# Blockchain application within a multi-sensor satellite architecture

**Rohit Mital**
*SGT KBRWyle*
**Jack de La Beaujardiere**
*U. of Maryland, College Park*
**Rohan Mital**
*U. of Colorado, Colorado Springs*
**Marge Cole**
*NASA ESTO and SGT KBRWyle*
**Charles Norton**
*NASA Headquarters*

## ABSTRACT

With the thrust towards multi-sensor satellite architectures for earth and space exploration, such as constellations and swarms, new technologies are required to enable the transition to this future capability. One of the areas of interest is establishing secure, efficient and prioritized data and command communication pathways among ground and space-based sources for such systems.

This paper presents early research results on the potential role, capabilities and value of blockchain usage within constellation and swarm satellite architectures. It demonstrates the use of blockchain's smart contract and distributed ledger capabilities for secure and prioritized multi-sensor satellite collaborative data exchanges, as well as the logging and tracking of command and control events. Adapting and utilizing this emerging technology will aid in addressing technology gaps expected from future constellation flight architectures, such as managing collective computational operations (correlation), dynamic and autonomous observation planning, time-critical events, and provenance tied to ground and space-based autonomous operations and control recordkeeping. In this scenario blockchain is applied in encrypted command transmittal to multiple, yet specific, entities enabling acknowledgement transmittals, performance scalability, and automatic event-based triggering.

## INTRODUCTION

The blockchain testbed project was started with the intent to research and experiment with blockchain technology and to assess whether this technology could be leveraged as an approach for some general use cases: secure and prioritized multi-sensor satellite collaborative data exchanges and the logging and tracking of command and control events. Several experiments have been run throughout the course of the project, revealing some distinct advantages as well as limitations of this technology as it applies to multi-sensor satellite architecture. In general, some of the features that have been demonstrated with blockchain technology that are of interest to NASA include:

- Distributed, accurate, and secure logging and tracking of command and control events across a network of ground stations
- Autonomous control of satellite constellations, with or without satellite-to-satellite communication
- Secure and prioritized data and command communication among ground and space-based sources

While there are potential limitations of this technology that have yet to be fully explored in a realistic, simulated hardware environment, the potential for this technology to support secure, trusted, and autonomous future satellite operations management is palpable. This paper aims to describe the work that has been achieved so far in researching and experimenting with blockchain technology in this context.

## DEFINITION OF TERMS

**Blockchain:** Blockchain can refer to the distributed ledger, in which blocks of data are chained together by cryptographic hashing; or to the network of participants who share copies of the distributed ledger and interact with transactions.
**Ledger:** In blockchain technology, the immutable, cryptographically secured record of events that is shared by all the participants in the blockchain.
**Transaction:** An entry onto the blockchain's ledger. It does not necessarily represent a financial exchange, it can be any data.
**Smart contract:** An extension of blockchain technology where transactions are associated with code that runs on the blockchain network. This gives the transactions automated functionality, and allows their outcome to depend on the logic dictated by the code.
**Chaincode:** Another name for a smart contract, used by the Hyperledger Fabric blockchain implementation.
**Node:** A computer that is connected to the blockchain network.
**Constellation:** A group of satellites working in concert.
**SQL:** Structured Query Language, a programming language used to interact with databases.
**JSON:** JavaScript Object Notation, a human- and machine-readable language for encoding data.
**REST server:** An interface used to interact with applications over the web.
**API:** Application Programming Interface, an interface for interacting with an application.

## TOOLS AND TECHNOLOGIES

### Blockchain

#### What is Blockchain?

Imagine a network consisting of hundreds of computers, each of which participate by sending and receiving data across the network. These transfers of data are called "transactions", and the transactions themselves contain information that is inherently valuable. In the Bitcoin world, for example, this would be cryptocurrency, but this data can be anything; asset data stored in key/value pairs, personally identifiable information, or even satellite commands. Now imagine that every transaction is stored in a ledger, and that this ledger is not contained on a single server but is continuously updated and duplicated across every computer in the network. Every participant in the network has a copy of this distributed ledger, which essentially describes the history of events on the network. This is the core of blockchain technology; *an immutable and distributed ledger of transactions, which are cryptographically secured, stored in data structures called "blocks."* With a basic understanding of blockchain technology in place, we can break down this definition into finer detail.

#### Breakdown of the Basic Components

Due to its distributed nature, the blockchain has no single point of failure. In addition, the entire network must come to a consensus prior to a transaction being appended to the ledger. Transactions cannot be altered or removed from the ledger because each block is cryptographically linked to the one prior. Altering or removing any single transaction would alter the entire chain; for one to get the entire network to come to a consensus over a fraudulent transaction would mean possessing most of the computing power in the network or taking over every machine in the network that is validating transactions. This fault tolerance and resiliency is referred to as "immutability" in the blockchain world and it ensures that the ledger of transactions remains accurate. Fig. 1 below provides a simplified visual representation of the distributed and automated nature of blockchain technology in the context of satellite/ground station communication. Notice how each peer has a copy of both the chaincode as well as the ledger. Notice also how even if one or two nodes are compromised in some way, data can still be replicated across any other available pathways.

