want to be involved in a netting solution. This may include more than one edge if there are two or more obligations from the same obligor to the same obligee.
- The node sends a scan request (SCAN-REQUEST) message to each node with which it shares a payment obligation.

[14] This same situation arises in centralized systems with transparent queues (See Table 1). To the extent that obligors do not wish to inform obligees about impending payments, netting cannot be achieved in either centralized or decentralized systems.

[15] The Ubin phase 2 report referes to their gridlock resolution process, which includes the final netting proposal and payments, as their *cycle-solver* algorithm. The recursive graph scanning algorithm can be thought of as a component of the broader cycle-solver algorithm.

Figure 1: Obligation network involving banks A through Q.



- Any node sending a SCAN-REQUEST message starts a timeout timer and waits for a scan acknowledgement (SCAN-ACK) message to be returned to it before the timer expires. If the timeout timer expires then it presumes that the request has failed and that the recipient will not respond. Any messages received after a timeout must be ignored.
- Nodes that receive a SCAN-REQUEST message must respond with a SCAN-ACK message. The SCAN-ACK contains no information other than that the SCAN-REQUEST was received. A node that issues a SCAN-ACK is committed to issuing a subsequent scan response (SCAN-RESPONSE).
- The recipient of a SCAN-REQUEST proceeds as a newly reached node.
- When all data has been collated by a node then it sends a SCAN-RESPONSE message to whichever node sent it a SCAN-REQUEST.
- The SCAN-RESPONSE message will indicate the final status of a request. This may indicate a set of edges found during scanning, or that another scan has already been seen with a lower scan ID within a set time frame (eg the last 20 seconds). The timing can be altered to be appropriate for the size of the network.
- If a node receives a SCAN-RESPONSE reporting a scan collision in which a lower value scan ID (treating scan IDs as a large integer) has already been seen then the scan will be stopped. If the recipient of the SCAN-RESPONSE is, itself, waiting to return data to an earlier requester then it will issue a SCAN-RESPONSE message to its requester indicating the scan collision.
- When a node has received all pending SCAN-RESPONSE messages it aggregates all of the results into one larger message, removing any duplicates.

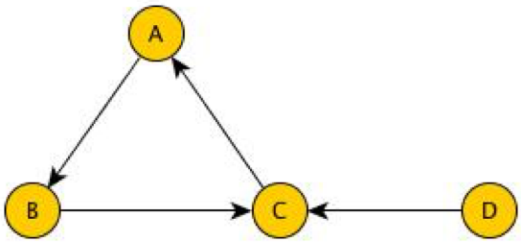
When the final SCAN-RESPONSE message is received by the initiator its final aggregation of results will represent all of the edges in the graph, and the full graph can be trivially recomputed.

We illustrate the recursive graph scanning algorithm in terms of the simple example shown in Figure 2.

While any node may initiate a scan, for the purpose of illustration it is convenient to assume it is started by node A.

