Applying Blockchain to Energy Delivery Systems

DESIGN DOCUMENT

Team: sdmay20-12
Client: Grant Johnson
Adviser: Manimaran Govindarasu

Team Members:
Anthony Cosimo - *Test Engineer*
Jacob Dawson - *Report Manager*
Keegan Bloedel - *Meeting Facilitator*
Katherine Ringgenberg - *Meeting Scribe*
Steven Rein - *Software Architect*
Dakota Moore - *Cybersecurity Manager*

sdmay20-12@iastate.edu
https://sdmay20-12.sd.ece.iastate.edu

Revised: 3 October 2019

Executive Summary

Development Standards & Practices Used

- Agile Software Development - two-week sprints
- Test-driven Development
- Continuous Integration and Development
- Peer Reviews

Summary of Requirements

1. API
    1.1. API shall expose Smart Contract functions to Web Requests from the User Interface
    1.2. API requests require authentication
    1.3. API shall still be functional when the Blockchain network is down
2. Blockchain Network
    2.1. Blockchain Network shall have a minimum of five nodes
    2.2. Blockchain Network shall have multiple orderer nodes
3. Smart Contract Layer
    3.1. Functions within the Smart Contract Layer shall be able to read, update, delete, and query data stored on the ledger
4. User Interface
    4.1. Allow user to display and query performance data stored in Blockchain network
    4.2. Allow user to issue operator commands
5. System-level
    5.1. Each system shall be fault tolerant to other systems.
        5.1.1. For example, if the Blockchain Network is down, the API shall return an appropriate error code to the UI, indicating that the Blockchain Network is down. The UI will then display a message notifying the user of the Blockchain Network being down
    5.2. Each component shall be deployed on Linux virtual machines made available through PowerCyber resources

Applicable Courses from Iowa State University Curriculum

- COM S 309: Software Development Practices
- COM S 319: Construction of User Interfaces
- S E 329: Software Project Management
- S E 339: Software Architecture and Design

New Skills/Knowledge acquired that was not taught in courses

- Knowledge of HyperLedger Fabric and other HyperLedger technologies
- Knowledge of Blockchains concepts applicable to the problem domain
    - Orderers
    - Self-Signing Certificates
    - Node interactions
    - Raft Consensus
- Understanding and implementation of Docker tooling for containerized applications

Table of Contents

List of Figures/Tables/Symbols/Definitions

1 Introduction

1.1 Acknowledgment

Our team would like to thank and acknowledge PowerCyber Labs for allowing us to use their compute resources, Manimaran Govindarasu for advising our project, and Grant Johnson for being the client to our project.

1.2 Problem and Project Statement

1.2.1 Problem Statement

Energy Delivery Systems are deployed in an environment that is geographically distributed utilizing public internet infrastructure for communications. The integrity of measurements, commands, and authenticity of control devices performing communication are critical for trusted operations.

1.2.2 Proposed Solution

The purpose of this project is to develop a blockchain Energy Delivery System solution to solve the problem statement. The project is driven by the need described in the problem statement above, to create a secure network for a geographically diverse Energy Delivery System. A report on the security of Energy Delivery Systems from the National Research Council in 2012 stated that a powerful terrorist organization could cause a blackout for millions of people for many weeks (Terrorism and the Electric Power Delivery System, 2012). To help mitigate this risk, the use of blockchain technology would provide a secure network for managing Energy Delivery Systems by removing the dependency on a central service. Additionally, a permissioned blockchain system ensures a higher level of security when concerned with cyber-attacks like a fifty-one percent attack. With this project we hope to deploy a permissioned blockchain system for interaction between devices on an Energy Delivery System and users managing those devices.

1.3 Operational Environment

The operating environment for this project is servers that will store the project and other data we may use, such as simulation data. Therefore, it should not be subject to any harsh weather conditions, and the servers should be properly maintained. These servers will be a Linux-based and deployed on a network that will allow for secure permissioned communication between all nodes.

1.4 Functional Requirements

1.4.1 API

1.4.1.1 The API shall expose the Smart Contract functions to Web Requests from the User Interface and/or Devices.

1.4.1.2 The API calls shall be authenticated to the Blockchain Node System.

1.4.1.3 Requests to the API shall require authentication.

1.4.1.4 When the Blockchain network is down the API shall return an appropriate error code indicating the loss of the Blockchain Network.

