Proving Ethereum for the Clearing Use Case

Emerald Performance Testing Technical Paper

David Creer, Richard Crook, Mark Hornsby, Nicolás González Avalis, Mark Simpson,
Nick Weisfeld, Ben Wyeth, Ivo Zieliński

September, 2016

Executive Summary

The current cross-border clearing and settlement value chain is costly, inefficient, and not transparent. A distributed ledger solution ('blockchain') applied to the cross-border clearing and settlement process is expected to enhance visibility, minimise settlement risk, lower costs, and increase transaction speed. Some estimates place the benefit of distributed ledger solutions in cross-border clearing and settlement at $50-60 billion on a cost base of $150 billion[1].

The Innovation Engineering team at Royal Bank of Scotland[2] (RBS) has built a Clearing and Settlement Mechanism (CSM) based on the Ethereum[3] distributed ledger and smart contract platform. The aims of the initial performance testing were to evaluate the technology and understand any current limitations.

GFT Technologies SE[4] (GFT) helped RBS to performance test the application on Google's Cloud Platform[5].

Ethereum required some modifications to tune its focus from a hugely distributed, public system to a faster moving, private ledger aimed at speed and throughput.

The test results evidenced a throughput of 100 payments per second, with 6 simulated banks, and a single trip mean time of 3 seconds and maximum time of 8 seconds. This is the level appropriate for a national level domestic payments system.

The modifications and setup specifics will now be used to take the project, which RBS is intending to open source, forwards.

Introduction

Emerald was setup to explore the creation of a Deferred Net Settlement (DNS) system like FasterPayments[6] using distributed ledger technology.

Existing domestic payment systems are based on a trusted central authority like the Bank of England. Cross currency international payments have no central authority. Distributed ledgers, which allow consensus without a central authority, could be a good fit for international payments. However, a first achievable step might be to put a domestic system live, using a distributed ledger, to prove out the technology.

The Single European Payments Area[7] (SEPA) has lacked a FasterPayments type system. This is being addressed with the SEPA Instant Credit Transfer scheme[8] (ICT). This scheme allows for more than one Clearing and Settlement Mechanism (CSM). The interface defined therein uses ISO 20022[9] and offered a great opportunity; if Emerald looked like an ICT compliant CSM from the outside, it could be developed as a viable alternative for banks wishing to use it.

The ICT defines a payment as the originating bank sending a payment to the beneficiary bank, the beneficiary bank releasing funds to the destination account, and the originating bank receiving notification that the payment has succeeded. At the time of writing, the round trip time for this was not finalised (by the consultation process), but is expected to be somewhere between 10 seconds and 25 seconds.

The goal of the Emerald performance testing was therefore set as:

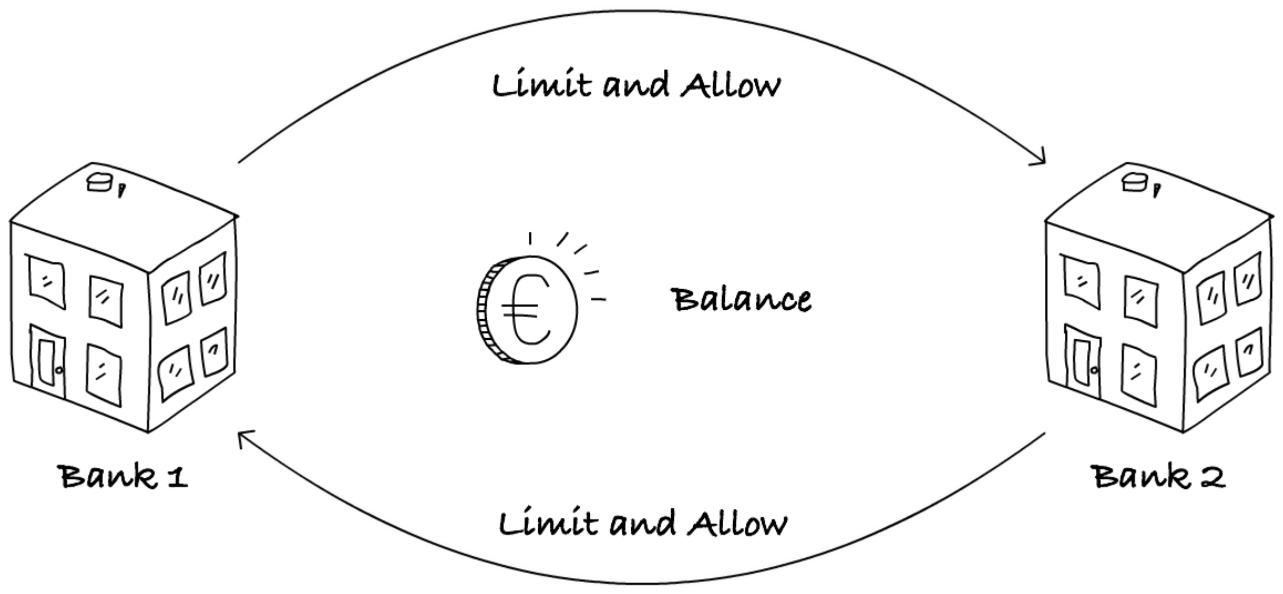
- 100 transactions entering the system every second (sufficient for a national level domestic payment system)
- A round trip time of under 25 seconds (sufficient to meet the ICT requirements)

