UBIN PHASE 2

Technical Documentation

Hyperledger Fabric

This report describes the technical design and observations of the Hyperledger Fabric prototype in Ubin Phase 2.

Table of Contents

1 Platform and System Designs

Hyperledger Fabric is a platform for distributed ledger solutions underpinned by a modular architecture aimed to deliver high degrees of confidentiality, resiliency, flexibility and scalability. It is intended to allow for pluggable executions of various parts and suit the multifaceted nature and complexities that exist over the economic ecosystem. For this project, the prototype is developed using Hyperledger Fabric version 1.0.1, utilising the Go programming language for smart contracts (chaincodes) and Node.js for the application layer.

Hyperledger Fabric offers the capability to create channels, enabling a gathering of participants to share a ledger of transactions that are only privy to these participants. This is an alternative for systems where a few members may be contenders and do not need transactions to be known to competing members.

Hyperledger Fabric prevents double spending by having peers (i.e. participation nodes) validate the transactions against the endorsement policy to ensure correct allotment and authentication of the signatures. The endorsement policy is defined per chaincode to determine the number of endorsements and signatures (from the endorsing peers) required per transaction. Peers will also perform a versioning check to ensure data integrity. The Orderer receives endorsed transactions, packages them into blocks and broadcasts to all participants in the channel. The participants in the channel then validate these transactions before committing them to the ledger.

In Hyperledger Fabric all peers maintain the complete ledger, but only a subset of them called endorsing peers, execute transaction proposals. The set of endorsers required for endorsement is defined by the chaincode via the endorsement policy. This characteristic of separating consensus from execution and validation makes Hyperledger Fabric modular. The net result is that Fabric implementation is able to benefit from an ecosystem of different consensus protocols for implementing the ordering service. Separating the execution, ordering and validation phases also enables it to scale independently.

To maintain the privacy of fund transfers between banks, a bilateral channel is created for each pair of banks present in the network. In the example of a four banks network, Bank A would have four bilateral channels, one each with Bank B, Bank C and Bank D. In each bilateral channel, the bank will maintain its channel-level account which allows for the flexibility to assign a fixed amount of liquidity for transacting between a particular counterparty. For ensuring transactional validity, in the prototype, the endorsement policy for all bilateral transactions is set to require both parties in each bilateral channel to endorse before committing into a block.

Additionally, each bank is also a participant in two multilateral channels: funding channel and netting channel, where all banks would be participating in. An illustration of what a network of four banks would look like is as follows (Figure 1):

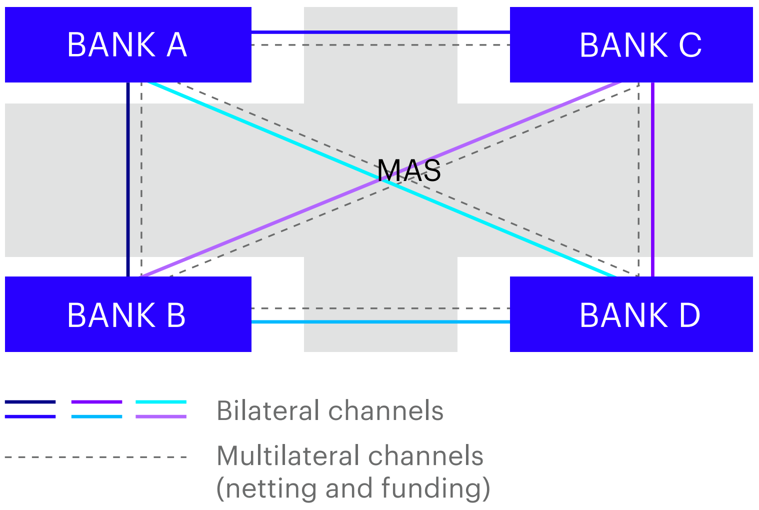


Figure 1: Hyperledger Fabric Channel Setup

The funding channel is used for banks to perform fund movements across the channel-level accounts in their bilateral channels. This multilateral channel will allow for auditability and traceability of such transactions. Both participants from the source channel are required to provide their endorsement for cross-channel fund movement to ensure that the fund being moved out is legitimate.

The purpose of the netting channel to cater for gridlock resolution where participants in the network can propose a set of unsettled payment instructions for the current gridlock resolution cycle. Since gridlock resolution potentially involves all parties in the network and all participants must agree to the final outcome, the endorsement policy set the netting channel chaincode requires all participants to endorse any transaction within the channel.

The Hyperledger Fabric prototype includes MAS as a participant in all channels, both bilaterally and multilaterally, providing MAS with the ability to audit and track all transactions in the Fabric network. Although MAS will not be directly involved in any bilateral transactions, MAS will be playing a special role in the netting channel to settle resolved transactions accordingly once a successful gridlock resolution solution is found.

1.1 Logical Architecture

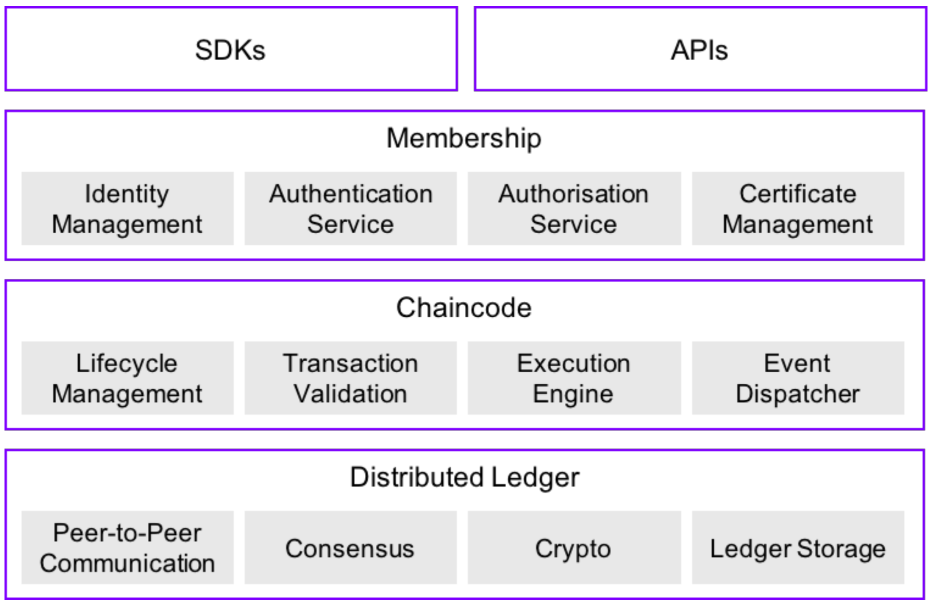


Figure 2: Key Components

Hyperledger Fabric is made of the following logical components:

- SDKs and API layer: The API layer is the client application that interfaces with the Fabric network through Fabric SDKs
- Membership layer: Authenticates, authorises and manages identities on Fabric
- Chaincode layer: A set of codes that define the business logic and are executed against the ledger's current state database. Responsible for processing transactions and determining if transactions are valid against business logic.
- Distributed ledger: Contains the blocks in the chain and state data maintained by each peer

1.2 Physical Architecture

Ubin Phase 2 Hyperledger Fabric prototype was deployed on 13 Azure virtual machines (VMs). Each virtual machine is 1 core, 3.5 GB RAM running on Ubuntu Server 16.04 LTS.

The VMs are organised as:

- 1 Orderer VM
- 12 Fabric Peer VMs:
    - 1 for MAS, acting as both, the Regulator and a Central Bank
    - 11 for the commercial banks (1 VM per participating bank)

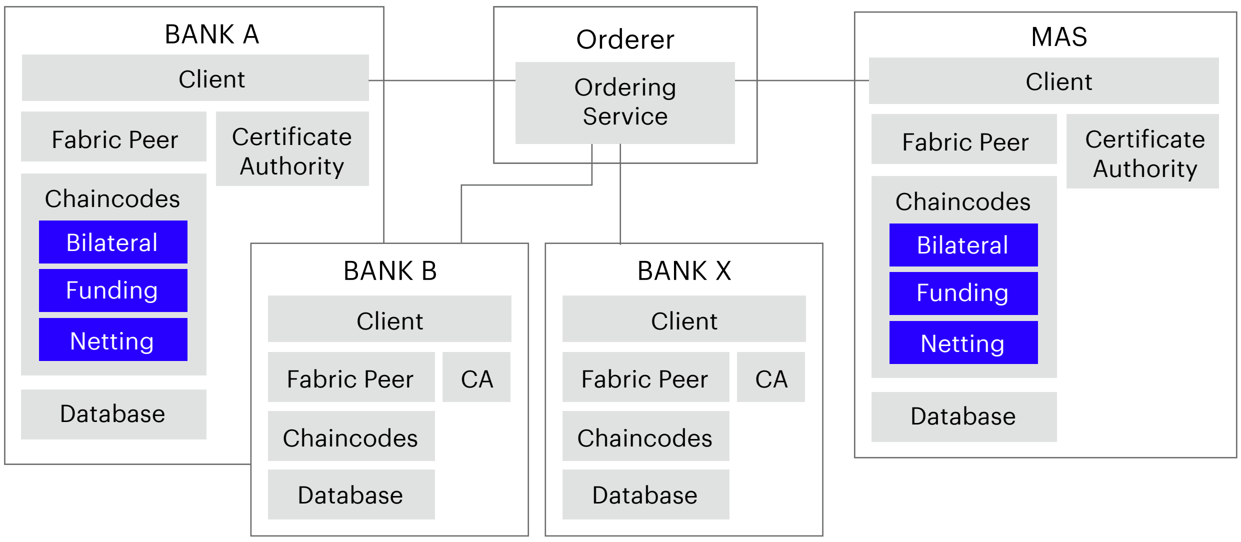


Figure 3: Ubin Phase 2 Hyperledger Fabric Network

In Ubin Phase 2, Fabric services (except the client) are started using Docker containers. The components deployed in the Fabric environment are:

- Orderer: In Ubin Phase 2, a solo orderer is deployed in a dedicated Docker container. The orderer is started up when the base Fabric network is started
- Fabric peer: Receives ordered state updates in the form of blocks from the ordering service and maintains the state and the ledger.
- Client (API layer): Runs on Node.js and connects to the Fabric network through Fabric Node SDK. This is run as a separate service.
- Certificate authority (CA): Component that governs identity registration and certificate management. There is one CA per organisation and the CA is started when the base Fabric network is started.