1.4.2 Blockchain Network

1.4.2.1 The Blockchain Network shall consist of at least five nodes for transaction consensus.

1.4.2.2 The Blockchain Network shall consist of multiple nodes to act as orderers.

1.4.3 Smart Contract Layer

1.4.3.1 The Smart Contracts shall implement the blockchain functions to read, update, delete, and query data stored on the ledger.

1.4.3.2 Users that are assigned to a channel are the only users allowed to utilize the Smart Contracts functions made available to that channel

1.4.3.3 Each Smart Contract shall define more than one endorsing node for consensus.

1.4.4 User Interface

1.4.4.1 The User Interface shall be rendered in a modern web browser.

1.4.4.2 If a user has the authority to request specific measurements and/or metrics and does so via the User Interface, then the User Interface shall display those requested measurements and/or metrics.

1.4.4.3 If a user has the authority to issue specific operator commands, then User Interface shall provide the ability to issue those specific operator commands to the ledger.

1.4.4.4 Given the API is down the User Interface shall remain functional.

1.4.4.5 A user shall be able to view the results of metrics and measurements they had previously requested.

1.4.5 Operational Environment

1.4.5.1 The blockchain network shall run on a Linux-based virtual machine provided by PowerCyber resources.

1.4.5.2 The API shall run on a Linux-based virtual machine provided by PowerCyber resources.

1.4.5.3 The User Interface shall run on a Linux-based virtual machine provided by PowerCyber resources.

1.4.5.3.1 The User Interface should be interacted with using a modern web browser, Chrome will be the supported browser.

1.5 Non-functional Requirement

1.5.1 API

1.5.1.1 The API shall have swagger documentation for each endpoint.

1.5.1.2 The API shall produce query responses in under x seconds

1.5.1.2.1 Need to investigate reasonable value for this.

1.5.1.3 The API shall have continuous integration and automated deployment.

1.5.2 Blockchain Network

1.5.2.1 Each node on the network shall be ran using Docker containers to allow for reliability when ran on different types of machines.

1.5.2.2 The Blockchain Network shall be deployed to PowerCyber resources by implementing a CI/CD pipeline.

1.5.2.3 The Blockchain Network should be able to run in a Linux-based system via the services provided by Amazon Web Services, however this is not a direct stipulation by the client, just a potential for additional complexity if needed.

1.5.3 Smart Contract Layer

1.5.3.1 The Smart Contract Layer shall have automated deployment and automated testing.

1.5.3.2 The Smart Contract Layer shall make updates to the Blockchain Network.

1.5.3.3 The Smart Contract Layer controllers and models shall contain documentation that describe their purpose and intended use.

1.5.3.4 The Smart Contract Layer shall have unit tests for all controllers.

1.5.4 User Interface

1.5.4.1 If the blockchain network is not available, the User Interface shall display loss of communication to the user.

1.5.5 Maintainability

1.5.5.1 The documentation made available through the project's wiki is descriptive enough for the client to understand how to modify the project properly when needed.

1.6 Intended Users and Uses

Our Energy Delivery System will have two main types of users: human users and devices. The use cases of these users will consist of the following:

1.6.1 Human users should be required to login into the system through a web user interface before interacting with the system.

1.6.2 Querying metrics and measurements from the blockchain system. This will be done by the human users.

1.6.3 Posting updated commands for devices to use. This will also be done by the human users, although there are potential use cases within the problem domain where the devices will give or share commands with each other.

1.6.4 Receive and execute commands provided by the user. This will be done by the devices.

1.6.5 Periodically post updated measurements. This will be done by the devices, although there is potential for the human users to make changes to the domain specific data, in cases of errors with metrics or measurements.

These use cases and users may have overlapping interactions with the system as mentioned, but generally the interactions will consist of the primary user in each use case, as mentioned above.