1. A initiates the scan and requests nodes B and C continue it. A will record edge AB.

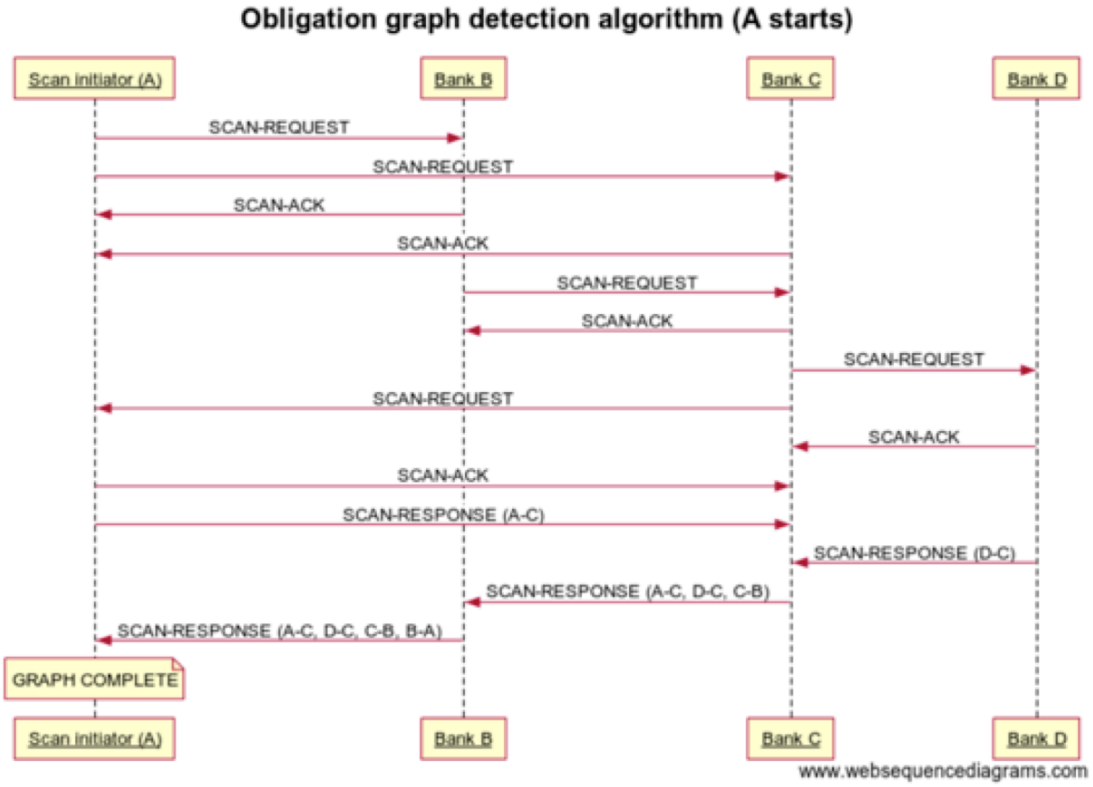
Figure 2: Obligation network involving banks A through D.



2. Node B subsequently requests that node C continue the scan, while node C requests both nodes B and D continue.
3. Node D responds with the details of the edge DC.
4. At this point it is possible that either node B, C or A may take different actions, dependent on the timing of the scan, but node C will ultimately bundle the reply from D regarding edge DC, while adding details of the edge CA. Node B offers details of the edge BC
5. Node A collates the replies from B and C, forming a view of a graph with edges AB, CA, BC and DC.

The above steps are illustrated in Figure 3.[16]

Figure 3: Illustration of the recursive graph scanning algorithm.



[16] In the Appendix, we provide a technical description of the implementation of the recursive graph scanning algorithm using Corda and provide a link to the software.

5.1.1 Scan Complexity

The scan complexity is a function of the worst-case graph walk. In most instances, our graph will be predominantly a tree structure, but our worst-case scenario is a purely linear graph, initiated by one of the least-connected nodes. In this, worst-case, scenario our scan complexity is $O(n)$ (twice linear) where n is the number of nodes in the graph. More typically the complexity is $O(\log(n))$ as we are performing a tree scan.

5.1.2 Concurrent Scanning

Given that any node might wish to initiate a scan at any time, it is possible for two or more nodes to start simultaneously. In this case there must be a procedure that ensures all but one deterministically back-off. This is achieved by having each scan initiator generate a unique scan ID and ensuring that the lowest value ID always succeeds. To ensure fairness in the selection of scan IDs, an algorithmic approach is required, and compliance can be enforced by each node within the scan.

Here is one such proposal. When the initiating node is ready to start scanning it computes a SHA2-256 hash over its pseudo-anonymous identity, concatenated with a starting timestamp. This hash value will be the scan ID. The SCAN-REQUEST message will include both the identity, timestamp, and scan ID. Recipients of scan requests will also compute the same SHA2-256 hash over the identity and timestamp to ensure that the initiator computed a fair scan ID. The randomizing nature of the hash computation means that no one node will be favoured over any other.