- Chaincodes: There are 3 chaincode containers per peer VM: bilateral chaincode, netting chaincode and funding chaincode. These containers are started when the chaincodes are instantiated on a peer.
- Database: This is the ledger storage storing the blocks within the chain and the data states. In Ubin Phase 2, CouchDB is used as the database, a JSON document store that allows rich queries. Each peer VM has its own CouchDB container that is started when the base Fabric network is started.

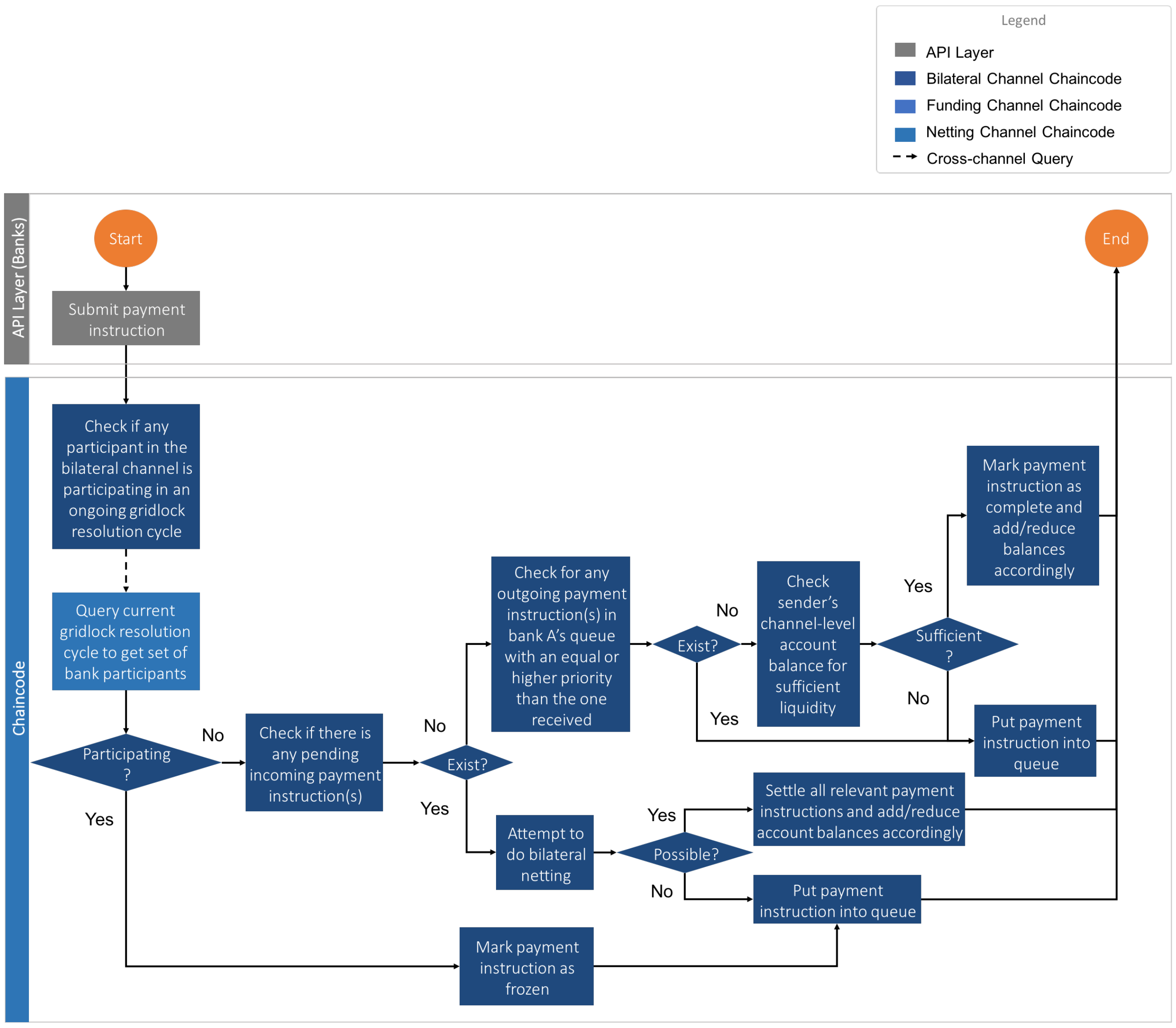
2 Functional Requirements

2.1 Fund Transfer

Fund transfer between banks are executed in the bilateral channel which both banks are part of. If there is sufficient liquidity in the channel-level account to fulfil the incoming payment instruction, it will be settled immediately (i.e. the balance of the sender's channel-level account will be decreased while increasing the balance of the receivers account and marking the payment instruction as complete directly).

As part of how channels function in Hyperledger Fabric, both banks will have visibility over all states in the channel which includes all pending payment instructions and channel-level account balances of both parties in the channel. Therefore, the bilateral channel design inherently supports bilateral netting to settle queued payment instructions during fund transfer which is implemented as part of the design of the Hyperledger Fabric prototype. The aim of having bilateral netting in place during fund transfer is to reduce the likelihood of requiring a triggering of multilateral gridlock resolution which will potentially involve all banks in the network. This introduces the possibility that queued payment instructions in a bilateral gridlock can be resolved within the bilateral channel without having to depend fully on gridlock resolution with multiple parties in the network.

The following flow diagram (Figure 4) depicts the logical flow of a fund transfer on the Hyperledger Fabric prototype:



Legend
API Layer
Bilateral Channel Chaincode
Funding Channel Chaincode
Netting Channel Chaincode
Cross-channel Query
API Layer (Banks)
Chaincode
Start
End
Submit payment instruction
Check if any participant in the bilateral channel is participating in an ongoing gridlock resolution cycle
Query current gridlock resolution cycle to get set of bank participants
Participating ?
No
Yes
Check if there is any pending incoming payment instruction(s)
No
Exist?
Yes
Check for any outgoing payment instruction(s) in bank A's queue with an equal or higher priority than the one received
No
Exist?
Yes
Check sender's channel-level account balance for sufficient liquidity
Sufficient ?
Yes
No
Mark payment instruction as complete and add/reduce balances accordingly
Put payment instruction into queue
Attempt to do bilateral netting
Possible?
Yes
No
Settle all relevant payment instructions and add/reduce account balances accordingly
Put payment instruction into queue
Mark payment instruction as frozen

Figure 4: Fund Transfer Process Flow

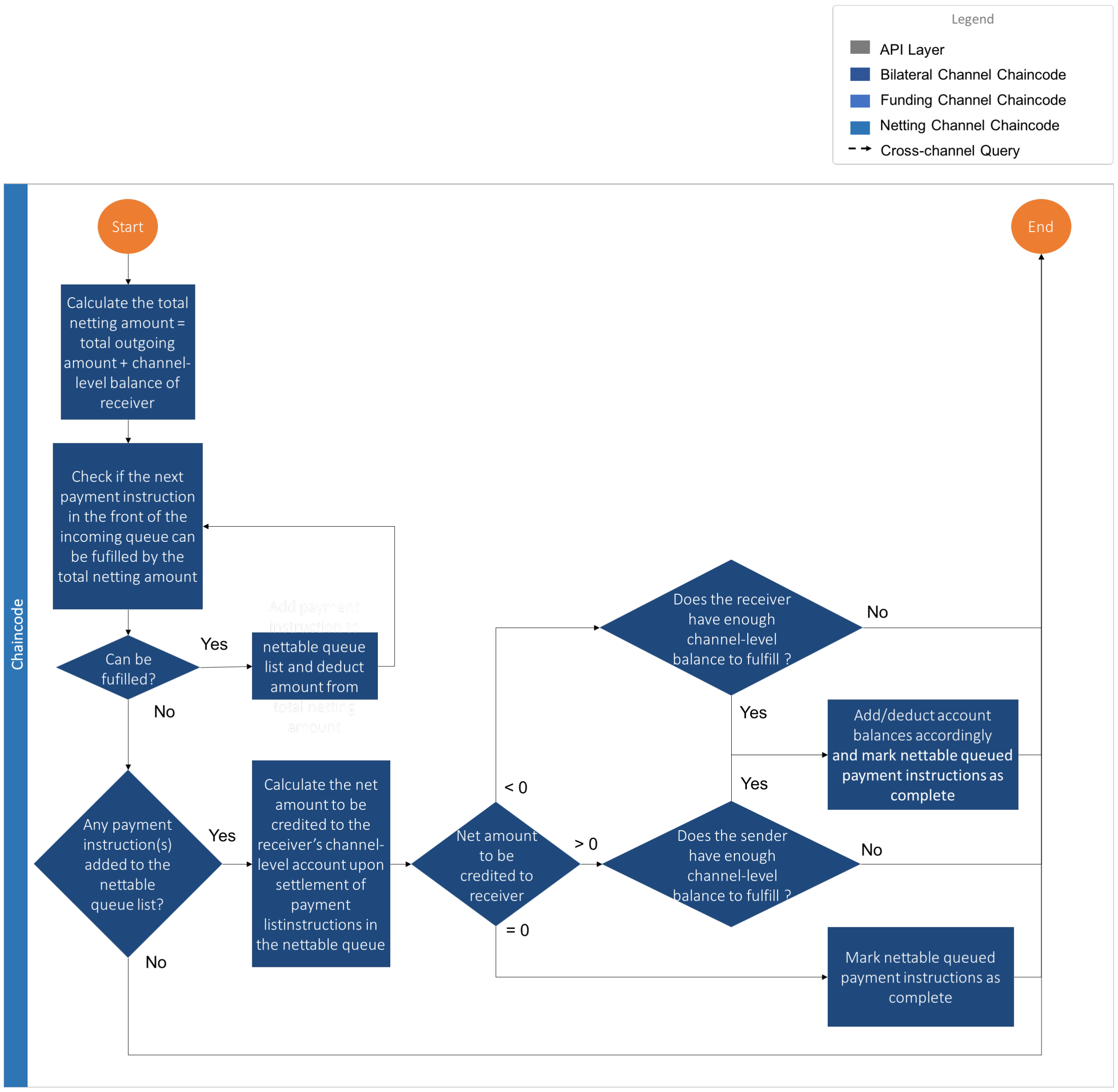
2.1.1 Bilateral Netting

Bilateral netting is attempted when a bank has insufficient liquidity to fulfil a new payment instruction, but has pending incoming payment instructions in queue. A simple example to illustrate bilateral netting is as follows:

- Bank A performs a fund transfer of $5,000 (Transaction 1) to Bank B. Given that its current channel-level balance is $1,000, it has insufficient funds to be able to fulfil the payment instruction. Assuming that in the meantime, Bank A has a pending incoming queued payment instruction of $4,000 (Transaction 2) from Bank B, Bilateral netting would be possible in this case as Transaction 1 and 2 combined would equate to a net transfer of $1,000 from Bank A to Bank B.
- Since bank A has sufficient liquidity to fulfil this, both transaction 1 and 2 will be marked as complete and $1,000 will be debited from Bank A's channel-level account and credited to Bank B's channel-level account.