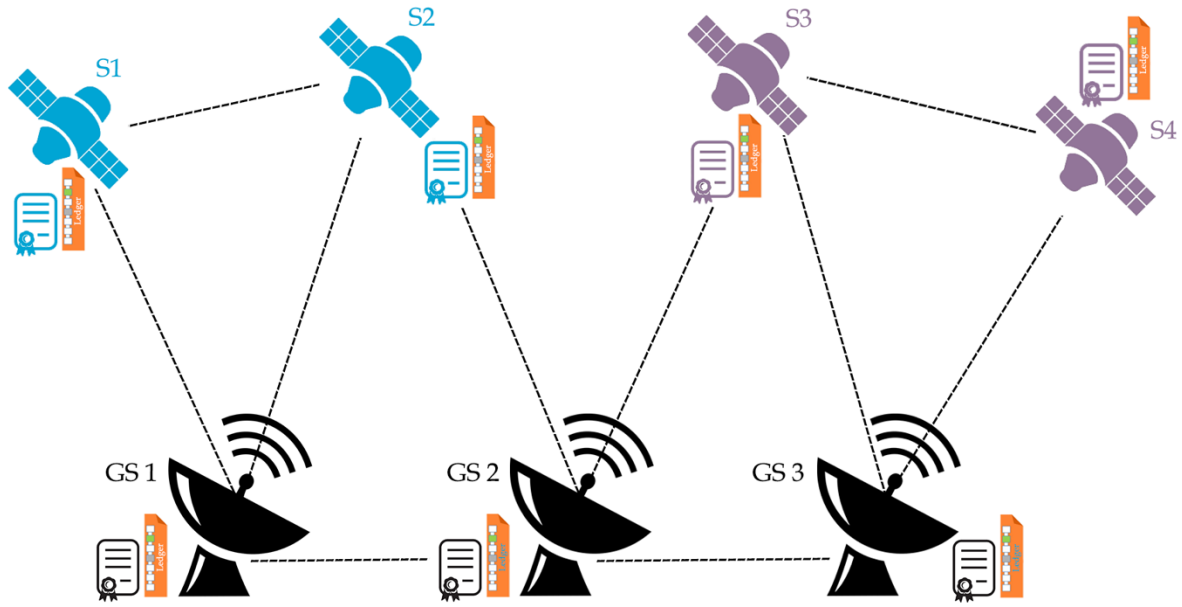


Fig. 1. Conceptual view of a blockchain network

We call each participant in a blockchain network a "peer." Peers can play different roles in the network depending on the implementation, but a common thread between all implementations is that these peers run "nodes" which contain both the distributed ledger as well as the logic that defines how different transactions should be processed. This logic is commonly referred to as a "smart contract" or "chaincode." Chaincode provides another major advantage to blockchain technology beyond distribution: automation. Using predefined code (typically written in Go or NodeJS, though this also varies by implementation), network administrators can define exactly how a transaction should be handled whenever it is sent or received and have that logic run on the network itself. To use an example in the context of satellite commanding, let's say that we have a situation where multiple ground stations are attempting to send commands to the same satellite simultaneously; we want to execute both commands, but one should take priority over the other. If these commands have been defined in chaincode as transactions, that prioritization logic can be handled autonomously by the chaincode running on the blockchain network. The commands themselves and any results will then be recorded and sent as transactions to the rest of the network's distributed ledgers. This is a simple example of how one might make use of this distributed and autonomous blockchain technology in a meaningful way, though the possibilities are extensive.

**Types of Blockchains**

Different blockchain implementations come to consensus over the accuracy of the distributed ledger in different ways. This is largely determined by the type of blockchain being implemented. There are three main types of blockchains: Public, Private, and Consortium, also called permissioned or community blockchains.

In a public blockchain, anyone can join the network and begin sending and receiving transactions as soon as they have downloaded a full node containing a copy of the ledger and chaincode. These implementations currently use what is called the "Proof of Work" (PoW) consensus mechanism, which was designed to be resilient and entirely trustless. This form of consensus requires peers on the network to solve mathematical problems which are computationally difficult, but easy to validate. Every peer on the network must validate and agree on any given solution before a block is added to the ledger. As one might imagine, while the system is highly resilient to attack and tampering, it is not very efficient and is computationally expensive; at the time of publication, it takes anywhere from 7-20 minutes for a transaction to be added to the ledger on the largest public networks, such as Bitcoin and the public Ethereum blockchain.

A private blockchain, by contrast, more closely resembles a centralized system. In a private implementation, only explicitly authorized peers can participate in the network. Administrators can also define whether certain participants have access to reading/sending different transaction types as well as which peers on the network will act

as “validator nodes”, which validate the transactions. This method of defining exactly how many peers will act as validators is referred to as “Practical Byzantine Fault Tolerance (PBFT)”. While PBFT is less resilient than the PoW mechanism used in public chains, it is significantly faster and does still provide more or less fault tolerance depending on how many trusted validator nodes are defined on the network.

A consortium, or permissioned blockchain, is the same as a private implementation but with more modularity built in. For example, participants can be allowed to join the network based on definitions and policies in code, rather than having to be explicitly authorized by an administrator. This extends to participants’ ability to read and send transactions; access to network resources can be defined in code, allowing the network to moderate peers and network activity autonomously.

## Hyperledger

### Hyperledger Fabric

Hyperledger Fabric [1] is an open-source implementation of a consortium blockchain developed by the Linux Foundation’s Hyperledger organization. It can be used to create permissioned blockchain networks with multiple channels of communication, and offers modular, customizable options for securing and managing the network. The chaincode on a Fabric network is written in a lightened version of Node.JS [2], and is executed by the peers within container environments, meaning that many hardware and operating system versions can be supported.