It is worth noting that if two nodes initiate scans at the same time, but the sets of payment obligations reachable from both are not connected, then both nodes may complete scans of their respective graphs. Both may attempt to find netting solutions independently of each other.

5.1.3 Privacy Concerns

The scanning process involves sharing payment information with parties that are not involved on either side of a transaction. Implementation of this scheme may require that participants agree in advance to allow the required payment information to be shared. Doing so without protecting the information may not be desirable, or even allowed, in many applications.

An option to protect information is to consider using pseudonyms, created by banks to protect their identities from other banks, in order to limit information sharing. In such a system, the link between banks' true identities and their pseudonyms should, and could, be known by the regulator. A process for creating, managing, and retiring pseudonyms is found within Corda's confidential identity framework. Pseudonyms, alone, are insufficient to prevent information leakage, but could potentially be combined with encryption and secure computing enclaves, such as Intel's SGX, to make it impossible for an observer to gather any insights into payment information.

With a secure computing enclave, even the operator of the CPU in which the enclave exists would be unable to determine the data that is being used to compute a netting solution. Such data would only reside in an encrypted form with pseudoanonymous identifiers.

5.2 Stage 2: Plan

Having completed a scan and thus having knowledge of existing payment obligations for its connected components of the network, a node is ready to make a liquidity saving proposal. The detect phase provides the necessary information to construct an in-memory graph representation, and from this nodes are able to use standard approaches to finding cycles within that graph representation. Simple sets of payment obligations may have only one or two cycles, larger sets may have significantly more. Consider our earlier example in Figure 1 again:

In this payment graph, we can see several examples of cycles with shared edges. For example, the cycles BPLEDB, BPLEFDB, and BPLEFDCB all share the edges BP, PL, LE. Similarly, the

cycles BPLEFDB and EFGHE are quite different but share the edge EF. In cases where we have multiple cycles that share a common edge, resolving any one will also eliminate the other cycle too. Similarly, though, a shared edge may indicate that it would be advantageous to try to solve two or more cycles simultaneously.

It is also important to recognize that we have no guarantees that any particular set of payments may be possible. There is no concept of "locking" the state of the payment network while the decentralized LSM process operates, so while the LSM is running, individual payment obligations that are considered to be part of the netting solution may be settled unilaterally, and thus not be available to be re-settled as part of a netting solution. In such cases some possible netting approaches that are produced by the planning stage may prove to be irrelevant, so the planning stage should not try to produce a single netting transaction. Instead it should calculate many possible independent netting transactions, prioritizes them, and try them in turn.

One option for a liquidity savings proposal is to try and clear all existing payment obligations simultaneously. This is the unconstrained solution, as it would be the solution if there were no liquidity constraints. In fact, the sum of liquidity savings obtained from settling the net amounts associated with any collection of cycles cannot exceed the savings from settling all payments simultaneously. Hence, the unconstrained solution maximizes netting efficiency.

5.2.1 Unconstrained Solution

Denote the set of banks in the system by $N = \{1, ..., n\}$. Start with a matrix P which represents all of the payment obligations revealed by the current scan. Let p_{ij} denote the payment from bank i to bank j, and there are t^{ij} such payments. The net debit (or credit if the debit is negative) position of bank i is given by

$$d_i = max \left\{ 0, \sum_{j \neq i} \sum_{k=1}^{t^{ij}} p_{ijk} - \sum_{j \neq i} \sum_{k=1}^{t^{ji}} p_{jik} \right\}. \tag{1}$$

The first double-summation term in (1) adds up all of the outgoing payments from i and the second double-summation term in (1) adds up all of the incoming payments to i. The unconstrained solution can be obtained if each bank i is willing to to provide liquidity d_i.

