A BLOCKCHAIN-BASED CONSENSUS SLICING MECHANISM FOR DISTRIBUTED SDN CONTROL PLANE

Shanqing Jiang[1,2] and Lin Yang[2]

[1]School of Cyber Science and Engineering, Southeast University, Nanjing, China
[2]Institude of System Engineering AMS PLA, Beijing, China

ABSTRACT

In large-scale distributed Software Defined Networks (SDNs), a logically centralized network view is required in the physically distributed control plane to provide correct application decisions. Distributed SDN controllers communicate through east-west interfaces to achieve consistency of the global network view and coordination of control decisions. The consistency of the global network view requires synchronization of various network states in multiple controllers in SDN. Different network states differ in update rate, data size, application access method, etc. These factors affect the consensus protocol and synchronization strategy used for state synchronization. Therefore, we propose a consensus slicing mechanism that dynamically adjusts the synchronization strategy according to application requirements, minimizing consistency differences among controllers and reducing synchronization overhead. Then, we use a storage approach combining blockchain with distributed databases to store synchronized state information, improving the efficiency and anti-tamper ability of state information. Simulation experiments validate the effectiveness of the proposed approach.

KEYWORDS

Consensus slice, Blockchain, SDN controller, Distributed control plane, State synchronization

1. INTRODUCTION

Software-Defined Networking (SDN) separates the control plane from the data plane in traditional networks, creating a logically centralized network management mode and providing flexible programmable interfaces for network applications. The SDN control plane has a global view of the network to facilitate the management of the entire network, such as computing forwarding paths and traffic load balancing. As network scale expands and application scenarios become increasingly complex, a single controller cannot meet the reliability and scalability requirements of the network. A single controller faces a single point of failure, and switches located far away will suffer from long response delays, which may result in delayed flow rule installation.

Distributed SDN controllers solve these problems effectively. Multiple physical SDN controller instances manage the entire network, dividing it into multiple domains, with each domain's SDN controller responsible for managing a portion of the forwarding devices in the data plane. Distributed SDN controllers enhance the robustness and scalability of the control plane. Physical distribution enables switch to be closer to controllers, reducing response delays. However, physically distributed multiple controllers still need to maintain a logically centralized network view to provide correct forwarding decisions for network applications. Distributed SDN

David C. Wyld et al. (Eds): DBML, CITE, IOTBC, EDUPAN, NLPAI -2023
pp.121-135, 2023. IJCI – 2023 DOI:10.5121/ijci.2023.120210

controllers maintain communication between each other through east-west interfaces to maintain consistency of the global network view and coordination of control decisions.
Distributed SDN controllers use consensus protocols and a series of synchronization strategies to achieve global network state synchronization. There are two general consensus protocols: strong consistency consensus protocol and eventual (weak) consistency consensus protocol. The strong consistency protocol requires all controller instances to synchronize the latest network state at any time, which can ensure that network applications can access the latest state data on any controller instance. However, the interaction process of the strong consistency protocol is relatively complex, and it takes a long time to go through the consensus process from state update to application access. Typical strong consistency protocols include RAFT and PBFT. The eventual consistency protocol periodically broadcasts messages between controllers to make the network view converge to a final consistent state. The eventual consistency protocol reduces synchronization overhead and increases network availability. However, it can cause a temporary inconsistency in the network view of different controllers, affecting the correct decision-making of application programs. Typical eventual consistency protocols include Anti-Entropy.

The consistency of the global view requires various network states in SDN to be synchronized among multiple controllers. Network state includes node, topology, flow table, and other information, which together form the network view of SDN. As shown in Figure 1, the distributed SDN control plane consists of multiple physically distributed controllers, each responsible for managing a portion of switches in the data plane. The set of controllers and switches it is responsible for is called a control domain. The controller in each domain is responsible for issuing forwarding policies to the switches within the domain, while also updating the local network view of the domain. Controllers between different domains communicate with each other through east-west interfaces, exchanging their respective domain's network views and synchronizing to achieve consistency in the global network view. For example, when host 1 in domain A wants to communicate with host 2 in domain C, a forwarding path needs to be established between them. Controller A can calculate a forwarding path from the source host to the destination host based on the global network view, which benefits from the real-time synchronization of network state information such as host location and network topology among multiple controllers.

