FLIGHTCHAIN

RESEARCH INTO THE USABILITY AND PRACTICALITIES OF BLOCKCHAIN TECHNOLOGY FOR THE AIR TRANSPORT INDUSTRY

WHITE PAPER

CONTENTS

INTRODUCTION

PROJECT BACKGROUND

Blockchain has been has been heralded as a transformational technology. While several use cases have been identified by airlines and airports, research is required to establish the suitability and practicalities of using blockchain to establish a 'single source of truth' for various data sets in use across the highly-connected air transport industry. SITA recognizes that there is also a real need for the industry to take the right approach, to ensure governance, standards, compliance, security and more.

For this reason, SITA Lab, the technology research team at SITA, initiated a project to investigate the provision of a single version of the truth for flight status data. Called FlightChain, this is an air transport industry blockchain research project established by SITA Lab and defined in conjunction with Heathrow Airport Holdings Limited (HAL) and International Airlines Group (IAG). In addition, Geneva Airport, and Miami International Airport joined the project part way through, demonstrating the scalability of the platform.

In this research, FlightChain is a private permissioned blockchain (implemented on both Ethereum and Hyperledger-Fabric) that stores flight information on the blockchain, using a smart contract to arbitrate potentially conflicting data. Data from LHR, BA, GVA and MIA is merged and stored on the blockchain.

During the course of this project the answers to the following questions were sought:

- How do you setup, secure and manage a permissioned blockchain?
- Who manages and controls permissions and access?
- How do you write a smart contract and who signs off on smart contract logic - is it similar to an Airports Council International (ACI) or International Air Transport Assocation (IATA) standard?
- How do you update a smart contract?
- How do we keep some data private and some public?
- Is there a need for an air transport industry vertical blockchain – one blockchain running many apps, or one blockchain per app?
- If there is a trusted transparent verifiable ledger of flight data, does it change anything?
- What are the comparisons and contrasts between Hyperledger Fabric and Ethereum blockchain offerings?

IONS

WHY FLIGHT DATA?

As noted above, the focus of this project was on blockchain technology. We selected flight data as a use case to test blockchain's capabilities, to explore implementation complexity, and to identify the performance of blockchain. In addition to the learning objectives we selected flight data for the following reasons:

- Flight data contains no Personally Identifiable Information (PII) or commercially sensitive data which means partner airlines and airports are comfortable sharing this data for the project.
- The "flight data problem" is a well-known issue in the industry - namely, there are multiple copies of subsets of flight status data and the data that does exist is not easily accessible by all parties. There is no single source of the truth about all flight data.

This lends itself to the use of blockchain – there are multiple writers of data and there is a need for a distributed data set.

WHY ETHEREUM AND HYPERLEDGER?

This project was implemented on both Ethereum and Hyperledger Fabric. Implementation on multiple blockchains allowed the team to identify aspects of blockchain (good and bad) which may apply to all implementations and which are specific to a vendor.

These blockchain implementations were chosen because they support smart contracts and private permissioned blockchains. In addition, Ethereum is a well-established implementation of blockchain, and Fabric has significant backing from the Hyperledger consortium, in particular, IBM.

WHY BLOCKCHAIN?

There are several established technologies available to solve the "flight data problem"; a centralized database (e.g. SQL) with a CRUD API; a decentralized database (e.g. Cassandra, Hazelcast).

Blockchain is also an appropriate technology choice for the following reasons:

- **Distributed immutable ledger** – Blockchain implementations provide a cryptographically immutable transaction ledger that is distributed to all participants in the network. Thus all participants will have a complete copy of all transactions on the ledger and have confidence that the record of transactions is true and consistent for all participants.
- **Multiple writers** – Blockchain provides a mechanism for multiple writers to update a common data set, where the data set is visible to all participants in the blockchain.
- **Absence of trust** – Blockchain has advantages over centralized/distributed databases in the case where there is an absence of trust between writers of the database. This is because all transitions are immutably recorded and shared on the ledger such that readers and other writers can decide whether to accept or ignore the transactions of any participant.
- **Shared Control** – The use of a smart contract allows different organizations to share control of the data through an approved and shared set of business rules codified by the smart contract. This disintermediation approach enables shared control of the data and presents a key differentiator in contrast to the trusted intermediary model exhibited by a centralized or decentralized database.