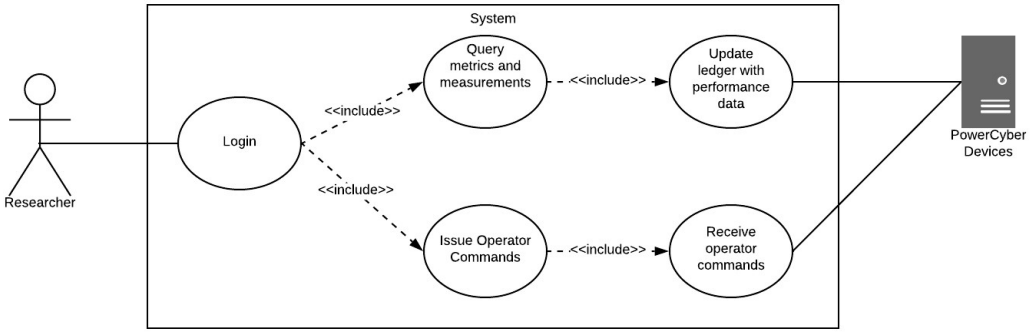

Figure 1: Use Case Diagram

1.7 Assumptions and Limitations

1.7.1 Assumptions

1.7.1.1 Server hardware and operating system environment will be made available through PowerCyber and these resources will be sufficient for the development.

1.7.1.2 We will be provided access to devices that use PowerCyber.

1.7.2 Limitations

1.7.2.1 There is no budget for the project, thus, we are constrained to using PowerCyber resources.

1.7.2.2 If the PowerCyber resources are found to be insufficient, the sponsor will either arrange for different resources or modify the scope to work with the existing resources.

1.7.2.3 Hyperledger Fabric must be used as the permission-based distributed ledger framework.

1.7.2.4 All Hyperledger Fabric related code must be written in JavaScript.

1.8 Expected End Product and Deliverables

1.8.1 A Fully Functional Blockchain Node System

There will be a Node System that is running within Virtual Machines within the PowerCyber network. The nodes in the Node System will help to endorse information updates to the ledger, provide immutability of the data within the ledger, and be fault tolerant to losses of nodes within the network. Communication with these nodes will be done by a multi-node ordering service. Unlike other blockchain systems, the orderer will allow our data to be deterministic instead of probabilistic. Additionally, the blockchain Node System will have smart contracts installed and instantiated, this will be for interaction between the nodes and the API system which will be utilized by the users for the provided use cases.

1.8.2 Authenticated Calls from API to the Blockchain Node System

There will be an API that can utilize the Smart Contracts in order to create, read, update, delete, and query data available on the Node System. All calls to the Smart Contracts from the API shall be authenticated and verified by the allowed participants in each channel. The Smart contracts will be able to receive data from the nodes along with updating the data on the nodes.

1.8.3 Web-based User Interface

The web-based user interface will contain web pages that will allow authenticated users to query and view measurements, and post commands to devices.

1.8.4 Project Documentation

There will be a wiki that lives within the project repository that describes how to deploy updates to the Smart Contracts, Node System, API, and Web-based User Interface. This wiki will also have documentation regarding the development environment setup and required dependencies.

1.8.5 Continuous Integration and Deployment Infrastructure

There will be a CI/CD solution in place for the project. Continuous Integration will run on all merge requests and will run on the master branch weekly to ensure that the master branch remains clean. Continuous deployment will be available for the blockchain network, API, and web-based user interface.

2. Specifications and Analysis

2.1 Proposed Design

The software solution consists of the following components: a User Interface, an API, and a Blockchain Network. The API receives requests from the User Interface to view performance data and issue operator commands to PowerCyber devices. The API will authenticate those requests with the Membership Service Provider. After a request has been authenticated, the API will make a

request to run a smart contract on the Blockchain Network that either queries performance measurements or updates operator commands for the PowerCyber devices.

