Chapter 4

Blockchain enabled Protection Scheme for Bipolar DC Microgrid-A real-time Simulation

Protection systems in distribution and transmission networks play a crucial role in modern smart grid environments by isolating faulted sections and ensuring the continuity of critical loads. Contemporary protection strategies are increasingly adopting cognitive approaches that adapt dynamically to extreme changes in the power system. However, the reliability and resilience of these data-driven solutions are highly dependent on the integrity, authenticity, and confidentiality of the control signals and information exchanged through the underlying relay communication networks. The design of protection systems includes communication enabled intelligence along with physical components to implement the protection algorithm. The resulting large scale cyber-physical formation become vulnerable to cyber threats. Securely transmitting relay settings between the control center and substations is essential to enable reliable and safe operations.

4.1 Introduction

The previous chapter discusses a unit protection scheme for bipolar DC microgrids. The proposed method in Chapter 3 is inherently resilient to the single ended cyber attack in the protected bipolar DC line. However, for faults during time synchronization attack the response of the proposed method may get delayed depending on the nature of TSA.

This chapter introduces a novel protection scheme for bipolar DC microgrids, leveraging the decentralized and immutable nature of blockchain technology to ensure secure and reliable data transmission. By implementing an Ethereum-based blockchain network, the proposed method secures the transmission of current measurements between the terminals of a protected DC line, thereby preventing tampering and unauthorized access.

The core of the proposed scheme involves RTDS to model the bipolar DC microgrid. Current measurements at the remote end of the protected line segment are transmitted using the socket protocol integrated within the RTDS. This setup ensures that any deviations in current, which may indicate faults or cyber intrusions, are promptly detected and addressed by the protection algorithm.

Additionally, the implementation includes a user dashboard that provides real-time monitoring and visualization of current measurements and blockchain transaction status. This dashboard facilitates user interaction and system management, ensuring transparency and enabling quick response to any detected issues.

By combining blockchain technology with real-time simulation, this research demonstrates a robust framework for enhancing the security and reliability of DC microgrids. Extensive simulations validate the proposed protection scheme, showcasing its effectiveness in various scenarios, including high-resistance faults, load fluctuations, and cyber attacks. Comparative analysis with existing techniques highlights the superiority of our approach in safeguarding the microgrid infrastructure against internal and external faults. This integration of blockchain technology not only secures data transmission but also enhances the overall resilience and efficiency of the energy system.

4.2 Blockchain Preliminaries and Considerations

Blockchain operates on a fully decentralized, peer-to-peer (P2P) network architecture and functions as a distributed, transparent public ledger, a data structure that is replicated and shared across all network participants. Each node in the network uses public-key cryptography to initiate transactions (any form of data exchange). These transactions are verified by peer nodes for signature authenticity and data integrity before being grouped into records called "blocks". Each block has a fixed capacity and is composed of a header and a body. The body stores the transaction data, while the header links the block

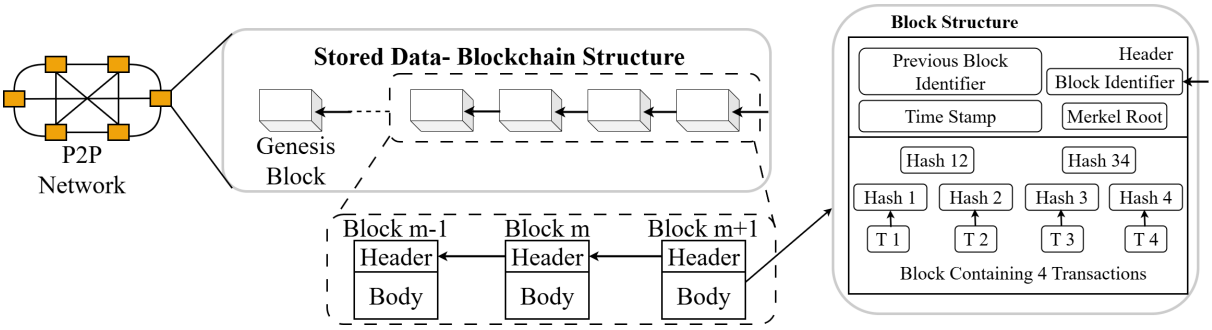


Figure 4.1: Blockchain data structure.

cryptographically to its predecessor, ensuring a sequential and tamper-evident chain. The first block on the blockchain is called the "genesis" block. The header of each block includes a unique identifier generated via a cryptographic hash of its transactions, the hash of the previous block, and a timestamp. Additionally, the header contains a Merkle tree root, which is derived by recursively hashing pairs of transaction IDs to create a hierarchical hash structure. Figure 4.1 illustrates a typical blockchain data structure for a peer-to-peer (P2P) network.

Newly generated blocks are permanently appended to the blockchain through a predefined set of rules known as a distributed consensus protocol, which guarantees agreement among all independent nodes on the shared global blockchain state, including the transaction content and its sequence. Numerous distributed consensus algorithms have been introduced, each influencing the scalability and performance of blockchain systems in different ways [119]. At a broader level, based on the intended application and the chosen consensus mechanism, blockchain systems can be categorized as either public or private. In public blockchains, any node is allowed to join the network, submit transactions, validate data, and publish new blocks while maintaining a complete copy of the distributed ledger. These networks typically support a large number of participants and rely on proof-of-work (PoW) consensus algorithms. In PoW, a miner node aggregates transactions into a block and must solve a complex computational puzzle before the block can be added to the chain. This process is intentionally resource-intensive to deter malicious activity and make unauthorized modifications prohibitively expensive.

In contrast, private blockchains require all participating nodes to be authenticated and clearly identified. These networks enforce stricter access controls and participant verification, allowing them to adopt more efficient byzantine fault-tolerant (BFT) consensus

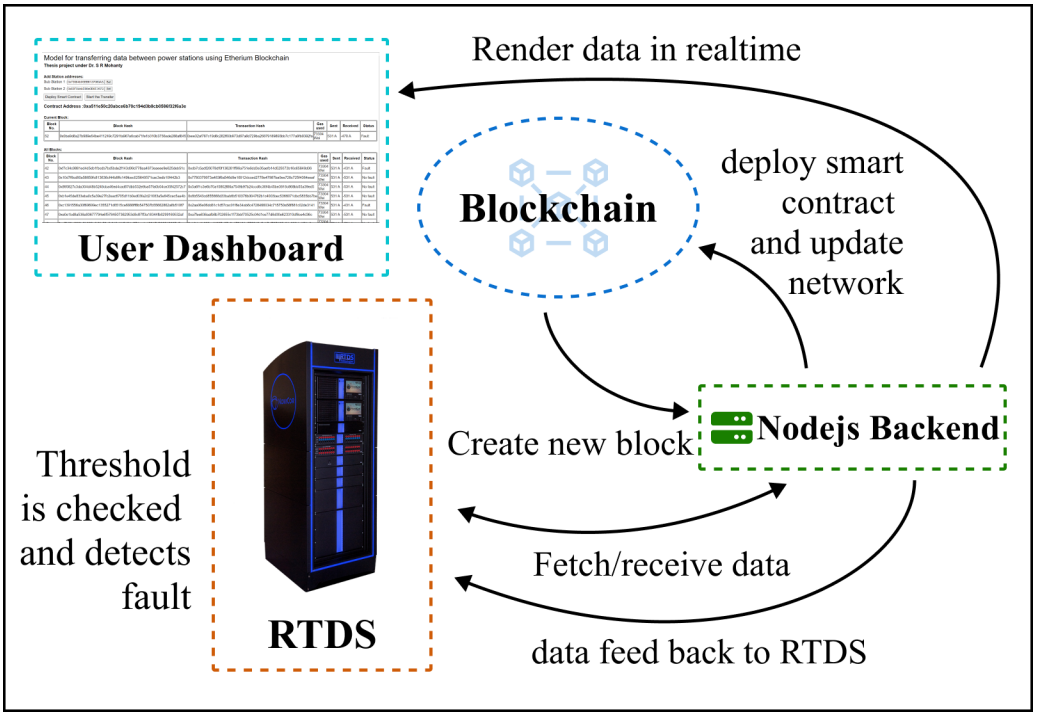


Figure 4.2: Blockchain model architecture.

protocols and voting-based mechanisms. As a result, they achieve agreement without relying on computationally demanding proofs, making them more suitable for controlled environments [119].

