CHAPTER 5

Development of a Blockchain-enabled Multi-Stage Proposal Verification in Multi-domain SDN Network

5.1 Introduction

In a multi-domain SDN network, where different administrative domains collaborate to provide integrated services, ensuring the integrity of network flows becomes a complex challenge. The seamless flow of data across these domains is essential for optimal network performance and user experience. Further, various malicious activities, such as rogue controller attacks, replay attacks, and Distributed Denial of Service (DDoS) attacks introduce vulnerabilities that can compromise flow integrity, leading to unauthorized flow manipulation and service disruptions [99]. Rogue controller attack is a type of attack in an SDN network in which the SDN controller is compromised to install malicious or unauthorized OpenFlow rules. These altered rules can disrupt network traffic, bypass security policies thereby undermining the trust model of the SDN architecture [108]. These altered rules can disrupt network traffic, bypass security policies thereby undermining the trust model of the SDN architecture. Various methods have been proposed in the literature to address these issues. However, the existing security mechanisms are often inadequate in addressing these issues, especially in dynamic and large-scale networks where multiple controllers operate independently. Further, most of the existing work has tried to compare the similarity of the flow rules installed on each domain to preserve the integrity of flow rules in a multi-domain network. However, this mechanism is not suitable in the practical scenario as each domain maintains a separate network and thus will have different flow rules. Researchers have started adopting Blockchain technology to secure the SDN networks. It is known for its decentralized, tamper-resistant, and transparent nature, and presents a promising solution to the security challenges faced by multi-domain SDN networks. By utilizing a distributed ledger, blockchain can ensure that all flow modification requests are securely recorded and verified across multiple domains.

The rest of the chapter is organized as follows: Section 5.2 presents the system architecture and detailed design of the proposed model. Section 5.3 describes the experimental setup and evaluation of the performance and security of the proposed work. Section 5.4 provides a discussion of the findings and limitations. Finally, Section 5.5 concludes the chapter and suggests directions for future research.

5.2 Proposed Model

In this chapter, we propose a blockchain-based approach to secure flow modification requests in multi-domain SDN networks called Cross-DistBlock. Our solution leverages the decentralized and tamper-resistant properties of blockchain to create a distributed trust model that enhances the security and integrity of flow modification requests. The system incorporates multi-stage verification of proposals using digital signatures, verification of participating controllers for consensus, consensus mechanisms, and adaptive policy enforcement to detect and mitigate malicious activities. Verification is further reinforced by a decentralized chaincode, which disseminates security event information across the network. Further, the adaptive policy enforcement mechanism dynamically responds to detected threats such as rogue controllers, replay attacks, and DDoS attacks by installing appropriate network policy on the OpenFlow switches.

Cross-DistBlock distinguishes itself by implementing a multi-stage flow verification process, which includes digital signatures, controller trust selection, and consensus-based validation, offering far more comprehensive security than the existing reputation-focused checks in BCS [8], BMC-SDN [22], and FRChain [101]. Additionally, the adaptive policy enforcement of Cross-DistBlock dynamically adjusts controller privileges based on real-time trust scores, allowing immediate restriction of suspected or rogue controllers. BMC-SDN employs a constant fading reputation system that forgets past actions at a steady rate, making it susceptible to attackers who intermittently switch between malicious and non-malicious actions. Similarly, BCS does not account for cumulative behavior, instead focusing on shorter-term reputation scores that may miss sporadic threats. By continuously assessing each controller's history, the proposed method minimizes vulnerabilities that could arise from short-term compliance periods. The proposed method also facilitates inter-domain collaboration through chaincode-based threat intelligence sharing, which addresses a significant limitation in BCS, BMC-SDN, and FRChain, where cross-domain coordination and response are absent. A detailed description of the architecture designed is explained in the following subsection.

5.2.1 Architecture

The proposed architecture for preserving flow integrity in a multi-domain SDN network leverages digital signatures, consensus mechanisms, and smart contracts on the blockchain (refer Figure 5.1). A detailed explanation of the architecture components and their interactions is given in the following subsection. As illustrated in the figure, the system operates in a multi-domain SDN environment where each domain is equipped with OpenFlow-enabled switches and an SDN controller integrated with blockchain functionality. When a flow modification request is initiated by a controller (Step 1), the controller first captures the essential flow attributes such as source and destination IP addresses, port numbers, MAC addresses, the intended action (forward or drop), and the assigned priority level. This information is packaged into a proposal object, which is digitally signed using the controller's private key to ensure integrity and authenticity. For intra-domain proposals, the signed proposal is sent only to peer controllers within the same domain for validation and consensus. For inter-domain proposals, the initiating controller communicates with controllers from other domains, triggering a broader validation process that must comply with global trust policies. The Certificate Authority (CA) plays a crucial role at this stage by authenticating the controller's identity and verifying its digital signature (Step 3). Once authenticated, endorsing peers/controllers verify the correctness of the proposal against the network state stored on the blockchain. This includes applying trust management rules and adaptive policy enforcement (Step 2), as defined in the deployed smart contract (chaincode). If the required number of endorsements is achieved, the proposal is forwarded to the Ordering Service (Step 4), which batches validated proposals into blocks, ensuring transaction ordering and consistency. The resulting block is then committed to the blockchain ledger, becoming an immutable record of the network state. Finally, the block is broadcast across all participating domains (Step 5) to maintain a unified and tamper-proof network view. In the next subsection, we formulate the system design of Cross-DistBlock.

5.2.2 System Model

The proposed model comprises a set of different administrative domains within the SDN network. Let D be the set of participating domains then D=$(D_1, D_2, ..., D_n)$. Each domain D_i has

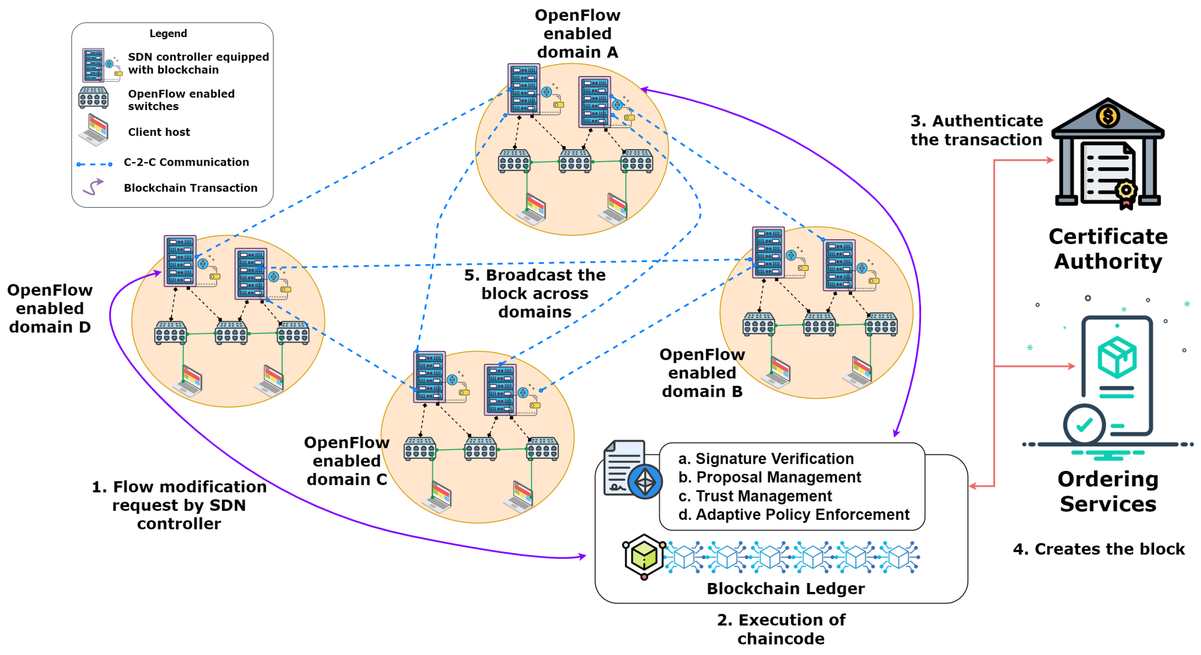
Legend
SDN controller equipped with blockchain
OpenFlow enabled switches
Client host
C-2-C Communication
Blockchain Transaction
OpenFlow enabled domain A
OpenFlow enabled domain B
OpenFlow enabled domain C
OpenFlow enabled domain D
1. Flow modification request by SDN controller
2. Execution of chaincode
3. Authenticate the transaction
4. Creates the block
5. Broadcast the block across domains
Certificate Authority
Ordering Services
a. Signature Verification
b. Proposal Management
c. Trust Management
d. Adaptive Policy Enforcement
Blockchain Ledger

Figure 5.1: Proposed Multi-Domain verification of network flows in SDN using Blockchain

its own set of controllers and OpenFlow switches). These domains can communicate with each other through inter-domain flow rules installed on the switches.

Multiple SDN controllers are located within each domain and are responsible for managing flow rules on OpenFlow switches. Let $C_{i1}, C_{i2}, \ldots, C_{ik}$ participate in verifying flow modification proposals from domain D_i. The network administrator assigns a trust value to each of the controllers at the time of network deployment. Let Γ_{ik} be the trust score of the controller C_{ik} in domain D_i. This trust score ensures that rogue controllers are excluded from the consensus process during proposal verification.