### Hyperledger Composer

Hyperledger Composer [3] is an open-source tool that is being developed by the Linux Foundation’s Hyperledger organization to simplify interacting with and creating blockchain networks. Composer offers tools to automate the deployment of a Fabric network and other accompanying tools, such as a REST server for interacting with the blockchain. It offers a web browser interface for editing the Fabric network’s configurations and chaincode, and can deploy those changes directly to the network.

| Date, Time | Entry Type | Participant | |
|---|---|---|---|
| 2018-08-06, 12:45:17 | SendCommand | US_3 (GroundStation) | view record |
| 2018-08-06, 12:44:54 | ActivateCurrentIdentity | none | view record |
| 2018-08-06, 12:44:48 | IssueIdentity | admin (NetworkAdmin) | view record |
| 2018-08-06, 12:44:34 | AddParticipant | admin (NetworkAdmin) | view record |
| 2018-08-06, 12:43:18 | ActivateCurrentIdentity | none | view record |
| 2018-08-06, 12:42:10 | IssueIdentity | admin (NetworkAdmin) | view record |
| 2018-08-06, 12:41:49 | IssueIdentity | admin (NetworkAdmin) | view record |
| 2018-08-06, 12:39:56 | AddParticipant | admin (NetworkAdmin) | view record |
| 2018-08-06, 12:39:33 | AddParticipant | admin (NetworkAdmin) | view record |

Fig. 2. A view of the ledger in Hyperledger Composer

## Ethereum

Ethereum is an open-source, public blockchain platform created by the Ethereum Foundation. For the purposes of this project, a private Ethereum blockchain was created. The Ethereum platform is lightweight and versatile but does not come with many tools out of the box and is command-line only. Ethereum smart contracts are written in Solidity, a language that was developed specifically for use in blockchain smart contracts.

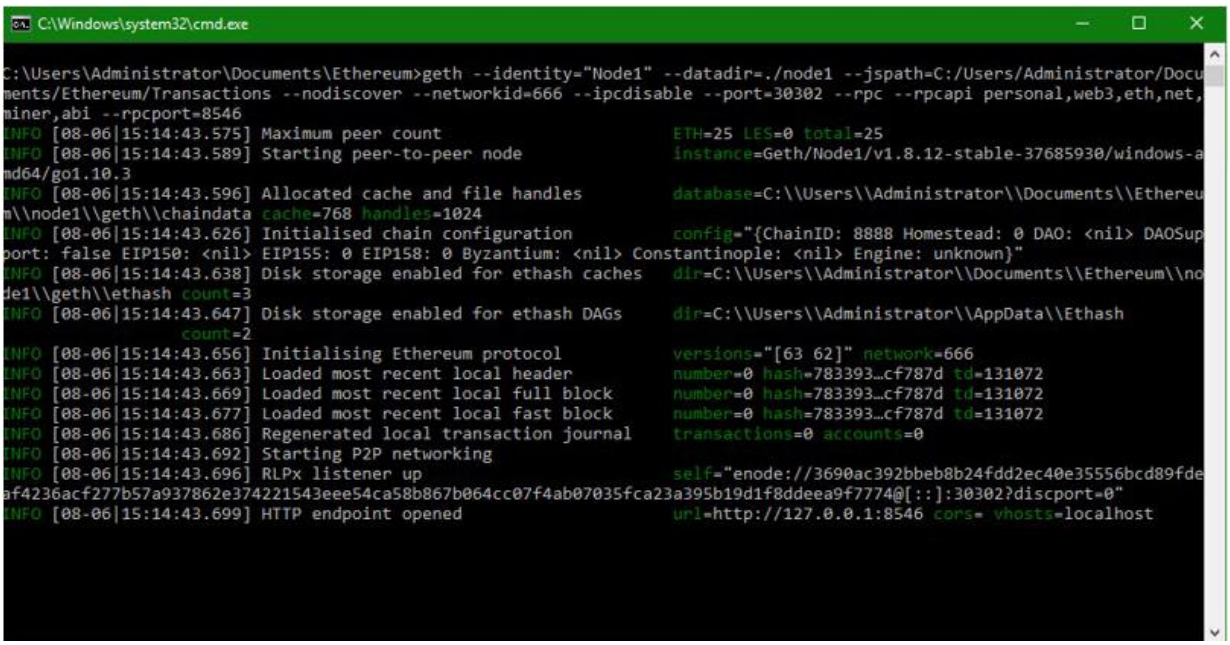


Fig. 3. The Geth command line interface

## Systems Tool Kit (STK)

### What is STK?

The Systems Tool Kit (STK) [4] is a powerful software tool provided by Analytical Graphics Inc (AGI) that allows users to model and simulate physics-based scenarios involving objects from land, sea, air, or space. It is primarily used to evaluate system performance in real or simulated time. This project relied heavily on STK in creating and running satellite simulation scenarios for use in various blockchain experiments, specifically to accomplish the following tasks:

- Setting up scenarios involving ground stations and satellites, where these objects are given user-defined locations/orbits as well as attributes (including sensor type/range, mass, tilt, etc.).

- Accurately modeling the scenarios while simultaneously collecting a variety of data points on the objects (stations and satellites) involved.