Three blockchain blocks could be needed for a round trip (the second block starts mining as soon as the first has finished so could not include the return message), so the aim was 10 seconds for a single trip and a block time well under 2 seconds.

Emerald

The underlying model takes a risk-based approach, tracking balances and limits. It is called the creditline model.

Creditline Model



For each pair of banks and a given currency, the creditline model stores:

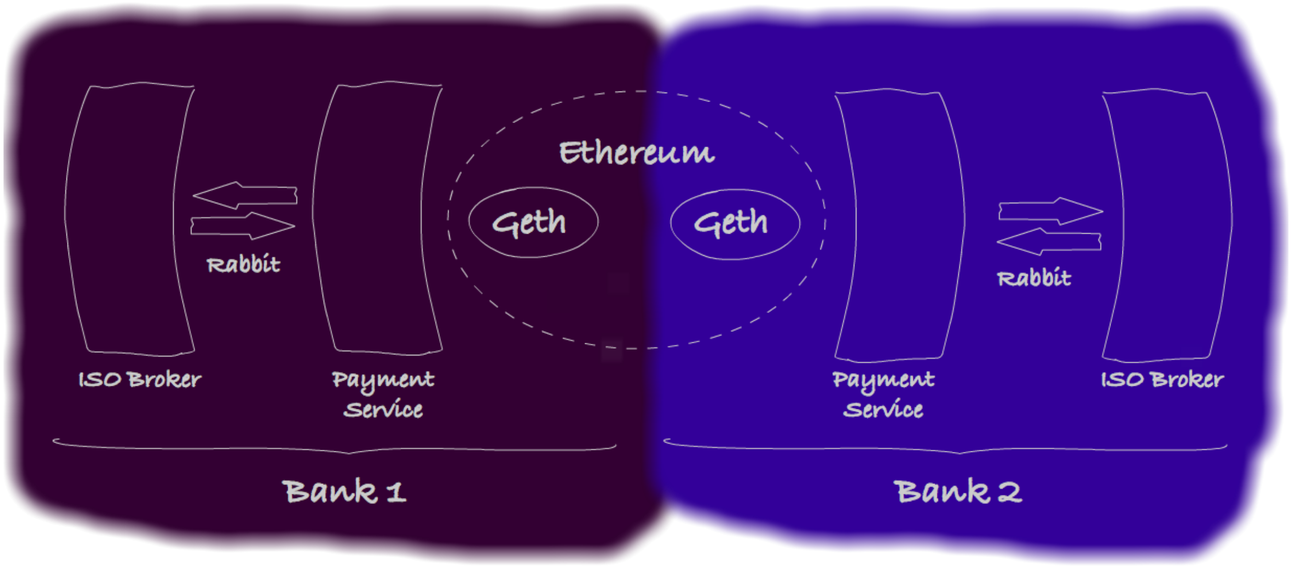
- the balance (who owes whom what)
- a limit for each bank
- an allow for each bank

Bank 1 can *limit* the amount of money Bank 2 can owe them. Bank 1 can *allow* itself to owe bank 2 a certain amount.