Let $S = (S_1, S_2, \ldots, S_n)$ be the OpenFlow switches managed by SDN controllers from domain D_i. These switches handle data packet forwarding based on flow rules installed. OpenFlow rules determine how data packets should be handled. Let F_j be the flow rules which contain the information about the match and instruction fields of flow rule structure.

The smart contract (chaincode) manages the proposal creation, verification, and consensus process as well as the trust score of each controller to prevent rogue controllers from participating in the verification process.

5.2.2.1 Proposal creation and Signature

When a controller C_{ik} in domain D_i needs to modify a flow rule, it creates a flow modification proposal. Let P_{ij} be the flow modification proposal from domain D_i regarding flow F_j.

The structure of F_j is defined by the following-

$$F_j = (dpid, inport, src\_IP, dst\_IP, src\_mac, dst\_mac, action, priority) \tag{5.1}$$

For each domain D_i and flow F_j, a proposal is created:

$$P_{ij} = Proposal(D_i, F_j) \tag{5.2}$$

This proposal is digitally signed using the controller's private key as-

$$S_{ij} = Sign(P_{ij}, PrivateK_{D_i}) \tag{5.3}$$

Where $Sign()$ is the signing function and D_i is the private key.

This proposal and signature are forwarded to the blockchain in the form of transactions. This transaction invokes the smart contract function for the proposal creation. Transaction of the proposal is presented as:

$$T_{ij} = \{P_{ij}, S_{ij}\} \tag{5.4}$$

5.2.2.2 Identification of Proposal Type

The controller has a global view of a particular domain. Therefore, controllers have the information of all the hosts connected to it. When a host requests a proposal creation, it checks the destination IP from the proposal structure. If the destination IP belongs to a host within the same domain, it is an intra-domain proposal. Otherwise, it is an inter-domain proposal.

To identify the proposal type we define a mapping function to determine the corresponding domain of a particular proposal. Let $M_d(IP)$ be the mapping function that maps an IP address to its corresponding domain.

$$M_d(P_{ij}(IP_j)) = D_i \quad if \;\; IP_j \in D_i \tag{5.5}$$

Now, we determine the domains of source and destination IPs in proposal F_j using the following-

$$D_{src} = M_d(P_{ij}(src\_IP)) \tag{5.6}$$

$$D_{dst} = M_d(P_{ij}(dst\_IP)) \tag{5.7}$$

Finally, we compare the domains of source and destination IPs of the proposal to determine the proposal type.

$$P\_Type = \begin{cases} Intra-Domain, & \text{if } D_{src} = D_{dst} \\ Inter-Domain, & \text{if } D_{src} \neq D_{dst} \end{cases}$$

This process of proposal type identification is necessary to trigger the appropriate security policies and consensus mechanisms.

5.2.2.3 Proposal Verification

After the identification of the proposal type, the next step is proposal verification. In the proposed work, the proposal verification involves into the following three stages:

a) **Stage I: Verification of the proposal's digital signature:** The first stage of proposal verification is to ensure the authenticity and integrity of the proposal using digital signatures. During the proposal creation, the controller signs the proposal using their private keys. Therefore, the smart contract uses the proposer's public key to verify the digital signature using the following function-

$$V_{ijk} = VerifySign(P_{ij}, PublicK_{C_{ik}}) \tag{5.8}$$

Where $VerifySign()$ is the verification function using C_{ik}'s public key. This verification ensures that the proposal has not been tampered with and that it originates from a legitimate source.

b) **Stage II: Verification of participating controllers:** Once the proposal's authenticity is confirmed, the next step is to evaluate the controllers involved in the verification process. We divide the controllers into three categories based on their behavior. These are- $Trusted$ (Υ), $Suspected$ (Ψ), and $Rogue$ $(\Re)$. Initially, all the controllers are allotted to the $Trusted$ list and are assigned a trust score (Γ) of one. Once, the controllers provide an invalid vote during the controller consensus their trust score is deducted by a factor of $(\delta = 0.1)$ and re-allotted to the $Suspected$ list. Finally, if the trust score goes below threshold $(\lambda = 0.6)$ then the controller is re-allotted into the $Rogue$ list. The categorization of controllers can be represented as

$$\Upsilon = \{C_{ik} \mid \Gamma_{C_{ik}} = 1\} \tag{5.9}$$

$$\Psi = \{C_{ik} | \; \lambda < \Gamma_{C_{ik}} < 1\} \tag{5.10}$$

$$\Re = \{C_{ik} | \; \Gamma_{C_{ik}} \leq \lambda\} \tag{5.11}$$

By setting a threshold slightly below one, controllers are warned (by being placed on the *Suspected* list) before being classified as rogue. Immediate addition to the *Rogue* list after a single invalid vote could lead to false positives. Therefore, this buffer period allows for more accurate identification of genuinely malicious or unreliable controllers.

This trust score ensures that rogue controllers are excluded from the consensus process during proposal verification. The trust score for any controller that provides an invalid vote is updated using the following equation-

$$\Gamma_{C_{ik}} = \Gamma_{C_{ik}} - \delta \tag{5.12}$$

When the following condition is met, the controller C_{ik} is included in the rogue list $\Re$.

$$if \;\; \Gamma_{C_{ik}} < \lambda, \;\; then \;\; \Re = \Re \cup \{C_{ik}\} \tag{5.13}$$

The rogue device list is shared in the Blockchain network to ensure all controllers are aware of the rogue controllers and are prevented from future participation in the consensus process. Therefore, this stage ensures that only reliable and trustworthy controllers participate in the proposal verification. Furthermore, this sharing of security events across SDN domains enables a secure dissemination of critical security information through collaboration among different administrative domains.

Next, the smart contract identifies the set of verified controllers that will participate in the consensus process (Stage III). The proposed work divides the participation of controllers into Intra-Domain and Inter-Domain proposal types. The detailed selection procedure is discussed below.

Intra-Domain: For intra-domain proposals, controllers within the same domain validate the proposal through the Verification Smart Contract. This is done to ensure that the verification process is localized, faster, and consumes fewer resources. Furthermore, this optimization prevents unnecessary load on the blockchain network and controllers.

Let D_x be the domain from where the proposal P_{ij} originates. Therefore, only controllers within the same domain D_x participate in the consensus provided these controllers are verified. Let V be the set of controllers that are verified and eligible to participate in consensus. This can be represented as follows-

$$V_{intra} = \{C_{x1}, C_{x2}, \ldots, C_{xk}\}$$
$$such\ that\ \Gamma_{xik} \geq \lambda,\ and\ C_{xk} \notin \Re$$

Inter-Domain: On the other hand, for inter-domain proposals, controllers from all domains participate to validate the proposal through cross-domain communication facilitated by Cross-Domain Contracts. Each controller participates in the consensus mechanism to make a more robust and distributed consensus process involving multiple domains. Furthermore, this helps to maintain a reliable network by penalizing controllers that frequently fail verification, thereby reducing the influence of potentially rogue controllers.

The controllers from all domains participate in the consensus, provided these controllers are verified. This can be represented as follows-

$$V_{inter} = \{C_{ij}\ from\ all\ D_i\}$$
$$such\ that\ \Gamma_{ijk} \geq \lambda,\ and\ C_{ijk} \notin \Re$$

c) **Stage III: Consensus among the controllers:** The final stage involves reaching a consensus among the verified controllers to validate the proposal. The participation of controllers in the consensus process depends on the *P_Type* classification.

Now, based on the *P_Type*, the verified controllers run the consensus among themselves. We utilize a Practical Byzantine Fault Tolerance (PBFT) consensus algorithm for proposal validation. The PBFT consensus algorithm ensures that the network can still achieve consensus and make correct decisions despite the presence of faulty or compromised controllers. Let f be the maximum number of faulty controllers the system can tolerate and n be the total number of participating controllers. Then, according to the PBFT consensus algorithm, there must be a $2f+1$ number of valid responses from the controllers.

The proposed work leverages proposal integrity to ensure that only valid proposals are accepted. Thus, each controller in the valid set (V_{intra}, or V_{inter}) evaluates the proposal independently and votes ($Vote = (v_1, v_2, ..., v_n)$) on its validity.

To validate the integrity of proposals, the controllers recalculate the hash (H'_j) of the received proposal and compare it with the provided hash (H_j). If they match, the corresponding controller votes the proposal as valid. Then, the smart contract collects all the valid votes from the participating controllers using equation 5.14.

$$Vote = \begin{cases} Valid, & \text{if } H'_j = H_j \\ Invalid, & \text{if } H'_j \neq H_j \end{cases}$$

$$\varphi = \sum_{k=1}^{n} Vote(Valid) \tag{5.14}$$

The proposal is accepted if a majority of controllers vote it as valid. If the proposal P_{ij} is rejected, then this will be added to the rogue proposal request.

$$P_{ij} = \begin{cases} Accepted, & \text{if } |\varphi| \geq (2f+1) \\ Rejected, & \text{Otherwise} \end{cases}$$

Once the decision is made about the proposal acceptance, the trust score needs to be updated. This update depends on the invalid votes received from the controllers. Therefore, the trust score is decremented for invalid vote providers by a factor δ using equation 5.12. Finally, the controllers are re-categorized based on the conditions set in equation 5.9, 5.10, and 5.11.

Therefore, this thorough verification process prevents rogue proposals from compromising the network and ensures that only legitimate and compliant proposals are executed.