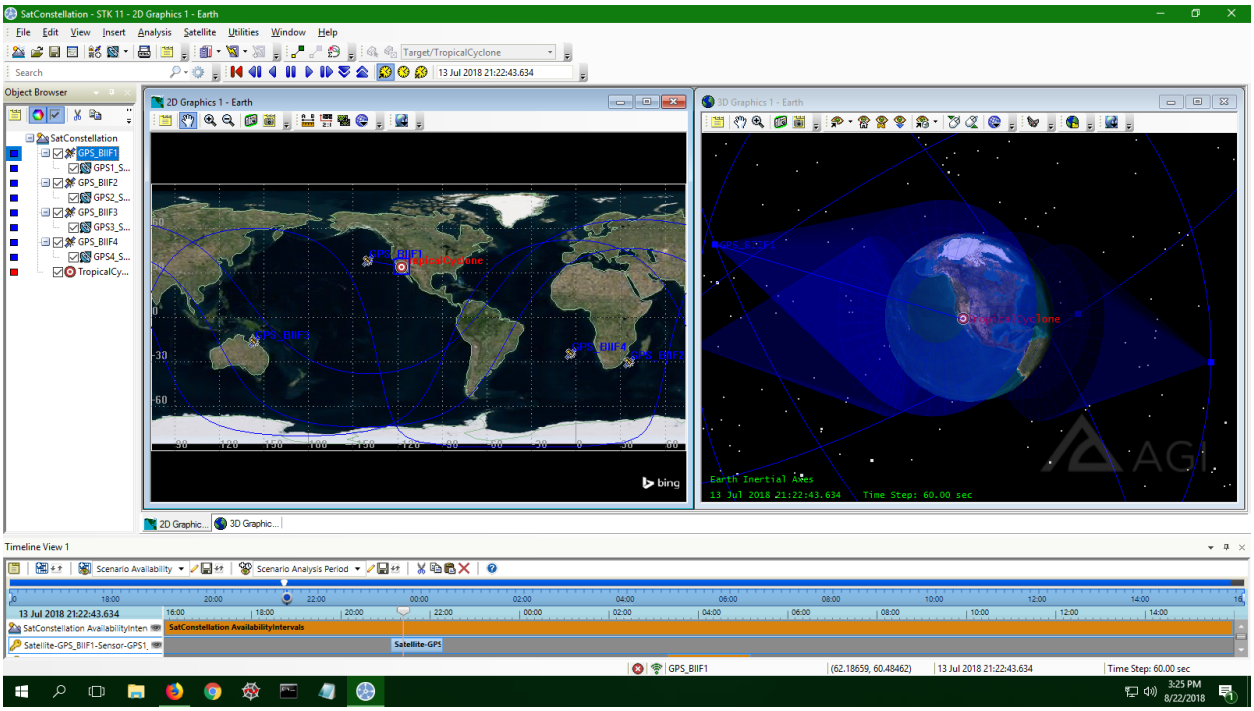

Fig. 4. STK main window view

**Connect/Object Model Libraries**

The Connect library and Object Model Library (OML) are APIs provided by STK that allow users to send information to and collect data from an STK scenario. They were key components in integrating STK with the blockchain.

The Connect library can be used to connect to STK over a TCP/IP connection and to send commands written in STK's own "Connect" language. These commands can be used to change aspects of a scenario as it runs. For example, creating new objects, changing attributes of objects, changing the speed at which the scenario plays out, etc.

OML is an STK API that can be implemented in a variety of high-level programming languages. It provides many useful objects, such as data providers, for collecting and processing information from an STK scenario. Information on specific objects (such as satellite location or a ground station's sensor range) or the scenario itself (such as the length of the scenario or names of objects in the scenario) can all be retrieved using OML.

**Integrating with the Blockchain**

Fig. 5 below provides a high-level overview of the various tools and libraries used when integrating STK with the blockchain network.

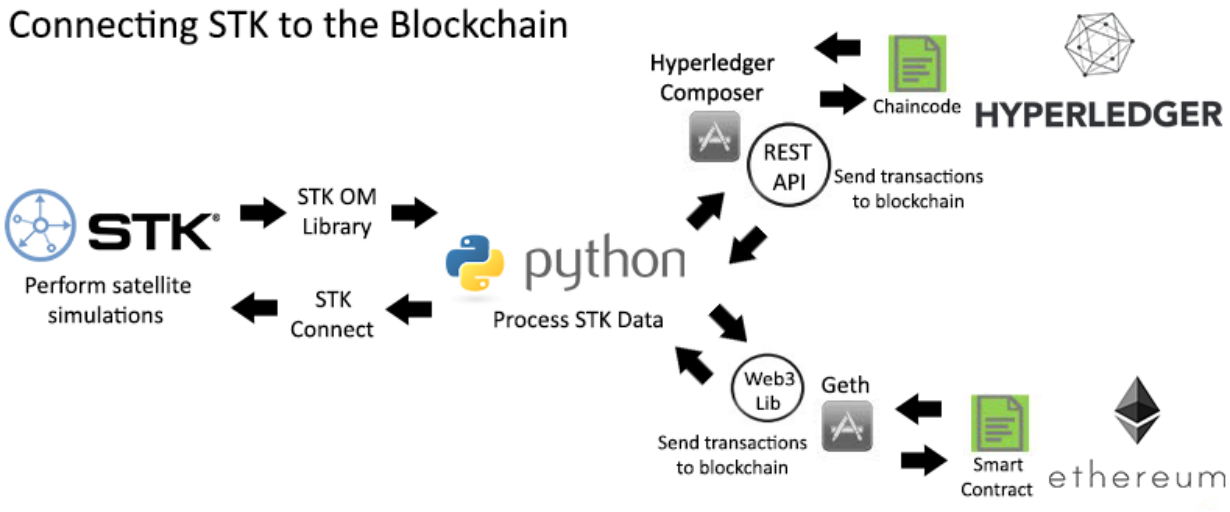


Fig. 5. Connecting STK to the Blockchain

The Connect library and OML were imported and used in Python [5] to change scenarios in real-time as necessary and to pull data on ground stations and satellites from STK. This data was manipulated and processed to prepare it to send to the blockchain network.