For the current implementation of the Hyperledger Fabric prototype, it is possible to switch bilateral netting off by changing a constant in the bilateral chaincode (`isBilateralNetting`). The intention of having the flag as a constant in the bilateral chaincode is so that all parties in the network are required to agree to toggling bilateral netting on/off and upgrade the bilateral chaincodes accordingly before the change can take effect.

The following diagram (Figure 5) illustrates the logical flow in the bilateral chaincode during bilateral netting:



Legend
API Layer
Bilateral Channel Chaincode
Funding Channel Chaincode
Netting Channel Chaincode
Cross-channel Query
Chaincode
Start
Calculate the total netting amount = total outgoing amount + channel-level balance of receiver
Check if the next payment instruction in the front of the incoming queue can be fufilled by the total netting amount
Can be fufilled?
Yes
No
Add payment instruction to nettable queue list and deduct amount from total netting amount
Any payment instruction(s) added to the nettable queue list?
Yes
No
Calculate the net amount to be credited to the receiver's channel-level account upon settlement of payment listinstructions in the nettable queue
Net amount to be credited to receiver
< 0
> 0
= 0
Does the receiver have enough channel-level balance to fulfill ?
No
Yes
Does the sender have enough channel-level balance to fulfill ?
Yes
No
Add/deduct account balances accordingly and mark nettable queued payment instructions as complete
Mark nettable queued payment instructions as complete
End

Figure 5: Bilateral Netting Process Flow

2.1.2 Cross-Channel Fund Movement

Cross-channel fund movement is a Ubin Phase 2 functionality designed specifically for the Hyperledger Fabric workstream. This feature enables a bank to manage and move funds between its bilateral channels.

For each of its bilateral channel, a bank would have a channel-level account. An advantage of having a bilateral channel per counterparty is that a bank can set "bilateral limits" on the amount of funds that can be used to transact with a particular counterparty. It is then possible that a bank may have sufficient liquidity overall to perform a fund transfer but the funds may not be in the particular bilateral channel with the target counterparty. This payment instruction will be queued in the specific bilateral channel only. Fund transfers with other counterparties can proceed if there is sufficient liquidity in the respective bilateral channels.

With cross-channel fund movement, a bank is able to move a specified amount of funds between channels to facilitate queue processing and future fund transfers. A cross-channel fund movement transaction involves three steps:

1. In the source bilateral channel, the requesting bank's channel-level account balance is deducted and a `moveOutFund` asset is created with the corresponding amount
2. The requesting bank then creates a `transientFund` asset in the funding channel. This asset is related to the `moveOutFund` through a common identifier. The asset also includes the direction of the fund movement (source and target bilateral channels) for traceability and auditability of the asset.
3. Then, the requesting bank creates a `moveInFund` asset in the target bilateral channel and the channel-level account balance in this channel is increased

These three steps and assets in cross-channel fund movement would be orchestrated by the application layer and recorded in the respective bilateral ledgers. The assets created provide full traceability of the fund movement transaction across the three channels. The sequence diagram below (Figure 6) illustrates cross-channel fund movement:

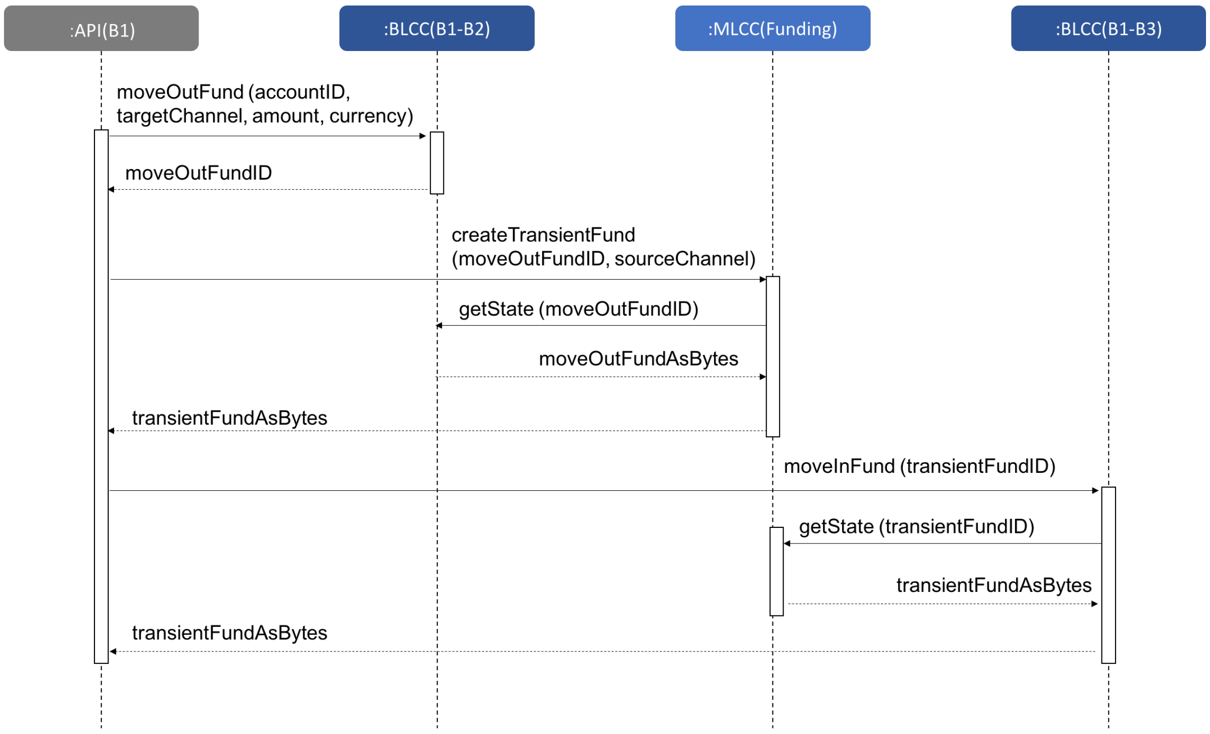


Figure 6: Cross-channel Fund Movement Sequence Diagram

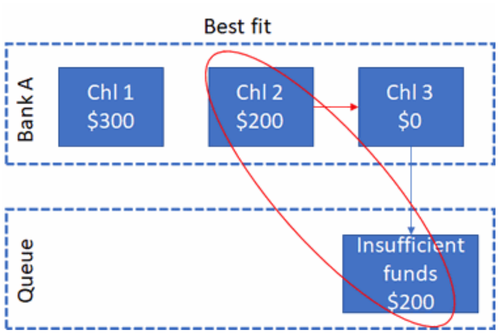
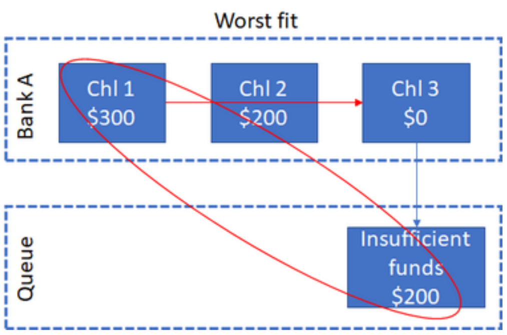
As an additional feature, there is also a functionality to suggest possible fund movement between channels by analysing the channel-level account balances and the outgoing queues per channel.

To illustrate, assume Bank A has a queued outgoing payment instruction in Channel A-B due to insufficient funds to fufill the payment instruction. Bank A has another channel A-C which has sufficient funds and no outstanding payment instruction in the queue. Bank A can execute the fund movement suggestion API to request for a possible fund movement suggestion. This function will propose a possible movement of funds from a bilateral channel with sufficient liquidity such as Channel A-C to a bilateral channel that has insufficient funds to fufill pending outgoing queued payment instructions such as Channel A-B. Once Bank A moves the funds as per suggested, Bank A will be able to settle the outgoing queue in Channel A-B.

In fund movement suggestion, Bank A will take all its balance per-channels, outgoing and incoming queues to calculate the overall position for each channel individually. Only channels with a positive position will be considered as a possible ‘source’ channel to move funds from. After obtaining the position, the source channel can be selected based on either ‘best-fit’ or ‘worst-fit’. The table below (Table 1) provides an overview of both algorithm:

2.2 Queue Mechanism

Table 1: Fund Movement Suggestion Algorithms

Best-Fit	**Worst-Fit**
The 'best-fit' will provide a fund movement suggestion based on the bilateral channel with excess liquidity of the amount closest to the amount required for queued payment instructions settlement.  Best fit Bank A Chl 1 $300 Chl 2 $200 Chl 3 $0 Queue Insufficient funds $200 To illustrate, assume that Bank A has the following channel-level accounts: • Channel 1: ○ Channel-level balance: $300 ○ No outgoing payment instructions • Channel 2: ○ Channel-level balance: $200 ○ No outgoing payment instructions • Channel 3: ○ Channel-level balance: $0 ○ 1 outgoing payment instruction of $200 Using the 'best-fit' algorithm, since there is an exact match of $200 in excess liquidity in Channel 2 to the amount required for settlement in Channel 3, a suggestion to move $200 from Channel 2 to Channel 3 will be provided.	The 'worst-fit' algorithm will provide a suggestion based on the bilateral channel with the highest amount of excess liquidity regardless of the amount required for queued payment instructions settlement.  Worst fit Bank A Chl 1 $300 Chl 2 $200 Chl 3 $0 Queue Insufficient funds $200 To illustrate, assume that Bank A has the following channel-level accounts: • Channel 1: ○ Channel-level balance: $300 ○ No outgoing payment instructions • Channel 2: ○ Channel-level balance: $200 ○ No outgoing payment instructions • Channel 3: ○ Channel-level balance: $0 ○ 1 outgoing payment instruction of $200 Using the 'worst-fit' algorithm, since Channel 3 has the highest amount of excess liquidity, a suggestion to move $200 from Channel 1 to Channel 3 will be provided.