5.2.2.4 Adaptive Policy Enforcement

There are two lists of controllers that require a policy adjustment (i.e. *Suspected* and *Rogue* list) when the proposal is rejected. The suspected controllers are less prone to malicious attacks on the network compared to the complete rogue controllers. Therefore, we designed two policies with appropriate actions considering the severity. The controllers from the *Suspected* list are treated with leniency and allowed only to participate during the controller consensus for voting. However, they are restricted from performing proposals for insertion and deletion of existing OpenFlow rules. On the other hand, the controllers from the *Rogue* list are treated strictly and completely isolated from the rest of the network by installing a tight policy on the OpenFlow switches.

We define a list of actions denoted as $A = \{A_1, A_2, A_3, A_4\}$ which are executed by the controller based on their behavior in the proposed work. Each of these actions is initiated by controllers and based on the controller category appropriate actions are executed. The list of actions with their description is presented in Table 5.1.

The actions listed in Table 5.1 are either allowed or restricted based on the controller category. In the below, we discussed the network policies for each category.

a) Trusted Controllers: Trusted controllers have full capabilities within the network, in-

Table 5.1: Action list with their description.

Action	Function Name	Description
A_1	$Proposal(D_i, F_j)$	Proposal creation for insertion and deletion
A_2	$VerifySign(P_{ij}, PublicK_{C_i})$	Signature verification of a controller
A_3	$Vote(v)$	Controller votes regarding the proposal integrity
A_4	$Block(P_{ij})$	Block all traffic from C_{ik}

cluding proposing new flow rules, deleting existing rules, and participating in consensus voting. All actions mentioned in the table are allowed. The network policy $(P1)$ can be represented as-

$$P1 = \begin{cases} allowed(A_1, A_2, A_3, A_4), & \forall\, C_{ik} \in \Upsilon \\ restricted(\emptyset), & \forall\, C_{ik} \in \Upsilon \end{cases}$$

b) Suspected Controllers: Controllers in the Suspected List are allowed to participate in the consensus voting process but are restricted from proposing new rules or deleting existing rules. The network policy $(P2)$ can be represented as-

$$P2 = \begin{cases} allowed(A_2, A_3), & \forall\, C_{ik} \in \Psi \\ restricted(A_1, A_4), & \forall\, C_{ik} \in \Psi \end{cases}$$

c) Rogue Controllers: Controllers in the Rogue List are completely isolated and cannot participate in any network activities. The network policy $(P3)$ can be represented as-

$$P3 = \begin{cases} allowed(\emptyset), & \forall\, C_{ik} \in \Re \\ restricted(A_1, A_2, A_3, A_4), & \forall\, C_{ik} \in \Re \end{cases}$$

Therefore, the proposed adaptive policy enforcement mechanism adjusts the capabilities of controllers based on their trust scores making it secure and resilient against the suspected and rogue controllers. This approach ensures that appropriate policies are enforced in real time whenever rogue activities are detected.

We have also prepared a flowchart for better visualization of the entire proposal verification process (refer Fig 5.2). The flowchart visualizes the steps from initiating a proposal request to the final decision of policy enforcement, emphasizing the role of different types of controllers (Trusted, Suspected, Rogue) at each stage.

A user initiates a flow modification request by submitting a proposal to the controller. This proposal is converted into a transaction and sent to the blockchain network. Next, the smart contract verifies the digital signature of the proposal using the proposer's public key. Once the digital signature is verified the SC determines the proposal type by comparing the source and destination IP of the proposal. However, if the signature verification fails, the proposal is discarded and no further action is taken. Therefore, stage (I) prevents unauthorized or malicious entities from injecting false proposals into the network.

In stage (II), the controllers are divided into two sets (inter-domain and intra-domain) to determine who will participate in the consensus process considering the trust score of each controller. Cross-DistBlock handles inter-domain proposal requests by including controllers from all participating domains in the consensus process, but only if they've been verified as non-rogue. This verification step ensures that only trusted controllers contribute to the decision-making process, thereby minimizing risks of compromised controllers influencing the outcome. Additionally, Cross-DistBlock employs chaincode to share threat intelligence across domains, so each domain is kept informed of any flagged controllers from other domains. This shared information strengthens inter-domain trust by allowing for coordinated responses to any emerging threats. By maintaining transparency and tracking trust scores across domains, Cross-DistBlock effectively mitigates cross-domain trust issues and limits the impact of potentially compromised controllers, even when domain interests may vary.

Stage (III) involves the consensus for the decision of proposal acceptance. Each participating controller independently evaluates the hash of the proposal and compares it with the provided hash to validate the integrity of the proposal. All the valid votes are collected by the SC and compared against the threshold set by the PBFT algorithm. Therefore, if the majority of

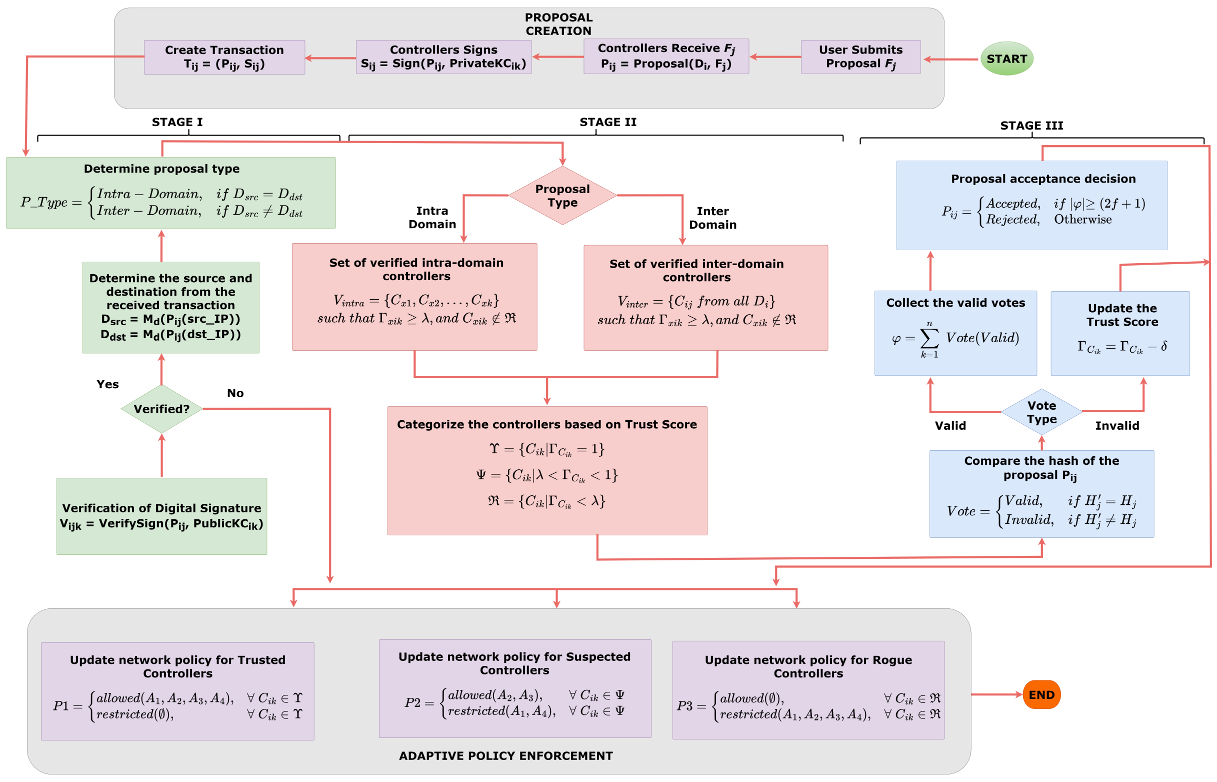
PROPOSAL CREATION
Create Transaction $T_{ij} = (P_{ij}, S_{ij})$
Controllers Signs S_{ij} = Sign(P_{ij}, PrivateKC$_{ik}$)
Controllers Receive F_j P_{ij} = Proposal(D_i, F_j)
User Submits Proposal F_j
START
STAGE I
STAGE II
STAGE III
Determine proposal type
$P\_Type = \begin{cases} Intra-Domain, & if\ D_{src} = D_{dst} \\ Inter-Domain, & if\ D_{src} \neq D_{dst} \end{cases}$
Proposal Type
Intra Domain
Inter Domain
Set of verified intra-domain controllers
$V_{intra} = \{C_{x1}, C_{x2}, \ldots, C_{xk}\}$ such that $\Gamma_{xik} \geq \lambda$, and $C_{xik} \notin \mathfrak{R}$
Set of verified inter-domain controllers
$V_{inter} = \{C_{ij}$ from all $D_i\}$ such that $\Gamma_{xik} \geq \lambda$, and $C_{xik} \notin \mathfrak{R}$
Proposal acceptance decision
$P_{ij} = \begin{cases} Accepted, & if\ |\varphi| \geq (2f+1) \\ Rejected, & \text{Otherwise} \end{cases}$
Determine the source and destination from the received transaction
$D_{src} = M_d(P_{ij}(src\_IP))$
$D_{dst} = M_d(P_{ij}(dst\_IP))$
Yes
No
Verified?
Verification of Digital Signature
V_{ijk} = VerifySign(P_{ij}, PublicKC$_{ik}$)
Categorize the controllers based on Trust Score
$\Upsilon = \{C_{ik} | \Gamma_{C_{ik}} = 1\}$
$\Psi = \{C_{ik} | \lambda < \Gamma_{C_{ik}} < 1\}$
$\mathfrak{R} = \{C_{ik} | \Gamma_{C_{ik}} < \lambda\}$
Collect the valid votes
$\varphi = \sum_{k=1}^{n} Vote(Valid)$
Update the Trust Score
$\Gamma_{C_{ik}} = \Gamma_{C_{ik}} - \delta$
Vote Type
Valid
Invalid
Compare the hash of the proposal P_{ij}
$Vote = \begin{cases} Valid, & if\ H'_j = H_j \\ Invalid, & if\ H'_j \neq H_j \end{cases}$
Update network policy for Trusted Controllers
$P1 = \begin{cases} allowed(A_1, A_2, A_3, A_4), & \forall\ C_{ik} \in \Upsilon \\ restricted(\emptyset), & \forall\ C_{ik} \in \Upsilon \end{cases}$
Update network policy for Suspected Controllers
$P2 = \begin{cases} allowed(A_2, A_3), & \forall\ C_{ik} \in \Psi \\ restricted(A_1, A_4), & \forall\ C_{ik} \in \Psi \end{cases}$
Update network policy for Rogue Controllers
$P3 = \begin{cases} allowed(\emptyset), & \forall\ C_{ik} \in \mathfrak{R} \\ restricted(A_1, A_2, A_3, A_4), & \forall\ C_{ik} \in \mathfrak{R} \end{cases}$
ADAPTIVE POLICY ENFORCEMENT
END