4.3 System Architecture

The system comprises an RTDS machine, an Ethereum blockchain network [120], a Nodejs backend, and a user dashboard. Figure 4.2 illustrates the main elements of the system architecture and presents the sequence of operations between each element.

4.3.1 Ethereum Blockchain

The Ethereum blockchain stands as a pioneering platform in the realm of decentralized technologies, offering a robust and versatile foundation for building decentralized applications (DApps) and executing smart contracts. Introduced by Vitalik Buterin in 2015, Ethereum has significantly expanded the capabilities of blockchain technology beyond its initial application in cryptocurrencies like Bitcoin.

At its core, Ethereum operates as a decentralized, open-source blockchain that sup-

ports a wide range of applications through its native cryptocurrency, ether (ETH). Unlike traditional centralized systems, Ethereum leverages a global network of nodes to maintain consensus and ensure the integrity of transactions and data without the need for a central authority. This decentralized nature enhances security, transparency, and resilience against tampering or single points of failure [121].

Ethereum provides a flexible platform for developing DApps, which are applications that run on the blockchain and operate autonomously without central control. Developers use Ethereum's programming language, Solidity, to write these applications. The Ethereum Virtual Machine (EVM) executes the DApps [122], ensuring they run consistently and securely across the network. This has led to the creation of a diverse ecosystem of applications, ranging from financial services (DeFi) and gaming to supply chain management and identity verification. Ethereum initially used a PoW consensus mechanism, similar to Bitcoin, which requires miners to solve complex mathematical problems to validate transactions and secure the network. However, to enhance scalability, security, and energy efficiency, Ethereum is transitioning to a proof of stake (PoS) mechanism through the Ethereum 2.0 upgrade [123]. In PoS, validators are chosen based on the number of ETH they hold and are willing to "stake" as collateral, significantly reducing the computational resources required. The DC microgrid is simulated in RTDS, and the current measurements at the two ends of the protected line segment are used for the protection decision. The Ethereum blockchain network records all data transmitted through the RTDS by the deployed smart contract. A smart contract is a set of rules that define how the network's nodes operate, ensuring the decentralized network functions correctly.

The user dashboard displays data in real time to the operator, ensuring that any malfunctions or faults can be addressed immediately. Additionally, the dashboard provides a comprehensive overview of system performance, facilitating efficient monitoring and management. It allows operators to analyze trends, optimize processes, and make informed decisions based on accurate and up-to-date information.

The backend, built on Node.js, is the lifeline of the entire system, connecting various tech stacks via application programming interface (API). Data received from the RTDS is directed to the backend server, which then transmits it to the blockchain network. The backend subsequently retrieves data from the blockchain and sends it to the next relay and the user dashboard.

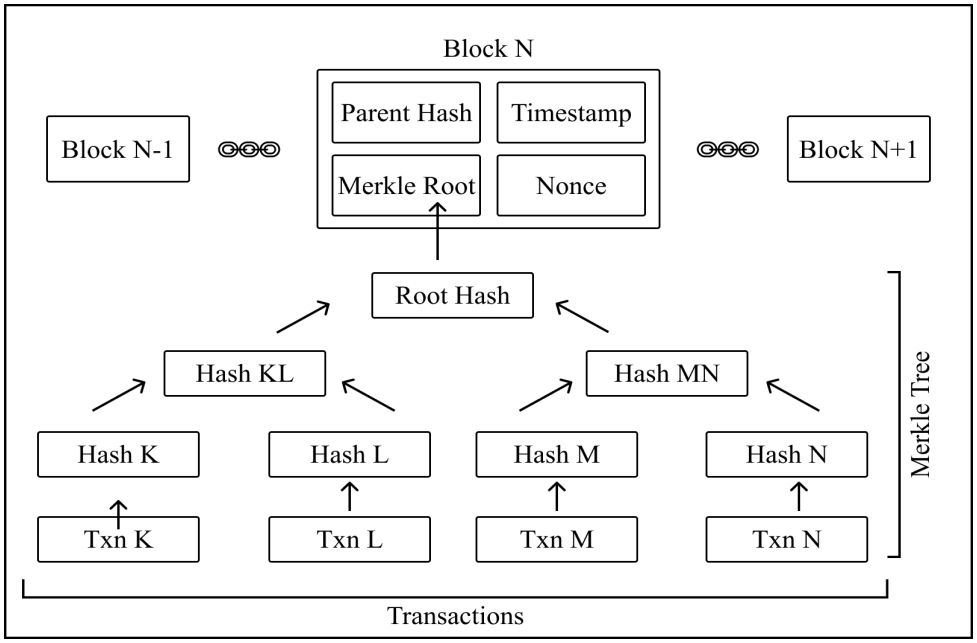


Figure 4.3: Block structure.

4.3.2 Block Structure

A block in the Ethereum blockchain is a collection of transactions. When users perform actions on the network (updating and executing smart contracts in this case), these actions are bundled and stored in the blocks. Figure 4.3 shows all the constituents of the block.

- Parent Hash: Hash of the previous block, ensures that the chain is continuous.
- Block Number: It is the position of the block in the blockchain.
- Timestamp: The time at which the block is created.
- Nonce: A value used to validate the proof-of-work.
- Merkle Root: It is a simple mathematical way to verify data integrity. In cryptocurrency, Merkle roots ensure that data blocks passed between peers are complete, undamaged, and unaltered. This is done by hashing pairs of data blocks repeatedly until a single hash, the Merkle root, is produced, allowing efficient and secure data verification.

4.3.3 Data Transmission Scheme

Data is transmitted from one end to the other in the protected line segment. RTDS simulates the real-time scenario where data is sent/received using sockets. The first transmission line is the RTDS to the Nodejs server.

Once data is received from the RTDS then it is directed to the express server (intermediater between RTDS and Nodejs), after that web sockets do the work of data transmission. Now data is finally received in the Nodejs environment where the actual Ethereum blockchain model is built. As the data is communicated using the express server in the last stage, the benefit of which is that whenever a new stream of data comes it automatically starts the process of sending data to the blockchain network.

The React JS library is used to integrate the Ethereum blockchain using Truffle Suite which is a development framework for building DApps.

To get a clear idea, let's divide the code into 2 sections:

1. **Interaction with the blockchain**:

   The initial step in interacting with the blockchain involves deploying the smart contract. To accomplish this, a dependency named Solc is installed and used, which compiles the Solidity language used for writing smart contracts. Once compiled, the smart contract needs to be deployed on a network. For testing purposes, the Ganache testing network is utilized. Additionally, managing transactions the Enkrypt wallet is employed.

   With the smart contract deployed, the subsequent step involves registering data sent by the first relay (node) onto the blockchain using functions specified in the smart contract. Following registration on the network, the data is retrieved by the second relay and transmitted back to the RTDS via the original data stream. Upon receiving the data, the RTDS assesses the threshold and determines the connection status, identifying any potential faults. Figure 4.4 shows the schematic of the whole process.