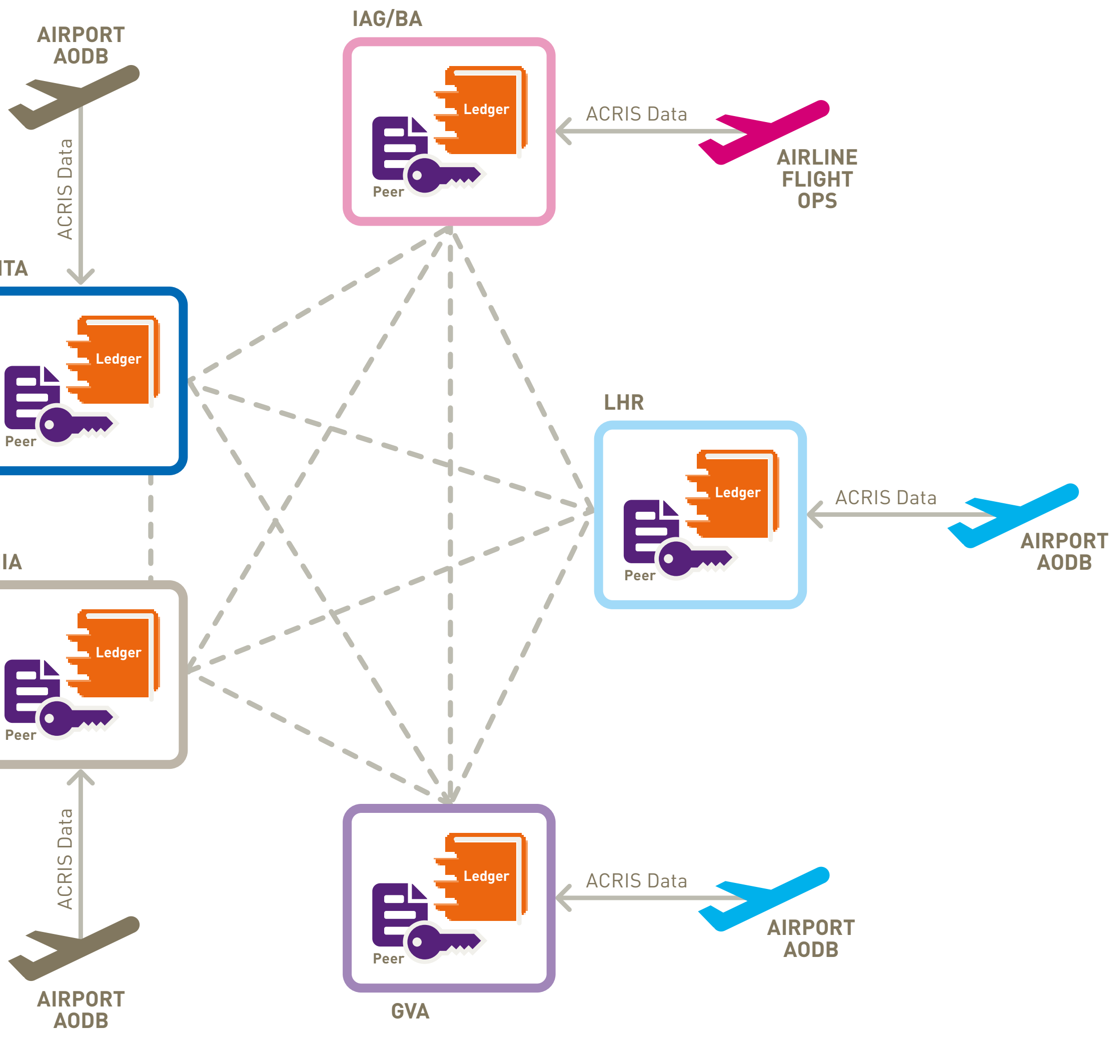
FLIGHTCHAIN CONCEPTUAL OVERVIEW

The FlightChain project consisted of:

- A private permissioned blockchain
- A smart contract running on the blockchain to merge flight data
- A source of operational real-time flight data from multiple airlines and airports.

This is illustrated below. As each operational data-source pushes flight data into the blockchain, the smart contract, running on each node, validates the data and writes this data to the blockchain ledger. The resulting set of data for a flight combines data from the operating airline, departure airport and arrival airport. This full data set can be queried from the blockchain.

During this project more than two million flight changes were processed by the smart contract and stored on FlightChain.



WHITEPAPER CONTRIBUTORS

Name	Title
Kevin O'Sullivan	Lead Engineer, SITA Lab
Sholeh Behzadpour	Innovation Technologist, IT Futures, HAL
Stuart Harwood	Heathrow Automation & Innovation, HAL
Harvey Tate	IAG Innovation, IAG

GLOSSARY

Term	Explanation
AODB	Airport or Airline Operating Database. This is the source for flight data from the airline or airport.
ACRIS	This is an ACI standards group. The flight data is stored in the blockchain in ACRIS data format. See http://www.aci.aero/About-ACI/Priorities/Airport-IT/ACRIS
Bitcoin Blockchain	This is the original blockchain network, supporting the bitcoin cryptocurrency. It is a public network. It does not support smart contracts.
Consensus	This is the term to describe how the distributed nodes in a blockchain network come to agreement on transactions submitted to the network. There are several different consensus algorithms. See glossary terms "Proof of ..."
DApp	Distributed App. An application using smart contracts are known as a Distributed App. A DApp may also include a user interface (UI) and some sort of distributed storage.
DLT	Distributed Ledger Technology - another term for Blockchain
Fabric	Fabric is one of the blockchain implementations under the Hyperledger consortium. It was developed by IBM and open sourced through Hyperledger.
Ethereum	Ethereum is an open source blockchain. There is a public Ethereum network. It can also be run as a private permissioned network (in this FlightChain project it was run as a private network). Ethereum supports smart contracts. https://www.ethereum.org/
Hyperledger	Hyperledger is a consortium of cross-industry companies working to advance maturity blockchain technology for use in enterprise space. Companies such as IBM, Accenture, Intel, American Express, etc. are open sourcing their own developments to speed up adoption of blockchain technology and to solve common problems.
Parity	Parity is an Ethereum client. An Ethereum network is made up of a series of clients, connected in a peer-to-peer manner. There is no Ethereum server. https://parity.io.
Proof of Authority	This is the consensus algorithm used in the Ethereum implementation of FlightChain. It relies on at least two nodes which have the authority to create new blocks. https://github.com/paritytech/parity/wiki/Proof-of-Authority-Chains
Proof of Stake	This is the consensus algorithm that relies on owners of bitcoin (or other cryptocurrency) to approve transactions.
Proof of Work	This is the consensus algorithm used on the bitcoin blockchain. It relies on miners solving mathematical problems (doing work). This may be computationally expensive for the miner, but simple for others to verify. It is a defense mechanism against bad actors manipulating the blockchain data.
Smart Contract	A smart contract is a software program that runs on the blockchain. It resides on all nodes and when data is written to the blockchain, it processes a set of business rules. The results of the smart contract are written on the blockchain.

FLIGHTCHAIN – KEY LESSONS

Together the team has compiled the key lessons from FlightChain from airline, airport and technology supplier viewpoints.

They are shared here for the air transport industry to consider as it examines the use cases and benefits of the blockchain for airlines and airports. It is important to note that these key findings relate to the use of a private permissioned blockchain and do not necessarily apply to public blockchains.

GOVERNANCE

KEY LESSON: A private permissioned blockchain still needs governance and operational oversight. Simply because it is distributed and decentralized does not mean it is self-managing. It is important to choose a governance model/organization that does not compromise the integrity of the blockchain.

In a public network there is no central leadership or control over the direction of the network – it is truly distributed and decentralized. This can result in slow decision making about the future direction of the blockchain, heated debate and splintering of a group. An example of this is the contention between the bitcoin development team (also known as Bitcoin Core) and the major mining consortiums which resulted in a fork of bitcoin in August 2017. As a consequence, two bitcoin blockchains emerged. Similar forks have happened in Ethereum (but in that case the fork was to reverse a hack of the network).