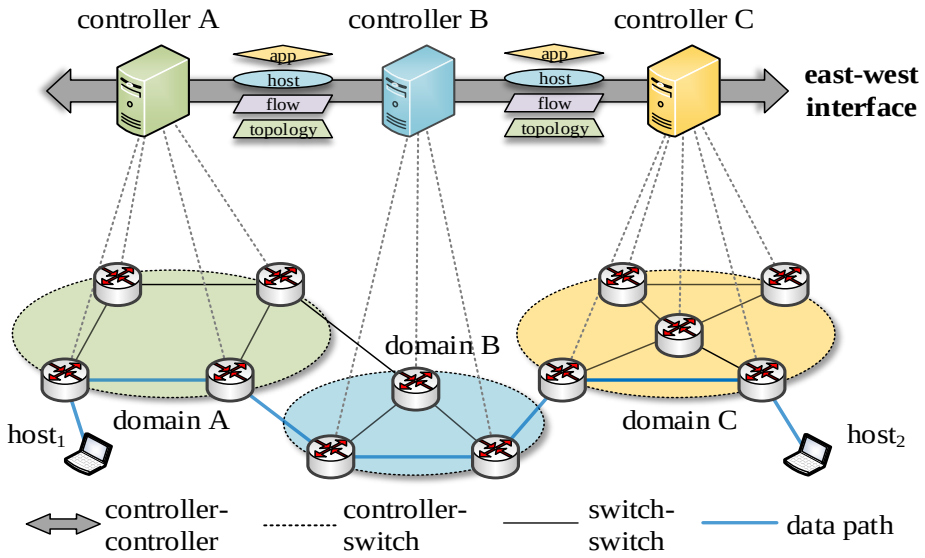

Figure 1. Multi-domain SDN networks: enable distributed SDN controllers to synchronize network states through east-west interfaces.

During the process of state synchronization, there are differences in the update rate, data volume, and application access mode of various network states that need to be synchronized, which affect the consensus protocols and synchronization strategies used for network state synchronization. Taking the ONOS controller as an example, network states are stored in a distributed data structure called "Stores," including controller master-slave relationships, host nodes, network topology, flow tables, and applications. For example, network topology is achieved through Anti-Entropy among controllers and is synchronized periodically, with full network topology information being synchronized each time.

In fact, maintaining consistency of the network view between multiple controllers is a very complex task. We can consider the synchronization of the network view among distributed SDN controllers as an instance of data synchronization in a distributed system. Therefore, the synchronization of the network view also follows the constraints of the CAP theorem: consistency, availability, and partition tolerance cannot be guaranteed simultaneously, only two of the three can be chosen at the same time while ensuring partition tolerance. Under the premise of ensuring partition tolerance, the synchronization of the global network view faces a trade-off between consistency and availability. Specifically, if consistency is emphasized, the network state should be synchronized among multiple controllers before providing access services (read/write) to the application. If availability is emphasized, the network state should respond to application access requests in a timely manner, even if the network state has not been fully synchronized across the network.

Currently, in the controller architecture implemented by the industry, the state synchronization among multiple controllers is mainly achieved through shared state storage space or message subscription and publishing. The strategies for updating the state among multiple controllers do not differ significantly. Onix and ONOS controller clusters both use the periodic full update mechanism based on the Anti-entropy algorithm to synchronize topology information. However, Onix's NIB database uses the synchronous mechanism based on RAFT for updating traffic state, while ONOS uses the synchronous mechanism based on Anti-entropy. Different controllers have few synchronization schemes to choose from when updating the state in the east-west direction, and the frequency of information updates cannot be adjusted. In recent years, although some studies have introduced adaptive synchronization mechanisms for network state in multi-controller scenarios, the consistency models proposed are relatively simple, achieving only the adjustment of synchronization rate [1,2] and the number of synchronization subjects [3].

Distributed SDN control planes may be vulnerable to information transmission security issues. When a distributed controller or an east-west channel is hijacked and some network state is maliciously tampered with, the consensus process can be disrupted, which affects the normal decision-making of network applications. For example, the literature cites attacks where malicious applications tamper with network state. Since most controllers lack access control for application API calls, maliciously registered applications may tamper with network topology data and even prevent other applications from receiving control messages. Therefore, maliciously tampering with network state simultaneously disrupts both consistency and availability. Currently, commercial SDN controllers, such as the representative ODL and ONOS, only use Anti-Entropy and RAFT protocols to achieve consensus on network state, and neither of these protocols can resist BFT-like security threats.