2. **Rendering of the User Dashboard:**

   The User Dashboard consists of two parts. The first part involves registering

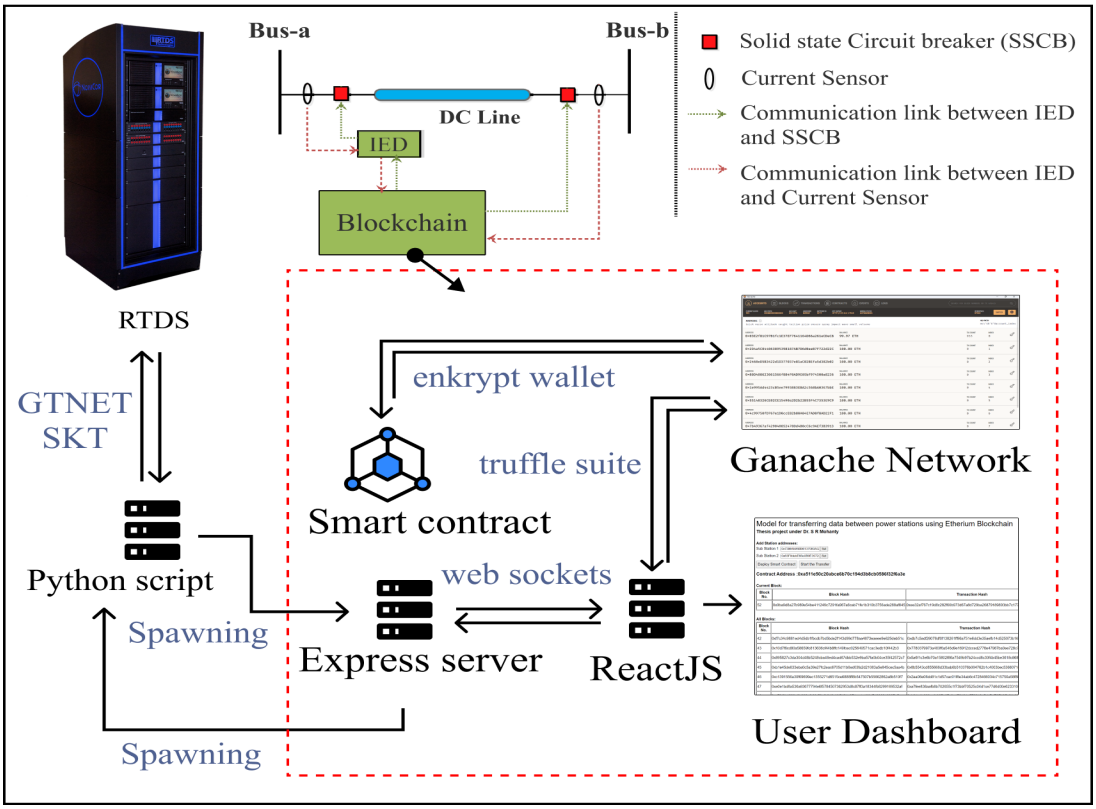


Figure 4.4: Data transmission flow.

nodes/relays and deploying the smart contract. Once deployed, a contract address is generated where all transactions occur, and records are maintained for all the blocks.

The second part displays data immediately after a block is created on the blockchain. This includes information related to the generated block, such as block number, block hash, transaction hash, gas used, and other parameters recorded on the block and provided by the RTDS, Figure 4.5 displays the actual view of the user dashboard.

4.3.4 Data Storage Scheme

Data is stored in blocks created on the Ethereum blockchain network and is structured using Merkle trees, which provide a hierarchical and efficient way to organize and verify large datasets. Each leaf node represents a transaction, and each non-leaf node is a hash of its children, culminating in a single root hash known as the Merkle root. This structure allows for efficient data verification and integrity checking. Figure 4.6 gives the schematic

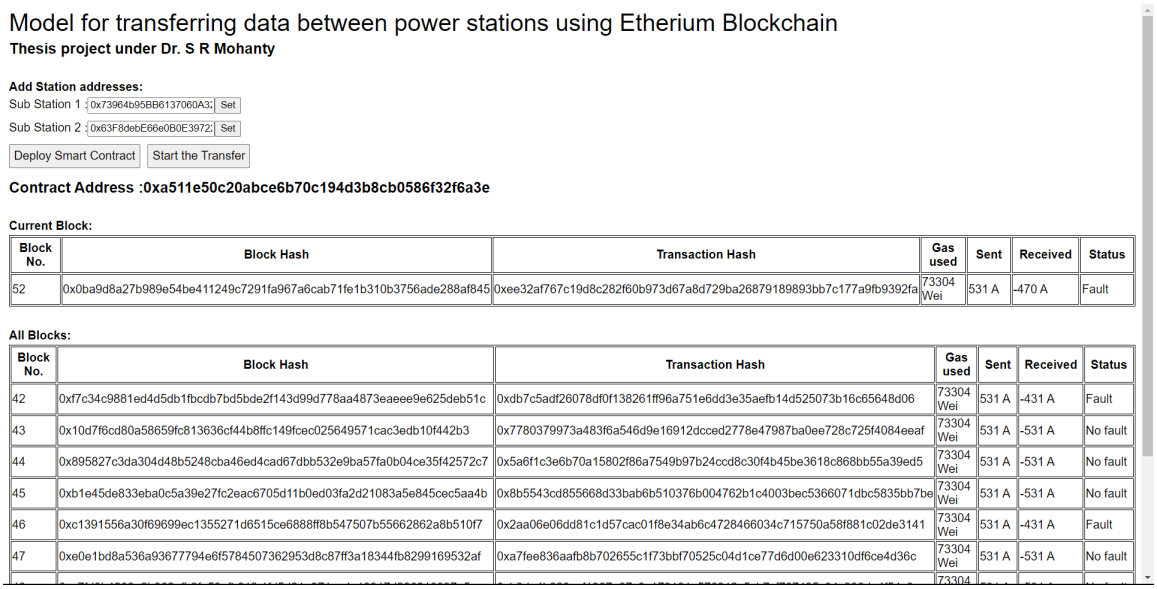


Figure 4.5: User dashboard.

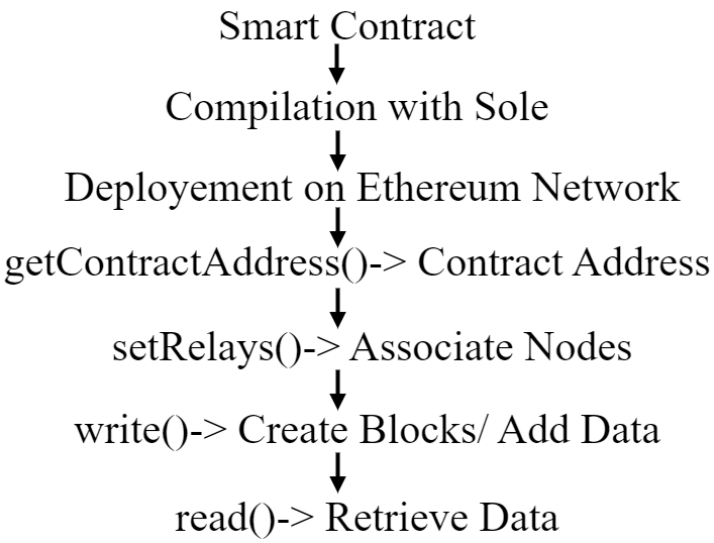


Figure 4.6: Data updation process.

of the process.

Key Functions in Data Management

1. getContractAddress(): This function deploys the smart contract and returns its address, serving as a unique identifier for accessing and interacting with the contract.

2. setRelays(): Associates specific addresses with relay nodes, granting them administrative rights to read and write data on the smart contract. This function ensures that only authorized nodes can perform critical operations.

3. write(): Updates the state of the smart contract by adding new data and creating new blocks. This function is essential for maintaining the dynamic nature of the blockchain as it continuously evolves with new transactions.

4. read(): Retrieves the current state of the blockchain, allowing users to access and verify stored data. This function provides transparency and accessibility, ensuring that data can be audited and validated by any network participant.

Security and Integrity

1. Cryptographic Hashing: Cryptographic hashing ensures the integrity and security of the data stored on the blockchain. Each block's hash is unique to its content, and any alteration in the data will result in a completely different hash, making tampering easily detectable.

2. Decentralization: The decentralized nature of blockchain technology enhances data security by eliminating single points of failure. Data is replicated across multiple nodes, and consensus mechanisms ensure that all copies of the ledger are consistent and accurate.