In the case of Hyperledger Fabric, we interacted with the network using the Hyperledger Composer application. Using Composer's REST API, STK data was sent in JSON format using a "post" request in Python. Composer subsequently ran the transactions we had defined in chaincode using this data as input and the data processed by the blockchain was then pulled using a "get" request. The processed data can then be passed back to STK where necessary using STK Connect, and this process continues until the experiment completes.

The Ethereum implementation uses an application called Geth [6] to interact with Ethereum nodes. The Web3.JS [7] API, which is used by the Geth Javascript console to interact with the network, is available as a library for Python. Using this library, it was possible to connect to an Ethereum node and send transactions to the Ethereum smart contract (recall that "smart contract" is another name for "chaincode"). The data processed by the smart contract can then be queried, sent back to STK as necessary using STK Connect, and this process continues until the experiment completes.

## Amazon Web Services (AWS)

### What is AWS?

Amazon Web Services [8] (AWS) is a collection of tools provided by Amazon that allow users to access on-demand cloud computing platforms. They provide a collection of virtual computers that are available all the time and feature most of the attributes of real, modern-day computers, such as CPUs/GPUs for processing, different operating systems, web servers, databases, etc.

We used several of the resources offered by AWS in this project to run our experiments through shared cloud servers. The two major resources used in this project were the Elastic Compute Cloud (EC2) and CloudFormation.

### EC2

Amazon Elastic Compute Cloud (EC2) is a web service that provides compute power in the AWS cloud. For this project, several instances of EC2 were created to host servers for running STK, Hyperledger Composer, and a Hyperledger Fabric blockchain network. These servers included:

- A Windows 64-bit Server instance, used to run the STK software and the Python scripts used in each experiment. This server also ran the Ethereum nodes used in experiments that tested the Ethereum blockchain implementation.

- Several Ubuntu Server instances, used to host the Hyperledger Fabric blockchain networks and Hyperledger Composer. When running experiments on the Windows server, PuTTY was used to connect to these instances when sending/receiving data to/from the blockchain.

**CloudFormation**

Amazon CloudFormation allows users to easily create and manage a collection of AWS resources using templates. The templates describe which resources are needed, along with any dependencies or runtime parameters required to run the necessary application. CloudFormation then takes care of launching and provisioning these resources and dependencies.

This project used CloudFormation to launch templates for both the Hyperledger Fabric as well as Ethereum blockchain networks, both of which were provided as samples by Amazon. This allowed us to run the blockchain networks on AWS, making it easier and more efficient to interact with them from our Windows Server, which was also running on AWS. The templates required a variety of prerequisite requirements to be fulfilled, such as adding user permissions, creating an EC2 instance with a policy attached that provides permissions for various Amazon resources, providing an Amazon Virtual Private Cloud (VPC) network with custom settings, etc. These prerequisites were properly configured per Amazon's documentation prior to launching the templates using CloudFormation.

## USE CASES

This section details the experiments that were designed to provide examples for the project's use cases. Most of the experiments described here are a culmination of other, smaller experiments into a single project that demonstrates multiple features at once. A complete list of the experiments that were run, along with a short description of each, can be found in Table 1 below.

Table 1. Descriptions of experiments conducted

| Experiment Name | Description |
|---|---|
| **Contention** | Multiple stations are sending commands to multiple satellites. When there is contention over which satellite should respond, the blockchain handles the logic autonomously and sends the command to the nearest satellite. |
| **Distributed & Autonomous Observation** | A satellite detects a tropical cyclone forming off the California coast. It notifies the blockchain, which utilizes its distributed and autonomous capabilities to proactively alert other satellites so that they can respond appropriately. |
| **Encryption** | Commands contained within transactions are encrypted before they are sent to the ledger, so they can only be read by participants with the decryption key. Timestamps and other metadata remain open for other participants to view. |
| **Event-driven** | When commands are executed within the chaincode and by satellites, the ledger entry for the command is updated with an Event. The Event is caught by participants listening for it and trigger a transaction on a second blockchain ledger for record-keeping. |
| **SQL Integration** | Information is pulled from the blockchain distributed ledger and stored in a centralized SQL Server database. |
| **Chain Trimming** | Once the blockchain exceeds a specified size, data in the distributed ledger is archived in a text file. The old blockchain is deleted and a new chain is started. |

| | |
|---|---|
| **Access Channels** | Ground stations from different countries are put in separate access channels, giving them access only to specific network resources defined by the admin. |

Table 2 maps all the features and use cases this project aimed to demonstrate to the ways they were implemented. Some were already provided by the blockchain platforms used, while others needed external libraries or coding to be implemented. The features and use cases in column 1 are implemented either directly by one of the blockchain platforms used, in which case the specific feature of the implementation is listed in column 3 or 4, or the feature is implemented in one of the experiments, in which case the experiment name is listed in column 2.