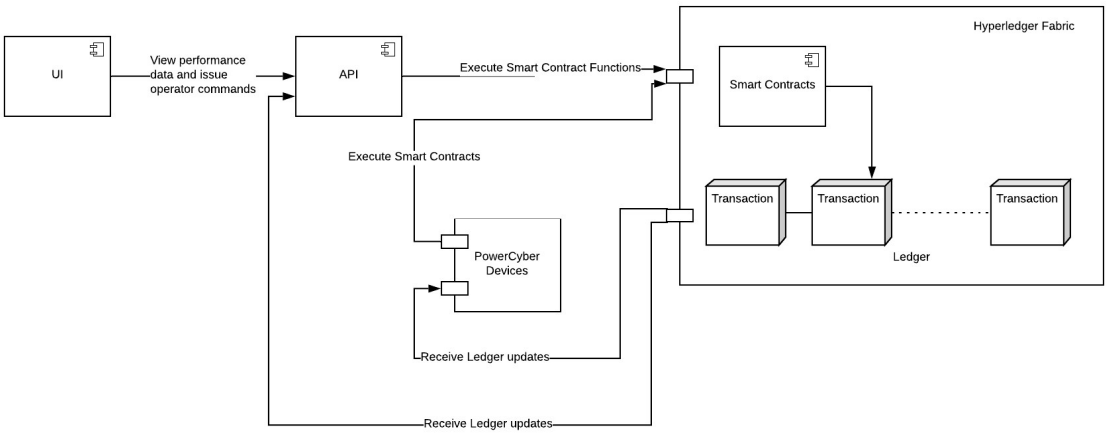

Figure 1: Architecture Diagram

2.1.1 API

2.1.1.1 The API will run smart contracts when new performance data has been received by a PowerCyber Device.

2.1.1.2 The API will run smart contract when an authenticated user requests to view performance data.

2.1.2 Blockchain Network

2.1.2.1 The Blockchain Node System will report ledger updates to the API.

2.1.2.2 The Blockchain Node System will receive requests to run smart contracts from the API.

2.1.2.3 Given that the entity who has requested to run a Smart Contract is authorized for that requested data, the Blockchain Node System will run that Smart Contract in order to update the ledger.

2.1.3 Smart Contract Layer

2.1.3.1 Given that the entity who is requesting to query entries via a Smart Contract is authorized to do so, the Smart Contract Layer will be able to query entries in the Blockchain Network.

2.1.3.2 Given that the entity who is requesting to create an entry via a Smart Contract is authorized to do so, the Smart Contract Layer will be able to create an entry in the Blockchain Network.

2.1.3.3 Given that the entity who is requesting to create an update to an entry via a Smart Contract is authorized to do so, the Smart Contract Layer will be able to create an update to an entry in the Blockchain Network.

2.1.4 User Interface

2.1.4.1 The User Interface will allow a user, based on their authority, to request PowerCyber Device metrics.

2.1.4.2 The User Interface will allow a user, based on their authority, to issue operator commands on PowerCyber device(s).

2.1.5 The User Interface will allow a user to login.

2.1.6 Given that a user with required permissions to create a new account, the User Interface shall allow the user to create a new user.

2.1.7 PowerCyber Devices

2.1.7.1 The PowerCyber Devices will receive and perform operator commands from the Blockchain Network.

2.1.7.2 The PowerCyber Devices will report performance data to the Blockchain Network.

2.2 Design Analysis

2.2.1 Infrastructure Analysis

2.2.1.1 Currently, we plan on using our Blockchain Network to handle the messaging between peers. We believe that given the need for a secure network with the need for remote access to data, deploying a Blockchain Network will be the best solution.

2.2.2 Front-end Analysis

2.2.2.1 Currently, we plan to use a web framework to handle the UI of our software. We are currently in the research phase for our UI because the final scope of our project has yet to be determined, but we plan on using a software such as Node.JS for developing the front-end due to the need for easy implementation of our API.

2.3 Development Process

Our team will be taking an agile approach to the project. We will be using two-week sprints with a retro and demo at the end of each sprint as necessary. For all merge requests, there shall be tests to accompany all new work introduced into the master branch, therefore, we will be following a Test-Driven Development process for the project. All merge requests must pass all component and system-level tests and be reviewed and approved by at least one other member on the project.

2.4 Design Plan

The software solution will consist of a Django API, Hyperledger Fabric distributed ledger with the Convector suite for developing Smart Contracts, and React.js UI.

The API will process requests from the UI and make requests to Smart Contracts when necessary. All requests made to the API, except for requests for signing in, will be authenticated requests. If the Smart Contract Layer were to become unresponsive, the API will remain functional outside of the interactions with the Smart Contract Layer.

The Blockchain Network will consist of at least five nodes and multiple orderers to share messages between peers. The Blockchain Network will have a Raft Ordering Service for transaction ordering. The world state ledger will utilize CouchDB. Hyperledger Fabric cryptographic generation tools will be used to utilize self-signed certificates instead of an external certificate authority to authenticate interactions to and within the Blockchain Network. Measurement data will be stored on the Blockchain Network along with issued operator commands. The purpose of doing so is to provide and immutable transaction history of operator commands and reports of measurement data.