In a private network, things are very different. Participants join by invite only and different participants may have different levels of access. This means that some entity is responsible for the governance – issuing invites to participate, managing identity, access and permissions.

Typical governance responsibilities include:

- Adding an account or organization to the network, so that organization can perform transactions on the network
- Defining and managing permission levels
- Deploying and upgrading smart contracts on the network
- Adding nodes to the network
- Managing upgrades to the system
- Revoking access

The requirement to have governance oversight is a significant difference from a truly distributed decentralized blockchain. It is obviously important that participants trust the governing entity, and have visibility of, and a stake in, the rules governing the network. It should also be noted that the governance over a network does not imply absolute control because some actions still require consensus of other participants – therefore it should be seen more as a caretaker role (albeit a caretaker with lots of privileges).

Industry organizations like ACI, IATA, or SITA could act as the trusted organization to setup and manage a private blockchain for the air transport community.

MATURITY OF TECHNOLOGY

KEY LESSON: It is still early days in the technology lifecycle for blockchain. A blockchain can be complex to set up and manage, especially when compared to point and click cloud services like AWS or Azure. Look for 'blockchain-as-a-service' offerings, and beware of vendor claims around maturity.

Either ensure you have correct skillsets, or use 'blockchain-as-a-service' offerings or simply wait until the technology matures.

We are still relatively early in the lifespan of blockchain technology, and certainly very early when it comes to repurposing it for uses outside of bitcoin. Ethereum was launched in 2015, and Hyperledger Fabric v1.0 launched in 2017. This is reflected in the lack of tooling when it comes to setting up, managing and monitoring a blockchain.

FlightChain was developed using a 'roll your own' approach to deploying Parity and Fabric onto AWS Ubuntu VMs, in order to get some direct experience and understanding of the complications of implementing a blockchain. One of the key lessons of FlightChain is that it would be complex to scale a network to many participants, especially when onboarding new airlines and airports after initial setup. Setting up a node consists of multiple manual steps for installing software, creating accounts, distributing this information to pre-existing nodes, restarting those nodes, etc. It is a process that is prone to error and resists automation.

However, there are also several blockchain-as-a-service (BAAS) offerings, which should simplify this operational overhead.

- **Microsoft Azure** – Microsoft have a relationship with Ethereum and offer Ethereum (and some other blockchain implementations) on Azure.
- **IBM Bluemix** – IBM have a Fabric implementation.

Of course, even though the blockchain is offered as a service, there will still be requirements for the service to be appropriately managed, maintained and governed per the requirements of the community using the blockchain.

Overall, any organization looking to deploy a blockchain solution needs to ensure the correct skillset is in-house, or consider a BAAS offering, or else wait for the technology to mature.

It should also be noted that there is a lot of activity around making blockchain more enterprise friendly and it is evolving rapidly through multiple competing vendor approaches.

SMART CONTRACTS

KEY LESSON: Smart contracts are programs that can update the state of data on the blockchain and are a key element to most enterprise blockchain use cases.

KEY LESSON: Smart contracts can be complicated to define, update, redeploy and get all participants into sync again. Strong lifecycle management is required.

KEY LESSON: Smart contracts have no legal status, however industry standards could be encoded in smart contracts.

A smart contract can be viewed simply as a set of business rules that are executed as a transaction on the blockchain. A blockchain is deployed onto the network and, in the case of Ethereum, runs on all nodes (Fabric has a slight variation). This means that all participants can have confidence that not only is the data consistent on all nodes in the network, but how users transact and interact is the same too – nobody gets preferential treatment.

When executing a smart contract, the participant can supply input parameters. The smart contract must be deterministic and executed on multiple nodes (the validator nodes). Assuming all nodes agree with the output of the transaction (consensus is achieved) then the transaction output is committed to the blockchain. Executing a smart contract typically updates the 'world state' of the blockchain, either by transferring assets, or in the case of FlightChain, updating flight data.

In FlightChain, the smart contract is responsible for applying the business rules about which an entity can update particular elements of the data. For example, Ryanair would not be able to update the flight data for a British Airways flight or Munich Airport would not be able to update the flight data for a flight from Heathrow to Madrid. See Appendix C for the complete logic of the FlightChain smart contract.

Industry standards as smart contracts?

While the current FlightChain smart contract logic is very simple, it is easy to see this evolving and becoming more sophisticated in how it manages conflicting information. As new rules are defined, there must be consensus amongst participants before a new smart contract is created and deployed on the blockchain.

A suggested way to achieve this consensus is through industry bodies such as ACI and IATA. However, instead of the current approach to developing industry standards, which is a taskforce to discuss and create a document (a recommendation or a resolution), these taskforces could instead create something much more deterministic – the smart contract itself. All parties could review the actual code implementation of the smart contract and once signed off it would be deployed. This process is a significant improvement as it avoids variations in interpretation of the documented standard which often occurs as a written document is misinterpreted by many different airlines, airports and IT suppliers.

Legal status of smart contracts

The term 'smart contract' is a misnomer, and it does not necessarily imply any special legal status. (Smart contract is also used in conjunction with terms such as 'programmable economy' and 'code as contract'). A smart contract has no special legal status, and certainly, in the near term any business relationship between parties using data on a blockchain will need to be backed by standard due diligence and negotiation of terms and conditions. That said, one of the expectations of blockchain is that smart contracts will streamline B2B engagement by removing friction associated with establishing trust between parties.

SYSTEM SECURITY

KEY LESSON: All the standard enterprise security risks apply to blockchain, with the additional complexity of managing a system distributed across multiple enterprises. It is only as secure as the weakest link.

In a private permissioned blockchain, failure to appropriately secure the network will result in leak of data and potentially loss of control of assets stored on the blockchain. So of course, security is as important for a private blockchain as it is for any other corporate IT system.

Securing access to the blockchain

Basic access to participate in the blockchain is managed by software configuration and network configuration. Each node has its own blockchain node address (known as an enode in Parity) and typically each node runs on its own server, so has its own IP.

The network configuration lists the valid enodes, therefore to breach this level of security a bad actor would need to modify the configuration on one node (note – not all nodes, just one) to add their node address to the permitted list. In addition, the bad-actor would also need to modify the firewall rules on that node to allow communications between the now breached node and the bad node.