In a centralized system, each bank i typically pre-specifies an amount l_i of liquidity that they are willing to provide to the queue. The goal of a centralized queue is to settle all the payments in the queue. However, this is only possible if the liquidity provided by each bank is enough to cover their net position (if negative). Specifically, settling all payments in the queue requires $l_i \geq d_i$ for all i. If this condition is not satisfied, then it is not possible to settle all payments in the queue and some must be discarded (technically, set aside until the next matching cycle or until the bank decides to make the payment outside the queue). We refer to a situation where the set of payments cannot be settled with the available liquidity as *uncovered*.

As discussed above, banks can use different criteria for deciding which payments to eliminate from the queue when the payment set P is uncovered by the available liquidity. A main advantage of the decentralized approach is that banks can decide after seeing a netting proposal whether or not they are willing to provide a prescribed amount of liquidity. Banks can assess a take-it-or-leave-it offer by comparing the benefits they get from having a set of payments settled atomically at that instant against the instantaneous cost of liquidity provision and/or side payments.

In fact, the current benefits and costs of settling a payment now are uncertain, because not settling now means settling at a future instant, the timing of which may be influenced by future payment arrivals, decisions to broadcast these arrivals as new payment obligations and decisions to include payments in new offers and accept them. Modeling this decision in a way that fully takes into account all these future trade-offs may be computationally intractable.[17] Instead, we assume

[17] We would have to specify a distribution on all payment arrivals, specify equilibrium strategies of the dynamic game in which proposals are made and the benefits and costs of accepting proposals are specified in terms of the likelihood of future payments arrivals, the inclusion of current payments in future proposals and the likelihood of accepting a payment in a future payment proposal given the computed equilibrium strategies (if these exist and are unique).

banks can determine a constant, instantaneous delay cost that reflects the current per dollar benefit of processing each individual payment as an immediate atomic transaction versus having it remain unsettled and awaiting inclusion in another (future) proposal or being reassigned for gross settlement. This benefit should depend on the priority level of individual payments, but is otherwise assumed to be constant per dollar within payments of the same priority.

Specifically, if we allow for an arbitrary number of priority levels for outgoing payments $h = 1, ..., H$, then at each instant we assume the marginal benefit of settling a dollar's worth of a priority h payment immediately is b^h. Likewise, we assume that each bank i has a known per dollar cost of providing liquidity at the current settlement opportunity, which we denote by c^i. The cost of providing liquidity at the current instant does not depend on the priority of the payment, but may differ across banks (and intraday).

Using these parameters banks can assess the instantaneous benefits and costs of any netting proposal. In particular, we can evaluate the unconstrained solution that clears all existing payment obligations. The same approach can be used to evaluate proposals for clearing subsets of payments.

We start by determining the net benefits of solutions without side payments. That is, we assess whether banks might accept netting proposals that involve each bank providing the liquidity to cover its own net debit position.

In this case the computation of net benefits is straightforward. Using the above notation, the net benefit to bank i of the unconstrained solution would be:

$$\sum_{j \neq i} \sum_{k,h} b^h \times p^h_{ijk} - c^i \times d_i. \tag{2}$$

In principle, we might expect that any bank i will agree to the unconstrained solution so long as (2) is positive. However, this solution may not provide a fair division of surplus and for these reasons may be rejected by some participants. Again, the same can be true of any proposal involving subsets of P. To address the issue of fairness we introduce the notion of *side payments*.

5.2.2 Proposals with Side Payments

We begin the discussion of side payments with a very simple example in which there are only two participants in the system, whom we conveniently name bank A and bank B. Suppose bank A owes bank B $100 and bank B owes bank A $80. If bank A and bank B are able to net these obligations, then they will only need $20 worth of liquidity. So far, we have assumed that the individual who is a net debit position (in this case bank A) be the one to provide the required liquidity (in this case $20). However, is this fair? If bank B did not submit his payment bank A would have to provide $80 worth of liquidity. And, of course, if bank A did not submit her payment bank B would have had to provide $100 worth of liquidity. The point is that both participants are made better off due to the netting so perhaps they both should contribute to the liquidity cost. Or, more pertinently, bank A might have an expectation that bank B should contribute to the liquidity cost and hence not accept a proposal by bank B that does not meet this expectation.