Therefore, in response to the aforementioned open issues, we will focus on exploring the logical principles of state synchronization between distributed SDN controllers and investigating how the state synchronization process can resist BFT-class threats. We establish a universal adaptive mechanism for network state synchronization called "Consensus Slicing among Distributed Control Planes", which implements security checks on consensus slicing based on multi-chains to

prevent malicious tampering of network state during the synchronization process. A multi-state adaptive synchronization strategy is designed to dynamically adapt to application requirements, minimizing synchronization overhead while maximizing network availability.

Therefore, this paper first classifies and describes the characteristics of SDN network states, then models the availability and consistency requirements of network applications in a fine-grained manner, and finally proposes an adaptive consensus slicing algorithm that utilizes blockchain and distributed databases to achieve synchronized sharing of network states in multi-domain networks. In order to obtain intuitive quantitative results, we conducted experiments to verify time and performance costs.

2. RELATED WORK

In this section, we first review the network state consistency problem in multiple controller SDN. Then we summarize some current research on state synchronization strategies between multiple controllers.

2.1. Consistency among Multiple Controllers

There has been a lot of research in terms of the consistency problem in multiple controllers SDN. Muqaddas et al. [4] studied the inter-controller communication traffic generated in ONOS clusters to maintain globally consistent network state and detailed the types of network state that need to be synchronized. Yu et al. [5] systematically introduce the consistency problem in SDN networks. They presented several scenarios of consistency violating, including packet inconsistency, flow table inconsistency, and network-level inconsistency. The relevant solutions up to that time were also introduced, which focusing on the consistency problems in both data plane and control plane in SDN. Hu et al. [6] concluded the consistency problems among multiple controllers, including controller state consistency and control strategy consistency. There exist many reasons for inconsistency: out-of-sync between controllers, concurrent policy conflicts, propagation delays, etc. Differently, Foerster et al. [7] focus on the consistency of the entire SDN network, summarizing consistency into connection consistency, policy consistency, and capacity consistency. Zhang et al. [8] considered the placement of SDN controllers in the in-band control plane. They described the trade-offs between switch-controller latency and controller-controller latency and the possible Pareto frontier.

2.2. Adaptive Synchronization Strategies

In order to trade off the consistency and availability, many studies have focused on designing adaptive state synchronization strategies in distributed control plane. Zhang et al. [2] quantified the consistency level of controllers based on the characteristics of SDN and proposed an adaptive synchronization strategy, in which controllers were divided into three kinds of roles with different synchronization periods. However, this work did not consider the specific networking states and synchronization overhead. Poularakis et al. [9] investigated the problem of optimal synchronization rates among controllers in distributed SDN systems. The authors considered two different objectives: maximizing the number of consistent controller pairs, and maximizing the performance of affected applications. Aslan et al. [10] investigated an adaptive consistency strategy of distributed SDN controllers based on tunable consistency model "Apache Cassandra". The proposed strategy that used online clustering techniques can map an application-specific indicator into consistency level of the distributed controllers. Bannour et al. [11] propose an adaptive continuous consistency model for large-scale distributed SDN networks (C-CDN). An intelligent Quorum-replicated consistency was used to achieve fine-grained control over the level of consistency. Compared with the ONOS's static consistency model, the approach can achieve

the minimum of application's inter-controller overhead. Bannour et al. [3] transferred the eventual consistency model to an adaptive multiple level consistency model, which can minimize the cost of state synchronizing while maintaining the application's performance requirement.

3. SYSTEM DESIGN

The concept of "slicing" in the network field is defined as a technology that virtualizes network physical resources [12, 13]. Network slicing divides the physical network into multiple logical subnets with specific functions and mutual isolation through virtualization techniques to meet the specific requirements of network applications or services, such as mobility, latency, security, and reliability. Network slicing includes isolated subsets that define available virtual resources (compute, network, storage), as well as a set of rules for identifying traffic running on these resources. Network slicing technology can save the cost of deploying dedicated networks, increase the elasticity of network services, and achieve good fault isolation.