The Smart Contract Layer will send ledger updates to the API and to PowerCyber devices respectively. The updates will include operator commands and measurement updates reported by PowerCyber Devices. All updates will require endorsements from at least three nodes, given that there are five total nodes. The Smart Contract Layer will allow users and PowerCyber devices to invoke, through the API or directly, Smart Contract functions. These functions will be written in JavaScript and will provide the ability to read, write, query, and delete information stored in the Blockchain Network. The Smart Contract functions must install on at least one Peer Node and instantiated on at least three other nodes for endorsement.

The UI will allow users to login, request measurements and metrics they are authorized to query, and issue operator commands that they are authorized to issue. If the API were to shut down for any reason the UI will remain responsive outside of the interactions with the API.

3. Statement of Work

3.1 Previous Work And Literature

An article titled “Blockchain technology in the energy sector: A systematic review of challenges and opportunities” gives a lot of information on blockchains and how they can be used in energy solutions. One example this article gives is Brooklyn MicroGrid, which is a blockchain-based person-to-person energy trading platform.

The Brooklyn MicroGrid system completed a three-month trial with the community. The article was written in February 2019. Upon further research into Brooklyn MicroGrid, it appears they are still up and running and have future goals of expanding and having fully automated transactions. This energy trading platform allows consumers to “sell their energy surplus directly to the neighbors by use of Ethereum-based smart contracts” (Andoni et al). This system’s ledger records “contact terms, transacting parties, volumes of energy injected and consumed by metering devices and crucially the chronological order of transactions” (Andoni et al). The article goes more in depth about blockchain potential and specifics involving the energy sector.

Our project will be looking at securing the communication network in an Energy Delivery System through the usage of blockchain. This differs from the Brooklyn MicroGrid system because our system will be constrained by a necessity for high quality integrity within the data, whereas the Brooklyn MicroGrid system is concerned with the sale of an energy surplus.

3.2 Technology Considerations

HyperLedger Composer, a tool used for building a blockchain network and implementing smart contracts using HyperLedger Fabric, has recently been deprecated. To work around this, we will investigate an alternative called Convector, supported by Hyperledger Labs. Convector provides similar functionality that Composer provides with some additional features such as an API Server Generator, configuration file generation, Smart Contract boilerplate code, etc. Also, convector allows the user to make configurations to a definitions JSON file for exposing Smart Contract functionality to an API.

HyperLedger Fabric will be used for configuring and deploying our blockchain network. HyperLedger Fabric utilizes YAML configuration files for structuring and setting up the network. HyperLedger Fabric also uses Docker Composer for defining, creating, and deploying the containerized nodes to ensure all environment requirements are met. HyperLedger Fabric works using orderer nodes that exist as the central communication for the network. These nodes ensure consistency is maintained with the state of the Ledger. HyperLedger Fabric has a key value database called CouchDB to deal with transaction logs and ledgers (NoSQL document store). CouchDB is the default database for HyperLedger Fabric, therefore the best choice when working with Fabric.

For development of our user interface and request API there are many technologies on the market to consider. With that in mind, there are three web frameworks the team has looked into. Django, written in Python, provides rapid development and a plethora of tools for development for teams. Flask, also written in Python, provides a simple and flexible developer experience. Comparing the two, Flask is most likely preferred if the focus is gaining experience and learning. Django would be preferred if the focus is on the final product and maintainability, being the older of the two. The final web framework we have considered is Node.js. The benefits of Node.js for our team are that we are already doing some programming in JavaScript through other components of the project. Also, Node.js integrates well with HyperLedger Convector, with the Convector API Server Generator producing Node.js. For the web user interface ReactJS will be used. The team decided upon this to give ourselves the opportunity to work with a framework that is growing within the developer community. Additionally, ReactJS is primarily used for building single page applications, which fits our expected solution for the web user interface.

3.3 Task Decomposition

When stating that an implementation of a feature will be happening, this suggests that the implementation will include the necessary source code, along with the implementation of any appropriate tests e.g. unit, integration, smoke.