We need a fair way of allocating costs associated with obtaining the joint benefit. Economists often turn to the Shapley value (Shapley 1953) in situations like this. There is a solid justification for using the Shapley value in cost allocation problems. Roth and Verrecchia (1979) show that, under reasonable assumptions, it provides the same expected utility the participants would expect to get from bargaining to an uncertain outcome.

To define the solution suggested by Roth and Verrecchia, we must define the Shapley value of each participant in any netting proposal, and to do this, we first must introduce the notion of a *characteristic function*. The characteristic function $v(S)$ defines the total net benefit that can be obtained by a group of banks $S \subseteq N$ by settling their combined payment obligations on a net basis.

Using the above notation,

$$v(S) = \sum_{i \in S} \left[\sum_{j \neq i} \sum_{k,h} b^h \times p^h_{ijk} - c^i \times d_i \right]. \tag{3}$$

That is, $v(S)$ is computed as the sum of the accrued benefits from payments settled minus the combined costs of supplying the liquidity required to settle them. Then, following Shapley (1953) the Shapley value for bank i is given by

$$w_i = \sum_{S \subset N} \frac{(s-1)!(n-s)!}{n!} \left[v(S) - v(S-i) \right], \tag{4}$$

where s is the number banks in group S. The Shapley value of bank i is the weighted sum of the terms $[v(S) - v(S-i)]$, which represent bank i's marginal contribution to coalition S. It can therefore be interpreted as a bank's expected marginal contribution to a coalition of banks that seek to net payments, based on the assumption that each bank's sequential arrival to the coalition is determined randomly.

Each bank i's share of the cost burden of providing liquidity to settle the payments in P, denote this by C_i^P, is then defined as the gross benefit to bank i of having its own payments in P settled minus its Shapley value:

$$C_i^P = \sum_{j \neq i} \sum_{k,h} b^h \times p_{ijk}^h - w_i. \tag{5}$$

A property of the Shapley value is that it assigns the total value of the coalition to its members (efficiency), so that the sum over all banks of the terms in (4) equals $v(N)$. It follows that the sum of the cost shares in (5) over all of the banks cover the actual cost of the proposal: ie $\sum_{i=1}^n C_i^P = \sum_{i=1}^n c^i d_i$.

To actually implement these cost shares we propose the use of side payments. That is, each bank provides liquidity equal to its net debit position and then side payments are made to so that the final cost share to each bank equals the amount specified by (5). Formally, side payments from bank i to bank j associated with any solution of the type given by (5) can be defined as

$$SP_{ij} = \begin{cases} 0 & \text{if } d_i > 0 \\ (c_j d_j - C_j^P) \frac{C_i^P}{\sum_{s:d_s=0} C_s^P} & \text{otherwise.} \end{cases} \tag{6}$$

As a simple (but not overly trivial) illustration let us consider three banks: A, B and C. Assume bank A owes bank B \$100, bank B owes bank C \$80 and bank C owes bank A \$70. Bank A has a net debit position of \$30 and both bank B and bank C have net credit positions of \$20 and \$10, respectively. $V(S) = 0$ for all $S \subseteq N$ except $S = N$. When all three join together there is a netting cycle that clears \$250 worth of payments with \$30 in liquidity. Assume that all the payments are the same priority and that the benefit per dollar to each bank of settling those payments at the current instant is $b = .05$. In addition, assume that the instantaneous, per-dollar cost of liquidity provision is the same for all banks and is equal to $c = .1$.[18] Then the total net benefit that can be obtained by the three banks by settling their combined payment obligations on a net basis is equal to $v(A, B, C) = .05 \times \$250 - .1 \times \$30 = \$9.5$.