When the system is asked to make a payment from Bank 1 to Bank 2, it checks to make sure that the new balance would not exceed the amount Bank 1 allows itself to owe Bank 2 and the limit that Bank 2 has set for the amount Bank 1 can owe them. If the new balance is within these limits, the payment is made and the balance adjusted.

If this system is used to model a DNS, when the banks wish to settle, a single payment is sent using the Real Time Gross Settlement System (RTGS) and the balance changed accordingly.

System Architecture



The version of Emerald that was performance tested has three components in a given deployment in a bank:

- ISO Broker - this is a java-based module that accepts payment requests either over RabbitMQ[10] or REST. It converts these to an internal representation and sends them over RabbitMQ to the Payment Service using JSON-RPC.
- The Payment Service used for the performance testing was written in NodeJS[11]. In the latest version this has been re-written using Java. This decision was made so that ConsenSys'[12] Truffle and EtherPudding libraries could be used to deploy and provide a promisified proxy to the solidity[13] contract, enabling us to focus on the contract.
- The Solidity contract - this contained the data of the creditline model, and allowed for transactional payments to be processed.

All the components were containerised using Docker[14] so that it was relatively easy to deploy many stacks of components representing each bank.

Emerald is now a private, shared source project.

Implementation Learnings

Ethereum was chosen as the distributed ledger because it is one of the most established smart contract platforms that is live in the public domain. The developers liked the idea of the freedom of a smart contract platform, where the smart contract defines the functionality, over a specific product which is well suited to this application but has less inbuilt flexibility.

It should be noted that a consortium Ethereum network was used in preference to the public Ethereum network which is aimed at a very different use case (public Ethereum is tuned to have relatively long lived blocks and the volume of transactions is currently not of the same scale as a payments system).

The contract was written in a language called Solidity.

Some of the methods in the contract were transactional - e.g. sendPayment. Transactional methods, which change the state of the data, only return a transaction id when run. To get information out of Ethereum, Ethereum's event/logging framework was used. Exceptions contain no information so it is hard to use them for situations in which there are many error cases.

All of this meant that to understand what happened to a payment the system essentially:

- submitted the sendPayment method call request and got back a transaction id
- polled the logs looking for the corresponding event
- read the event to understand what had happened to the call
- used the bespoke returnType field to understand what the outcome was

Testing the contract was relatively easy. Initially, Truffle had a built-in framework that worked well with ethereumjs-testrpc. Later, some integration tests using the docker images were built.

Performance Testing

Initial performance testing focussed on results, rather than total understanding. As such, a path was picked through the solution space that led to a result consistent with a small domestic payment system. With more work, it is believed that higher volume payment systems might be achievable. Issues found have been noted for further investigation.

What was Tested?

The plan was to run one stack (ISO Broker, RabbitMQ, Payment Service, Geth (ethereum node)) per bank. During the test, transactions to other banks would be submitted to all banks.

To simplify results gathering, the time measured was that from when the payment was submitted to when the receiving bank was alerted to the payment. The aim was to get this well under 10 seconds with the system coping with 100 transactions per second entering the network.

Test Setup

Containerised stacks were deployed on Google Cloud Platform using Kubernetes[15], which allowed for easy horizontal scaling and topology changes. The timing of transactions was measured by comparing log timestamps from different transaction phases. Google's Big Query was used to aggregate the desired statistics.

GFT prepared an automated, multimode Ethereum network deployment, which allowed for testing various topologies. The initialisation procedure of the bank stack supplied by RBS' engineers was woven into the setup. Every test run was preceded by a fresh deployment of a new Ethereum blockchain, ensuring its initial state remained constant in every case.

The frequency and volume of requests sent to the stack were orchestrated using JMeter[16] in distributed mode.

During testing, GFT explored many different topologies. The setup that gave the best performance had:

- 1 mining node
- 1 node per bank stack
- all the nodes connected to all other nodes (full mesh network)

Testing initially used geth 1.4.3 and later switched to 1.4.7 to pick up some block syncing performance improvements (though this uncovered some Ethereum issues). The very different use case for Emerald meant that some code changes were required to hit the targets.

Issues Faced

The following is a high level list of issues faced during testing:

BlockTime - this is hard coded to 13 seconds in Ethereum and damping means that while it is possible to start with a low difficulty to achieve fast block times, this trends upwards over time. To help with this, the DurationLimit was set to 2 in protocol_params.go.