3.3.1 Requirements Gathering

3.3.1.1 Gather Functional Requirements

3.3.1.2 Gather Non-functional Requirements

3.3.2 Gain Domain Knowledge

3.3.2.1 Meet with PowerCyber research team

3.3.2.2 Research Energy Delivery Systems

3.3.3 Implement a functional five node Blockchain Network

3.3.3.1 Implement CI/CD pipeline

3.3.4 Implement the Smart Contract Layer

3.3.4.1 Create, append, and query measurement and metrics data on nodes

3.3.4.2 Create, append, and query operator command transactions on nodes

3.3.4.3 Implement CI/CD pipeline

3.3.5 Implement the API

3.3.5.1 Implement endpoints for logging into software suite

3.3.5.2 Integrate with Smart Contract Layer

3.3.5.3 Implement endpoints for querying measurement and metrics data

3.3.5.4 Implement endpoints for querying operator commands history

3.3.5.5 Implement endpoints for creating operator commands

3.3.6 Implement the UI

3.3.6.1 Implement routes for querying measurement and metrics

3.3.6.2 Implement routes for issuing operator commands to specific devices

3.3.6.3 Implement routes for querying operator commands on specific devices

3.3.7 Integrate with PowerCyber Devices

3.3.7.1 Implement PowerCyber devices reporting performance data to Blockchain Network

3.3.7.2 Implement PowerCyber devices receiving operator commands from Blockchain Network

3.3.8 System-level Testing

3.3.8.1 Implement end-to-end tests for querying performance data

3.3.8.2 Implement end-to-end tests for issuing operator commands

3.4 Possible Risks And Risk Management

Title: Blockchain could possibly be the wrong solution for the problem we intend to solve.

Risk: Avoid

Information: The time to complete blockchain consensus may take too long for the desired use cases. Also, the resources required to deploy and maintain a blockchain may not be valuable enough for the desired use cases.

Mitigation Action: We will work with the PowerCyber team to determine appropriate use cases (measurements and commands) that would benefit from using a Blockchain Network.

Title: Team's lack of knowledge on the domain could lead to easy to detect flaws being introduced into the project.

Risk: Mitigate

Information: The architecture of the project could be at risk due to the lack of knowledge on the domain. Classes, dataflow, and test assertions could be invalid resulting in a time consuming refactor.

Mitigation Action: Gain domain knowledge by visiting PowerCyber, interviewing domain experts, and asking questions to the client and advisor when necessary. Due to the team's frequent two-week sprints and high client interaction, increased communication and quick feedback can be easily achieved.

Title: Integration of system components could fail

Risk: Mitigate

Information: The integration between the Smart Contract Layer and the API, the PowerCyber Devices and the Blockchain Network, etc. is at risk due to the team's inexperience with Smart Contracts and PowerCyber Devices.

Mitigation Action: Utilize and maintain Swagger documentation and automate integration and acceptance tests.

3.5 Project Proposed Milestones and Evaluation Criteria

- Successful communication between five nodes in the blockchain network occurs.
    - Occurs when five nodes can successfully communicate.
- Smart contracts within nodes can carry out a task.
    - Occurs when the task that the smart contract intended to carry out happens successfully.
- Successful communication between blockchain nodes in regard to the problem domain.
    - Occurs when blockchain nodes are successfully communicating using definitions of the data within the problem domain.

- API provides reasonable responses queried by the user.
    - Occurs when the API functionality exists for all defined smart contracts and responses are given successfully in a reasonable time interval.
- User interface makes a successful API call.
    - Occurs when the API call is triggered by the user interface successfully carries out its function.
- User interface provides accurate and visually pleasing data.
    - Occurs when the client is satisfied with the readability of displayed metrics and the user interface, overall.

3.6 Project Tracking Procedures

We are going to use Gitlab Issues to track our progress. Gitlab Issues provides our team burndown charts and allows us to use custom formats for tracking. We will use these issues to track individual and team progress throughout the project. These issues include a title and description on what will be worked on, the ability to select due dates and labels, and the ability to even assign members to each issue. When an issue is completed, it is easily marked as so and archived, leaving a detailed list of what was worked on for the project.

3.7 Expected Results and Validation

Our desired outcome is to produce and implement a suite of software that utilizes a secure Blockchain Network and can integrate with devices used at PowerCyber. This software suite shall add integrity to the existing system at PowerCyber.

To confirm that our solution works at a higher level the following will need to be found true by the end of the project.