Inspired by network slicing, we focus on the state synchronization of the SDN multi-controller control plane. The current state synchronization mechanism of distributed SDN controllers is not flexible enough. The personalized requirements of network applications cannot be met, and the balance between the consistency and availability of state synchronization between controllers cannot find the optimal solution. Not to mention, the integrity and trustworthiness of network state cannot be guaranteed. These shortcomings severely hinder the deployment and application of SDN on a large-scale wide area network. A flexible and adaptive state synchronization mechanism that is configurable according to application requirements is required for physically distributed controllers.

To address this, we propose the SDN consensus slicing mechanism. By designing a customizable network state consensus mechanism with specific functions to meet the differentiated performance or security requirements of network applications. Consensus slicing includes logical slices that define available consensus resources (consensus nodes, consensus protocols), as well as synchronization policies (synchronization frequency, amount of synchronized data) for network state synchronization among multiple controllers. Consensus slicing customizes the optimal consensus and synchronization strategy for state data between different domains in the distributed SDN network to meet the differentiated requirements of consistency, availability, and security for different applications. Especially for the potential risk of tampering with network state data, the integrity and trustworthiness of information are ensured by setting BFT-like consensus protocols.

3.1. Blockchain-Based Consensus Slicing Model

We propose a consensus slicing model based on blockchain. Generally, consensus and synchronization have conceptual differences. Consensus refers to the agreement among multiple subjects on a certain event and emphasizes the outcome, while synchronization refers to maintaining the same clock frequency or data replica among two or more subjects and emphasizes the process. In the control plane of a multi-controller SDN, it is necessary to maintain a globally consistent network view for each physically distributed controller, i.e., multiple controllers form a consensus on the global network view, and the synchronization of various network states among multiple controllers is essential in this process. Moreover, as the network state is frequently updated, the consensus on the global network view needs to be maintained frequently, requiring the continuous operation of the synchronization process. Therefore, we figuratively express consensus slicing as a time-sliced synchronization strategy of set *S* (multi-

dimensional network states) and set *C* (physically distributed multiple controller instances) in the time dimension, as shown in Figure 2.

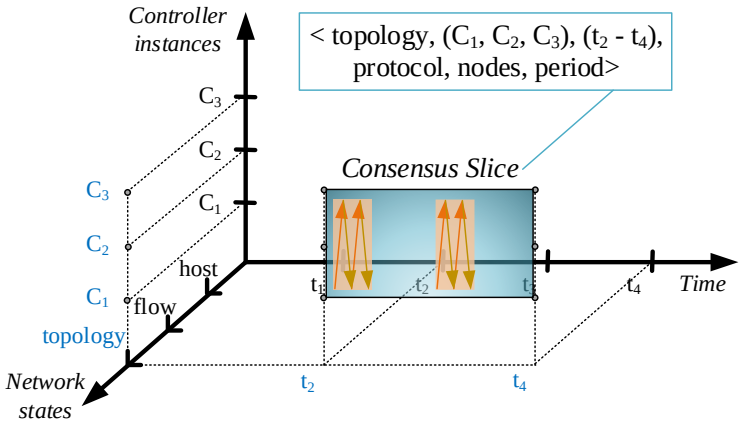

Figure 2. 3D schematic diagram of consensus slice

More specifically, each consensus slice includes a series of entities and strategies for achieving consensus, including: the objects of consensus (various network states), the subjects of consensus (multi-controller entities), the consensus protocol, the consensus nodes, the duration of consensus, and the synchronization strategy (synchronization period) within the time window. As shown in Figure 2, the blue rectangle consisting of <*topology*, (C_1, C_2, C_3), (t_2, t_4)> becomes a consensus slice in the three-dimensional space, which is used to maintain the consensus among controllers (C_1, C_2, C_3) regarding the network topology between t_2 and t_4. It also includes attributes such as the applied consensus protocol, the number of consensus nodes, the synchronization time period, and the synchronization data volume. During the duration of the consensus slice, multiple controllers first synchronize the state information according to the slice attributes, and the orange part in the figure shows the information interaction process of state synchronization. During this period, network state data cannot be applied for reading and writing. In the remaining blue part of the slice, network applications can access the synchronized state data.