Figure 5.2: Flowchart of the proposed proposal verification process

controllers (including the Trusted and Suspected ones) vote in favor, the proposal is accepted. Otherwise, the proposal is rejected. On the other hand, if the hash of the proposal does not match, the trust score of the corresponding controller is decremented and forwarded for the network policy update.

Based on the updated trust score, the suspected and rogue controllers will have a new policy installed on them. The adaptive policy enforcement mechanism isolated the rogue controllers by installing OpenFlow rules to block traffic from these controllers. Therefore, the proposed multi-staged approach strengthens the security of the network by ensuring that only authentic, untampered, and properly vetted proposals are accepted. In the next subsection, we discuss the smart contract design of the proposed work.

5.2.3 Chaincode Design

Chaincode is a self-executing program that executes actions without requiring manual intervention once predefined conditions are met. It is similar to the Smart contracts of Ethereum Blockchain. It plays a crucial role in simplifying and automating the proposal verification process in our work. Apart from automation, the chaincode provides a transparent and immutable ledger. Therefore, every action taken by the controller is transparent, providing clear accountability. Once these actions are recorded on the blockchain, they cannot be altered, ensuring the integrity of verification. Further, the chaincode operates in a decentralized manner and therefore, does not rely on a single entity.

The key components of chaincode design require managing the proposal verification, trust score of controllers, and adaptive policy enforcement. We have two structures for proposal and controller. The proposal structure includes details of the proposer, OpenFlow rules parameters, and verification status. The controller structure includes its ID, domain, trust score, and status (Trusted, Suspected, Rogue). The chaincode also includes functionality for adding controllers, categorizing controllers, handling proposal verification, voting for the final decision, and finally updating trust scores.

Algorithm 9 Proposal Creation Algorithm

Input: dpid: Datapath ID of switch, **s_ip**: Source IP, **d_ip**: Destination IP, **in_port**: Ingress Port, **s_mac**: Source MAC address, **d_mac**: Destination MAC address, **action**: Action to perform, **p**: Priority of the proposal

Output: P_ID : Proposal ID

```
1: Initialize a new proposal object
2: P_ID ← $NewProposal(dpid, s\_ip, d\_ip, inport, s\_mac, d\_mac, action, p)$
3: Compute the cryptographic hash of the proposal
4: Hash ← $cHash(Proposal)$
5: Store the proposal on the blockchain
6: Return P_ID
```

Algorithm 9 presents the steps for the submission of a new proposal to the blockchain, which calculates its hash, and stores it. Algorithm 10 presents the voting consensus for proposal integrity. Since we used the PBFT algorithm for the voting among controllers, it allows $\frac{n-1}{3}$ numbers of malicious controllers that provide false votes. Therefore, at least $2f+1$ valid votes are required to reach the consensus despite the presence of faulty controllers. Next, the trust score is decremented for the controller that provides invalid votes (refer Algorithm 11) and controller status is updated based on the trust score.

Algorithm 10 Voting Consensus Algorithm

Input: controller[]: List of valid controller instances, **P_ID**: Flow modification proposal to be validated.

Output: Status : Consensus status (True/False)

```
1:  Compute Fault Threshold
2:  $\mathbf{n} \leftarrow \mathbf{len}(\mathbf{controller}[])$
3:  $\mathbf{f} \leftarrow (\mathbf{n}-\mathbf{1})/\mathbf{3}$
4:  Initialize Vote Count
5:  $\mathbf{validVotes} \leftarrow \mathbf{0}$
6:  Collect Votes
7:  for $c \in controller[]$ do
8:      if verifyProposal(P_ID) then
9:          $validVotes \leftarrow validvotes + 1$
10:     end if
11: end for
12: Check PBFT condition
13: if $\mathbf{validVotes} \geq (\mathbf{2} * \mathbf{f} + \mathbf{1})$ then
14:     return $True$
15: end if
16: return $False$
```

Algorithm 11 Update Trust Score Algorithm

Input: controller[]: List of valid controller instances participating in the consensus

Output: TrustScore[]: Updated trust score of each controller.

```
1:  for $c \in controller[]$ do
2:      if isValid[c] == False then
3:          $TrustScore[c] \leftarrow TrustScore[c] - 0.1$
4:      end if
5:      Classify controller status based on trust score.
6:      Condition 1:
7:      if $TrustScore[c] < 0.1$ then
8:          $Status[c] \leftarrow$ 'Suspected'
9:      end if
10:     Condition 2:
11:     if $TrustScore[c] < 0.6$ then
12:         $Status[c] \leftarrow$ 'Rogue'
13:     end if
14:     Condition 3:
15:     if $TrustScore[c] > 0.6$ then
16:         $Status[c] \leftarrow$ 'Trusted'
17:     end if
18:     Store updated controller state on the blockchain
19: end for
```

5.3 Experimental Results

The experimental evaluation aims to demonstrate the effectiveness, performance, and scalability of the proposed blockchain-based proposal verification system for multi-domain SDN networks. The evaluation focuses on verifying the system's ability to detect, and prevent rogue flow modification requests and maintain network security and integrity. In the next subsection, we discuss the environment setup for the simulation of the proposed work.

5.3.1 Environment Setup

We created a virtual Hyperledger Fabric to simulate the Blockchain network. It is a modular architecture that allows for highly customized configurations, enabling organizations to tailor the network according to specific needs, enhancing scalability and efficiency. Other frameworks such as Ethereum lack this level of modularity. Therefore, Hyperledger Fabric is a more suitable choice for multi-domain SDN environments compared to Ethereum.

We used the Python-based Ryu controller to handle the SDN network. It is a lightweight, flexible, and easy-to-deploy SDN controller, making it ideal for research and prototyping compared to other controllers such as OpenDayLight, Beacon, ONOS, Faucet, Floodlight, etc [34]. However, the ONOS controller also supports modular architecture. These controllers are complex and resource-intensive, which can introduce additional overhead in a research setting.

The experimental setup created for this simulation is presented in Figure 5.3. We used four docker containers to deploy and run the experiment. The first container is used for Certificate Authority (CA) that manages certificates for network participants, ensuring secure identity management. The ordering services are deployed on the second container. They maintain a log of all transactions and deliver it to the appropriate peers for validation. The remaining two containers are used to deploy two separate administrative domains in which virtual topology and chaincode are deployed. The chaincode encapsulates the logic for adding controllers, verifying proposals, managing trust scores, and applying adaptive policies on the switches.

We created a custom Python script that generates the virtual topology in the Mininet. The topology configuration is presented in Table 5.2. We used a linear topology in our experi-

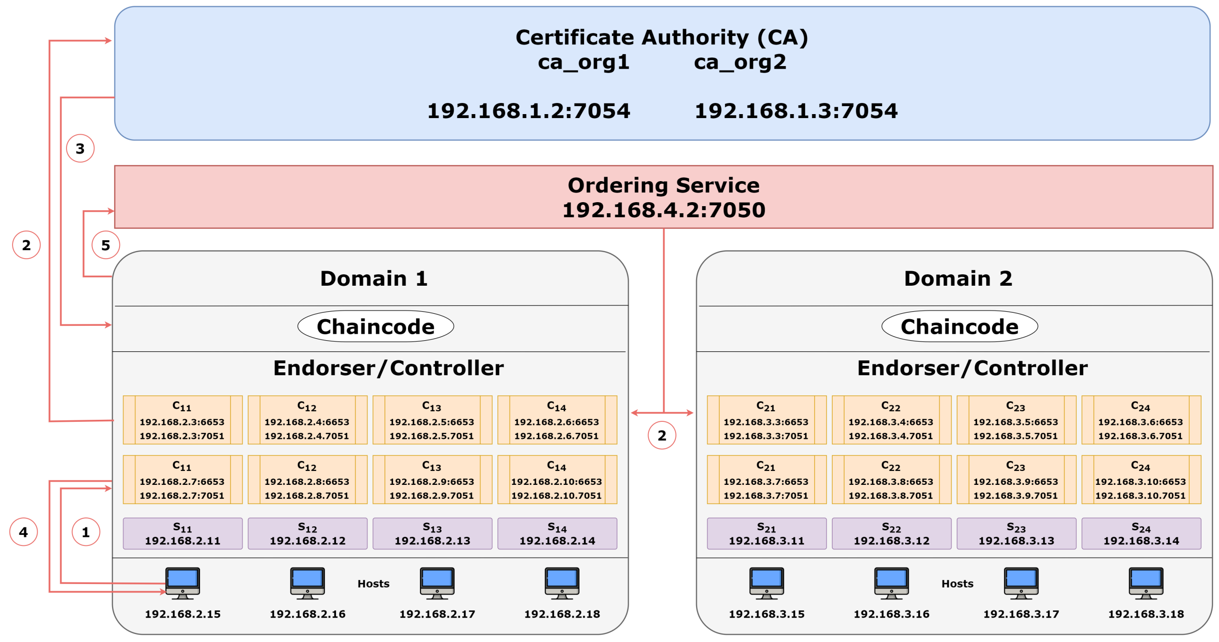

Figure 5.3: Experimental topology for simulation of the proposed work.