Second Time Resolution - due to Ethereum having a target block time of 13 seconds, it only needs second time resolution. It guards against blocks getting too far into the future by pausing. When block times are less than 1 second, this gets invoked fairly quickly. It was decided that it was safer to turn off the guards, rather than port Ethereum to millisecond resolution, given the timescales. Small changes were made in worker.go and block_validator.go to remove this and BlockFutureErr as issues.

Forks - as block times diminished, running multiple miners started mining blocks at the same time. For reasons discussed later, it was felt having one miner was a reasonable way to proceed, not least because it allows the presumption that all blocks are valid.

Funding - although Emerald does not really need gas, it was necessary to continuously fund accounts to pay for the volume of work being attempted.

Multiple Threads Mining - commitNewWork could be called in several ways that sometimes resulted in two threads mining the same block with slightly different sets of transactions in. This was turned off in worker.go.

Damping - one new block every second would be appropriate. Damping is not needed for this use case so to increase stability difficulty was hard coded (block_validator.go).

Blocks from Same Source - with only one miner, all the blocks come from one source. Ethereum guards against this so the hashLimit and blockLimit were changed in fetcher.go.

Logging and Metrics Capture - these were turned off to help - certainly high log levels stressed performance though this might have been related to the disks in the VMs being used.

Block Syncing Times - block sync times seemed to be the limiting factor - getting the blocks out to the nodes. This needs the most investigation going forwards.

Results

The best results achieved were for a 6 node setup - thus emulating a small domestic setup with 6 banks, with 100 transactions entering the network each second, spread across the banks. In this setup a steady state was achieved with the following transaction times:

- minimum - 665 milliseconds
- mean - 2805 milliseconds
- maximum - 8085 milliseconds

Learnings

Bearing in mind that the Ethereum platform has not been aimed at this use case before, the results have been encouraging. Clearly, more investigation needs to be undertaken to make this work at scale.

Most interesting was that a lot of the complexity in Ethereum is not needed in this private network use case.

Gas - is used in Ethereum to stop infinite loops taking out the whole network. However, Emerald only has a small number of well-tested contracts. The need for a gas-like concept tracking the cost of operations (it is not possible to tell in advance whether a program will complete) led to the development of the EVM (Ethereum Virtual Machine) and to solidity. Not needing this concept means that in principle Emerald could use the richer and more mature JVM and perhaps Java[17] rather than Solidity.

Gas is probably part of the reason that transactions seem to be run twice on each mining node.

Damping - is not needed in the Emerald use case. Fixed block times would make the system much more predictable.

Proof of Work - is an amazing concept that allowed Bitcoin to solve the double spend problem by creating distributed consensus about the order in which transactions are applied to the blockchain. A key point is that it costs so much (kit, electricity) that miners are incentivised to make sure that the Bitcoins they get rewarded with for mining a block maintain their value - i.e. they will not cheat.

At its heart, it is a way of a dictator electing themselves so that they can say "I am going to put these specific transactions, in this order, in the next block". The election involves solving an, on average, hugely compute-intensive problem.

This works fine when the amount of work to find the next block is massive - but Emerald needs small, consistent block times. Arguably, the trust issues that lead to the need for proof of work do not exist in the payments world - for example, a bank is trusted to put the right amount for a payment in the destination account, so banks implicitly trust each other right now.

A much simpler round-robin-based approach might be more suitable for Emerald - each bank taking it in turn to pick the transactions into the next block.

This is discussed much more thoroughly in R3's excellent "Introducing R3 Corda" white paper[18].

Mined Time To Event Time - most of the time lost was between a transaction being mined in a block and then the event being fired on the two nodes that care about this event (payer and payee).

Next Steps

There are several possible paths that can be explored as next steps to improve performance:

- More testing
- Fork or Contribute to Ethereum
- State Channels
- Look at other technology

More Testing

To further prove Ethereum in this use case, the next step would be to delve into the mined time to event time issue. In order to identify the next series of bottlenecks, there are several things to be explored here:

- Is the network being stressed? If so, is it blocks or transaction related?
- How to optimise block propagation?
    - Do more, smaller blocks perform better than fewer, bigger blocks?
- The non-mining clients have to run the transactions on receipt of the block they are in to establish the end state of the block and generate the log messages - do they need more CPU?
- Disk issues? Logging etc?