- The software suite can receive performance data from PowerCyber devices
- The software suite can store performance data in its Blockchain Network
- Users of the software suite can query performance data from the software suite's UI
- Users of the software suite can issue operator commands to PowerCyber devices from the software suite's UI
- Operator commands issued through the software suite's UI will be entered into the Blockchain Network and distributed to the PowerCyber devices
- PowerCyber devices can receive and execute commands issued to them via the Blockchain Network
- Removing Blockchain peer nodes does not interrupt validation and committing to the ledger
- Removing Blockchain orderer nodes does not interrupt validation and committing to the ledger
- Removing communications to the Blockchain Network or the API results in a displayed error and continued responsiveness of the UI

4. Project Timeline, Estimated Resources, and Challenges

For the majority of section 4, when stating that an implementation of a feature will be happening, this suggests that the implementation will include the necessary source code, along with the implementation of any appropriate tests e.g. unit, integration, smoke.

4.1 Project Timeline

Tasks	Sept 3rd - Oct 14th	Oct 15th - Oct 28th	Oct 29th - Nov 11th	Nov 12th - Dec 9th	Dec 10th - Jan 6th	Jan 7th - Jan 27th	Jan 28th - Feb 24th	Feb 25th - Mar 23rd
Requirements Gathering								
Gaining Domain Knowledge								
Prototyping								
Blockchain Network								
Smart Contract Layer								
Implement the API								
Implement the UI								
PowerCyber Integration								
System-level testing								

Table 1: Timeline of Proposed Work Schedules

We foresee the Requirements Gathering stage to continue until Oct 14th. There are still some non-functional requirements we need to clarify with the client. The prototyping stage is for gaining background knowledge on the tech stack we will be working with. This should help clarify which technologies we decide to use for the project. Implementing a Blockchain Network will take close to a month due to the team's inexperience with the technology and gaining access to the resources for hosting the Blockchain Network. So far, from what we have noticed during our experiments with Hyperledger Convector, implementing the Smart Contract Layer should not be too difficult. Hyperledger Convector provides a simple way to define Smart Contracts and to create automated tests for those Smart Contracts.

The means of how we intend to implement the API is still under design consideration. At this moment, we are unsure if we should use the auto-generated backend available through the Convector Suite or by implementing a Django API to handle requests from the UI. The estimation of implementing the API may change in the future, depending on what we learn during the prototyping stage.

Implementing the UI shall be close to straight-forward. The framework that we use for the UI does not depend on the decisions made for the API.

The time to complete integration with PowerCyber devices is currently unknown. The estimation in the Gantt chart is our best guess as of right now. We will have a more accurate estimation for this task once we have visited with the PowerCyber team.

Addressing technical debt will always be necessary for projects of this size. We will need to constantly revisit tasks that we have completed in the past and address any updates that are necessary at that time.

4.2 Feasibility Assessment

Depending on the use cases we learn about when we visit with the PowerCyber team, the correct use case will be chosen that is best suited for a Blockchain implementation given the latency and data criticality constraints. Some foreseen challenges that we see as of now is the time for the nodes to come to a consensus when a Smart Contract is executed. Depending on the requirements of the devices at PowerCyber, the time to come to a consensus may be too slow. For example, a device may need to update the Blockchain Network every two seconds, we can foresee our nodes taking longer than two seconds to come to a consensus. Thus, rendering our software suite incorrect for that use case.

If there are some use cases that we can handle due to the bottleneck of coming to a consensus across all nodes, then our project will obsolete provide a simple and easy way for researchers to view measurements and metrics and to issue operator commands to PowerCyber devices. While doing so, our software suite will be providing immutable performance data and operator commands transaction history.

4.3 Personnel Effort Requirements

Task		Time to Perform Correctly
Requirements Gathering		Total Expected Hours: 30
	Gather Functional Requirements	15
	Gather Non-functional Requirements	15
Gain Domain Knowledge		Total Expected Hours: 22

	Meet with PowerCyber research team	2
	Research Energy Delivery Systems	20
Implement a functional five node Blockchain Network		Total Expected Hours: 20
	Implement CI/CD pipeline	20
Implement the Smart Contract Layer		Total Expected Hours: 60
	Create, append, and query measurement and metrics data on nodes	30
	Create, append, and query operator command transactions on nodes	10
	Implement CI/CD pipeline	20
Implement the API		Total Expected Hours: 45
	Implement endpoints for logging into software suite	5
	Integrate with Smart Contract Layer	15
	Implement endpoints for querying measurement and metrics data	5
	Implement endpoints for querying operator commands history	5
	Implement endpoints for creating operator commands	15
Implement the UI		Total Expected Hours: 45
	Implement routes for querying measurement and metrics	10
	Implement routes for querying operator commands on specific devices	20
	Implement routes for issuing operator commands to specific devices	15
Implement System-level Tests		Total Expected Hours: 45
	Implement end-to-end tests for operator commands	25
	Implement end-to-end tests for performance data	25

Table 2: Personnel Efforts Requirements per Task

4.4 Other Resource Requirements

The only additional resource needed for our project is access to the PowerCyber facilities.