We have established a mathematical model for consensus slicing, which is a subset of the Cartesian product of multi-dimensional consensus attributes, as shown in the formula below:

$$Consensus\ Slice = \{(s_i, c_i, t_i, cp_i, cn_i, sp_i) \mid$$

$$s_i \in S, c_i \subseteq C, t_i \subseteq T, cp_i \in CP, cn_i \subseteq CN, sp_i \in SP\}$$

1) *CP* is a collection of consensus protocols, such as PBFT, Gossip, Raft, PoB, etc. Different consensus algorithms adopt different state synchronization mechanisms, which differ in terms of synchronization time overhead and mitigation of security risks, while ensuring the goal of consistent global network status. Different consensus algorithms perform differently in terms of security, real-time performance, reliability, and trustworthiness. For example, the PBFT consensus algorithm is geared towards Byzantine faults, with high security, but the consistency synchronization time is longer and difficult to meet the needs of network status with higher real-time requirements, such as traffic scheduling.
2) *CN* is the set of consensus nodes, and the number of consensus nodes affects the degree to which the state is replicated across multiple blockchain nodes or distributed database replicas. The more consensus nodes there are, the higher the consistency, but the overhead due to consensus and writing also increases.

3) *SP* represents the synchronization time period, which indicates the frequency of synchronization of the state to the consensus cluster. For highly dynamic states, it is important to choose a suitable update frequency that matches their rate of change to ensure the real-time nature of the data.

3.2. System Framework and Function Modules

To implement the mechanism of consensus slicing, we designed a framework for collecting and synchronizing network status information in a prototype system, which includes a network status collection module, an application service level agreement (SLA) generation module, a consensus slice generator, and a multi-chain and distributed database for synchronizing and storing network status, as shown in Figure 3.

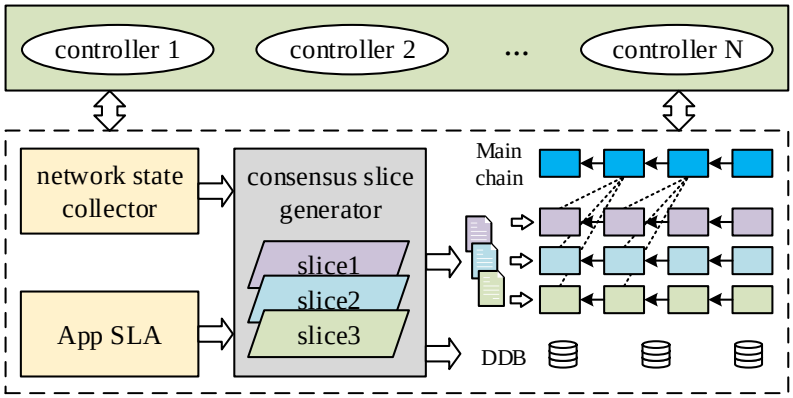

Figure 3. System framework

1) The network status collection module is used to collect and classify various types of status information within the control domain. Inspired by Onix's NIB and ONOS's Store, we divide network status into information such as controller, switch, host, mastership, link, topology, flow rule, application, and network configuration according to their attributes and sources.

2) The application requirement mapping module is responsible for obtaining the SLA requirements of upper-layer applications on the controller in real-time. Different network applications have different requirements for the availability and consistency of the global network status when facing different network status, and the application requirement mapping module is needed to identify and clarify the differentiated requirements of real-time network applications.

3) The consensus slice generator first adaptively selects the best consensus and synchronization strategy for each type of network status. Then it collects information from the data plane and the control plane within the domain, synchronizes the network status information through the designed smart contract to the sub-chains of the blockchain according to the generated strategy, and uploads the consensus strategy and key network status information to each sub-chain or stores them in the distributed database.

4) The distributed database is used to store real-time data with a faster data production rate, such as real-time flow table data and network topology. The blockchain is used to store data with slower data production rate and higher security requirements, such as node identity information. This classification will be described in detail in Section 4.1.

5) The main chain of the blockchain is used to store the consensus strategy information and consensus process records of each sub-chain corresponding to the consensus slice, ensuring the global consistency and tamper resistance of the consensus strategies for each type of network status among all controllers. The main chain can obtain the overall view of the consensus strategies for the global network status, and can also detect any abnormality in the operation of the consensus slice and switch the slice accordingly.