Table 2. Implementations of features and use cases

| Features and Use Cases | Experiments | Hyperledger Fabric and Composer | Ethereum* |
|---|---|---|---|
| **Prioritized Commands/ Contention** | Contention | | |
| **Collaborative data exchanges** | Distributed & Autonomous Observation | | |
| **Log and track C&C events** | (Built-in blockchain functionality) | Historian/Ledger | Ledger |
| **Dynamic and autonomous observation planning** | Distributed & Autonomous Observation | | |
| **Command provenance and recordkeeping** | Contention | Historian/Transaction metadata | Ledger |
| **Encrypted command transmittal** | Encryption | Encryption library | |
| **Command transmittal to multiple, yet specific, entities** | Contention, Distributed & Autonomous Observation | Access Control Language (ACL) | |
| **Acknowledgement transmittals** | Event-Driven, Contention | Events | |
| **Automatic event-based triggering** | Event-Driven | | |
| **Performance scalability** | SQL Integration, Chain Trimming | | |
| **Satellites in a string commanding successors** | Distributed & Autonomous Observation | | |
| **Data downlink in multiple parts** | SQL Integration | | |

*Ethereum is a modular framework that can support many of these features, but they are not provided by default

## Distributed and Autonomous Observation

### Introduction

The goal of this experiment was to test the autonomous, distributed capabilities of blockchain technology and assess their usefulness to NASA. The experiment provides a solid example for the following use cases:

- Collaborative data exchanges
- Dynamic and autonomous observation planning
- Command provenance and recordkeeping
- Command transmittal to multiple, yet specific, entities
- Satellites in a string commanding successors

**Scenario**

We set up a scenario in STK with four satellites and a target location on the coast of California which was labeled "Tropical Cyclone". The scenario takes place over the course of a full day in STK time, and the satellite orbits were set up such that three of the satellites would have about 2 hours of access time to the Tropical Cyclone (access time refers to time periods in which the target location is within the satellite's field of view). One satellite would not have any access time to the target location. Access time to the Tropical Cyclone in this experiment is dependent upon the sensor range of the satellites; a greater sensor range expands the satellite's field of view in STK, consequently increasing the amount of time the target location is in view.

The intended outcome of this scenario was to have a satellite detect the Tropical Cyclone and notify the blockchain network. That information could then be distributed autonomously to other peers on the network, including the other satellites, allowing them to proactively increase or decrease their sensor range once they meet certain requirements specified in the network's chaincode.

**Approach**

To accomplish this, we set up a blockchain network to include the four satellites as well as an arbitrary number of ground stations. The ground stations are represented by a Python script, which sends/receives data to/from STK and a Hyperledger Fabric blockchain network. The satellites send the following information to the blockchain network every STK hour: satellite name, latitude, longitude, current time, and status (where "status" is data that lets the blockchain know whether the satellite has detected any meteorological phenomena of interest). The first satellite to detect the Tropical Cyclone sends a transaction to the blockchain network with this information. The chaincode will then store this information in state variables so that it knows the following information: whether meteorological phenomena were detected, which satellite detected it, at what time, and what the location of the phenomenon is.

At this point, when the other satellites send their information to the blockchain, the chaincode will check its state variables to see whether the Tropical Cyclone was detected. If so, it will run an additional check to see whether the sending satellite meets the requirements to increase its sensor range. In our experiment, the requirements were simply that the satellite is within 40 degrees latitude and longitude of the detected phenomenon, though these requirements are arbitrary and can be defined in code as desired.

**Conclusions**

The outcome was as intended at the outset of the experiment. The blockchain network was able to autonomously control the sensor range of the satellites by processing and reacting to information sent to it by satellites on the network. Information was replicated as expected to all participants in the network and stored accurately in the distributed ledger. This all occurred autonomously through chaincode installed on the network.

The advantages to using blockchain in this case are:

- The ability to control the satellites autonomously without satellite-to-satellite communication. If we assume the satellites are not running a full blockchain node (which is reasonable considering their limited memory and storage capacity), blockchain will still automatically replicate information by moving through any available pathways. For example, if the ground station network is connected, as soon as information is sent

to the blockchain, every ground station will have that data stored in the ledger and the appropriate stations can communicate with the other satellites.

- An accurate and immutable record of events over the network that is autonomously duplicated and stored in the distributed ledger.

- A trustless system. Stations that we might not want to view the data can still be used as peers to get the information where it needs to go. Access control capabilities prevent these stations from viewing or tampering with data they are not authorized to interact with.

Limitations that require further experimentation in a simulated hardware environment include:

- Time for validation of transactions over the network. We do not want to miss windows of opportunity when making observations.

## Access Channels

### Introduction

The goal of this experiment was to test the possibility of having separate, secured channels of communication on a blockchain network. The experiments demonstrated the following use cases:

- Encrypted command transmittal
- Command transmittal to multiple, yet specific, entities
- Command provenance and recordkeeping

### Scenario

A scenario was created with two ground stations, one based in the U.S. and one based overseas. Both stations are on the same network and share the same ledger, but we want to limit the visibility of certain transaction to only U.S. stations. This represents the fact that certain satellites are of a more sensitive nature, such as those operated by the military. While the communications to and from those satellites need to remain confidential, it would still be advantageous to be able to use the full blockchain network for uplinks and downlinks. For this reason, the use of access channels on a blockchain was investigated.