ments. This choice was made to facilitate clear and manageable interactions between controllers and domains, enabling a straightforward analysis of performance and security metrics. Two sets of controllers are created, one for each domain. We selected 16 controllers, with 8 per domain, to align with the PBFT consensus requirements, allowing for a minimum of 2 controllers to act as rogue in each domain during the intra-domain proposal verification experiment without compromising the overall consensus. These controllers act as endorsers and validators in the consensus process. The switches can communicate with the controllers through the southbound protocol called OpenFlow (version 1.3).

Table 5.2: SDN Configuration.

Devices	Configuration
SDN Controller	Ryu (Version)
No. of Domains	2
No. of Controllers	16 (8 per domain)
No. of Switches	8 (4 per domain)
No. of Hosts	8 (1 per switch)

The flow of messages for every transaction follows the following steps.

a) Submits flow modification proposal: The client/host sends the flow modification proposal to the SDN controller for endorsement (Marked with 1 in Figure 5.3).

b) Endorsement of the proposal: The certificate authority checks the membership of the endorser/controller to authenticate the transaction (Marked with 2). Then it executes the chaincode that is deployed for the proposal verification (Marked with 3). If the authentication fails, the endorsement result is returned to the host (Marked with 4).

c) Ordering Service: After the transaction has been approved by the peers, the transaction is sent to the ordering services (Marked with 5). The orderer then creates a block and broadcasts it to all the peers from different domains (Marked with 6).

d) Update the ledger: Finally, the newly created block is updated in the ledger by network peers/controllers.

5.3.2 Performance evaluation

In this subsection, we measure the performance of the proposed method in terms of latency due to the signature verification, voting scheme, and adaptive policy enforcement to check the strength of the proposed work. In our comparative analysis, we evaluated the performance of our proposed method against three other blockchain-based approaches: FRChain [101], BCS [8], and BMC-SDN [22]. These methods have been widely recognized for their effectiveness in securing SDN networks, particularly in scenarios involving rogue controllers and other malicious activities.

We carefully implemented the BCS, BMC-SDN, and FRChain methods to enable a meaningful comparison with our proposed Cross-DistBlock approach. To maintain consistency, we used Mininet and the Ryu controller as our primary network emulation environment, aligned with the environment configurations used in each original method. The smart contracts from each of the existing methods were translated to chaincode, as this allowed us to standardize the blockchain implementation across our test cases. Each competing method was implemented according to the original paper's specifications. The constant fading reputation system of BMC-SDN was configured to match its original decay parameters, ensuring that the forgetting rate mirrored the behavior outlined in their methodology. Similarly, BCS was set up with two redundant sub-controllers as per its specifications, and FRChain was optimized for a small-scale network to reflect the original method's system constraints. To evaluate the performance consistently, we used the same performance metrics specified in the original studies, such as attack detection rate, consensus latency, and attack detection time. This ensured that the results accurately reflected the effectiveness of each method. Where specific network conditions or parameters (e.g., traffic volumes, attack types) were mentioned in the original studies, we applied the same conditions to replicate the intended test environment.

5.3.2.1 Latency

The total latency incurred in the proposal verification process can be a combination of three main components- Time for digital signature verification (T_{sign}), voting time (T_{voting}), and policy enforcement time (T_{policy}). Therefore, the latency of the proposed work can be expressed as:

$$T_{latency} = T_{sign} + T_{vote} + T_{policy} \tag{5.15}$$

The time taken to verify the digital signature of the proposal P can be expressed as:

$$T_{sign} = f_{sign}(P) \tag{5.16}$$

where, $f_{sign}(P)$ is the function representing the time complexity of signature verification.

However, the number of controllers participating in voting depends on whether the proposal is intra-domain or inter-domain. Let K number of controllers participate in the voting. Then, for intra-domain $K = N_{intra}$ and for inter-domain $K = N_{inter}$.

Therefore, the time taken to complete the voting can be expressed as:

$$T_{vote} = f_{vote}(K) \tag{5.17}$$

The time taken to enforce the adaptive policies can be expressed as:

$$T_{policy} = f_{policy}(P1, P2, P3) \tag{5.18}$$

where, $P1$, $P2$, and $P3$ are the policies mentioned in subsection 5.2.2.4.

We measure the latency for proposal verification on the virtual setup presented in Figure 5.3. Initially, we measure the average time required for voting with varying numbers of proposals. The number of proposals ranges from 10 to 1000.

Table 5.3: Average latency for signature verification, voting, and policy enforcement with increasing proposal request.

No of Proposals	T_{sign} (ms)	T_{vote} (ms)	T_{policy} (ms)	T_{total} (ms)
100	50	176.0	51	277.0
150	74	265.2	70	409.2
200	99	354.4	98	551.4
250	126	439.5	122	687.5
300	149	528.0	150	827.0
350	171	617.5	179	967.5
400	202	707.0	195	1104.0
450	223	798.5	220	1241.5
500	248	885.0	250	1383.0
550	270	973.0	275	1518.0
600	299	1063.0	298	1660.0
650	318	1152.5	318	1788.5
700	351	1240.0	347	1938.0
750	379	1330.5	373	2082.5
800	401	1423.0	397	2221.0
850	433	1511.5	424	2368.5
900	457	1602.0	449	2508.0
950	483	1694.5	473	2650.5
1000	504	1785.0	500	2789.0

Table 5.3 presents the latency for each primary component of proposal verification. From the table, we can observe that the average time taken for signature verification fluctuates between 0.3 to 1.2 ms per proposal. Again, the consensus time taken by the controllers to vote on the proposal's validity varies between 1.7 to 2 ms per proposal. Finally, the time required to enforce network policies based on the voting outcome alternates between 0.8 to 1 ms per proposal. Further, T_{sign} and T_{policy} introduce minor fluctuations in total latency and have less impact on overall latency compared to T_{vote}. The consistent voting time (T_{vote}) also ensures that the consensus process remains robust even as the number of proposals grows. Therefore, total

latency grows linearly with the increase in the number of proposals. This linear relationship indicates that the system handles scalability effectively, without introducing exponential delays.

5.3.2.2 Throughput Analysis

We also measure the throughput of the proposal verification. Throughput is a measure of the number of proposal requests the system can process in a given period. To perform this experiment, we consider the same setup as before.

We use the total latency values (T_{total}) from Table 5.3, to calculate throughput for each number of proposals. We calculate throughput (T_p) by dividing the number of proposals by the total latency time.

$$T_p = \frac{No.\ of\ proposals}{T_{total}} \tag{5.19}$$

From the Figure 5.4, we can observe a slight variations in throughput due to the minor fluctuations in latency components like T_{sign} and T_{policy}. However, throughput remains fairly consistent across different loads, indicating that the system handles scaling well without significant drops in performance.

5.3.3 Security Analysis

We simulate and evaluate the performance of the proposed work under various attack scenarios such as rogue controller attacks, replay attacks, and DDoS attacks. We discuss the detailed analysis of these scenarios in the following subsection.

5.3.3.1 Rogue Controller Attack

Here, a rogue controller sends malicious flow modification requests on the blockchain. We have prepared a script that generates a synthetic OpenFlow rule which is irrelevant to the original network configuration. The steps for simulating malicious flow injection is presented in Algorithm 12. This algorithm tries to install a malicious flow on switch S_{11} from controller C_{11}. We remove the key from the CA to perform this experiment.