4. CONSENSUS SLICING MODEL

4.1. Network Status Classification in SDN

We draw inspiration from the NIB of Onix and the Store mechanism of ONOS, where multiple controllers locally store network state data structures within their respective domains. Through interactions and synchronization among controllers, various types of network states across multiple domains are formed into a globally consistent consensus.
From an ontological perspective, we analyse these network states and find that they can be classified into three dimensions: entity, state, and transaction. These three categories provide a primitive level of specification for network view in SDN. We define entities as individuals that exist in the physical network. In a running SDN network, controller nodes, switch nodes, and links can be regarded as entities. Each entity has multiple attributes, and the value of each attribute at a specific time is the state, such as the switch port status and the real-time available bandwidth of the link. Transactions refer to the actions executed by entities at a specific moment, which may change the state value of the entity itself or other entities, such as a controller issuing flow table rules to a switch.

From a temporal perspective, entities are relatively persistent information. Once a network device or controller code is deployed in the network system, its identity will persist unless removed from the network. State values are real-time changing information, but they may remain unchanged for a period until modified. Transactions in the network are instantaneous or continuous information, recording the momentary behaviour of network entities. From a microscopic time-scale perspective, continuous transactions are also composed of discrete transactions, such as the operation of network services in SDN, which are formed by multiple control flow table registrations and data plane traffic interactions.

By discussing the above definitions, we find that network protocols are various communication specifications established on the basis of entities, state values, and transactions. For example, the OpenFlow protocol specifies the communication rules between controllers and forwarding devices, including the interacting entities, state values that need to be synchronized and modified, and the upward or downward interactions. From the usual research perspective, the information consensus among multiple controllers is considered an important foundation for east-west protocols. In contrast, based on the above analysis, the fundamental problem that information consensus needs to solve is the consistency of <*entity, state value, transaction*>, and other information in the same time and different spaces.

Table 1. Network status classification and description

State category	Explanation
Controller	Store the controller identity and status information.
Switch	Store the switch device identity and configuration information.
Host	Store a list of network hosts.
Mastership	Store the mapping of master-slave relationships between controllers

	and switches.
Link	Store the link relationships and statuses among all devices.
Topology	Store the topology relationships between switches.
Flow rule	Store real-time flow table rules and statistics information.

4.2. Fine-Grained Slicing with Adjustable Consensus Mechanism

In distributed systems, the time model is usually classified into synchronous, asynchronous, and partially synchronous models. Real-world distributed network systems typically adopt partially synchronous models, in which the message communication and transmission delay between processes have upper bounds for most of the time. In distributed SDN, there is a certain communication delay between multiple physically distributed controllers, which will slow down the consensus process of the global network state, affecting consistency and availability. For weak consistency consensus protocols, communication delays affect the convergence speed of the global state. For strong consistency consensus protocols, communication delays will exponentially slow down the message confirmation among multiple nodes, affecting the submission of the final state.

We have modelled the availability and consistency requirements of network applications at a finer granularity. We establish a mapping relationship among "network application-controller-network state-duration", that is, within a specified time, the network application will require consensus to be formed among multiple controllers for a certain type of network state. Based on this mapping relationship, regardless of whether a final consistency or strong consistency protocol is adopted, the fine-grained requirements of specific network applications can ensure that the required network state is confirmed and synchronized among the involved controllers, and the data of the state can be returned to the application without waiting for the global consensus among all controller instances. That is, the availability requirements of network applications are prioritized, effectively preventing the problem of global consensus failure caused by communication delays between controllers.

We build a fine-grained service level agreement (SLA) model for network applications to describe their real-time consistency and availability. The fine-grained SLA is expressed as following, which declare that the network application requires the specified network states reach consensus among specified controllers.

$$SLA(app) \propto F(controllers,\ status,\ duration)$$

Based on the defined relationship, the related states can be accessed by application after the involved controllers receiving and confirming these states, rather than reaching global consensus among all controllers. This mechanism provides a cost-effective consensus process and prevents the consensus timeout problem caused by communication delay among controllers. A realistic example can be demonstrated in Figure1 to illustrate the significance of the proposed model. Assuming there exists a dedicated application on controller A for maintaining links and traffics monitoring between *host1* and *host2*, the link and flow status should be guaranteed for prior consensus between controller A and B through our proposed fine-grained mechanism.