4.3.5 Network Scheme

The network architecture of proposed blockchain-based system leverages the Ganache network for development and testing purposes. Nodes, representing relays, are connected in a decentralized manner, with relays acting as intermediaries to facilitate communication between the nodes and the blockchain network. The backend server coordinates data flow between the RTDS machine, the Ganache blockchain network, and the user dashboard.

The proposed system uses the Ganache network, a personal blockchain for Ethereum development, which simulates the Ethereum blockchain for testing purposes. While Ganache does not employ a real consensus mechanism like PoW, it allows for the simulation of block creation and transaction validation, enabling developers to test and debug their applications in a controlled environment. Figure 4.7 shows the Ganache user interface.

Security is a critical aspect of our network scheme. The cryptographic hash ensures that the hash of each block is unique to its content, making any alteration easily

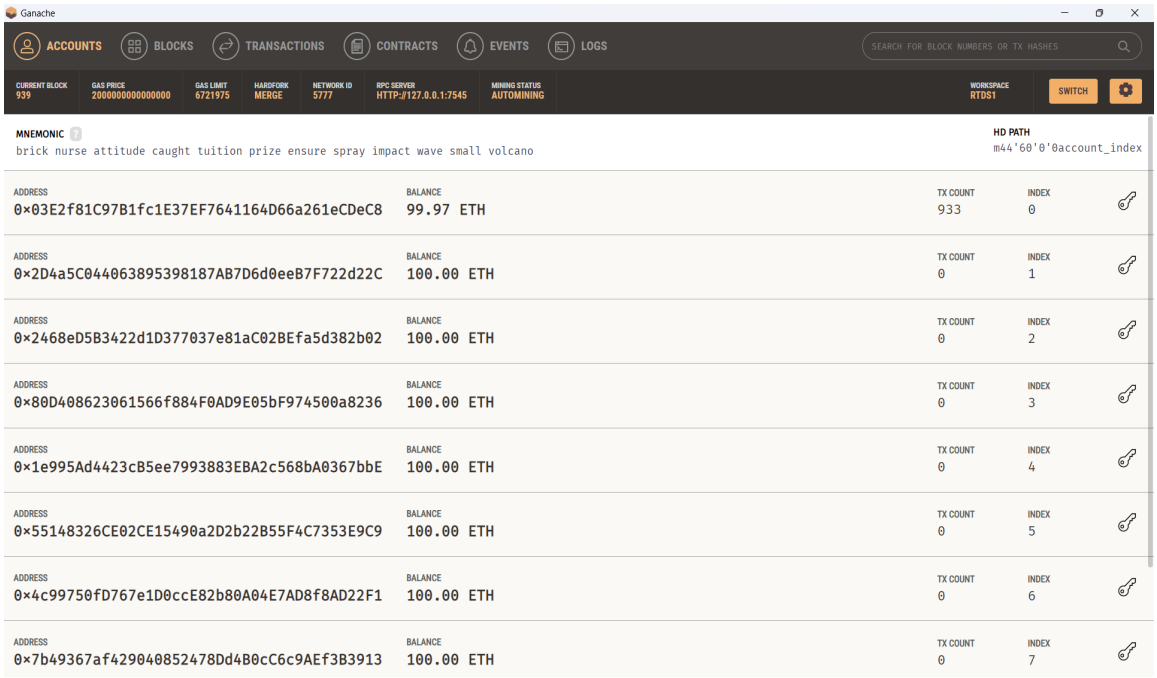


Figure 4.7: Ganache GUI.

detectable. The decentralized nature of the blockchain eliminates single points of failure, as data is replicated across multiple nodes. Smart contracts automate transaction management, providing an additional layer of security by enforcing predefined rules and conditions.

4.4 Blockchain-Based Secure Transmission Mechanism

In this method, the secure transmission of data between two nodes (relays) involves two stages. The first stage includes the communication between RTDS and the host PC using the Python environment through the general-purpose TCP/IP network socket protocol. The second step consists of interaction with the blockchain using Python.

4.4.1 Integration of RTDS with Blockchain Network

The integration of a blockchain-enabled protection scheme for a DC microgrid necessitates real-time data transmission between the RTDS and the blockchain network. This is achieved through the GTNET SKT, which allows the RTDS to communicate with external systems.

The DC microgrid is simulated in RDTS to get the real-time data, such as current

measurements. This data is transmitted over a network socket to the blockchain-enabled system, where it is securely recorded and analyzed. The use of the GTNET SKT facilitates this data exchange, ensuring that the blockchain receives accurate and timely information for processing.

4.4.2 Configuration of User Dashboard

A user dashboard is configured for real-time monitoring and control of data transmission. The dashboard is designed using HTML, CSS, and JavaScript using React.js. Web3.js is used to connect the dashboard frontend to the Ethereum blockchain network. This connection enables our dashboard to interact with smart contracts and retrieve blockchain data.

A specific function is implemented to retrieve relevant data from the blockchain, such as transaction details, block information, and smart contract states. The dashboard has the real-time updates to display live data from the blockchain. A listener is established to monitor blockchain events and update the dashboard interface dynamically whenever new data is available.

The dashboard has functionality to initiate and track transactions directly from the dashboard. It provides features for sending transactions, monitoring transaction status, and viewing transaction history as shown in Fig. 4.5. Following are the detailed walk-through of the user dashboard operation:

1. **Entering station address:**

   The user begins by entering the addresses of the two stations involved in the system. These stations represent the endpoints of the protected line segment in the DC microgrid.

   The addresses are set through the user interface, ensuring that the smart contract will recognize and authorize these stations for reading and writing data.

2. **Deploying the smart contract:**

   After entering the station addresses, the user deploys the smart contract. Deployment involves compiling the contract using Truffle and deploying it to the local Ganache network. This step initializes the smart contract and sets up the blockchain

for data transactions. Upon successful deployment, the smart contract address is displayed on the dashboard, indicating that it is ready for interaction.

3. **Connecting to the system:**

   With the smart contract deployed, the user then connects to the system. This connection process involves initiating a WebSocket connection to the backend server, which listens for incoming data streams from the RTDS.

   Once connected, the dashboard displays a confirmation message, indicating that the system is ready to start transactions.

4. **Starting data transactions:**

   The connection established acts as a client, receiving real-time data from the RTDS via the Python script using the GTNET SKT socket protocol.

   As data is received, it is processed and prepared for blockchain transactions. The data is written to the blockchain using the smart contract's write function, and the transaction details are displayed on the dashboard.

   Each data point is recorded as a transaction on the blockchain, ensuring immutability and security [124]. The dashboard provides real-time monitoring, showing transaction hashes, block numbers, gas used, and the status of each transaction.

4.4.3 Data transmission to NodeJS environment using spawning

The integration of real-time data transmission between RTDS and a NodeJS environment is critical for the blockchain-enabled protection scheme. By using NodeJS to handle data received from RTDS, the data can be efficiently managed and processed [125], ensuring seamless interaction with the blockchain. This process involves spawning a child process in NodeJS to handle the data transmission. To achieve efficient data transmission, the spawning feature in NodeJS is used. Spawning allows us to create child processes that run independently of the main NodeJS application. This is particularly useful for handling data streams from RTDS, enabling us to process and forward the data to the blockchain network without blocking the main event loop of the NodeJS application [126].

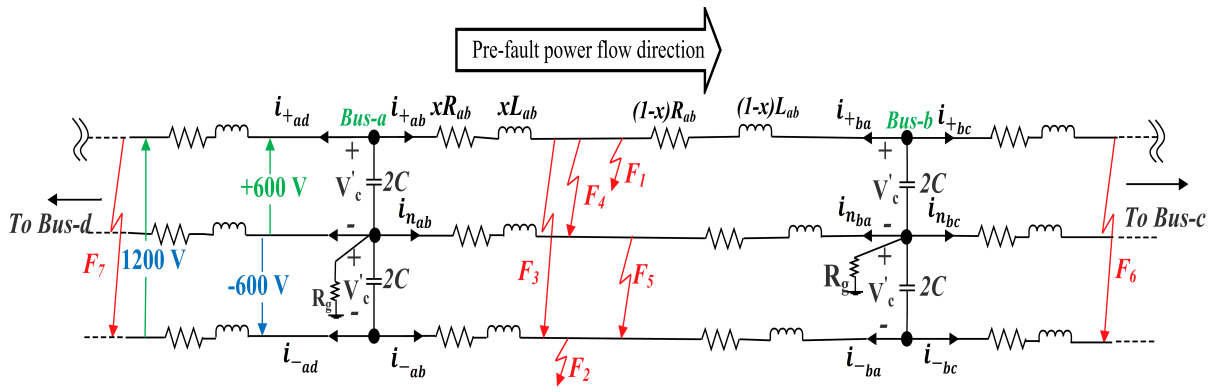


Figure 4.8: A section of bipolar DC microgrid.