To compute the Shapley value for each bank, we list all the possible orderings of the banks and then take the average over all orderings of the marginal contributions of each bank to the total net benefit.[19] Since there are three banks, there are six possible orderings as shown in Table 2.

The calculation of the marginal contribution can be understood as follows. Let us take the order A,B,C as an example. If bank A pays \$100, the benefit is \$5 and the cost is \$10. So bank A would not choose to clear its payment obligation to bank B. The marginal contribution of bank A under this ordering is therefore \$0. Next, if both bank A and bank B form a coalition, the total settlement benefit is \$9 and the total liquidity cost is \$10. So they would choose not to clear their payment obligations. The marginal contribution of bank B is therefore also \$0. Finally, if all three banks form a coalition, the total benefit of clearing their obligations is \$12.5 and the total liquidity cost is \$3. With a net gain of \$9.5, all three banks would choose to clear their obligations. The marginal contribution of bank C is thus \$9.5.

[18] Both b and c should be small relative to magnitudes of liquidity provided. They only reflect the costs and benefits of settlement now versus an uncertain point in the near future. It is also reasonable to set b smaller than c. This is consistent with assumptions made in Bech and Garratt (2003, 2012). If it were not, then one might expect the bank to settle the payment via the standard RTGS stream.

[19] This is the same computation as (4).

Table 2: Shapley value equals the average of each bank's marginal contributions over all of the different orderings.

Order	MC A	MC B	MC C
A,B,C	0	0	9.5
A,C,B	0	9.5	0
B,A,C	0	0	9.5
B,C,A	9.5	0	0
C,A,B	0	9.5	0
C,B,A	9.5	0	0
Shapley value	3.16667	3.16667	3.16667

In this case, the average marginal contribution is the same for all three banks since each is pivotal in completing the netting cycle the same number of times. Therefore, the Shapley value is the same for all three banks.

Next, we compute bank i's share of the cost burden of providing liquidity to settle the proposed payments. In this simplified setting: $(C_A^P, C_B^P, C_C^P)\&(b \times p_{AC} - w_A, b \times p_{BC} - w_B, b \times p_{CA} - w_C)\&(.05 \times 100 - 3.16667, .05 \times 80 - 3.16667, .05 \times 70 - 3.16667)\&(1.83333, 0.83333, 0.33333)$. These cost shares can be realized by having bank A provide \$30 in liquidity at a cost of \$3 and having banks B and C make side payments to bank A of \$0.83333 and \$0.33333, respectively. One can argue that this solution is more fair and equitable than bank A providing all the liquidity and receiving no side payments.

5.3 Stage 3: Execute

Having identified a set of payment obligations for which a net settlement is possible, the node running the LSM may initiate sub-flows to each participant. Each will be asked to construct a payment to one or more other participants, on the basis that was agreed to during the planning stage. Each of these payments will be bundled together into one single atomic transaction. The transaction will then be sent to each participant to be checked, verified, and signed, before being finalized through the notary service and stored into each relevant node's vault.

6 Concluding Remarks

This paper makes a proposal for a decentralized liquidity savings mechanism and it argues that such a scheme could be welfare improving. The potential for welfare improvement comes, in part, from the ability of decentralized proposals to utilize private information of the participants. This occurs because proposers know their own information and can learn about the information of others through repeated interaction. In essence, we are advocating that a market solution might dominate the planner's solution, however, this remains an open question.

One potential obstacle to realizing welfare gains is the degree of complexity involved in determining netting solutions and proposing cost allocations. There is a lot of information to process. Banks need to constantly assess their own liquidity situation, forecast their future liquidity needs and availability, and, ideally, aggregate information from accepted and rejected proposals over time in order to determine other banks payment characteristics. It is possible that there is too much complexity for treasury managers or any human to process in real time. There is thus likely to be a role for Artificial Intelligence or machine learning.