This type of attack would allow someone to replicate all data on the network.

Permission to transact

The above attack would allow a bad-actor to have read-only access to the data. In order to transact and participate in the network by executing transactions, a bad-actor needs to have an account and have sufficient privileges and (depending on the nature of the network) sufficient cryptocurrency.

This can be done by getting administrator access and creating such an account, or simply getting access to an existing participant's account.

It is important to note that the attack vectors are similar to those on a traditional enterprise system, and similar defenses and monitoring need to be in place. The added complication of a distributed system like a blockchain network is that you are relying on all participants to have these defenses in place. It is not necessary to breach all nodes on a network, only one. As with all security, the chain is only as strong as the weakest link.

PRIVATE VS PUBLIC

KEY LESSON: For most enterprise use cases a private managed blockchain network will be required, not a public network.

A private blockchain is functionally the same as a public blockchain (e.g. distributed peers, ledger of events, smart contracts, cryptocurrency), with the principal difference being over who can participate and the consensus models.

Public Network

In a public blockchain like Bitcoin or Ethereum, anybody can join, add a node, be a miner, participate in consensus, view transactions and (on Ethereum) execute smart contracts. A public blockchain is truly distributed in that no individual or corporation controls access. Everybody has access to the network, and has visibility of all data on the network.

One benefit of a public network is that due to the large number of nodes participating, it is computationally very expensive to take over and manipulate the network to alter the ledger. While this is commonly referred to as 'more secure' it is more accurate to refer to it as 'more tamper-proof'. There are many examples of security breaches that result in loss of cryptocurrency – a blockchain is not a magic security blanket.

The public nature of the data in blockchain is one of the downsides when viewed from an enterprise perspective. In most cases in a business network it is a regulatory or commercial requirement that access to data is limited to authorized people or organizations. In a public blockchain, this is hard to do.

Private Network

A private network addresses these enterprise concerns. It will be setup by an individual or an organization and participants require an invitation to join. A private network is also typically a permissioned network. Different participants have different levels of access – e.g. view only, transact, verify.

In a private network, the consensus model can also be radically different from the typical "proof of work" consensus models of bitcoin blockchain, or public Ethereum blockchain. This type of network can use "proof of authority" which removes the requirement for mining and increases the transaction throughput.

It should also be noted that a private permissioned network is no longer a truly distributed network because some entity must be ultimately responsible for its governance - and therefore in a position of trust.

DATA PRIVACY

In FlightChain, the data was stored unencrypted on the blockchain. And of course, the nature of blockchain is that all participants have all the data on blockchain. This is fine for the specific needs of this group for this research project, but in many other cases it will be necessary to encrypt the data, or at least provide a mechanism for limiting access to the data.

In Ethereum the options are limited – the data must be encrypted at source and stored encrypted on the blockchain. Whoever needs the data must have the private key to decrypt it after reading it from blockchain. This adds a processing overhead, albeit not an overhead that happens on the blockchain. It also adds the overhead of key management between participants.

Hyperledger Fabric has a different approach and subsets of participants on a blockchain can set up their own channel for sharing sensitive data. This data will not be stored on the main blockchain, and is kept private between participants on the channel. An example of this would be the sale of a property – the transfer of ownership would be stored on the main chain as this is public information, but the cost of the property stored on a separate channel and kept private between seller and buyer.

SYSTEM PERFORMANCE, SCALABILITY & RESILIENCE

KEY LESSON: Frequency of block mining is a limit on performance. In a private network this is largely mitigated.

KEY LESSON: Fabric is more complicated than Ethereum, but it is designed for high throughput.

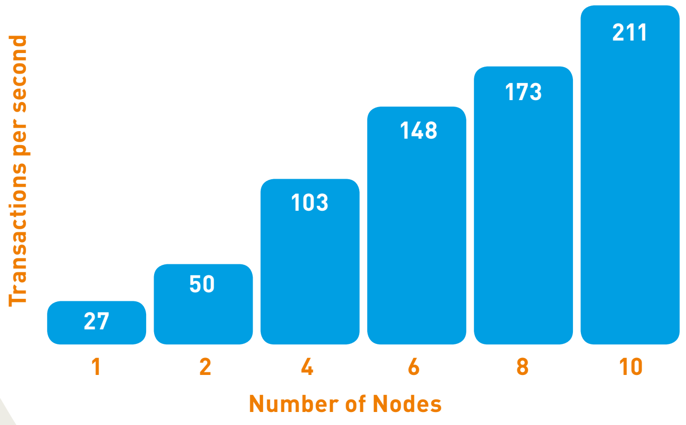
KEY LESSON: A distributed system is resilient by nature. But while we can say the network as a whole holds a single version of the truth, for an individual node we can only say 'it holds a single version of the truth, eventually'.

Performance

One of the major questions about blockchains is the transaction throughput and the scalability. While a blockchain can scale to thousands of nodes by design, this does not correlate to an improvement in transaction throughput, in fact, it can often cause a longer transaction execution time because the transaction has to execute on more nodes for consensus to be achieved.

Ethereum Parity

A single Parity node can typically handle about 30 transactions per second. To improve this performance, transactions can be load-balanced across multiple nodes. Below is a diagram indicating the transaction throughput at various levels of load-balancing. Of course, real results will vary based on the complexity of the smart contract (if any).



This is well below the requirements of a real-time transaction system like a passenger services system (PSS) or a payment system. However, scalability and throughput on blockchain is an area of much research at the moment.

Hyperledger Fabric

The stated aim for Fabric is to support 100,000 transactions per second. Fabric has a very different architecture to Parity to support this. With Fabric, there are dedicated Endorsement Peer nodes for running the smart contracts (known as chain codes). Consensus is achieved on these nodes before the data is written to the ledger via separate Ordering nodes. Endorsement Peers can be scaled arbitrarily and independently of the Ordering nodes.

Resilience and maintaining data integrity

A blockchain is a distributed system. It is designed to have any number of participating nodes, and each node generally has a copy of all the blockchain data. So, what actually happens if a node goes down and then re-joins the network – how is integrity of data preserved and what is the source of the truth?