Table 4.1: Trajectories of superimposed currents (symmetrical domain) during various faults.

Faults	Internal Fault		External Fault (F_6)		External Fault (F_7)	
	Δi_2	Δi_1	Δi_2	Δi_1	Δi_2	Δi_1
PGF	Q-1	Q-1	Q-4	Q-4	Q-2	Q-2
NGF	Q-1	Q-3	Q-4	Q-2	Q-2	Q-4
PNF	Q-1	Origin	Q-4	Origin	Q-2	Origin

4.5 Protection Scheme

The protection method used in this case is the same as discussed in Sections 3.3 and 3.4 of Chapter 3. Different Internal and External faults are simulated for a bipolar DC microgrid as shown in Fig. 4.8.

Table 4.1 shows the location Δi_2 and Δi_1 in Δi-plane, for different internal and external faults.

4.6 Performance Evaluation

This section presents the evaluation of the proposed solution in terms of overall communication efficiency. The relay at the two ends of the protected line segment (unit protection scheme) is simulated as a virtual node within the considered blockchain network hosted by a machine with i7-12700, 3.6 GHz CPUs and 32 GB memory.

The proposed model is validated for both the transmission of current measurement and the trip signal from the remote end of the protected line segment to the local end.

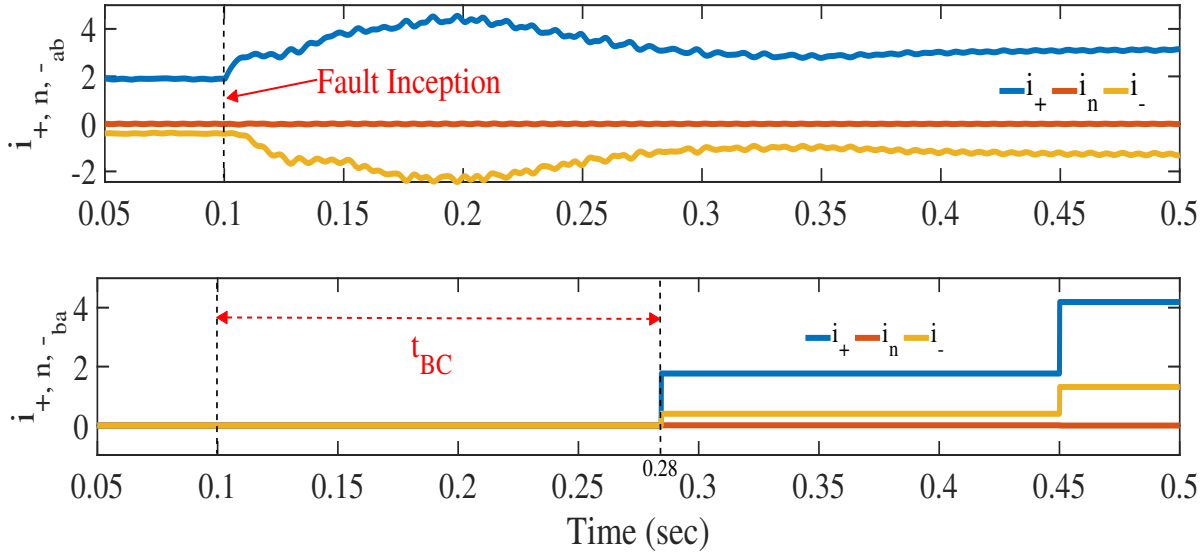


Figure 4.9: Pole domain currents during internal PGF (F_1).

Since the protection scheme has already been validated in Chapter 3, only internal faults are simulated to evaluate the performance of the blockchain-based scheme. The fault inception time is 0.1 sec for all the cases.

Three types of internal faults PGF, NGF, and PPF are simulated and the results are discussed one by one.

4.6.1 Internal PGF

Figures 4.9 and 4.10 show the pole domain and symmetrical domain currents, respectively, at the local and remote ends of the protected line during internal PGF (F_1 in Fig. 4.8). During internal PGF both, $i_{2_{ab}}$, and $i_{2_{ba}}$ increases in positive direction after fault and therefore $\Delta i_{2_{ab}}$ as well as $\Delta i_{2_{ba}}$ are positive. The latency (t_{BC}) in this case is 0.18 sec (Fig. 4.10). It is to be mentioned that the transmission of the data started after the detection of the disturbance.

There are several sections in the real-time simulation model, each with different latencies. The blockchain latency, defined as the time taken to record a transaction and retrieve the updated state from the blockchain, was measured throughout the operation. The trip signal is generated after fault detection using both end data therefore, the overall latency in this case also include the latency due to protection delay (t_{PD}).

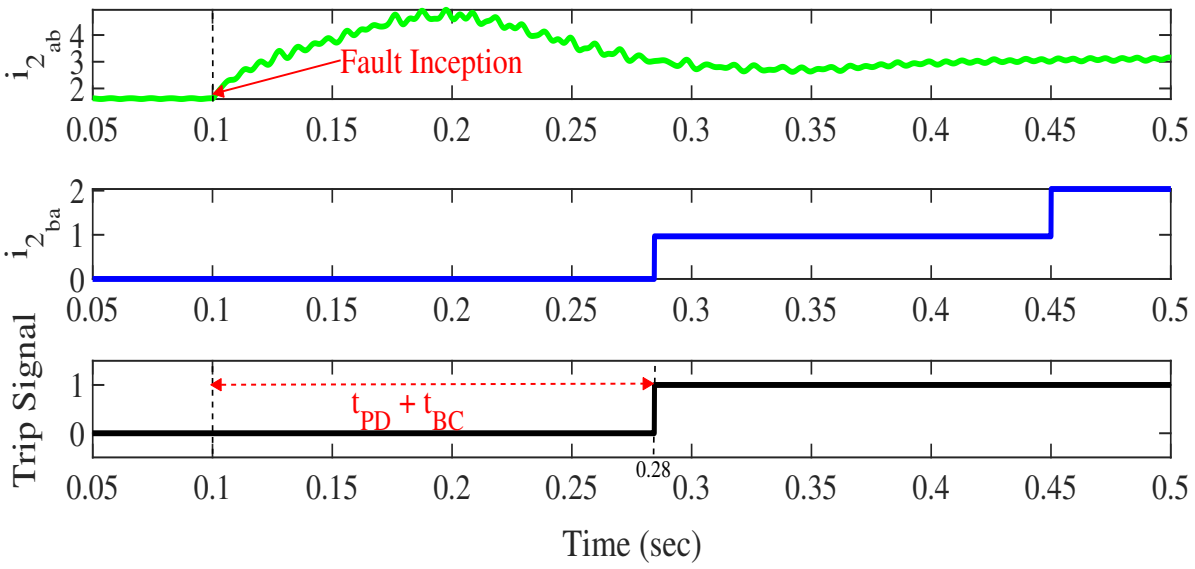


Figure 4.10: Symmetrical domain currents during internal PGF (F_1).