2.2.1 Queue Settlement

Before proceeding with queue settlement on the bilateral channels, there will be a check to see if any of the channel's participants are currently taking part in a current ongoing multilateral gridlock resolution cycle (via a cross-channel query). If either party in the channel is indeed part of an ongoing multilateral gridlock resolution cycle, queue settlement will not be executed as this will potentially have an impact on the current multilateral gridlock settlement. As part of the implementation of the API layer for the Hyperledger Fabric prototype, the queue settlement function is called after functions which can potentially change the order of queues or alter channel liquidity. These functions are as follows:

- Pledge
- Move in fund
- Queue cancellation
- Toggle queue hold/resume
- Queue reprioritization

Due to this nature of when the queue settlement is triggered, if the order of queues or channel liquidity is altered, there may be new opportunities that open for bilateral netting. Therefore, as part of the flow for queue settlement, bilateral netting is also included.

The following diagram (Figure 7) illustrates the logical flow of queue settlement for the Hyperledger Fabric prototype:

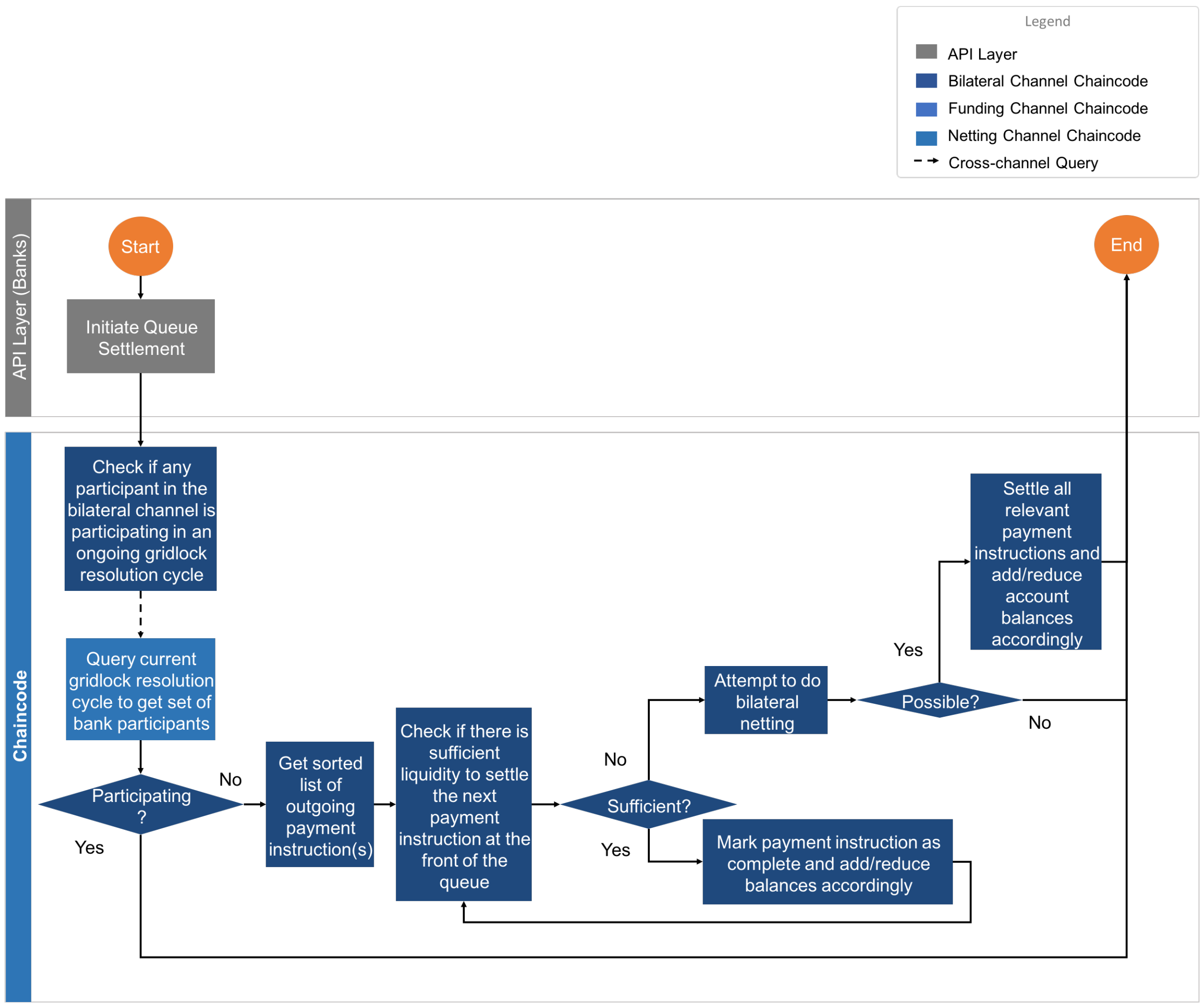

Legend
API Layer
Bilateral Channel Chaincode
Funding Channel Chaincode
Netting Channel Chaincode
Cross-channel Query
API Layer (Banks)
Chaincode
Start
Initiate Queue Settlement
Check if any participant in the bilateral channel is participating in an ongoing gridlock resolution cycle
Query current gridlock resolution cycle to get set of bank participants
Participating ?
No
Yes
Get sorted list of outgoing payment instruction(s)
Check if there is sufficient liquidity to settle the next payment instruction at the front of the queue
Sufficient?
No
Yes
Attempt to do bilateral netting
Possible?
Yes
No
Settle all relevant payment instructions and add/reduce account balances accordingly
Mark payment instruction as complete and add/reduce balances accordingly
End

Figure 7: Queue Settlement Process Flow

2.2.2 Queue Management

As part of queue management, there are several functions implemented which can be performed by the user to manipulate the bilateral channel queues. These functions include:

- Queue cancellation
- Queue reprioritisation
- Toggle queue hold/resume

2.2.2.1 Queue Cancellation

When a payment instruction is cancelled, the corresponding `QueuedTransaction` state will be converted into a `CompletedTransaction` state with status set as `CANCELLED`

2.2.2.2 Queue Reprioritisation

On the ledger, priorities are represented by integers where higher priority payment instructions are assigned a greater number in the `priority` field. Reprioritising of payment instructions would thus only require an update of the according `QueuedTransaction` state.

It is important to note that in the current Hyperledger Fabric prototype implementation queues are ordered according to priority followed by creation time. The impact of this is the order in which the queues are reprioritised would not affect the final queue order as the sorting is not done according to update time.

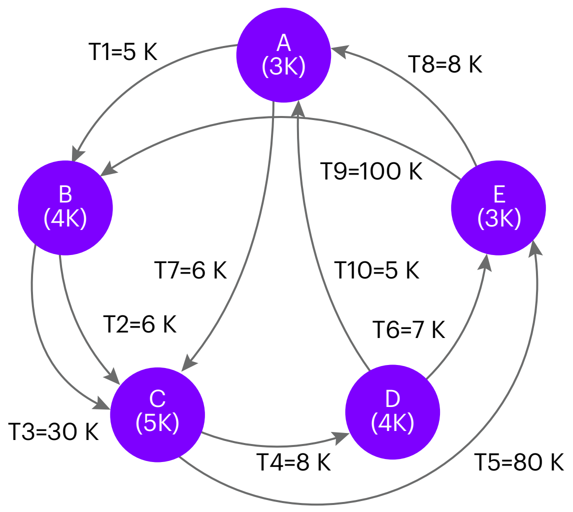
2.2.2.3 Toggle queue hold/resume

As part of the design of the Hyperledger Fabric prototype, the `Status` field in the `QueuedTransaction` asset determines whether a queued payment instruction is `ACTIVE` or on `HOLD`. Toggling between the two statuses would simply require an update of the corresponding state on the ledger. Since no other fields will be altered, the payment instruction will retain its queue position prior to being set on hold.

2.3 Gridlock Resolution

To illustrate the design for gridlock resolution across the 3 platforms, this section will refer to a common gridlock scenario as per described below (illustrated in Figure 8). There are more examples of gridlock/deadlock scenarios described in Section 7 Testing which also includes the resolution per workstream.

- There are 5 participating banks (Bank A, Bank B, Bank C, Bank D and Bank E) with starting balance of $3,000, $4,000, $5,000, $4,000 and $3,000 respectively
- There is a total of 10 payment instructions (T1, T1, T3, T4, T5, T6, T7, T8, T9 and T10), each of which have the sender and receiver detailed in the table below where a negative value indicates amount to be paid while a positive value indicates amount to be received
- All payment instructions are of `Normal` priority
- The banks have insufficient liquidity to settle the first payment instruction in their outgoing queues



5 BANKS

A	Bank A
B	Bank B
C	Bank C
D	Bank D
E	Bank E

10 TRANSACTIONS

	A (K)	B (K)	C (K)	D (K)	E (K)
Starting balance	3	4	5	4	3
T1	-5	+5			
T2		-6	+6		
T3		-30	+30		
T4			-8	+8	
T5			-80		+80
T6				-7	+7
T7	-6		+6		
T8	+8				-8
T9		+100			-100
T10	+5			-5	

Figure 8: Gridlock Scenario

2.3.1 Algorithm Description

For the Hyperledger Fabric prototype developed, EAF2 was used as the algorithm for illustrating the concept of gridlock resolution however, the pluggable nature of Hyperledger Fabric allows the implementation of other algorithms which may have specific optimisation. In this section, we describe the use of the EAF2 algorithm with Fabric for gridlock resolution.