Consider the scenario where a node is disconnected from FlightChain for a period of time: the current state of the blockchain is defined by the 'BlockNumber' – that is the number of blocks of data that are chained. Each time a block of transactions is added to a blockchain this number is incremented. When a node joins a blockchain for the first time, or rejoins after a period of being disconnected, it will identify what the current network BlockNumber is and it will request data from its peers to bring itself back into sync with the rest of the network. This resync is an automatic process on a blockchain. Of course, during this period this node will not have the correct version of the truth (flight data in the case of FlightChain).

Because of this any external business service relying on the data will need to be aware that querying data from this particular node may result in incorrect data until it is fully synced. Therefore we may interpret the blockchain as providing a single version of the truth, eventually. There are solutions for this problem and they generally rely on a business service relying on an off-chain copy of the data and resorting to on-chain data if/when independent validation is required at a subsequent stage.

FlightChain has demonstrated that blockchain is a viable technology choice for the use case of providing a single source of truth for data, specifically real-time flight information. While it could be argued that there are alternative and proven technology choices for simply sharing data (e.g. Cassandra or Hazelcast), the use of blockchain, and smart contracts in particular, provides 'shared control' of the data set and improved trust in the data.

It is still early in the technology lifecycle, and even during this project there were many changes made to the Ethereum and Fabric platforms as both are under rapid evolution. The current lack of maturity in the toolsets for operation of a blockchain makes it complicated and error prone to implement a network across many different airlines and airports. For this reason, Blockchain-as-a-Service is a compelling option.

While the FlightChain smart contract is relatively simple, it provided an important role in controlling access to the data. In a real-world network, it is important to manage the changes to this contract as it affects all participants. It may be necessary for signoff from industry bodies such as ACI and IATA. One can imagine a future where industry standards are written directly as smart contracts instead of published as PDF documents.

While no decisions or commitments have been made, logical next steps for this project would be to:

- Add many more airlines and airports to FlightChain to get a more complete data set.
- Add more sophistication to the FlightChain smart contract.
- Identify a business model to fund the operation of FlightChain.

Airlines, airports and other industry stakeholders interested in the FlightChain project can contact the team lead, Kevin O'Sullivan at SITA Lab.

APPENDIX A - ETHEREUM

This section contains some low level technical details on the Ethereum implementation of FlightChain, which will be most relevant to architect, developer and support teams.

FlightChain was implemented using Parity, one of several Ethereum client apps. The information below is specific to this client, although the implementation details will not significantly change with other clients (e.g. geth).

VM SYSTEM REQUIREMENTS

CPU

The system requirements are relatively modest. For FlightChain, each node is running on an Azure DS2 V2, or an AWS t2.small instance – and even then, the CPU consumption is generally less than 10%. Although, it must be noted that the transaction throughput for FlightChain at this stage is < 5 per second.

Memory

Equally the memory requirements are quite low – the parity client consumes < 0.5 Gb. This of course will vary depending on the number of smart contracts deployed and the use of those contracts, and the number of transactions recorded over time.

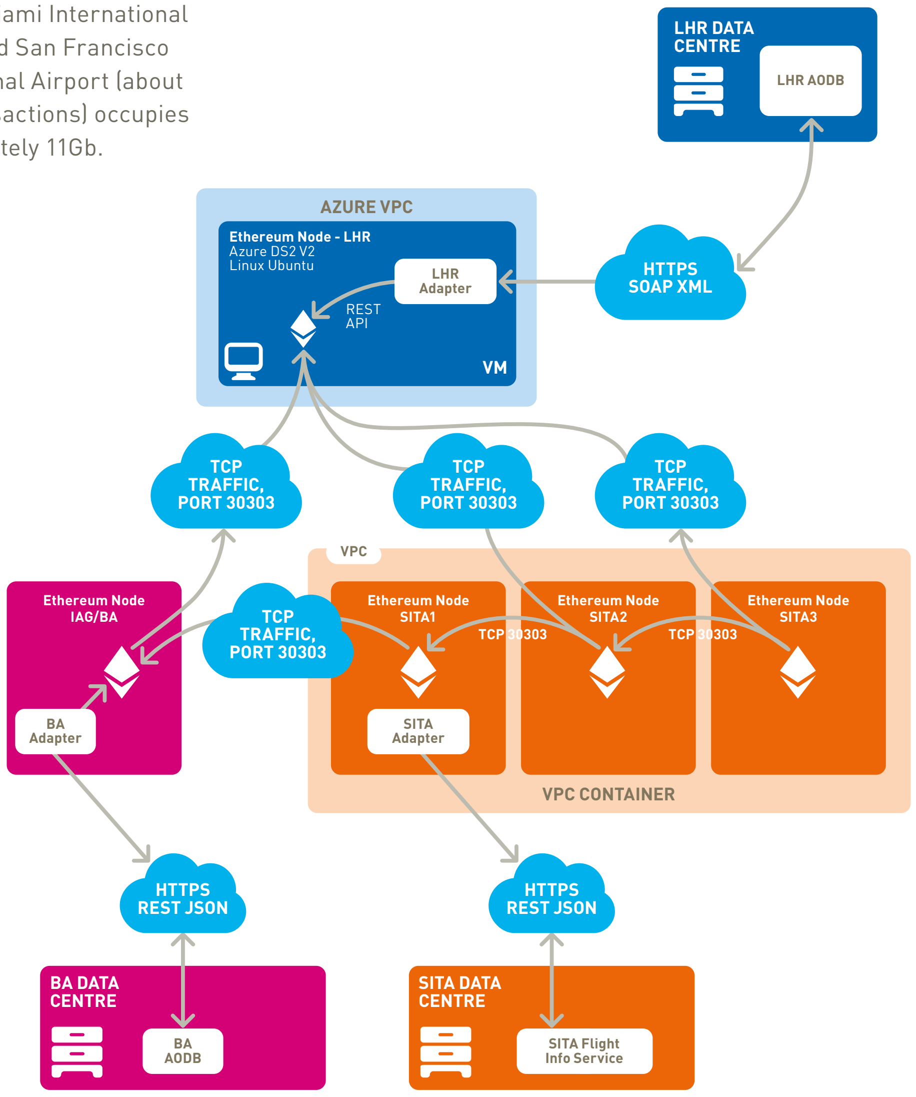
Disk Requirements

The disk space requirement has a direct correlation with the number of transactions (in FlightChain, the number of flight updates) - the more flight data recorded, the more space required. Disk utilization is efficient, and there is not any significant overhead required by Parity beyond the data passed to the FlightChain contract. As an illustration, three months of flight updates for all flights into and out of London Heathrow, Geneva Airport, Miami International Airport and San Francisco International Airport (about 1.3m transactions) occupies approximately 11Gb.