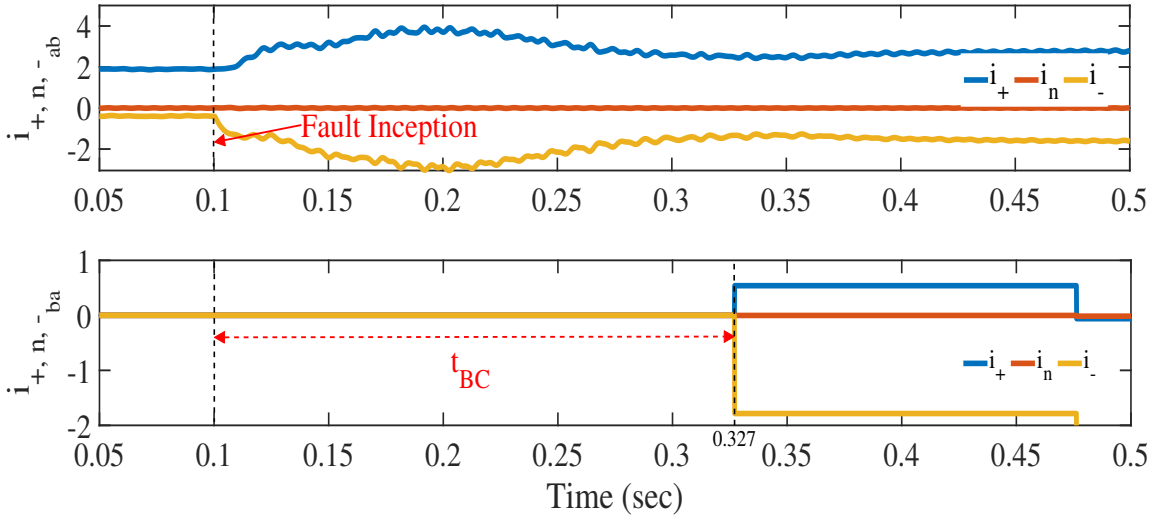


Figure 4.11: Pole domain currents during internal NGF (F_2).

4.6.2 Internal NGF

Figures 4.11 and 4.12 show the pole domain and symmetrical domain currents, respectively, at the local and remote ends of the protected line during internal NGF (F_2 in Fig. 4.8). During internal NGF both, $i_{2_{ab}}$, and $i_{2_{ba}}$ increases in positive direction similar to internal PGF. The latency in this case is 0.22 sec.

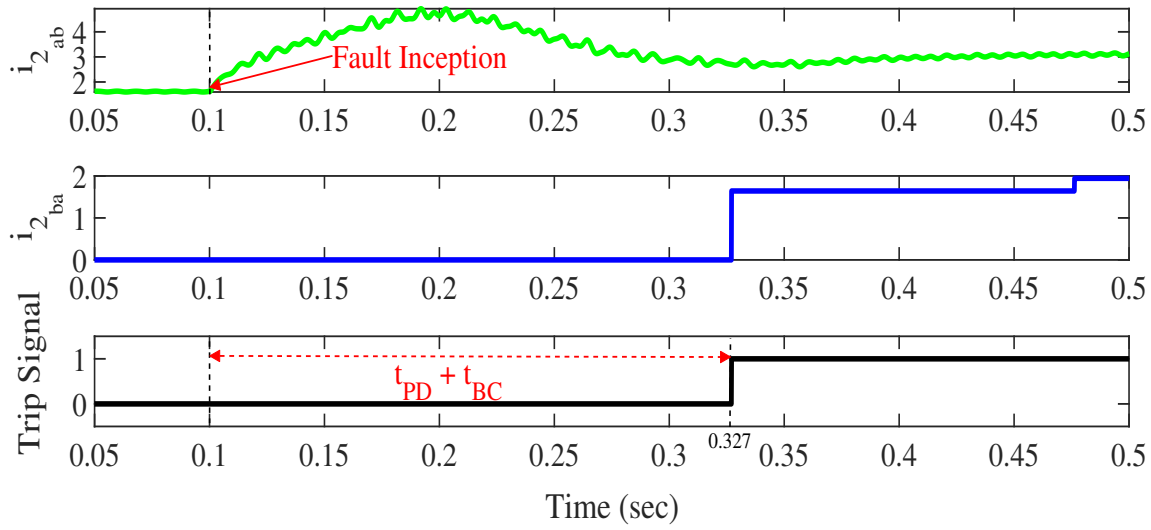


Figure 4.12: Symmetrical domain currents during internal NGF (F_2).

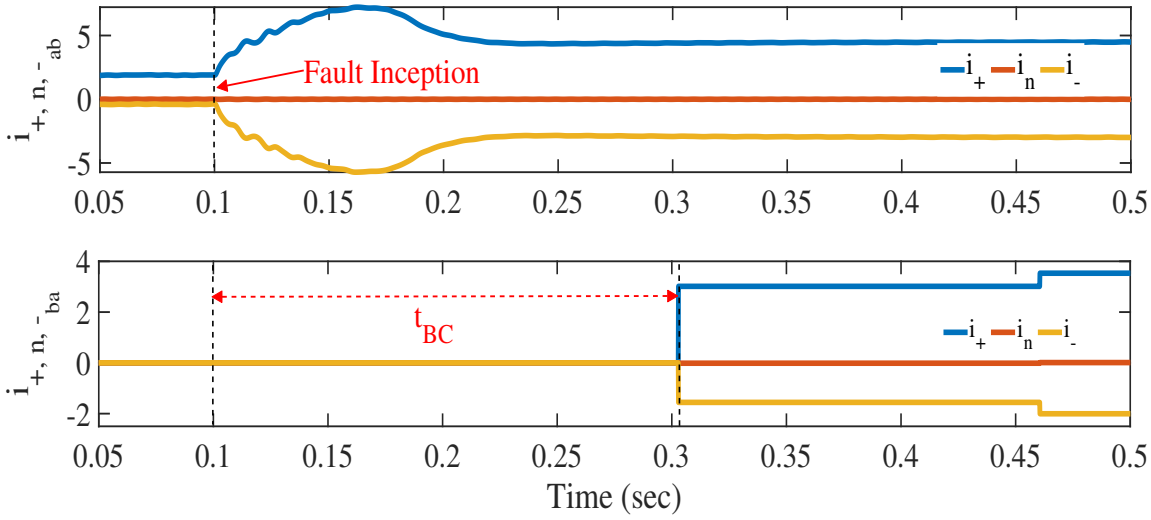


Figure 4.13: Pole domain currents during internal PPF (F_3).

4.6.3 Internal PPF

Figures 4.13 and 4.14 show the pole domain and symmetrical domain currents, respectively, at the local and remote ends of the protected line during internal PPF (F_3 in Fig. 4.8). Similar to all the internal faults, during internal PPF both, $i_{2_{ab}}$, and $i_{2_{ba}}$ increases in positive direction. The latency in this case is 0.205 sec.

Although the proposed method can secure data transmission using blockchain, the latency is the main challenge.

In order to further analyze the data transmission latency, the trip signal (binary signal) is transmitted from one end to the other for different internal fault conditions and

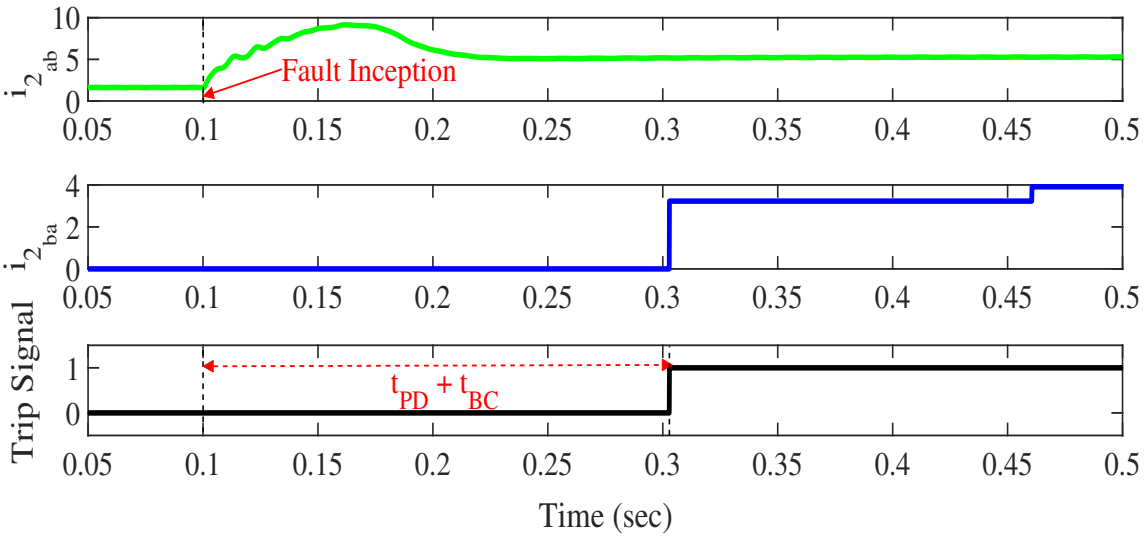


Figure 4.14: Symmetrical domain currents during internal PPF (F_3).