This would need to be driven by an actual use case as the domain this is to be deployed in will define many of the requirements and possibly the topology of the application across banks.

Fork or Contribute to Ethereum

Ethereum is very much aimed at being a public system. Many of Emerald's requirements are at odds with the broad direction Ethereum is taking. That said, it would be immensely powerful to extend Ethereum to support both use cases. Some conversations with the core Ethereum team would resolve whether contributing or forking would make the most sense.

State Channels

Global consensus blockchains have inherent performance issues due to the need to get all the information out to everyone all the time. However, given that the model represents many bilateral relationships which are independent of each other, what level of global consensus is required?

In a recent paper written by Vitalik Buterin[19], co-founder of Ethereum, he pointed out that using state channels to take most of the actual transactions off the blockchain could result in many orders of magnitude of performance improvements.

The broad idea is that clients agree off chain on the current state and periodically the latest agreed state is updated to the chain. The period is something which could be defined at the network level or bilaterally.

The "agreement on state" process involves an ordered sequence of states being signed by both parties - essentially containing the new balance between them. If there is ever an argument about the current state then a contract can be written that allows the highest sequence number signed by both parties that either of them can present to the ledger to be used to update the ledger.

It is a very elegant way of speeding things up at a cost of some complexity, e.g. how do you setup the state channels? Something similar is being done on Bitcoin[20] by the Lightning Network[21].

Other Technologies

Not necessarily needing global consensus (at least in the DNS use case) might allow for other technologies to be considered.

R3 Corda, Big Chain DB[22], Hyperledger[23] and Interledger[24] are possible alternative technologies that take very different approaches that might be more appropriate.

Conclusion

In summary, with some modification, Ethereum can scale to payment volumes consistent with a domestic payment system. There are some more avenues to investigate that would lead to it supporting much higher volumes. There are other technologies that might be considered too.

References

	Note	Reference
1	Ripple cost savings	https://ripple.com/insights/ripple-and-xrp-can-cut-banks-global-settlement-costs-up-to-60-percent/
2	Royal Bank of Scotland	http://www.rbs.com/
3	Ethereum	https://www.ethereum.org/
4	GFT	https://www.gft.com/
5	Google Cloud Platform	https://cloud.google.com/
6	Faster Payments	http://www.fasterpayments.org.uk/
7	SEPA	http://ec.europa.eu/finance/payments/sepa/index_en.htm
8	SEPA ICT Documentation	http://www.europeanpaymentscouncil.eu/index.cfm/knowledge-bank/epc-documents/draft-sepa-instant-credit-transfer-rulebook-for-public-consultation-and-response-template/
9	ISO 20022	https://www.iso20022.org/
10	Rabbit MQ	https://www.rabbitmq.com/
11	NodeJS	https://nodejs.org/en/
12	ConcenSys	https://consensys.net/
13	Solidity	https://solidity.readthedocs.io/en/develop/
14	Docker	https://www.docker.com/
15	Kubernetes	http://kubernetes.io/
16	JMeter	http://jmeter.apache.org/
17	Java	https://www.java.com/en/
18	R3 Corda White Paper	https://r3cev.com/blog/2016/4/4/introducing-r3-corda-a-distributed-ledger-designed-for-financial-services
19	R3 Ethereum Paper - written by Ethereum co-founder Vitalik Buterin - includes State Channels.	https://www.scribd.com/doc/314477721/Ethereum-Platform-Review-Opportunities-and-Challenges-for-Private-and-Consortium-Blockchains
20	Bitcoin	https://bitcoin.org/en/
21	Lightning Network	https://lightning.network/
22	Big Chain DB	https://www.bigchaindb.com/
23	Hyperledger	https://www.hyperledger.org/
24	Interledger	https://interledger.org/
25	Fantastic Bank of England articles explaining what money is.	http://www.bankofengland.co.uk/publications/Documents/quarterlybulletin/2014/qb14q1prereleasemoneyintro.pdf http://www.bankofengland.co.uk/publications/Documents/quarterlybulletin/2014/qb14q1prereleasemoneycreation.pdf
26	GFT Testing Methodology	http://www.gft.com/blockchain
27	Emerald GitLab Repository	https://gitlab.com/emerald-platform/emerald
28	Emerald website	http://emerald-dl.com/