NETWORK REQUIREMENTS

A blockchain is a distributed network, and the nodes all need to communicate with multiple other nodes to ensure the data is kept in sync. In the case of Parity, peer-communication is over port 30303 (by default). Therefore, the only network firewall requirement is that this port must be open and each node should see multiple other nodes. It is not a strict requirement that all nodes see all other nodes, but the more nodes in communication with each other, the more robust and resilient the network.

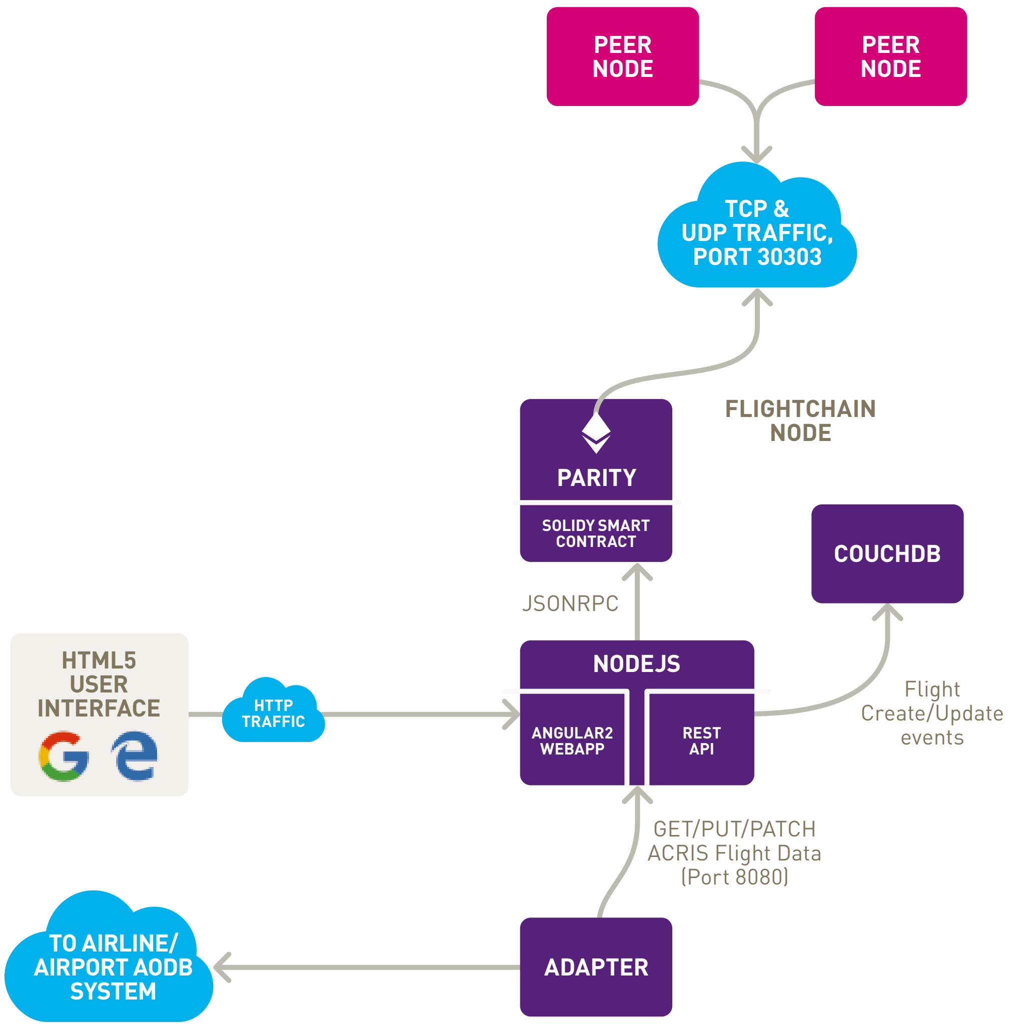
On each node, the Parity client also listens on port 8545 for JSON RPC calls to interact with the blockchain data.



NODE ARCHITECTURE

To the right is the architecture for each node in the network. In FlightChain, each node is an Ubuntu VM running the following processes:

- **Parity -** The Parity clients running across all the nodes form the p2p Ethereum network.
- **Node JS -** Node bridges to Parity over the JSON RPC interface and exposes a GET/PUT/PATCH API interface to the Adapter. The Node app also receives all transaction events emitted by Parity and writes flight transaction updates to CouchDB for fast lookup of data.
- **Couch DB -** Opensource database used to store a copy of the flight data. This is used as an index into the blockchain for fast lookup of data.
- **Adapter -** The adapter is a bespoke process that interfaces to the airline or airport AODB system and converts the data into the ACRIS standard data format and merges changes into the FlightChain network.



SMART CONTRACT

Ethereum uses a language called Solidity for Smart Contracts. Solidity is statically typed language supporting inheritance and importing of third party additions through library plugins. Solidity programs are compiled to bytecode and then deployed onto Ethereum blockchain. When compiled they also create an application binary interface (ABI) stub file which can be used to simplify invocation of the contact. When deployed on the blockchain, each smart contract has its own address.

There is no practical limit to the number of smart contacts that can be deployed on an Ethereum blockchain. As of September 2017, there are 1.7 million contracts deployed on the public Ethereum network.

Input / Output

When invoking a smart contract, the client app can pass in parameters. These parameters are defined in the ABI file. The contract is invoked by sending the parameters to the local Parity client app on a single node. Ethereum will then distribute and invoke the transaction on all validating nodes. The smart contract can also generate output events. These events will be fired on every node – these events can be captured to get error output or to store generated output in a side DB for faster lookup.

Contract Lifecycle

This is the typical lifecycle of developing/invoking a smart contract. It assumes that you already have an account on the blockchain, along with some Ether.