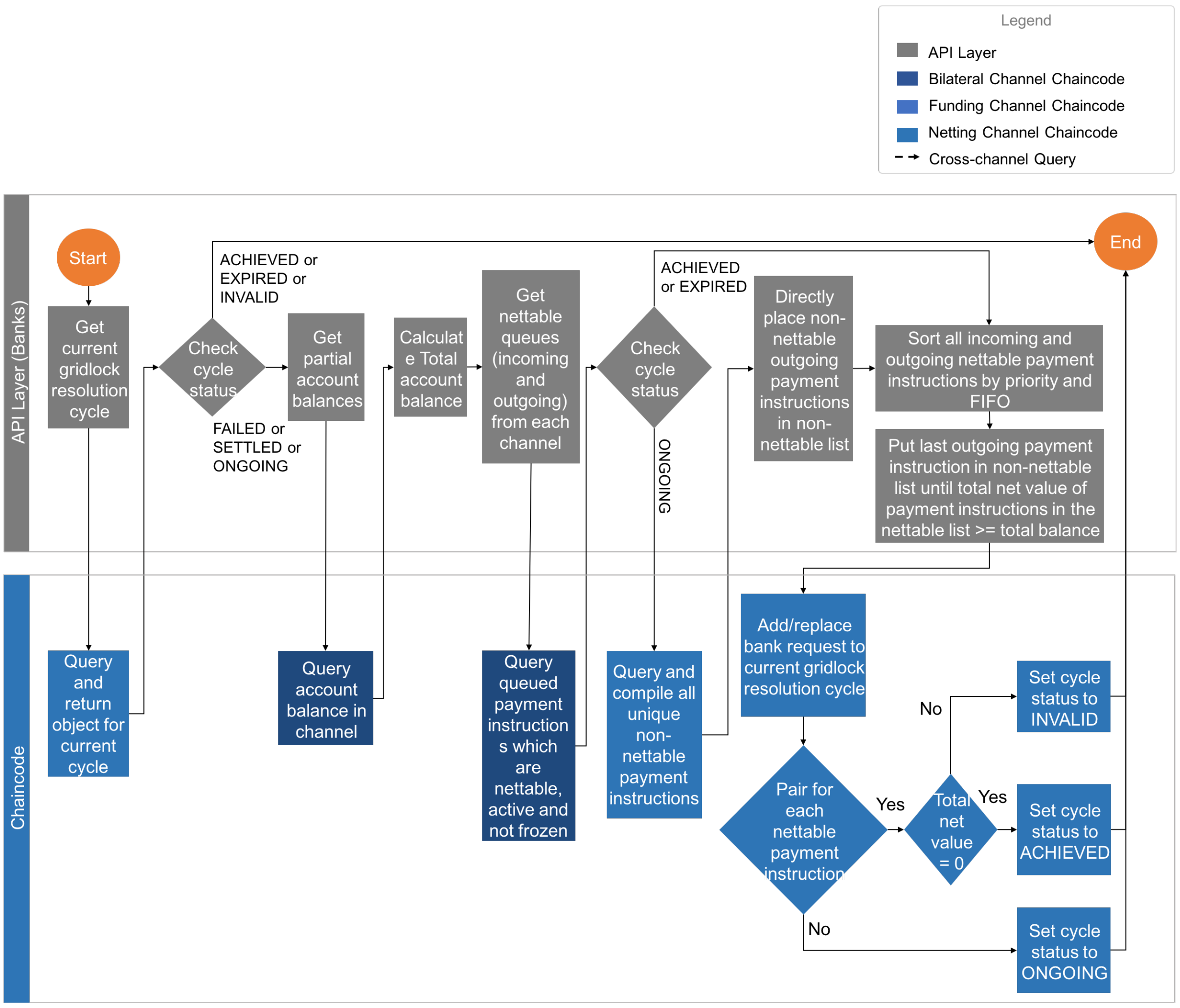
Gridlock resolution for the Hyperledger Fabric prototype is split into 2 main stages: Initiation/Participation and Settlement.

As part of the Hyperledger Fabric prototype implementation, the triggering of participation and settlement will be partially automated on the API layer:

- Commercial Bank Peers: Initiation and the initial participation of the current gridlock cycle will be done manually. Subsequent participation in the same cycle will be triggered on a fixed interval by the API layer.

- MAS peer: Repeatedly query and check the status of the current gridlock cycle to determine if settlement is required

2.3.1.1 Initiation/Participation



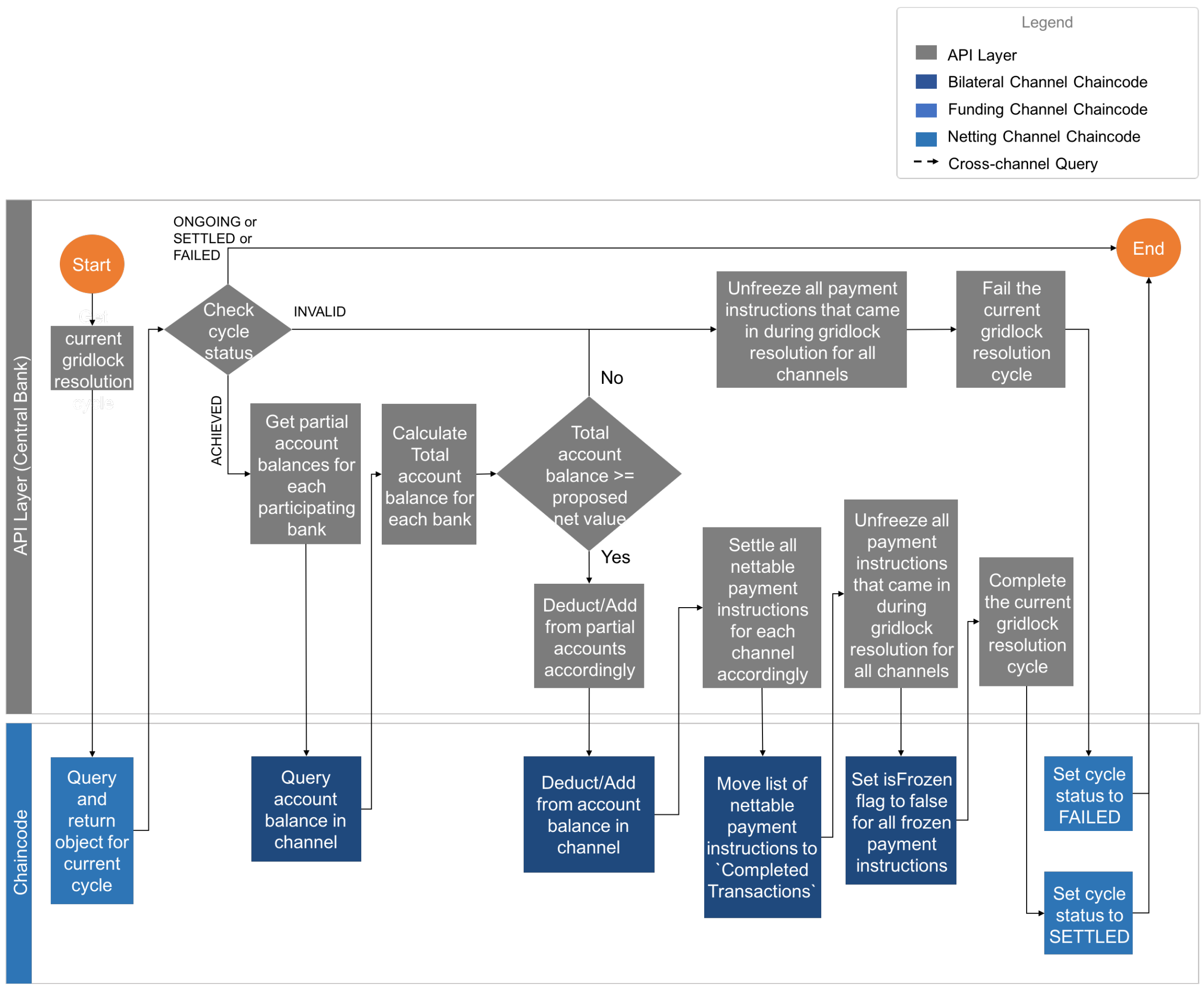
Legend
API Layer
Bilateral Channel Chaincode
Funding Channel Chaincode
Netting Channel Chaincode
Cross-channel Query
API Layer (Banks)
Chaincode
Start
Get current gridlock resolution cycle
Check cycle status
ACHIEVED or EXPIRED or INVALID
FAILED or SETTLED or ONGOING
Get partial account balances
Calculate Total account balance
Get nettable queues (incoming and outgoing) from each channel
Check cycle status
ACHIEVED or EXPIRED
ONGOING
Directly place non-nettable outgoing payment instructions in non-nettable list
Sort all incoming and outgoing nettable payment instructions by priority and FIFO
Put last outgoing payment instruction in non-nettable list until total net value of payment instructions in the nettable list >= total balance
End
Query and return object for current cycle
Query account balance in channel
Query queued payment instructions which are nettable, active and not frozen
Query and compile all unique non-nettable payment instructions
Add/replace bank request to current gridlock resolution cycle
Pair for each nettable payment instruction
Yes
No
Total net value = 0
Yes
No
Set cycle status to INVALID
Set cycle status to ACHIEVED
Set cycle status to ONGOING

Figure 9: Initiation/Participation Process Flow for Multilateral Gridlock Resolution

1. If there is a need for gridlock resolution, the bank will first check if there is any ongoing gridlock resolution cycle. If there is no ongoing cycle, the bank will proceed to initiate the a new "netting cycle". Otherwise, it will participate in the existing cycle.
2. The bank's API layer will retrieve all channel-level balances and active payment instructions (both incoming and outgoing) across all its bilateral channels.
3. The information will be consolidated on the API layer. Channel-level balances will be summed to get the full balance of the bank while queued payment instructions across all bilateral channels will be merged and sorted according to priority and time of creation.
4. The full list of payment instructions will be separated into two buckets, nettable and non-nettable payment instructions. The separation is done based on the criteria that the final overall balance of the bank should not be in deficit after all nettable payment instructions are accounted for.
   a. If the bank is participating in an ongoing gridlock resolution cycle, the current non-nettable list in the cycle should also be taken into consideration (i.e. non-nettable payment instructions in the current gridlock resolution cycle should also be in the proposed non-nettable list)
5. The proposed list of payment instruction reference IDs of all nettable and non-nettable queued payment instructions will then be submitted into the netting channel along with the net value of the nettable payment instruction list

6. The gridlock resolution cycle will be marked as `ACHIEVED` if the new proposal by the bank results in the following:
    a. There is a matching pair of reference IDs for all nettable payment instructions proposed for the cycle
    b. The total net value of all payment instructions equals to 0
7. Banks will continually participate in the cycle until a resolution is found. If there is no solution found within the stipulated time (set to 5 minutes as a default), the gridlock resolution cycle will be marked as `EXPIRED`.

2.3.1.2 Settlement



Legend
API Layer
Bilateral Channel Chaincode
Funding Channel Chaincode
Netting Channel Chaincode
Cross-channel Query
API Layer (Central Bank)
Chaincode
Start
current gridlock resolution
ONGOING or SETTLED or FAILED
Check cycle status
INVALID
ACHIEVED
Get partial account balances for each participating bank
Calculate Total account balance for each bank
Total account balance >= proposed net value
No
Yes
Unfreeze all payment instructions that came in during gridlock resolution for all channels
Fail the current gridlock resolution cycle
End
Deduct/Add from partial accounts accordingly
Settle all nettable payment instructions for each channel accordingly
Unfreeze all payment instructions that came in during gridlock resolution for all channels
Complete the current gridlock resolution cycle
Query and return object for current cycle
Query account balance in channel
Deduct/Add from account balance in channel
Move list of nettable payment instructions to `Completed Transactions`
Set isFrozen flag to false for all frozen payment instructions
Set cycle status to FAILED
Set cycle status to SETTLED

Figure 10: Settlement Process Flow for Multilateral Gridlock Resolution

Being the only participant in the network with a complete view of all channels, MAS will be performing settlement for the gridlock resolution cycle. While gridlock resolution is ongoing, MAS will periodically check the status of the gridlock resolution cycle. Once the status of the current gridlock resolution cycle is `ACHIEVED`, the settlement process will begin.