Questions also arise as to how well a decentralized LSM might perform under extreme stress scenarios. If voluntary proposals break down, then there is heightened potential for gridlock. For this reason, it might be desirable to have a more automated gridlock resolution mechanism in place, as a backstop, with proposals initiated by the central operator.

Appendix: Implementation of the Recursive Graph Scanning Algorithm Using Corda

The detect, plan and execute logic are all immediate subdirectories of the ubin-corda git repo, available at https://github.com/project-ubin/ubin-corda.

6.1 SCAN-REQUEST

The SCAN-REQUEST message has the following form:

- Initiator node identity: 32 bytes
- Timestamp: 8 bytes (integer, little endian, UTC time in microseconds since 1970-01-01:00:00:00)
- Scan ID: 32 bytes (SHA2-256 hash over the initiator node identity and timestamp)
- Time to live: 4 bytes (positive integer, little endian, timeout in microseconds)

The time to live (TTL) value is set by the initiating node, and will represent the total amount of time that the scanning process may take before timing out. As each new node propagates the SCAN-REQUEST it must reduce this value to ensure that it will time-out any failed requests with enough time to reply to the node that requested it to scan, and before that node times-out its request.

Nominally if the initial scan time is set to 6000000 microseconds (6 seconds) then each successive request should reduce this value by 100000 microseconds (100 ms). This allows for up to 60 scan propagations.

6.2 SCAN-ACK

The SCAN-ACK message has the following form:

- Scan ID: 32 bytes (the Scan ID from the original SCAN-REQUEST).

6.3 SCAN-RESPONSE

The SCAN-RESPONSE message has the following form:

- Scan ID: 32 bytes (the Scan ID from the original SCAN-REQUEST).
- Status: Integer
  - 0: Returning edge list
  - 1: Scan collided with a lower scan ID
- Edge count: Integer
- "edge count" edge response entries
- Liquidity offer count: Integer
- "liquidity offer count" liquidity offers

An edge response entry has the following form:

- Node ID 1: 32 bytes (node's pseudo-anonymous identity)
- Node ID 2: 32 bytes (node's pseudo-anonymous identity)
- Obligation value: Big decimal (value of the obligation)

A liquidity offer entry has the following form:

- Node ID: 32 bytes (nodes' pseudo-anonymous identity)

- Liquidity offer: Big decimal (amount of liquidity being offered by the node to help with net payments)

References

P. Angelini, An Analysis of Competitive Externalities in Gross Settlement Systems, Journal of Banking & Finance, 1998, 22, pp 1-18.

E. Atalay, A. Martin and J. McAndrews, "Quantifying the Benefits of a Liquidity-Saving Mechanism," Federal Reserve Bank of New York Staff Report, 2010, No. 447. Available at SSRN: https://ssrn.com/abstract=1151440 or http://dx.doi.org/10.2139/ssrn.1151440

A. Ball, E. Denbee, M. Manning and A. Wetherilt, "Intraday liquidity: risk and regulation," Bank of England Financial Stability Paper, 2011, No. 11.

M. Bech and R. Garratt, "The Intraday Liquidity Management Game," Journal of Economic Theory, April 2003, 109, 198-219.

M. Bech and R. Garratt "Illiquidity in the Interbank Payment System Following Wide-Scale Disruptions," Journal of Money, Credit & Banking. August 2012, 44(5), 903-929.

J. Chapman, R. Garratt, S. Hendry, A. McCormack and W. McMahon, "Project Jasper: Are Distributed Wholesale Payment Systems Feasible Yet?," Financial System, 2017, 59.

N. Davey and D. Grays, "How has the Liquidity Savings Mechanism reduced banks' intraday liquidity costs in CHAPS," Quarterly Bulletin, 2014, Q2.

E. Denbee, R. Garratt and P. Zimmerman, "Methods for Evaluating Liquidity Provision in Real-Time Gross Settlement Payment Systems," in Diagnostics for the Financial Markets – Computational Studies of Payment System, Simulator Seminar Proceedings 2009–2011, Chapter 3, Matti Hellqvist and Tatu Laine, Eds) Scientific Monographs E:45, 2012.