5. Testing and Implementation

5.1 Interface Specifications

All components within the software suite will have to agree on consistent models to represent performance data, types of devices to interact with, and operator commands. This is yet to be determined until we visit the PowerCyber team.

5.2 Hardware and Software

5.2.1 API

5.2.1.1 Unit tests and integration tests for the API will use the included unittest module provided by the Django framework. With this module we will write automated tests to make HTTP requests to our endpoints.

5.2.2 Blockchain Network

5.2.2.1 All new chaincode will be added to a chaincode subdirectory and will be tested when relaunching the Blockchain Network.

5.2.3 Smart Contract Layer

5.2.3.1 We will be using the Chai assertion library along with the Mocha test framework for testing models and controllers. The Chai assertion library can be used in a similar way as Jest is used, using expect style assertions. To keep our software suite consistent, we will be using expect style assertions in our Smart Contract Layer. The Mocha test framework will allow for easy asynchronous testing, serially running tests, accurate reporting, and mapping uncaught exceptions to correct test cases. Both Chai and Mocha are used by default in Hyperledger Convector.

5.2.4 User Interface

5.2.4.1 For writing unit tests, acceptance tests, and end-to-end tests, the Jest Javascript test runner will be used along with the React Testing Library. The Jest test runner will allow us to access the DOM via jsdom for testing React components and to use mocks. The React testing library will allow us to test React components without relying on their implementation details. For end-to-end tests we will remove the use of mocks and have all requests made to their respective endpoints.

5.2.5 PowerCyber Devices

5.2.5.1 Assuming we will not have access to PowerCyber devices often, we will have to examine some example performance data from the devices we will be working with and create a simulation data collection to use for integration tests between the Blockchain Network and PowerCyber devices.

5.3 Functional Testing

All components of the software suite will contain their own test suites that are automated. These test suites will consist of mostly unit tests besides the UI and API components. The UI and API components will have unit tests and integration tests. The UI will also have acceptance tests and end-to-end tests.

5.4 Non-Functional Testing

Unit tests for making unauthenticated requests via the API and to the Smart Contract layer will be in place to ensure that all requests must be authenticated. There will be performance tests for the Smart Contract Layer by using simulated data from the PowerCyber devices to profile the time it takes for all nodes to come to a consensus on Smart Contract execution.

5.5 Results

During our research phase, we learned that one of the software's we wanted to use for our project became depreciated and this led to us having to research further into possible replacements. This led to us using Convector, which we learned is supported by Hyperledger Labs. Due to our project still being in the early stages, we have yet to design or test any actual software.

6. Closing Material

6.1 Conclusion

So far, we have gathered requirements from the client and have experimented with Hyperledger Convector. We plan on finishing the gathered requirements phase and moving onto the prototyping phase. By the end of the prototyping phase we will have a more detailed project plan since we will know which technologies to use and how to integrate with the PowerCyber devices.

6.2 References

M. Andoni, V. Robu, D. Flynn, S. Abram, D. Geach, D. Jenkins, P. Mccallum, and A. Peacock, "Blockchain technology in the energy sector: A systematic review of challenges and opportunities," Renewable and Sustainable Energy Reviews, vol. 100, pp. 143–174, Feb. 2019.

Terrorism and the Electric Power Delivery System. (2012). National Academies Press, p.1.

6.3 Appendices

Framework and Library Documentation:

CouchDB:

http://docs.couchdb.org/en/stable/

HyperLedger Fabric:

https://hyperledger-fabric.readthedocs.io/en/release-1.4/

HyperLedger Composer (deprecated):

https://hyperledger.github.io/composer/latest/introduction/introduction.html

HyperLedger Convector:

https://docs.covalentx.com/article/71-getting-started

PowerCyber Labs:

http://powercybersec.ece.iastate.edu/powercyber/welcome.php

Django:

https://docs.djangoproject.com/en/2.2/

React.JS:

https://reactjs.org/docs/getting-started.html

Raft

https://raft.github.io/raft.pdf