- Write smart contract using solidity code (.sol files)
- Compile to bytecode and generate a .abi stub file (which contains details on how input/output is encoded)
- Deploy the smart contract bytecode to Ethereum blockchain
- Use .abi stub files to write code (e.g. JavaScript) to invoke the smart contract and listen for output events.

When updating the smart contract code and redeploying it, it is deployed to a new transaction address. And the pre-existing smart contract is not removed (after all, it is stored as a ledger entry and the ledger is immutable). Therefore, an important consideration is to ensure that all participants in a blockchain app are running against the same version of the smart contract. This is a significant management overhead, external to the blockchain operation itself, and can be especially complicated to manage with a large number of participants.

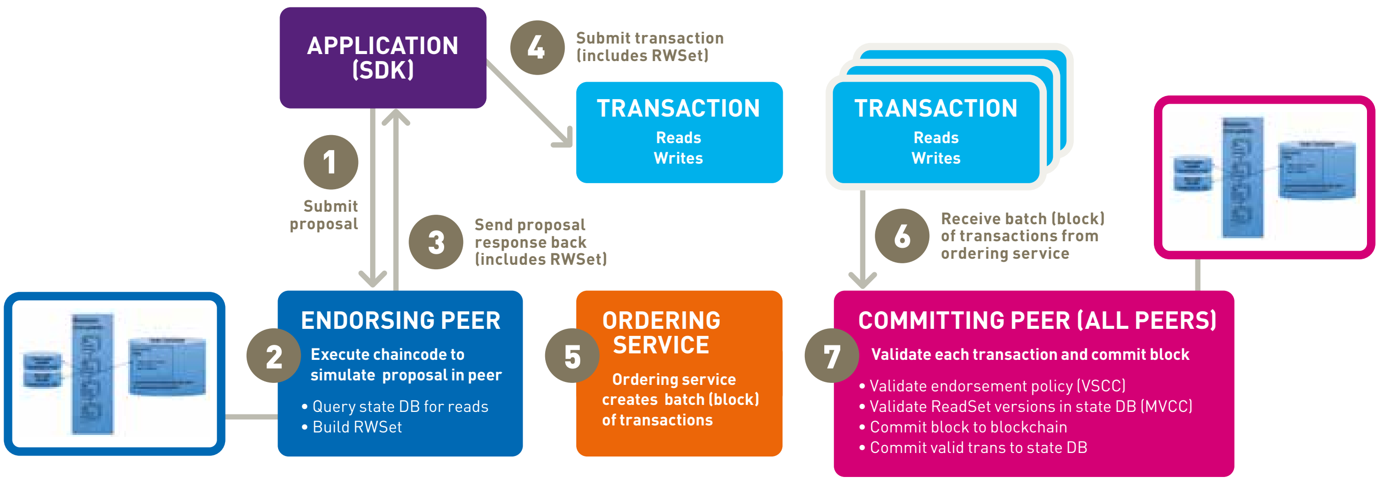
Solidity language:
https://solidity.readthedocs.io/en/develop/
Truffle framework:
http://truffleframework.com/docs/getting_started/project
Parity JSON RPC:
https://github.com/paritytech/parity/wiki/JSONRPC-parity-module

APPENDIX B - HYPERLEDGER – FABRIC

This section details the implementation specific details and notes related to Hyperledger Fabric v1.0. Fabric has a specific focus on permissioned blockchains for enterprise use at a large scale. The architecture of Fabric V1.0 is quite different to Parity in that consensus is separated from the ledger storage and there is a concept of 'channels', which are private ledgers between parties that need privacy. This design supports the key goals of Fabric - confidentiality, security and scalability.

The diagram below illustrates the basic flow in Fabric;

- An application invokes a smart contract (known as ChainCode in Fabric) on Endorser nodes. Consensus is achieved (or not) between multiple endorser nodes and a token passed back to the application
- The application can then submit this, or an ordering service, which will commit the data to the ledger.



APPLICATION (SDK)
4 Submit transaction (includes RWSet)
TRANSACTION Reads Writes
TRANSACTION Reads Writes
1 Submit proposal
3 Send proposal response back (includes RWSet)
6 Receive batch (block) of transactions from ordering service
ENDORSING PEER
2 Execute chaincode to simulate proposal in peer
• Query state DB for reads
• Build RWSet
ORDERING SERVICE
5 Ordering service creates batch (block) of transactions
COMMITTING PEER (ALL PEERS)
7 Validate each transaction and commit block
• Validate endorsement policy (VSCC)
• Validate ReadSet versions in state DB (MVCC)
• Commit block to blockchain
• Commit valid trans to state DB

Source:
https://www.ibm.com/developerworks/cloud/library/cl-top-technical-advantages-of-hyperledger-fabric-for-blockchain-networks/index.html

VM SYSTEM REQUIREMENTS

CPU
As with Ethereum, the system requirements are also modest. For FlightChain, each node is running on an AWS t2.small instance – and even then, the CPU consumption is generally less than 10%. Although it must be noted that the transaction throughput for FlightChain at this stage is < 5 per second.

Memory
Equally the memory requirements are quite low – the parity client consumes < 0.5 Gb. This will of course vary depending on the number of smart contracts deployed and the usage of those contracts, and the number of transactions recorded over time.