E. Denbee, R. Garratt and P. Zimmerman, "Identification of Over and Under Provision of Liquidity in Real-Time Payment Systems," Journal of Financial Market Infrastructures, December 2015, 4(2), 1-19.

M. Diehl, "Measuring Free-Riding in Large-Value Payment Systems: The Case of TARGET2," Journal of Financial Market Infrastructures, 2013, 1(3), 31-53.

M. Diehl and U. Schollmeyer, "Liquidity Saving Mechanisms: Quantifying the Benefits in TARGET2," Diagnostics for the Financial Markets–Computational Studies of Payment System, 2011.

European Central Bank and Bank of Japen, "STELLA Liquidity Saving Mechanisms in a Distributed Ledger Environment," September 2017, available at https://www.ecb.europa.eu/pub/pdf/other/ecb.stella_project_report_september_2017.pdf

D. Fudenberg and J. Tirole, Game Theory, MIT Press, 1991.

M. Galbiati and K.Soramäki, "Liquidity Saving Mechanisms and Bank Behavior in Payment Systems," Banking, Finance, and Accounting: Concepts, Methodologies, Tools, and Applications, 2015, 644-660.

Michael M. Güntzer, Dieter Jungnickel, Matthias Leclerc; Efficient algorithms for the clearing of interbank payments; European Journal of Operational Research 106 (1998) p. 212-219.

M. Jurgilas and A. Martin, "Liquidity-Saving Mechanisms in Collateral-Based RTGS Payment Systems," Annals of Finance, 2013, 9(1), pp.29-60.

A. Martin and J. McAndrews, "Liquidity-Saving Mechanisms," Journal of Monetary Economics, 2008, 55(3), 554-567.

Masashi Nakajima, "Liquidity Saving Mechanisms in Payment Systems and Settlement Liquidity," International Journal of Innovation in the Digital Economy (IJIDE), 2016, vol. 7, issue 4, 23-45.

Monetary Authority of Singapore, "Project Ubin Phase 2: Re-imagining interbank real-time gross settlement system using distributed ledger technologies," November 2017, available at http://www.mas.gov.sg/~/media/ProjectUbin/Project%20Ubin%20Phase%202%20Reimagining%20RTGS.pdf

Nakajima, M., "Liquidity Saving Mechanisms in Payment Systems and Settlement Liquidity: The Experience of Japan's Next-Generation RTGS Project," in Analyzing the Economics of Financial Market Infrastructures, Diehl, M. (ed.), IGI Global, 2015, 225-249.

Payments Canada, R3 and the Bank of Canada, "Jasper Project: A Canadian Experiment with Distributed Ledger Technology for Domestic Interbank Payments Settlement," October 2017, available at https://www.payments.ca/sites/default/files/29-Sep-17/jasper_report_eng.pdf

A. Roth and R. E. Verrecchia, "The Shapley Value as Applied to Cost allocation: A Reinterpretation," Journal of Accounting Research, Spring 1979, 17(1), 295-303.

David Seward, Saving Liquidity in a Liquidity-abundant World, OTC space.

R3 is an enterprise software firm using distributed ledger technology to build the next generation of financial services infrastructure.

R3's member base comprises over 80 global financial institutions and regulators on six continents. It is the largest collaborative consortium of its kind in financial markets.

Consortium members have access to insights from projects, research, regulatory outreach, and professional services.

Our team is made of financial industry veterans, technologists, and new tech entrepreneurs, bringing together expertise from electronic financial markets, cryptography and digital currencies.

corda is an open source, financial grade distributed ledger that records, manages and executes institutions' financial agreements in perfect synchrony with their peers.

Corda is the only distributed ledger platform designed from the ground up to address the specific needs of the financial services industry, and is the result of over a year of close collaboration between R3 and its consortium of over 80 of the world's leading banks and financial institutions.