For each participating bank in the gridlock resolution cycle, MAS does a validation check on whether it has sufficient liquidity to fulfil its proposal. If any bank does not have enough liquidity, the cycle will be marked as `FAILED` directly.

After verification of all bank proposals, funds will be added and deducted accordingly from the respective bank's balances. Since the actual balance of each bank is a sum of all bilateral channel-level account balances, there may be a possibility of debiting from multiple channel-level accounts if there is insufficient liquidity to fulfil the bank's gridlock resolution proposal. Note that for the implementation of this prototype, the order of channel-level account balances to deduct from is fixed based on the sequence defined in the API configuration files.

In the next step in the settlement process, MAS will submit the transaction reference IDs of all nettable payment instructions proposed in the gridlock resolution cycle to the respective bilateral channels to be updated as complete and `SETTLED`. During this step, all payment instructions which came in during the gridlock resolution cycle will also be unfrozen.

Once everything is completed, MAS will then proceed to mark the gridlock resolution cycle as `SETTLED`, ending the settlement process.

2.3.2 Illustration Using a Gridlock Scenario

2.3.2.1 Stage 1: Initiation/Participation

The following example illustrates what goes on in a typical gridlock resolution cycle:

	A (K)	**B (K)**	**C (K)**	**D (K)**	**E (K)**	
Starting balance	3	4	5	4	3	
T1	-5	+5				**Matched**
T2		-6	+6			**Matched**
T3		~~-30~~	~~+30~~			
T4			-8	+8		**Matched**
T5			~~-80~~		~~+80~~	
T6				-7	+7	**Matched**
T7	-6		+6			**Matched**
T8	+8				-8	**Matched**
T9		~~+100~~			~~-100~~	
T10	+5			-5		**Matched**
Netted balance	5	3	9	0	2	

Figure 11: Multilateral Gridlock Resolution Scenario Outcome

1. Assuming Bank A initiates the gridlock resolution cycle, Bank A will be proposing all its payment instructions T1, T7, T8 and T10 as nettable as these payment instructions have a net value of +2 while Bank A has a current balance of 3, resulting in a positive netted balance of 5.
2. Similarly, Bank B will also propose T1, T2, T3 and T9 as nettable as they add up to a net value of +69, resulting in a netted balance of 73.
3. Bank C however, will only propose T2, T3, T4 and T7 as nettable and T5 as non-nettable as inclusion of T5 will cause Bank C to have a netted balance of -41 (deficit).
4. In the same vein, Bank D will propose T4, T6 and T10 as nettable.
5. When it comes to Bank E, only T6 and T8 will be proposed as nettable while T5 and T9 will be proposed as non-nettable since T5 is already marked as non-nettable by Bank C and the inclusion of T9 will cause a deficit.
6. During the second round of participation and proposal by Bank B, only T1 and T2 will be proposed as nettable while T3 and T9 are marked as non-nettable as T9 is already marked by Bank E as non-nettable and with that as a consideration, T3 will now cause the netted balance for Bank B to be in deficit.

7. Similarly, when Bank C re-proposes, since T3 is already marked as non-nettable by Bank B, T3 will be removed from the nettable list, leaving only T2, T4 and T7 in the nettable list.
8. With the second proposal by Bank C, there is now a matching pair of for every proposed nettable payment instruction in the netting channel. The netting channel chaincode will now mark the cycle as 'Achieved' and is now ready for settlement

2.3.2.2 *Stage 2: Settlement*

Table 2: Remaining gridlock transactions after gridlock resolution

	A (K)	B (K)	C (K)	D (K)	E (K)
Closing balance	5	3	9	0	2
T3		-30	+30		
T5			-80		+80
T9		+100			-100

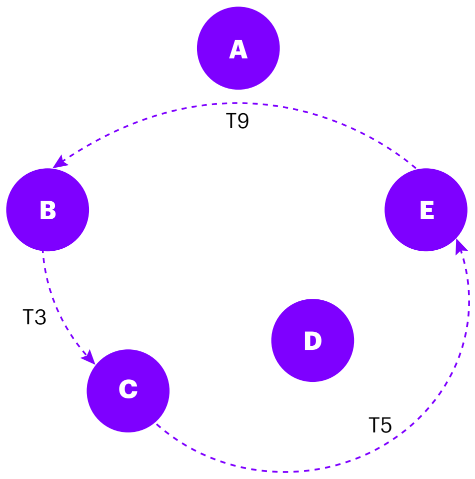


Figure 12: Remaining Gridlock Transactions

The closing balance of each bank is $5000, $3000, $9000, $0 and $2000 respectively after gridlock resolution. T3, T5 and T9 remain unsettled/remain in the queue.

To simplify gridlock resolution settlement, an approach is to have MAS play the role of a facilitator. As a facilitator of gridlock resolution settlement, once it is detected that a gridlock resolution cycle is achieved, MAS will calculate the resulting net balance of each participating bank to ensure that no bank ends up in a deficit at the end of settlement. After the check, MAS will then deduct and add funds appropriately to each bank's balances. Thereafter, MAS will mark all the appropriate payment instructions in their respective bilateral channels as settled.

An alternative approach which involve the decentralized settlement of gridlock resolution is also feasible in jurisdictions where regulatory policies and business controls deem this to be permissible. This alternative doesn't require a single facilitator like the MAS node. Instead, participating banks will submit their individual status to the netting channel. The banks will then determine the netting cycle to be complete once all updates have been received by all parties.

2.4 Pledge and Redeem

2.4.1 Pledge

A pledge is requested by a bank through MEPS+. The bank has to specify the pledge amount as well as the channel to pledge to. In MEPS+, MAS performs the necessary validation before approving the pledge request and debiting the bank's RTGS account. These are assumed to be pass-through steps in this prototype.

MEPS+ will then call the MAS DLT Pledge API to increase the channel-level account balance of the bank in the specified channel. This pledge function can only be executed by MAS.

Given that funds are only injected into the DLT upon debiting of the bank's RTGS account, there is no net increase of overall money supply at any one time.

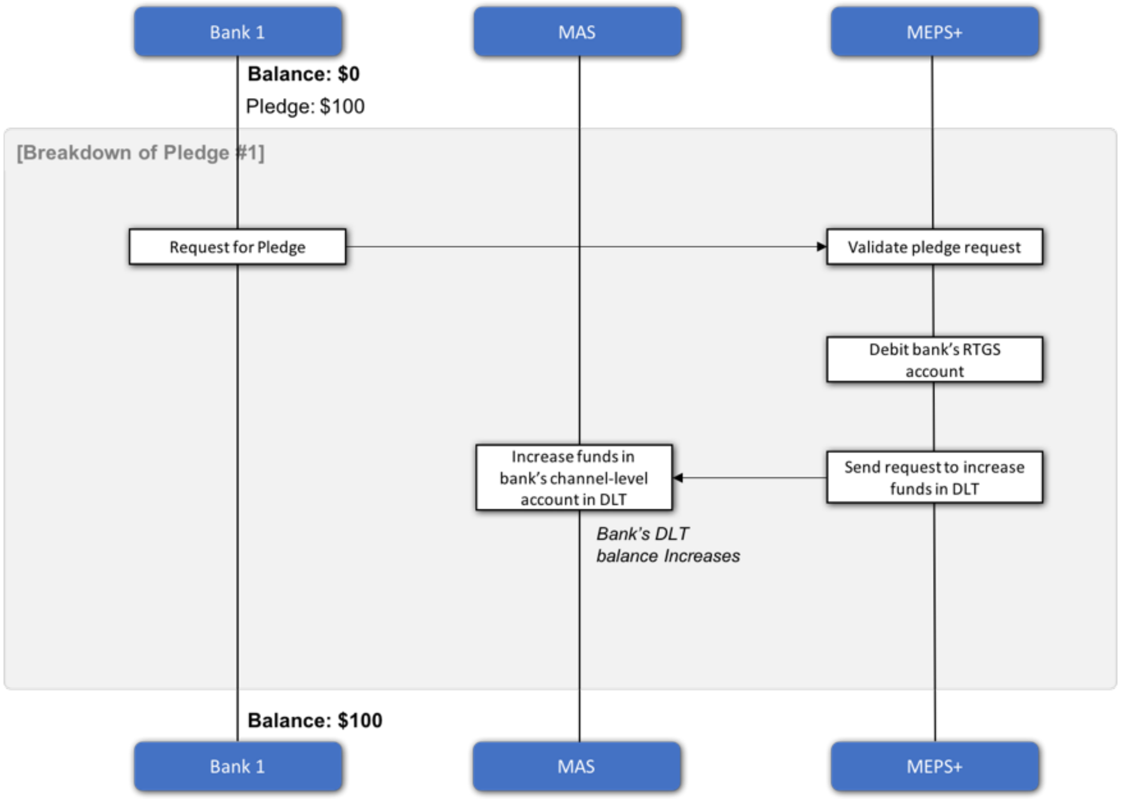

Bank 1
MAS
MEPS+
Balance: $0
Pledge: $100
[Breakdown of Pledge #1]
Request for Pledge
Validate pledge request
Debit bank's RTGS account
Send request to increase funds in DLT
Increase funds in bank's channel-level account in DLT
Bank's DLT balance Increases
Balance: $100
Bank 1
MAS
MEPS+

Figure 13: Pledge Sequence Diagram

2.4.2 Redeem

A bank sends a request to MEPS+ to redeem funds from its DLT account. As part of the request, the bank specifies the redeem amount as well as the channel to redeem from. MEPS+ will then call the MAS DLT Redeem API to attempt to reduce the bank's balance in the specified channel. This redeem function can only be executed by MAS.

If there are sufficient funds in the channel to redeem and the transaction is successful, MEPS+ will then credit the RTGS account in MEPS+. If the redeem fails, no change is made to the bank's balance and no funds are credited into the RTGS account.

Given that the bank's RTGS account is only credited upon successful withdrawal of funds in the DLT, there is no net increase of overall money supply at any one time.