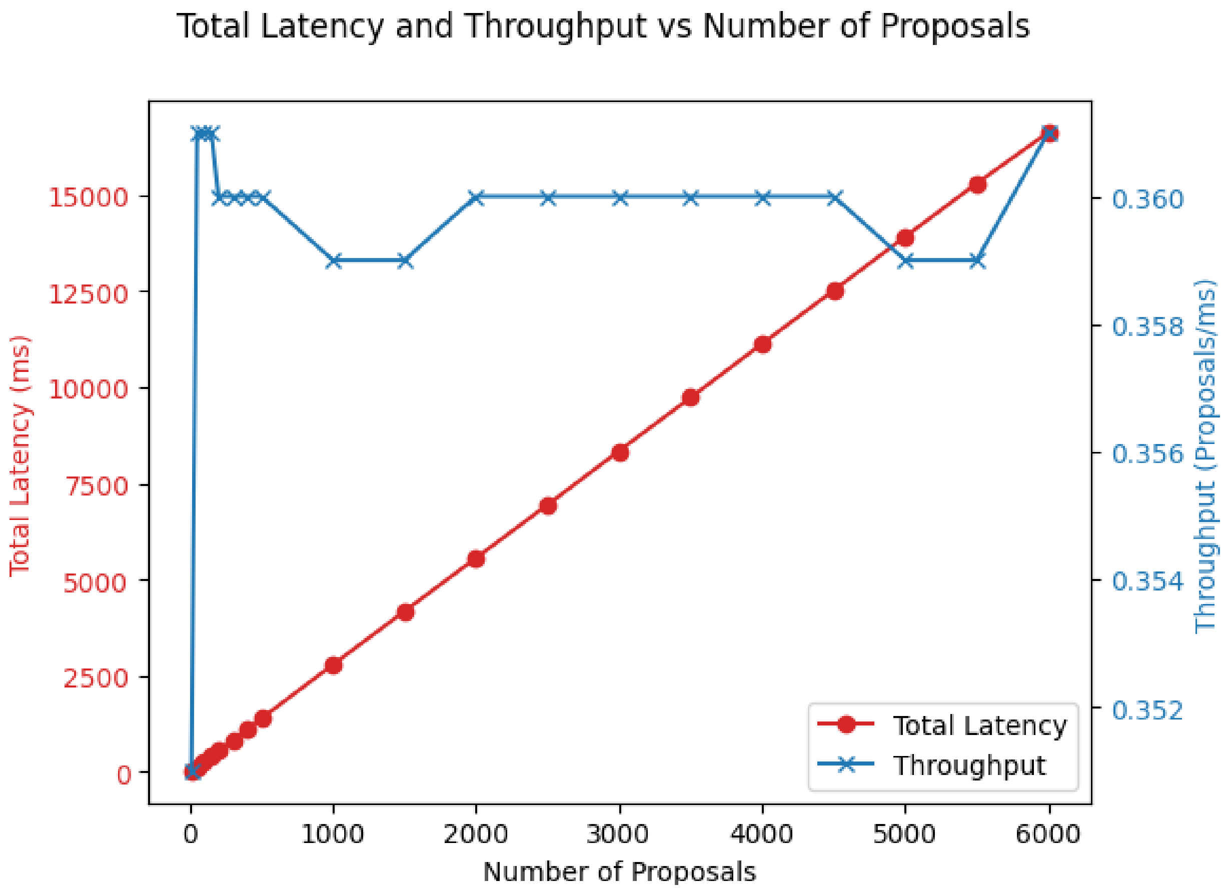

Figure 5.4: Throughput and total latency of the proposal verification with increasing number of proposal.

The CA maintains a list of valid public keys for all legitimate controllers. Each certificate binds the controller's identity to its public key, ensuring only authorized controllers can initiate flow modification requests. If the signature verification fails or if the proposer's public key is not found in the CA's list, the proposal is immediately flagged as rogue and rejected. Therefore, the rogue flow modification request is detected in Stage I of the proposed work by verifying the digital signature.

5.3.3.2 Replay Attack

In this type of attack, the attacker captures the legitimate flow modification requests and replays them to the network. To simulate this attack, we provide the legitimate flow details to the target controller C_{11} which then perform the attack on the network. The steps for replay attack is presented in Algorithm 13.

Since a replay attack reuses an old request, the digital signature remains valid, and

Algorithm 12 Simulating Malicious Flow Injection by Rogue Controller

Input: dpid: Datapath ID of switch, **s_ip**: Source IP, **d_ip**: Destination IP, **in_port**: Ingress Port, **s_mac**: Source MAC address, **d_mac**: Destination MAC address, **action**: Action to perform, **p**: Priority of the proposal, **C_ip**: IP of the controller

Output: $\mathbf{A_s}$: Action Status

```
1: Form a json file using Input for flow modification command
2: Create a hash of the flow
3: h ← $Hash(dpid + s\_ip + d\_ip + s\_mac + d\_mac + action + p + C\_ip)$
4: Crate an URL for flow modification
5: url ← $(http : //controller_ip : 8080/stats/flowentry/add)$
6: Apply REST post command
7: r ← $POST(url, h)$
8: Display the response
```

Algorithm 13 Simulating replay attack by legitimate controller

Input: C_ip: IP address of the SDN controller, **R_c**: A list of previously captured flow modification requests

Output: Re-sends each flow modification request to the controller and prints the HTTP response status

```
1: Start
2: for $request \in R_c[]$ do
3:     Construct the URL as:
4:     $url \leftarrow$ ”$http : // + controller\_ip + : 8080/stats/flowentry/add$”
5:     Send an HTTP POST request to url with request as JSON body
6:     Receive and store the HTTP response
7:     Wait for 1 second
8: end for
```

stage I of our method will likely not detect the replay attack. To counter replay attacks, the proposal includes a timestamp and a nonce (a unique, one-time-use number). Each new proposal must have a fresh timestamp and nonce that are unique and within a valid time window. The controllers check the timestamp and nonce to ensure the request is new and not a replay. If a proposal is detected as a replay (due to an invalid timestamp or nonce), it is rejected and an adaptive policy enforcement mechanism restrict the attacker from replaying the attack.

5.3.3.3 DDoS Attack

Here, the attacker generates a large volume of flow modification requests to overwhelm the controllers and network. The steps for DDoS attack on the controller is presented in Algorithm 14.

Algorithm 14 Algorithm for DDoS attack by rogue controller

```
Input: C_ip: IP address of the SDN controller, F_m: Flow entry data to be sent repeatedly,
R: Number of requests to simulate the attack
Output: Status : Action Status

1: Initialize an empty list called threads
2: for $i \in R$ do
3:     Create new thread
4:     Create a new URL to make flow modification request
5:     Sends an POST command to C_ip
6:     r ← $POST(url, F\_m)$
7:     Display the response r
8: end for
```

We plot a graph (refer Figure 5.5) to show the latency incurred due to the DDoS attack on the network. Additionally, it also measures the latency during proposal verification under normal conditions, during an initial DDoS attack, and during a subsequent DDoS attack. Additionally, we also compare the result with the proposed method and another without it.

We start the simulation with normal proposal verification from $t = 0$ to $t = 50$. Then, we introduce the attack at $t = 50$ with a large proposal request (30000 requests per second). In the proposed method, the graph shows that the latency starts to rise linearly with the increasing

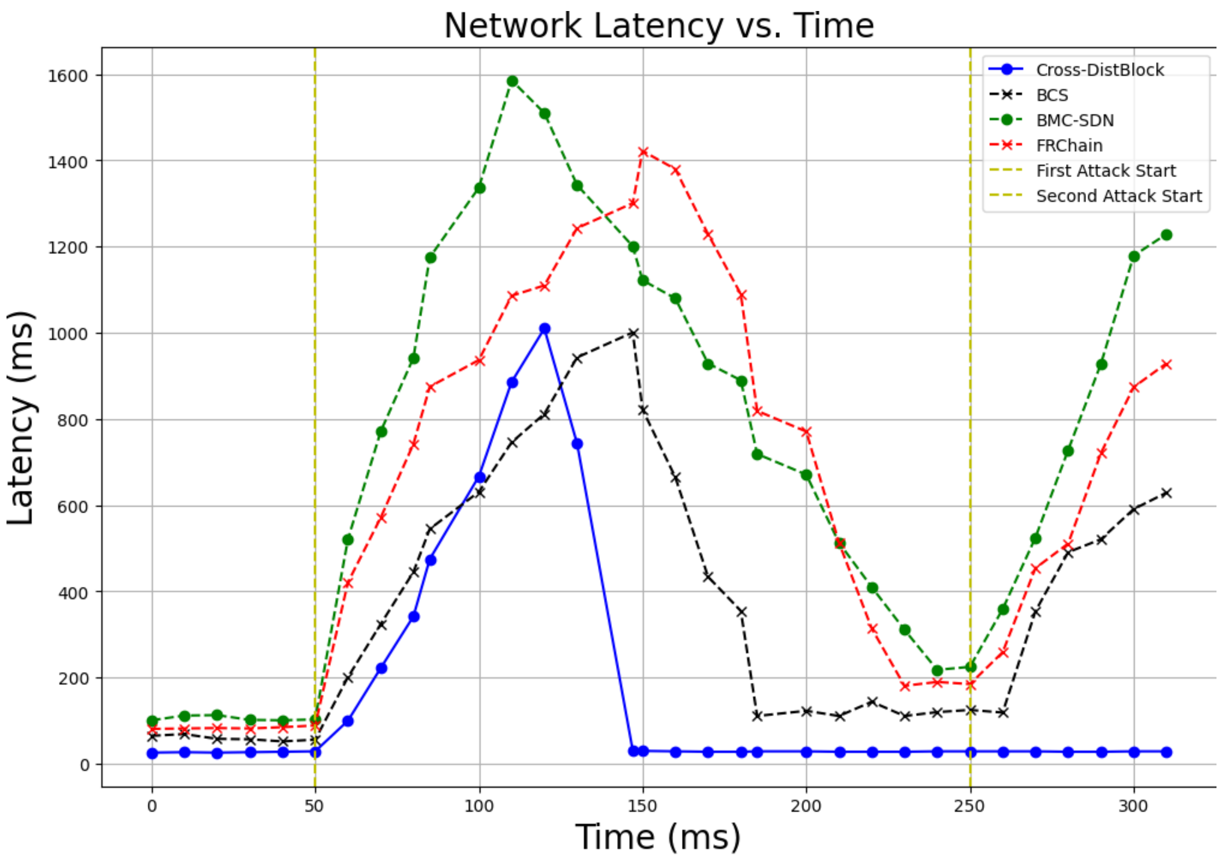

Figure 5.5: Latency comparison of the proposed method with other existing methods during DDoS attack.

number of proposal requests due to the attack in all four methods. At ($t = 120$), the DDoS attack causes a significant spike in latency. Then, the latency begins to stabilize at ($t = 147$) as the adaptive policy enforcement installs flow rules that block all traffic from the rogue controller and isolate the network in the proposed method. This demonstrates that the proposed method can stabilize the network in 97 ms, which matches the results obtained from average latency for proposal verification. However, in the existing methods, the DDoS attack persists and the latency keeps increasing and these methods take a longer time to detect and mitigate the attack.