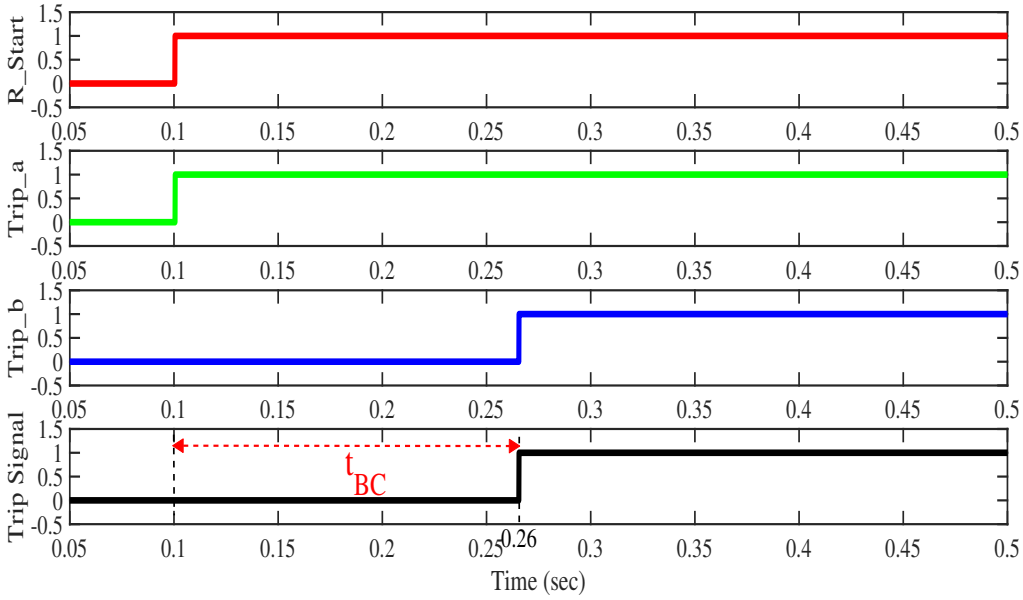


Figure 4.15: Trip signals at bus-a and bus-b during internal PGF (F_1).

the simulation results are given below.

Figure 4.15 shows the simulation result for the internal PGF. Similarly, Fig. 4.16 and Fig. 4.17 show the simulation results during internal NGF and PPF, respectively. The data transmission latency is 0.16 sec for PGF (Fig. 4.15), 0.20 sec during NGF (Fig. 4.16), and 0.15 sec for fault PPF (Fig. 4.17).

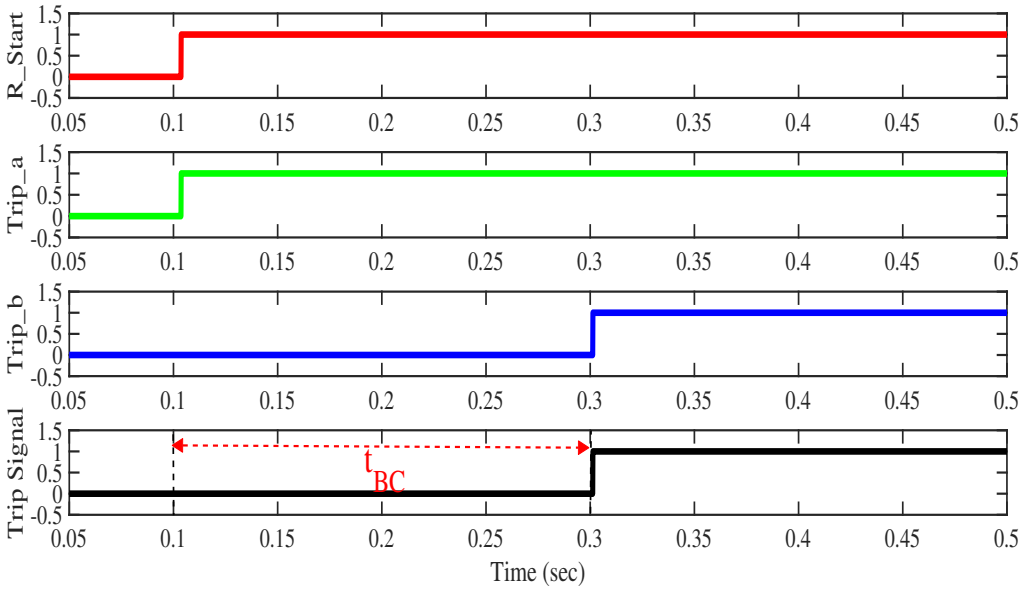


Figure 4.16: Trip signals at bus-a and bus-b during internal PPF (F_2).

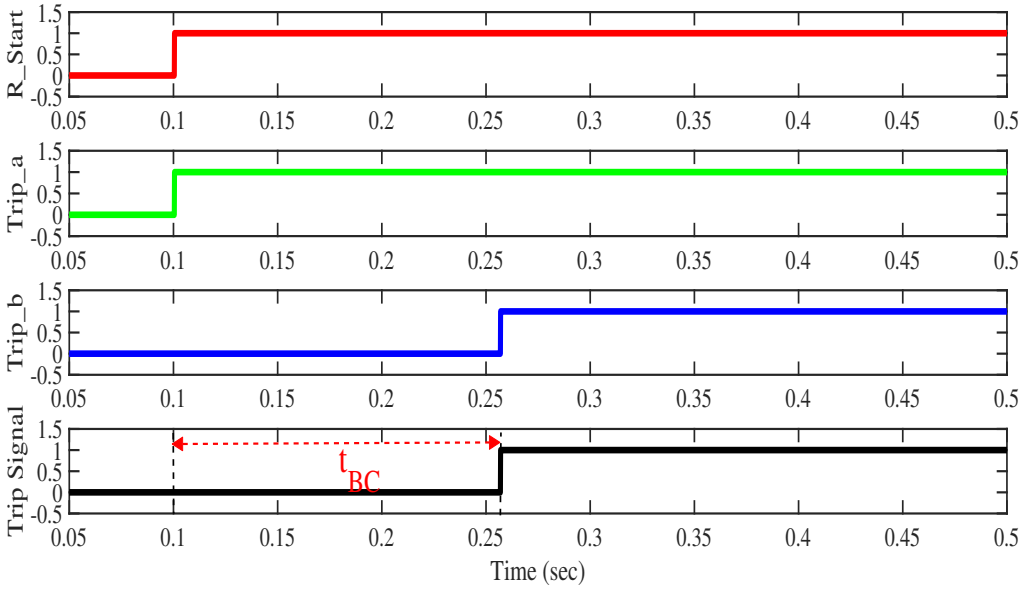


Figure 4.17: Trip signals at bus-a and bus-b during internal PPF (F_3).