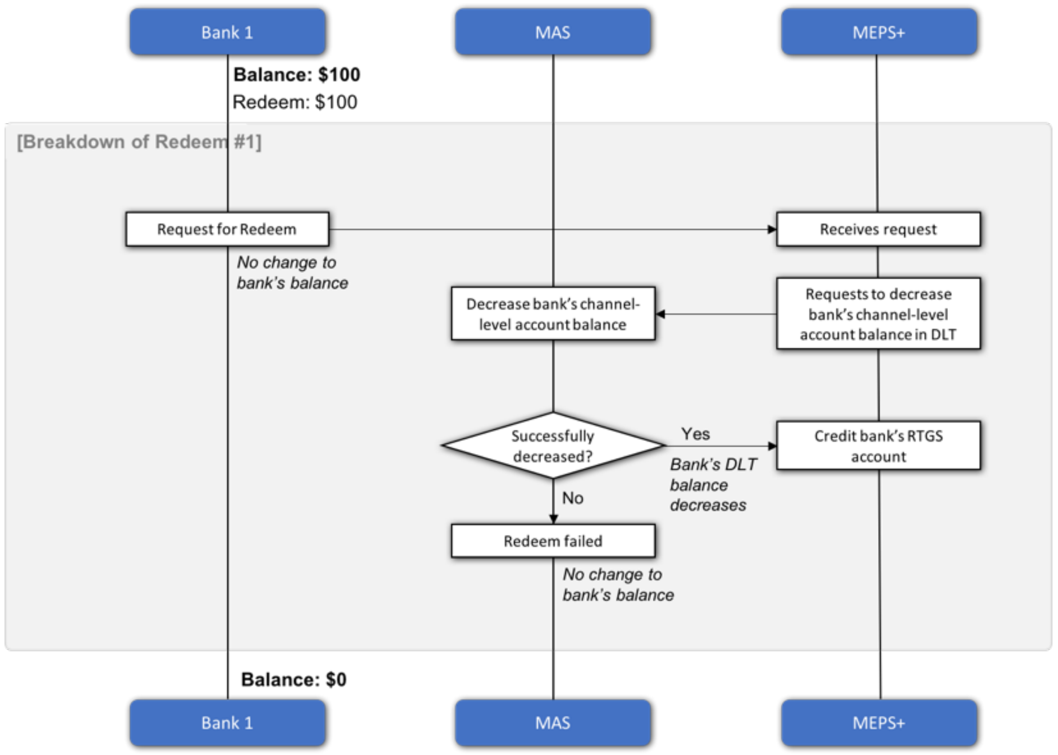

Bank 1
MAS
MEPS+
Balance: $100
Redeem: $100
[Breakdown of Redeem #1]
Request for Redeem
Receives request
No change to bank's balance
Decrease bank's channel-level account balance
Requests to decrease bank's channel-level account balance in DLT
Successfully decreased?
Yes
Credit bank's RTGS account
Bank's DLT balance decreases
No
Redeem failed
No change to bank's balance
Balance: $0
Bank 1
MAS
MEPS+

Figure 14: Redeem Sequence Diagram

2.5 Balance Enquiry

A specific bank can view all outgoing and incoming payment instructions that it is involved in as well as its current balance. No other banks should be able to view its total balance at any time.

The central bank or regulator should have the visibility of the balances of all banks in the network at any one point of time. This allows the central bank or regulator to monitor the overall liquidity in the network and allows it to detect if there are any fraudulent activities e.g. sudden reduction in balance when there is no redeem request. Given the central bank is responsible for pledge and redeem requests, it is also critical that it can monitor both the DLT accounts as well as the MEPS+ accounts to ensure the accounts can reconcile.

With the design of bilateral channels in the Hyperledger Fabric prototype, a bank will have one bilateral channel for every counterparty bank. A bank will then have one channel-level account per bilateral channel. To obtain its total balance, the bank can query its channel-level account across all its bilateral channels and obtain a sum of this.

It is possible for a bank to view the channel-level account of its counterparty in their shared bilateral channel. However, given the counterparty has other channels with other banks in the network, it is not possible for a bank to identify the total balance of its counterparty at any one time.

Aside from the bank pair, MAS is also a participant of all bilateral channels. Given MAS is a participant in all bilateral channels, the regulator has visibility to all channel-level accounts of all banks. It is possible for the regulator to query the balances of all channel-level accounts of a bank and performing a summation to get the total balance of one bank.

2.6 Manage Accounts

2.6.1 Onboarding a New Bank/Organisation

Adding a new bank or organisation to Hyperledger Fabric network would require the genesis block, otherwise known as the configuration block, to be updated with the required information of the new organisation.

For Ubin's design, there are several configuration blocks to be updated. This includes the configuration block for the ordering system and the configuration blocks for the two multilateral channels (netting and funding). It is important to note that although a configuration block update would not require any downtime of the Fabric network, any transactions done on the affected channels during the update may be affected and/or invalidated.

On top updating the configuration blocks in the respective channels and orderer, adding a new organisation to the Fabric network would also involve the creation of new bilateral channels with existing participating organisations in the network. This can be done once the configuration block for the ordering system has been updated with the new organisation.

2.6.2 Suspending a Bank/Organisation

The approach to suspend a bank or an organisation in Hyperledger Fabric is to update each channel configuration which the bank is part of, removing the according organisation group and also to update the ordering system channel configuration block to remove the organisation from the consortium list.

It is however important to note that for the version of Hyperledger Fabric used in this prototype (v1.0.1), if an org defined as part of the initial instantiation endorsement policy is suspended, the chaincode will not be able to be invoked by anyone and the chaincode cannot be upgraded with a new endorsement policy either as any upgrades depend on the endorsement policy defined **during the first instantiation**. The impact of this will be the most evident on the netting channel, whereby the endorsers are set to be all participants in the channel. Therefore, suspending a bank part of the initial consortium will cause the entire netting channel to stop functioning (i.e. the world state on the netting cannot be modified).

2.6.3 Update Bank Name

Bank names are not currently stored in the ledger as per current design. Only BIC (Business Identifier Codes) are used. If required, bank names can be stored as a separate field in the `Account` asset in each bilateral channel or on the API layer. Permissions can be set to only allow the regulator or the bank itself to update the field if stored as a separate field in the `Account` asset.

2.6.4 Key Pair Update

The procedures for updating a key pair on Hyperledger Fabric differs depending on the key to be updated. If the key pair to be updated is a parameter of the MSP (i.e. root/intermediate CA certificates or admin certificates), a change of configuration for the ordering system as well as all existing channels which the organization is currently part of is required.

If the key pair to be updated is not a parameter of the MSP (e.g. peers' or users' key pair) then there is no requirement to update the configuration blocks. The new key pair will have to be added/enrolled via the CA while the old certificates will be added to the Certificate Revocation List (CRL), maintained by the CA.

There will be no impact on the transaction history or the world state for the update of both type of key pairs. However, any transactions done at during configuration updates may be invalidated.

2.6.5 Secure Key and Certificate Storage

In Hyperledger Fabric, there is no requirement to store other participants' public keys or certs as this is all communicated via signatures attached during transactions. Participants' admin certificates and root/intermediate CA certificates will be stored in the channel configuration.

By default, the Fabric CA server and client store private keys in a PEM-encoded file, but they can also be configured to store private keys in an HSM (Hardware Security Module) via PKCS11 APIs. This behavior is configured in the BCCSP (BlockChain Crypto Service Provider) section of the server's or client's configuration file.

2.6.6 Identify Participants

The current version of the Fabric core has no specific functionality built in to obtain all participating organisations in the network. However, as part of initial network setup, the members of the consortium would have already decided and predetermined the participants and channels.

If required, a participating organization can also inspect the configuration/genesis block of a multilateral channel (i.e. funding or netting channel) to identify who are the participants in the network. This is possible as the current design requires all participants in the network to be part of the multilateral channels.

2.7 Versioning

2.7.1 Chaincode Version Check

Currently there is no built-in functionality on the Hyperledger Fabric core to check the current code version of each chaincode. However, since each version of chaincode spawns a new docker container, as long as these containers are managed properly, the current chaincode versions can be implied from the chaincode container names.

2.7.2 Chaincode Version Upgrade

A chaincode may be upgraded any time by changing its version, which is part of the SignedCDS. Other parts, such as owners and instantiation policy are optional. However, the chaincode name must be the same; otherwise it would be considered as a totally different chaincode.

Prior to upgrade, the new version of the chaincode must be installed on the required endorsers. Upgrade is a transaction similar to the instantiate transaction, which binds the new version of the chaincode to the channel. Other channels bound to the old version of the chaincode still run with the old version. In other words, the `upgrade` transaction only affects one channel at a time, the channel to which the transaction is submitted.

For simplicity, MAS can be the one triggering the `upgrade` command across all channels as it is part of all bilateral and multilateral channels in the network. The `upgrade` command can be done via a CLI container or via the Node SDK.

It is important to note that in the current version of Hyperledger Fabric, old chaincode containers will not be removed or killed automatically after a chaincode upgrade. Killing or removing of old chain containers is currently a manual process to be done by the administrator of the peer node.

2.7.3 Fabric Core Upgrade

As the current Hyperledger prototype for Ubin Phase 2 is implemented with docker containers in swarm mode, upgrading the Hyperledger Fabric core version will require an upgrade of the docker container image. Because of this, all docker containers will need to be brought down for and redeployed with the new image. This would however mean that all existing world state and transactional history will be wiped out.

3 Technical Specification

3.1 Transaction Validity

Hyperledger Fabric has a unique approach of ensuring the validity of every transaction and it boils down to the characteristics of the bilateral channel formed between two banks.