At $t = 250$, we send a second DDoS attack. In the existing methods, the latency spikes again and continues to rise, indicating network instability. This is due to their lack of an adaptive policy enforcement mechanism. However, in the proposed method, the latency remains stable due to the earlier policy installed. This shows the effectiveness of the adaptive policy enforcement mechanism in mitigating the impact of repeated attacks.

The absence of adaptive policies in existing methods became evident when we subjected the network to repeated DDoS attacks. During the first DDoS attack, all methods, including ours, successfully detected and mitigated the attack by initiating the voting, consensus, and block creation processes. However, when the DDoS attack was applied a second time, the limitations of the other methods were highlighted. FRChain, BCS, and BMC-SDN all had to undergo the entire process of voting, consensus, and block creation once again, which significantly increased the network's latency and processing overhead.

We provided a summary of the latency comparison due to the DDoS attack in Table 5.4. This table effectively demonstrates the robustness of the proposed method in maintaining network performance and security even during repeated attacks. The experimental results show that Cross-DistBlock achieves a latency improvement of 20.49% over BCS, 28.15% over BMC-SDN, and 49.21% over FRChain.

Table 5.4: Latency comparison due to the DDoS attack.

Methods	Latency in 1st attack (ms)	Latency in 2nd attack (ms)
Cross-DistBlock	97	0
BCS [8]	122	123
BMC-SDN [22]	135	133
FRChain [101]	191	190

We also measured the latency incurred by the network with increasing network demands, particularly in a high-traffic scenario. The latency graph clearly illustrates that as the number of proposal requests increases, all methods experience an increase in latency. However, the rate of increase differs significantly among the methods. For FRChain, BCS, and BMC-SDN, the latency rises sharply as the number of proposals grows. This is primarily due to their reliance on traditional consensus mechanisms, which require substantial computational resources and time, particularly in scenarios with a high volume of requests.

We also measure the scalability of the proposed method by increasing the request count per second. Fig 5.6 shows that our proposed method shows a more gradual increase in latency. Initially, when the proposal requests are relatively low (around 1,000 to 10,000), all methods perform comparably. However, as the request count reaches higher levels (20,000 to 50,000), the advantages of our method become evident. This demonstrates that our method not only scales better but also maintains more consistent performance under heavy load.

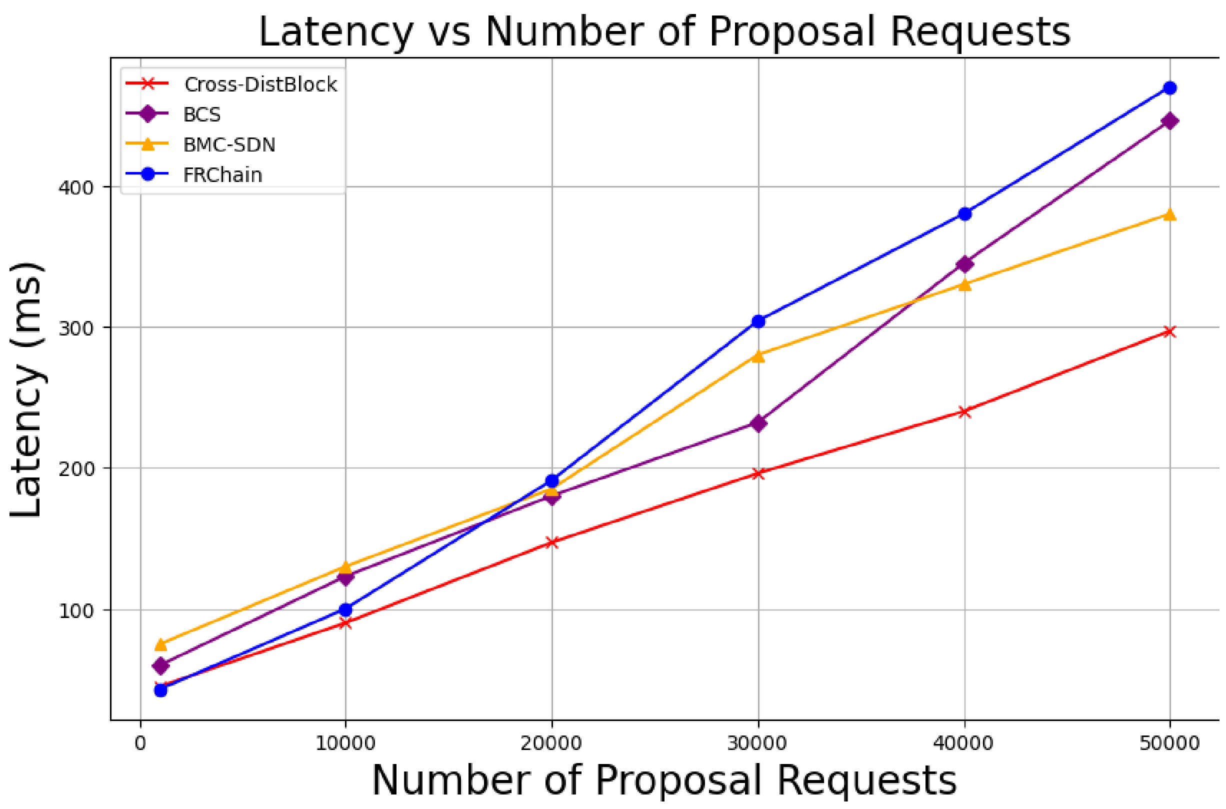

Figure 5.6: Latency comparison of the proposed method with increasing traffic during DDoS attack.

5.3.3.4 Controller Overhead

Since the controllers are given the role of endorser for proposal verification, the load on these controllers will increase, These controllers need to execute the chaincode functions to decide on the proposal. Therefore, we compare the CPU consumption of these controllers with and without the proposed method.

We simulate these two cases on the given experimental setup. In the initial period, the CPU consumption starts at 24% and increases up to 29% linearly. After a while, we perform the DDoS attack at around $t = 90$. Due to the attack, the CPU consumption rises to 42% while using the proposed method. After a while, the CPU consumption remains stable at 42% due to the adaptive policy enforcement at $t = 122$. On the other hand, CPU consumption spikes to 71% and remains high in cases without the proposed method. When we stop the attack a $t = 130$, the CPU consumption starts to drop.

At $t = 150$, we apply a second attack on the network. With the proposed method, CPU consumption remains stable at 29% due to its previously installed OpenFlow rules. However,