### Approach

To create these separate access channels, the built-in Access Control Language (ACL) functionality was used to define to define different access rules for the U.S. and non-U.S. stations. The ACL allowed us to define rules based on the IDs assigned to ground stations: if the prefix of the ID is “US”, the ground station is granted full access to all the resources and transactions on the network. If, however, the ground station has a different prefix to its ID, a participant connecting from that station is limited in their ability to send sensitive transactions, as well as to view those transactions on the ledger

With these ACL rules in place, we connected to the blockchain as a participant from the U.S. ground station. We were able to send the restricted transactions and retrieve them from the ledger. We also were able to see both ground stations on the network and view all the information associated with the other ground station. We then disconnected from the blockchain network and reconnected as a participant from the non-U.S. ground station. When attempting to submit the same transaction type as before, the peers in the network would not validate the transaction. Instead, an error message was returned informing us that we did not have the necessary permissions to send that type of transaction. When viewing the ledger, we could not see the transactions that had been sent when we were acting as the other ground station. We also could not see the U.S. ground station, or any information associated with it in the list of stations.

### Conclusions

The experiment successfully showed that separate access channels can be created on a permissioned blockchain. It is important to note that the ledger entries and resources were not simply made invisible to the non-U.S. ground station, but cryptographically obfuscated from them. A participant connecting from that ground station would be able to find the block containing the transaction on the ledger, but the transaction and its contents would be encrypted and unreadable to them. Since both stations are on the same network, they would both have a record of the transactions on their ledgers. This is important, because it means that the ground station that does not have permission to view the transaction could still be used as a point of connection for a satellite to pass the command contained within the transaction to it.

Note that, as with many aspects of a permissioned blockchain, different channels of communication that involve stations or participants being classified into different levels of access rely on the network Administrators to properly make those classifications when granting the station or participant access to the network. Assuming this is done correctly, there is no way for a participant in the network to escalate their privileges without invalidating their connection to the blockchain.

The advantages to using blockchain technology in this case are:

- The ability to make secured communications with a satellite through an untrusted ground station
- The ability to have separate channels of communication while keeping as many points of contact to satellites as possible

Limitations that require further experimentation:

- Verify that satellites can still decrypt the communication if it is relayed from a station that cannot decrypt it

## Integrating with Older Technology

### Introduction

The goal of this experiment was to both demonstrate the integration of blockchain with older technology as well as to test the performance scalability of blockchain technology. The motivations behind this experiment were twofold:

- Even if blockchain technology were to be adopted, NASA (and many other businesses, for that matter) may still need to integrate with older technology (for example, a centralized database like SQL)

- Performance across a large network, as well as limitations of satellite storage space and memory, are concerns that need to be addressed

This experiment demonstrated the following use cases

- Performance Scalability
- Data Downlink in Multiple Parts

The experiment was split into two parts: “SQL Integration” and “Archiving Old Blockchain Data.” These parts are described in detail below.

### SQL Integration

The aim of this experiment was to retrieve data from the blockchain network’s distributed ledger and store it in a centralized database. Because the experiments were run on a Windows Server instance, Microsoft SQL Server was chosen as the test system. Two main question were considered when designing and running the test:

- Can transaction data in the distributed ledger be parsed and stored in a SQL database in a consistent, clear, and meaningful way?

- How fast and efficiently can data be retrieved from the distributed ledger

Transaction information in both the Hyperledger Fabric and Ethereum blockchain ledgers is stored in JSON key/value pairs. Some key/value pairs, such as the timestamp and the transaction's unique identifier (transactionID), are system-defined and are common to all transactions. All other keys/values are user-defined.
Using the json [9] and pyodbc [10] Python libraries, it was possible to parse the transaction data and send it to the Microsoft SQL Server database. Each table in the database was named after a transaction type, which in the case of Hyperledger Composer is defined by a "class" key in every transaction. The transactionID was used when looping through the ledger to determine whether the transaction had already been added to the database, as this value is unique to every transaction. Finally, other keys contained within a transaction were assigned as columns to each table. In this way, we were able to successfully store every transaction as well as the contained data in an easy to understand and organized fashion inside of a SQL database.

Performance was good; thousands of transactions could be retrieved and stored in the SQL database in a matter of seconds. Of course, testing was run in an idealized environment, and a real-life scenario would be significantly slower. Another important point to note is that the querying capabilities of blockchain vary by implementation. For example, our simple Hyperledger Fabric network ledger is just a large list of transactions containing data, with no relational database capabilities. Therefore, even if a specific transaction id was known, the ledger still needed to be looped through if we wanted to add new transactions to the SQL database. However, some implementations store blocks data in raw format, using a leveldb or couchdb index, for example, to make lookups faster. This aspect of the technology is still being worked on and improved by many developers worldwide who are contributing to the open-source implementations.