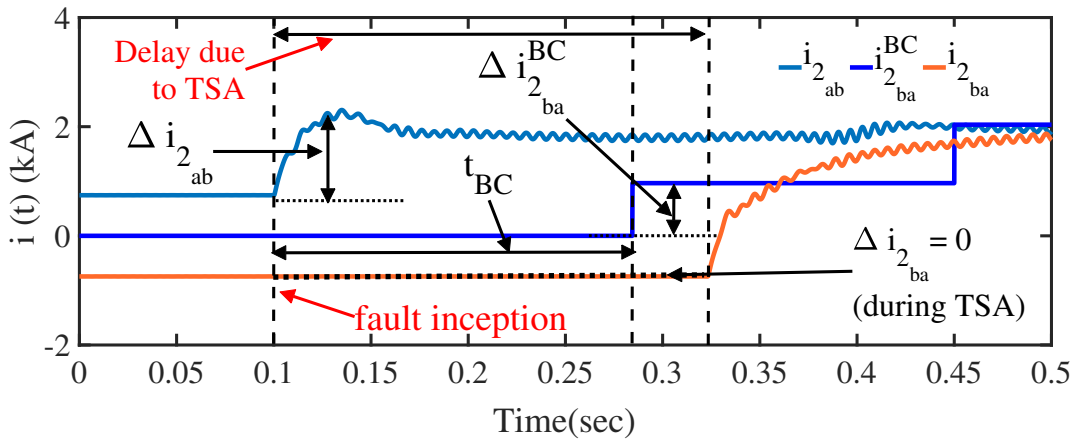


Figure 4.18: $i_{2_{ab}}$, and $i_{2_{ba}}$ during PGF simultaneous to TSA.

4.7 Discussion

The main objective of this chapter is to enhance the robustness of the communication-based protection scheme against cyber attacks. The blockchain-based data communication between the two ends of the protected line segment enhances the robustness of the communication system. Real-time simulation is completed using RTDS to communicate the measurement data for making protection decisions.

The simulation results are compared for a PGF (F_1 in Fig. 4.8) with blockchain-enabled communication and during an attack on both pole measurements as shown in Fig. 4.18. In Fig. 4.18, $i_{2_{ba}}(BC)$ is the value of $i_{2_{ba}}$ communicated using blockchain. It can be observed that the TSA introduced time delay can provide an intentional time delay, which can impact the performance of the fault detection method during internal faults. However, when the measurement data is communicated using tamper-proof communication medium (blockchain), only system system-introduced delay (t_{BC}) is present. In this case, the superimposed component ($\Delta i_{2_{ba}}^{BC}$) is positive, and the internal fault is detected.

Latency in the blockchain-based data communication is important aspect in the proposed method. In order to validate the performance in real-time, two different scenarios are simulated using a 2-peer model relay network:

1. Firstly, the current measurement values are communicated.

2. In the second scenario, the trip signal is communicated in the form of a binary value.

Every stage in the overall system introduces some time delay in the communication. This includes the time delay introduced by the WebSocket connection, which communicates

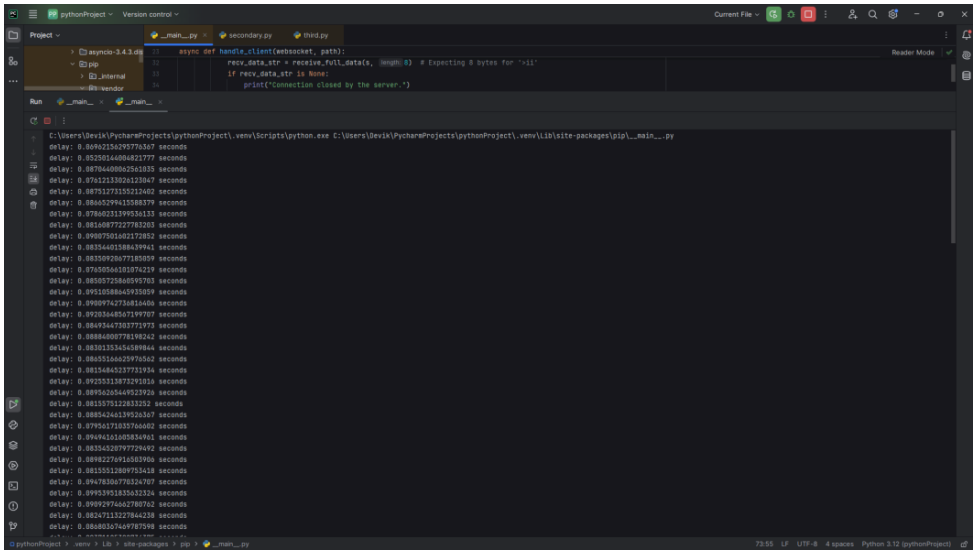


Figure 4.19: Websocket and blockchain, combined latency.

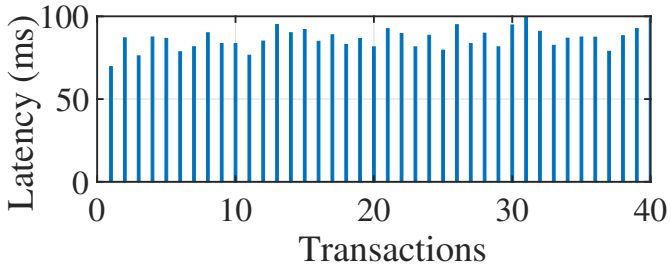
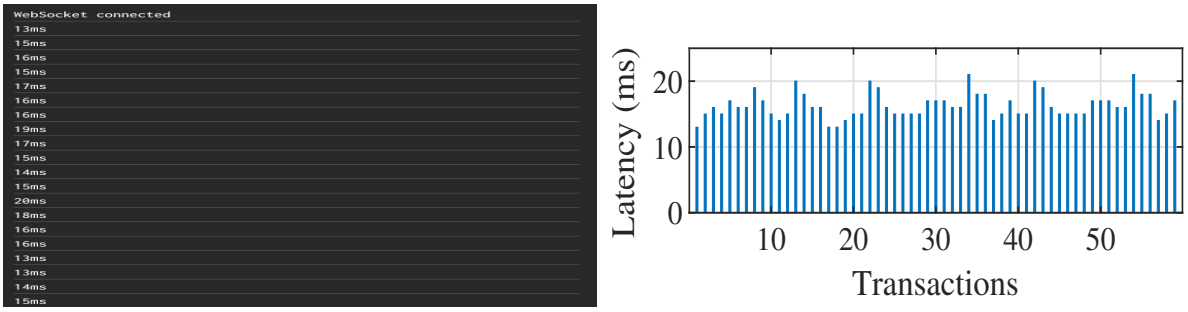


Figure 4.20: Websocket and blockchain, combined latency (bar plot).

data to the blockchain environment. Secondly, there is the latency associated with the socket, which communicates data for the RTDS. The combined latency including websocket and blockchain is given in Fig. 4.19, and Fig. 4.20.

The average latency of 16 milliseconds in blockchain is observed in the real-time environment. This includes transaction signing, network confirmation, and state retrieval. Each transaction's completion time, encompassing signing, sending, and confirmation, averaged 16 milliseconds, as shown in Fig. 4.21. Despite latency, the system maintained real-time data synchronization, facilitating prompt fault response. Future efforts may target latency reduction through blockchain optimization and exploring faster consensus mechanisms, enhancing scheme responsiveness and reliability.



(a) (b)

Figure 4.21: Latency in the proposed scheme (a) screenshot from the module (b) bar plot.

4.8 Summary

The blockchain-enabled protection scheme offers a robust solution for ensuring the security and integrity of the grid's operation. By leveraging blockchain technology, an improved mechanism is established using a decentralized and immutable ledger that records real-time data transactions, providing transparency and accountability In a protection system. This decentralized approach mitigates the risks associated with a single points of failure and cyber-attacks, ensuring a resilient and secure grid infrastructure. The seamless integration of various components, including Ganache, Truffle, ExpressJS, ReactJS, RSCAD, and Python scripts, demonstrates the effectiveness of the approach in building a reliable and efficient protection scheme for DC microgrids. Ganache and Truffle facilitate the development and testing of smart contracts, while ExpressJS and ReactJS power the user-friendly dashboard for intuitive interaction with the system. RSCAD and Python scripts provide robust simulation and control capabilities, ensuring accurate and responsive management of the microgrid.