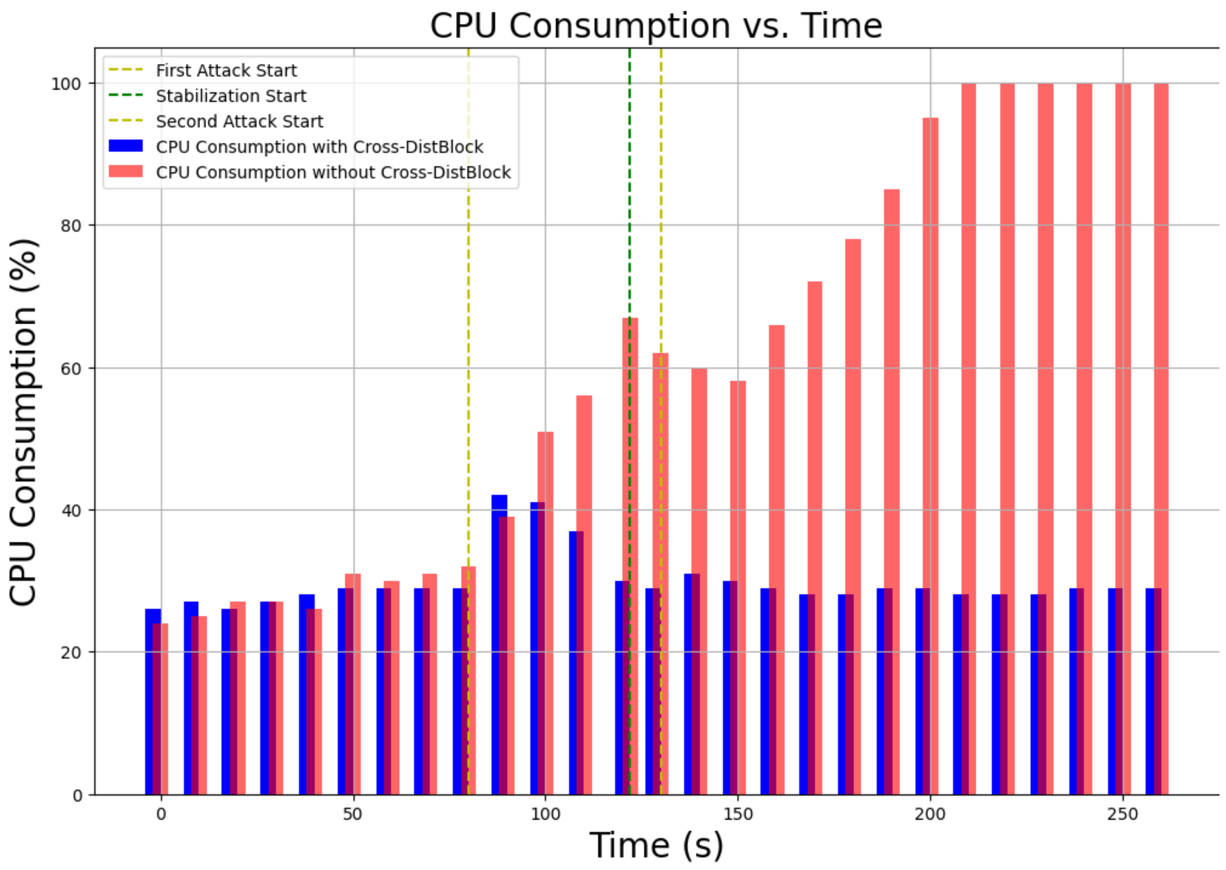

Figure 5.7: CPU consumption of endorser/controller due to proposal verification in DDoS attack scenario.

without the proposed method, the CPU consumption escalates and the system becomes unresponsive. Therefore, the proposed method effectively stabilizes CPU consumption after an attack, compared to a scenario where no mitigation measures are in place.

5.4 Discussion

In this subsection, we discuss the security and effectiveness of Cross-DistBlock.

Our approach leverages a chaincode that maintains a list of controllers along with an accumulated invalid vote count they have provided. This vote count serves as a historical record, indicating whether a controller has been consistently trusted, is suspected, or has become rogue. By accumulating these instances of invalid voting rather than relying solely on frequency, the system is designed to recognize sporadic patterns of threat behaviour over time. By focusing on cumulative inconsistencies rather than isolated incidents, our system can detect controllers exhibiting sporadic invalid votes over time, allowing us to flag intermittent threats even if they do not consistently vote invalidly.

In our setup, a controller's trust threshold is set at 0.6. This threshold enables each controller to cast up to four invalid votes before it is removed from the network. This buffer period allows for more accurate identification of genuinely malicious or unreliable controllers. Moreover, this system enables the network administrator to take corrective actions against controllers listed as suspected, even before they are formally classified as rogue. This proactive approach serves as an effective safeguard against colluding controllers that attempt to subtly manipulate the network.

Latency was a critical metric in evaluating the performance of the proposal verification process. The latency measurements included the time taken for digital signature verification, voting by controllers, and the enforcement of adaptive policies. Under normal conditions, the latency increased linearly with the number of proposal requests. However, during an attack, the latency spiked due to the additional processing required to isolate rogue components. The results showed that after the attack was mitigated, the latency stabilized and returned to acceptable levels. Without the proposed method, the latency continued to increase, highlighting the effectiveness of our approach in maintaining low latency during and after attacks.

Throughput was measured as the number of successfully processed transactions per unit of time. The results indicated that our system maintained a high throughput under normal conditions, with a slight decrease during attack scenarios. The introduction of the adaptive policy enforcement mechanism caused a temporary reduction in throughput due to the additional overhead of isolating rogue components. However, once the malicious entities were isolated, the throughput returned to normal levels, demonstrating the system's ability to recover quickly after an attack.

Rogue controllers pose a serious threat to the integrity of the SDN environment. Our approach successfully identified and isolated rogue controllers through a continuous evaluation of their trust scores. The smart contract's capability to maintain a rogue list and prevent these controllers from participating in the consensus process was crucial in ensuring the validity of network operations. Experimental results showed that when rogue controllers attempted to manipulate network rules, they were quickly detected, and their influence on the network was neutralized. The trust score mechanism effectively reduced the risk of malicious activities within the network, as controllers with trust scores falling below the threshold were promptly isolated, preventing further damage.

Replay attacks were addressed by incorporating digital signatures and hash verification within the proposal verification process. The system recalculated the hash of incoming proposals and compared them with the stored hash values. In cases where a mismatch was detected, the proposal was flagged as invalid, and no further action was taken. This approach successfully prevented replay attacks, as any attempt to resend a previously valid transaction was immediately recognized and rejected. Our experiments demonstrated that the hash-based verification process was robust against replay attacks, ensuring that only legitimate proposals were executed.

The proposed method effectively mitigates Distributed Denial of Service (DDoS) attacks by employing an adaptive policy enforcement mechanism. Therefore, it provides a substantial advantage over the existing methods. After the first DDoS attack, our system dynamically adjusted the security policies, isolating the malicious entities and ensuring that subsequent attacks did not require the same intensive verification process. As a result, when the second DDoS attack was launched, our method was able to maintain network stability with minimal additional latency, as the adaptive policies had already fortified the network against similar threats. The results indicate that while FRChain, BCS, and BMC-SDN are effective in their initial response to attacks, their lack of adaptive policies limits their ability to efficiently handle repeated threats, making them less suitable for dynamic and continuously evolving network environments.

During the attack, the network experienced a significant increase in CPU consumption, peaking at 42% when the first attack was detected. However, the adaptive policy quickly isolated the malicious traffic, stabilizing the CPU usage. When a second DDoS attack was launched, the CPU consumption remained stable, indicating that the network was resilient to subsequent attacks due to the preemptive isolation of rogue nodes. In contrast, without the proposed method, the CPU consumption continued to rise, reaching up to 90%, which severely degraded the network performance and demonstrated the importance of our method in maintaining network stability under attack conditions.

In our approach, once a controller is identified as rogue, it is immediately removed from participating in the consensus process to prevent further actions. Our proposed adaptive policy enforcement mechanism dynamically adjusts the capabilities of controllers based on their trust scores, enhancing security and resilience against suspected or rogue controllers. This design ensures that security policies are applied in real time as soon as any rogue behaviour is detected, limiting the impact on network operations.

Currently, our system can effectively block all network traffic originating from a rogue controller, safeguarding the network from any further malicious influence until the controller can be re-deployed by the network administrator. To further strengthen our approach, we can incorporate a mechanism that revalidates the proposals approved by a rogue controller, allowing us to rollback or remove any unauthorized flows it may have installed.

Increasing the number of controllers and the size of the network adds overhead, particularly due to the blockchain's consensus mechanism. Specifically, the blockchain's consensus mechanism incurs increased communication and computational demands as more controllers participate in the network.

To manage scalability, our approach includes an adaptive policy that adjusts the consensus participation scope of controllers based on the type of proposal request. Intra-domain consensus is conducted among controllers within the same domain, ensuring local reliability with minimized communication overhead. For inter-domain consensus, selected controllers from each domain participate to maintain a cross-domain agreement without requiring the involvement of every controller in the network. This selective inter-domain participation helps reduce the burden on the network, achieving a balance between security and performance as the network scales.

5.4.1 Use Cases

a) Smart City: In smart city deployments, different aspects of city infrastructure- such as transportation, energy grids, public safety, and communication networks are often managed by separate entities or departments, each representing a different domain. Multidomain SDN helps integrate these diverse networks into a unified management framework, allowing for coordinated traffic management, data sharing, and service provisioning across the city. For example, traffic management systems can be linked with public transportation networks and emergency services, enabling real-time adjustments to traffic flows in response to incidents or changing conditions. By providing centralized control and dynamic policy enforcement, the proposed security solution ensures seamless connectivity across multiple domains.

b) Healthcare System: Healthcare systems often operate across multiple domains, such as different departments within a hospital or between various healthcare providers. The proposed model's multidomain SDN approach ensures seamless and secure data flow across these domains, even when they are managed by different organizations or have distinct security policies. Additionally, healthcare IoT networks are vulnerable to rogue devices that could attempt to introduce malicious data or disrupt normal operations. The proposed model includes a mechanism to detect and isolate rogue devices or controllers by maintaining a trust score for each network controller.

5.5 Conclusion

In this chapter, we tackled the critical issue of securing flow modification requests in multi-domain SDN networks by introducing a blockchain-based approach called Cross-DistBlock. By leveraging the decentralized and immutable characteristics of blockchain technology, we established a strong trust model that ensures the integrity and authenticity of flow proposals. Our solution employs a stage-wise proposal verification process combined with adaptive policy enforcement, which effectively detects and mitigates threats from rogue or malicious controllers. Experimental results validate the effectiveness of our approach in maintaining network stability, even in the face of attacks like DDoS, rogue controller activities, and replay attacks. The successful integration of Hyperledger Fabric with Ryu SDN controllers highlights the practical feasibility of our framework, providing a secure and scalable solution for managing SDN networks across multiple domains. This work sets a solid foundation for advancing research in securing SDN environments, emphasizing the critical role of blockchain technology in enhancing network security and trust.