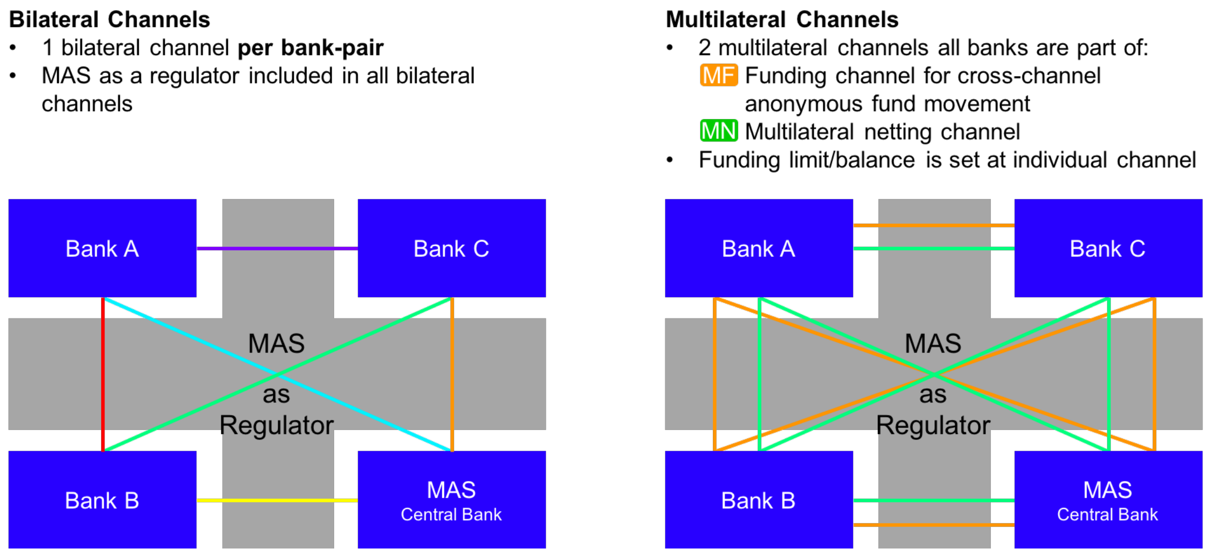
When there is a transaction between two banks that are engaged in a bilateral channel, the sender and receiver of the transaction would endorse the transaction. in the process of endorsing the transaction, the banks within the bilateral channel would ensure that the transaction proposal is well-formed. The banks would also search their transaction history to make certain that the transaction is new and has not occurred in the past, thus preventing a transaction from transacting twice.

The bank that is receiving the payment will also check the validity of the sender's signature. The sender bank is only authorised to perform the transaction operation on this particular bilateral channel.

The funding channel is used for banks to perform fund movements across the channel-level accounts in their bilateral channels. This multilateral channel will allow for auditability and traceability of such transactions. Both participants from the source channel are required to provide their endorsement for cross-channel fund movement to ensure that the fund being moved out is legitimate.

Hyperledger Fabric prevents double spending by having peers (i.e. participation nodes) validate the transactions against the endorsement policy to ensure correct allotment and authentication of the signatures.

3.2 Privacy


Bilateral Channels
- 1 bilateral channel **per bank-pair**
- MAS as a regulator included in all bilateral channels

Multilateral Channels
- 2 multilateral channels all banks are part of:
  MF Funding channel for cross-channel anonymous fund movement
  MN Multilateral netting channel
- Funding limit/balance is set at individual channel

Figure 15: Bilateral and Multilateral Channels Setup

The design of bilateral channels ensures that only the two banks involved and MAS have access and visibility to transactions within the channel. As such, Bank A can view transactions of Bank B that involve Bank A and Channel A-B, but is not able to view transactions of Bank B. As such, only the sender, receiver and MAS can see details of any fund transfer or queue mechanism transactions.

Banks can view the channel-level account of its counterparty in each bilateral channel. Pledge and redeem requests are visible to both banks in a bilateral channel. However, since a bank can only see the pledge and redeem requests for 1 channel of a counterparty and the particular channel-level account balance, it is not possible for a bank to deduce its counterparty's total balance, pledge or redeem requests. Hence, the total balance is only visible to the bank itself and MAS.

As part of cross-channel fund movement, the amount of funds being moved from one channel to another is visible to all participants in the network as a `transientFund` asset is created in the funding channel. This asset ensures traceability of the fund movement as it moves from a source bilateral channel to the funding channel, and finally to the target bilateral channel. As a result, all participants are able to identify the source and target channels involved and the amount being moved. However, it is not possible to determine the parties' overall account balance.

During gridlock resolution, banks participate in a cycle by providing the list of payment instructions that are nettable and non-nettable as well as the netted value as a result of their proposed nettable payment instructions. This information is shared in the netting channel and the only information shared for each payment instruction is the transaction ID. Given that a pair of transactions will appear in a gridlock cycle, it is possible for a participant to deduce the counterparties involved for particular transaction ID but it will not be able to decipher who is the sender or receiver, nor can it determine the amount of the transaction.

3.3 Technical Matrix

Application	**Version**
Hyperledger Fabric Core	V1.0.1
Docker	17.09.0-ce
Go	1.7.6
Node JS	6.9.5
NPM	3.10.10

4 Interface Specifications

Refer to:

https://github.com/project-ubin/ubin-docs/blob/master/api/UbinPhase2-FabricAPI.pdf

5 Key Observations and Findings

5.1 Privacy

Hyperledger Fabric is a permissioned network with the ability to set up private channels between participants where each channel maintains an independent ledger. Channel enables information to be shared between parties on a need-to-know basis. A channel is a data partitioning mechanism to limit transaction visibility only to stakeholders. Other members on the network are not allowed to access the channel and will not see transactions on the channel. Ledgers exist in the scope of a channel. This enables the setup of ledgers which can be shared across an entire network of peers (e.g. netting channel) and ledgers which include only a specific set of participants (e.g. bilateral channels).

By design and to emphasise privacy, given that fund transfer and queuing mechanisms occur within bilateral channels, the transaction details are only visible to the pair of banks in the bilateral channel and MAS (as a regulator). Both banks in this design can view the balances of both party's channel-level accounts. However, given that a bank can only view the balance of one channel-level account per counterparty, it is not possible for any bank to deduce the total balance or liquidity of a counterparty within the DLT.

The funding channel is intended to facilitate legitimate movement of funds across channels. In the current prototype, this channel is only used to ensure that funds being moved are tracked in the funding channel to allow for traceability. It is possible for network participants to identify the channels involved in each fund movement transaction. For future considerations, it is advised that the data is secured at the infrastructure level, such as deploying additional cryptographic functions to ensure confidentiality. With this, it is also possible to remove the funding channel altogether to allow higher efficiency.

In the netting channel, the identities of the banks participating in a gridlock resolution cycle, payment instruction IDs and net value of nettable payment instructions are included per bank. There is no individual transaction amount exposed. In a gridlock resolution cycle with few payment instructions, it is possible to deduce the amount for the payment instructions. However, given that netting is likely to involve multiple payment instructions per cycle, the amount for each particular payment instruction cannot be deduced.

5.2 Scalability and performance

In this design, it is required to set up [N x (N-1) / 2] + M channels to achieve the intended objectives, where:

N = number of participating nodes

M = number of multilateral channels, i.e. 2 for Ubin Phase 2 with a funding channel and a netting channel (e.g. for 10 banks, the design requires 47 channels i.e. [(10 x (10-1) /2] + 2)

The number of channels will increase with every new participant, which in turn increases the complexity in terms of network and channel management.

Bilateral channels allow bilateral netting to happen within a channel, without involvement of the rest of the network participants. This introduces the possibility that queued payment instructions in a bilateral gridlock can be resolved within the bilateral channel without having to depend fully on gridlock resolution with multiple parties in the network.

The design of an individual bilateral channel between pairs of transacting banks also introduces the need for bank operators to maintain the funds in each channel. This means that in addition to the total pledged fund from MAS to the participating bank in the DLT, these banks need to define and move specific amounts of funds between its bilateral channels where needed. The movement of funds across bilateral channels is driven by the bank users based on business operations demands. There is a possibility to automate fund movement to conform to the business operation rules. A possible solution for this is to design a fund movement algorithm that can be automated in the system.

An orderer broadcasts the transactions to all peers in a channel for validation before committing the transaction. For the purpose of prototyping, Ubin Phase 2's Hyperledger Fabric setup consists of one orderer which sends transactions to all peers in a channel for validation. A multiple node ordering service (e.g. Kafka) can be implemented for high availability of the ordering service to ensure it does not become a single point of failure.

The overhead caused by the setup of multiple channels in this design can be reduced by means of sharing hashed data among all participants while keeping private data with a limited set. This new component is underway and expected in future releases of Hyperledger Fabric.

Note: Kafka is an open-source stream processing platform that allows high throughput and low-latency processing for real-time data feeds.

Additional exceptional scenarios were considered during the project:

Exceptional Scenario	**Observations**
Impact of injecting transaction(s) to the network during gridlock resolution	Any functionality which reduces the liquidity or alters the queue positions of a gridlock resolution participant will be unavailable during multilateral gridlock resolution. New payment instructions will be queued until the current gridlock cycle has completed or timed out.

5.3 Resiliency

The design of Ubin Phase 2's Hyperledger Fabric prototype consists of one orderer to send transactions to all peers in a channel for validation. This presents a single point of failure to the network; when the orderer fails, no transaction can be ordered into blocks and committed to the chain. Fortunately, this issue can be resolved with the implementation of multiple node ordering service (e.g. Kafka) for high availability.

One key trade-off in the Hyperledger Fabric workstream design is the number of channels required to guarantee high levels of privacy, while achieving gridlock resolution. Hence cross-channel communication (e.g. movement of funds from one bilateral channel to another) becomes a vital part of the design. In the developed prototype, the orchestration logic to manage communication between two channels are programmed in a custom application using Node.js. At this point, cross-chain interaction is still not supported and is being planned for future platform release. Another alternative is to ensure there is a rollback mechanism for an application-orchestrated function that involves multiple chaincode executions.

In terms of node setup, higher resiliency be achieved by setting up reasonable and cost-effective redundant nodes in the system. This applies for bank nodes and MAS node.

Exceptional Scenario	**Observations**
Impact of removing 1 participating bank node during gridlock resolution	Only gridlock resolution will be affected as it requires all nodes to endorse any new netting proposal. If a node fails mid-cycle, no new proposals can be added until the failed node recovers or the cycle times out. In order to address such resiliency risks, the endorsement policy could be revised to require a set minimum number of participating nodes.

Impact of removing MAS during gridlock resolution	In this design, settlement for gridlock resolution is dependent on MAS to update balances and settle queues across all bilateral channels. An alternative decentralised settlement approach could be designed which will involve additional orchestration between channels.

5.4 Finality

For each transaction, the peers defined in the endorsement policy of the chaincode will endorse the transaction. The endorsed transaction is then sent to the orderer for it to order it into a block and broadcast it to the channel participants to validate and commit the block to their ledgers. This mechanism assures the finality of transactions within a channel.

However, transactions involving communication between channels remain subjected to orchestration by conventional technologies, as such orchestration logic is developed using a custom application on Node.js instead of within the chaincode. Another key observation is that endorsement for cross-channel fund movement only involves the sending and receiving banks who can collude to create more funds in the DLT. It is possible for MAS to trace such an occurrence but this would increase reliance on MAS as a governing entity.