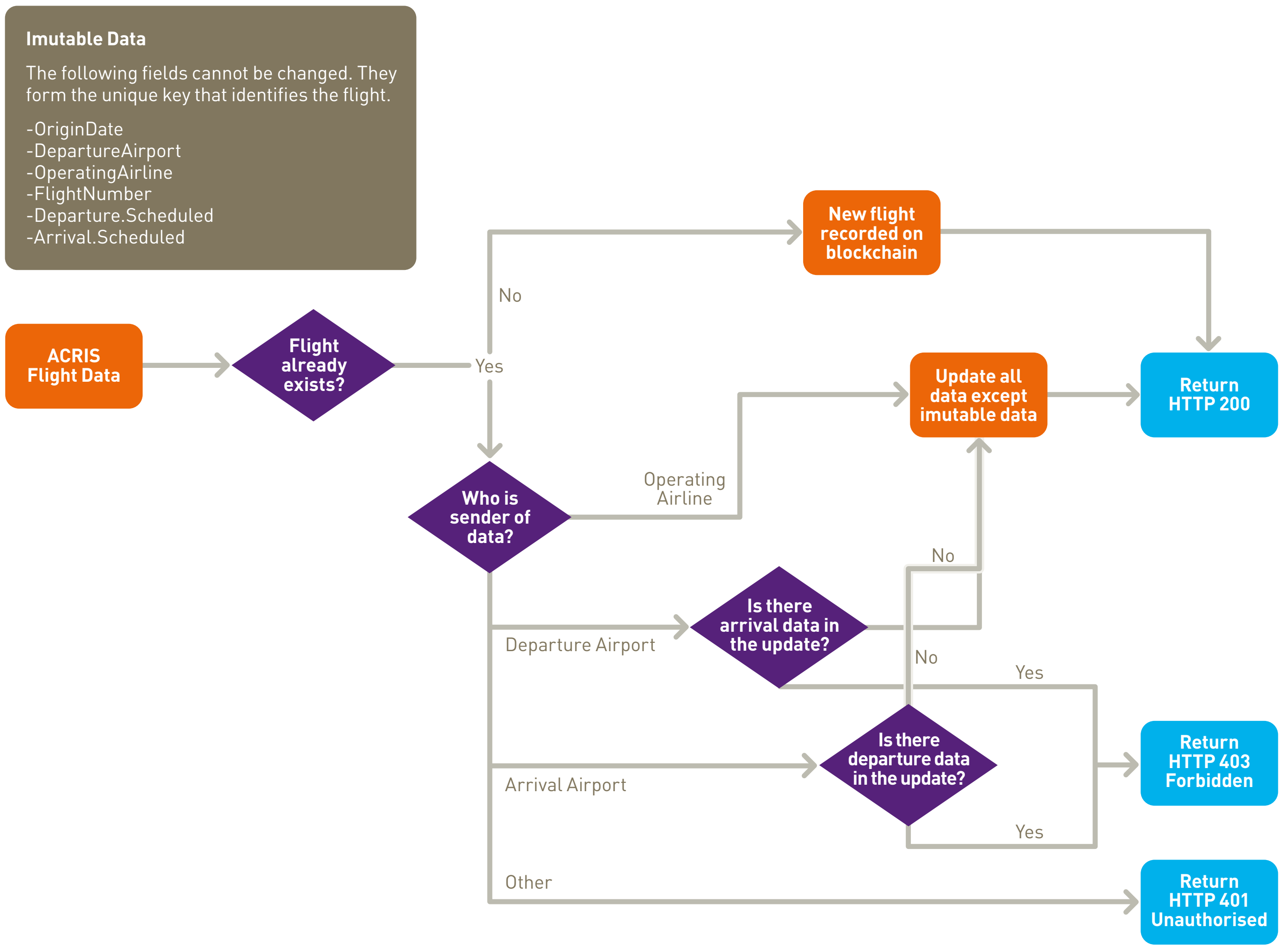
Disk Requirements
The disk space requirement has a direct correlation with the number of transactions (in FlightChain, the number of flight updates) - the more flight data recorded, the more space required. Disk utilization is efficient, and there isn't any significant overhead required by Parity beyond the data passed to the FlightChain contract. As an illustration, three months of flight updates for all flights into and out of LHR, GVA, and MIA (about 1.3m transactions) occupies approximately 11Gb.

SMART CONTRACT

Fabric Smart Contracts are known as chain code and have many similarities with Ethereum smart contact concepts. They can be written in 'go' and support for Java is in the pipeline. Chain code is deployed onto, and executed on, the Endorsing Nodes in a Fabric network. The chain code is executed across multiple endorsing nodes and when consensus is reached across these nodes the smart contract output can then be sent to the committing peer(s) and only at this point is the data stored on the network.

APPENDIX C – FLIGHTCHAIN SMART CONTRACT

The logic flow for the FlightChain smart contract is shown below. Note that the HTTP return codes indicate that there is a REST API wrapping the FlightChain smart contract.



Imutable Data
The following fields cannot be changed. They form the unique key that identifies the flight.
-OriginDate
-DepartureAirport
-OperatingAirline
-FlightNumber
-Departure.Scheduled
-Arrival.Scheduled
ACRIS Flight Data
Flight already exists?
No
New flight recorded on blockchain
Yes
Who is sender of data?
Operating Airline
Update all data except imutable data
Return HTTP 200
Departure Airport
Is there arrival data in the update?
No
No
Yes
Arrival Airport
Is there departure data in the update?
Yes
Return HTTP 403 Forbidden
Other
Return HTTP 401 Unauthorised

Create success. Together™

SITA AT A GLANCE

Easy air travel every step of the way.
Transforming air travel through technology for Airlines, at Airports, on Aircraft and at Borders.

- SITA's vision is: 'Easy air travel every step of the way'.
- Through information and communications technology, we help to make the end-to-end journey easier for passengers – from pre-travel, check-in and baggage processing, to boarding, border control and inflight connectivity.
- We work with about 400 air transport industry members and 2,800 customers in over 200 countries and territories. Almost every airline and airport in the world does business with SITA.
- Our customers include airlines, airports, GDSs and governments.
- Created and owned 100% by air transport, SITA is the community's dedicated partner for IT and communications, uniquely able to respond to community needs and issues.
- We innovate and develop collaboratively with our air transport customers, industry bodies and partners. Our portfolio and strategic direction are driven by the community, through the SITA Board and Council, comprising air transport industry members the world over.
- We provide services over the world's most extensive communications network. It's the vital asset that keeps the global air transport industry connected.
- With a customer service team of over 2,000 people around the world, we invest significantly in achieving best-in-class customer service, providing 24/7 integrated local and global support for our services.
- Our annual Air Transport and Passenger IT Trends Surveys for airlines, airports and passengers are industry-renowned, as is our Baggage Report.
- In 2016, we had consolidated revenues of US$ 1.5 billion.

For further information, please visit **www.sita.aero**

For further information, please contact SITA by telephone or e-mail:

Americas
+1 770 850 4500
info.amer@sita.aero

Asia Pacific
+65 6545 3711
info.apac@sita.aero

Europe
+41 22 747 6000
info.euro@sita.aero

Middle East, India & Africa
+961 1 637300
info.meia@sita.aero

Follow us on **www.sita.aero/socialhub**