**Archiving Old Blockchain Data**

As a follow-up test to the SQL integration experiment, we wanted to know if, after storing data in a centralized database, the old blockchain data could be deleted before starting a new chain and repeating the process over again. The blockchain ledger size constantly increases, which could prove to be problematic and inefficient in the long-term because of storage limitations and the inefficiency of retrieving ledger information as more data is appended. This experiment was run on a couple of Ethereum nodes using the Proof of Work consensus mechanism, as this was easy to implement, and the project was short on time when testing on this experiment began. The desired outcome was to achieve the following:

- Start up a new node connected to the same blockchain that can process and validate the same transactions
- Archive the pertinent data stored in the ledger
- Stop the old node (ledger data cannot be deleted while a node is running; attempting to force delete sometimes resulted in unexpected behavior, like a node having to be restarted before it would continue to process transactions)
- Delete the old ledger data to free up storage space

The full process would involve starting a new node (Node 2), waiting for it to begin processing the same transactions as the old node (Node 1), archiving data in the ledger stored on Node 1, stopping Node 1, and finally deleting the old ledger data. In this way, it was thought that downtime could be completely avoided, because Node 2 would already be processing transactions while Node 1's ledger was archived and deleted. However, of the solutions tested, it was found that achieving the desired outcome was not possible. The PoW consensus mechanism proved to be the main issue in these tests, for the following reasons:

- It was not possible to connect a new node to the same running Python script using the same chaincode without stopping the currently running node. This was a Web3 API limitation; only one instance could be open at once per script.

- Trying to run a new node that was connected to an entirely different Python script resulted in unexpected behavior. Sometimes the Python script attached to the new node would throw errors saying that the chaincode was not installed. Other times, a node would simply stop processing transactions until it was restarted. The results varied, but never produced the desired outcome.

The final working experiment that was set up worked by having Node 2 wait for Node 1 to completely stop before beginning to process new transactions. In this way, we could successfully archive old information, delete the old ledger data, and begin processing and storing new transactions without any issues. The limitation, of course, was the downtime between stopping Node 1 and deleting its stored data and processing transactions on Node 2.
It is important to note that the issues encountered while running this test were due to the inherent limitations of the implementation used, however Ethereum is highly modular and finding a workaround is possible given more time. In addition, a Hyperledger Fabric implementation using the PBFT consensus mechanism was not tested.

**Conclusions**

In conclusion, we believe that integrating blockchain technology with older technology, such as SQL, will not prove to be difficult. At its core, blockchain is not an entirely new technology; it is an amalgamation of existing technologies, such as private key cryptography, peer-to-peer networking, and the protocol governing consensus that varies between implementations. The blockchain ledger stores data using key/value pairs, database-like capabilities are possible in certain implementations, and there are many libraries currently available that allow one to interact with the blockchain in high-level languages like Python. Some blockchain implementations, like Ethereum, are also highly modular, allowing developers to create their own blockchain implementations and applications from scratch, which presents some exciting possibilities.

While we were not able to thoroughly test everything we had hoped to in these particular experiments, there were a couple areas of interest uncovered that require further research and analysis. These are listed below:

- Blockchain data can be stored in a variety of ways; some implementations are simply a big list of transactions, others use databases like LevelDB or CouchDB.

- Even if the issues encountered in the "Archiving Old Blockchain Data" experiment could be solved with a different implementation, it is not clear whether archiving old data and starting a new blockchain is the best and/or only solution to the problem of the ledger's constantly increasing size.

## FINDINGS

While blockchain technology offers many advantages for a multi-sensor satellite architecture, it is not without limitations. These advantages and disadvantages are discussed below, but some of them require further testing under more realistic conditions in order to fully assess them. It should also be noted that it would be possible to attain all the same advantages with technologies other than blockchain, but the resulting implementation would no doubt have its own shortcomings. It would therefore seem simpler to use blockchain technology, as it is an active area of development with a strong developer community.

**Advantages**

- Immutable, non-repudiable, distributed record of commands and communications
- Automation of satellite observations
- Automatic routing of commands to any ground stations with line of sight to satellites
- Downlink satellites data at any ground station and automatically distributed to all desired parties
- Pass secured, encrypted data through unsecured ground stations
- Fine-grained access control over distributed data

**Disadvantages**

- Blockchain consensus expends more computational power
- Ledger grows in size over time and must be stored in its entirety on all peers
- Latency between peers has a very high performance cost
- Commands must be validated for consensus before being passed to satellites

## FUTURE DIRECTIONS

The blockchain experiments were performed using simulated satellites in an ideal cloud computing environment. This does not reflect the limitations that would be encountered in the real world, especially in terms of connectivity and processing speed issues. The next step for this project's research would be to repeat the experiments conducted under more realistic conditions. This would include using simulated satellite hardware, simulating a larger blockchain network with more latency, and simulating connection downtimes from satellites losing line-of-sight to ground stations.

## ACKNOWLEDGEMENTS

The authors wish to thank SGT KBRWyle and NASA's Earth Science Technology Office for sponsoring this project.

## REFERENCES

The following are references for the software used:

1. Hyperledger Fabric [Computer Software] https://hyperledger-fabric.readthedocs.io
2. Node.JS [Computer Software] https://nodejs.org
3. Hyperledger Composer [Computer Software] https://hyperledger.github.io/composer/latest/
4. Systems Tool Kit [Computer Software] https://www.agi.com/products/engineering-tools
5. Python [Computer Software] https://www.python.org
6. Geth [Computer Software] https://geth.ethereum.org/downloads/
7. Web3.js library [Computer Software] https://web3js.readthedocs.io/en/1.0/
8. Amazon Web Services [Website] https://aws.amazon.com/console/
9. json library for Python [Computer Software] https://docs.python.org/3/library/json.html
10. pyodbc library for Python [Computer Software] https://pypi.org/project/pyodbc/