When controllers across multiple domains cannot timely share the network status within their own domains, the network view among controllers may become inconsistent. In order to measure the consistency among controllers, we introduce the factor of network status diversity based on the previous analysis. The greater the difference in network status between controllers, the weaker the consistency of the network. Different types of network status exist in each control domain, such as the addition of new forwarding nodes, link interruptions, and port traffic surges.

Based on the analysis of these three types of network status, we define the consistency level of each controller C_i as follows:

$$Cons = \sum_{j=1}^{a_1} \left| \Delta v_{ij} \right| + \sum_{j=1}^{a_2} \left| \Delta \mu_{ij} \right| + \sum_{j=1}^{a_3} \left| \Delta \omega_{ij} \right|$$

In above, Δv_{ij} represent the amount of change in the network entities but have not yet synchronized, and there are a_1 network entities; $\Delta \mu_{ij}$ represent the amount of change in the network states but have not yet synchronized, and there are a_2 network states; $\Delta \omega_{ij}$ represent the amount of change in the network transactions but have not yet synchronized, and there are a_3 network transactions.

The cost of maintaining consistency among multiple controllers mainly includes two aspects: the synchronization load of each controller and the synchronization overhead between controllers. We analyse the relationship between these costs and synchronization parameters. The synchronization load of a single controller can be quantified as the number of synchronizations per unit time, which is related to the consensus algorithm used during synchronization. To reduce synchronization overhead, a threshold upper limit is set for the synchronization period based on real-time application requirements. When the degree of non-consistency of each network status exceeds the response requirements of the application, the state synchronization is triggered, thereby minimizing communication overhead. For each type of status, the synchronization period needs to be determined based on the application with the highest demand in the current network.

$$Cost = \sum_{j=1}^{a_1} \eta_1 n_1 v_{ij} + \sum_{j=1}^{a_2} \eta_2 n_2 \mu_{ij} + \sum_{j=1}^{a_3} \eta_3 n_3 \omega_{ij}$$

In above, η_1 donates the impact factor of the consensus algorithm (PBFT) on the cost of synchronizing entities, and n_1 represents the maximum number of controllers that need to be synchronized with entities. η_2 donates the impact factor of the consensus algorithm (RAFT) on the cost of synchronizing states, and n_2 represents the maximum number of controllers that need to be synchronized with states. η_3 donates the impact factor of the consensus algorithm (Gossip) on the cost of synchronizing transactions, and n_3 represents the maximum number of controllers that need to be synchronized with transactions.

After analysing the metrics for measuring consistency and cost, it is necessary to find the optimal balance point between them to maximize benefits. The optimization goal is to calculate an optimal synchronization period that ensures that the synchronization level of network states meets application requirements, minimizes the differences in network states between different controllers, and maintains the lowest synchronization overhead. The optimization goal is shown below:

$$minimize(G) = \lambda_1 \ln Cons + \lambda_2 Cost\ ,\ \lambda_1 + \lambda_2 = 1$$

5. NETWORK STATE STORES OPTIMIZING

After completing the generation of consensus slices and the formulation of synchronization strategies, it is necessary to select an appropriate storage method for the network status information of multi-domain networks. The network view contains network status information in various dimensions. Some data describe network configuration and device attributes, and have characteristics such as low frequency, small changes, and high importance. Other data are real-time network status information, with characteristics such as strong real-time performance, large data volume, and tolerance for a certain degree of loss.

5.1. Differentiated Storage Mechanism

To address the different characteristics of network status information, we have designed a differentiated storage mechanism that fully leverages the advantages of blockchain and distributed databases. Network snapshot information is divided into two categories: real-time, high-frequency information (such as real-time control strategy messages, sampling messages, etc.) and low-frequency, critical information (such as network configuration information, device attribute information, etc.).

As shown in Figure 4, the snapshot information is used as input, and the storage medium is determined based on the type. If this type of network snapshot information belongs to real-time, large amounts of information, it is stored in multiple storage nodes of the database, and the distributed architecture achieves multiple backups of the data. If it is low-frequency, critical information, it is stored in a structured manner in the blockchain and the data is read and written through calling smart contracts. At the same time, the real-time data in the database is constantly checked to see if it meets the conditions for hash mapping. If it does, the data is clustered and all the data in the cluster is used as input to the hash algorithm, and the resulting hash value is stored in the blockchain.

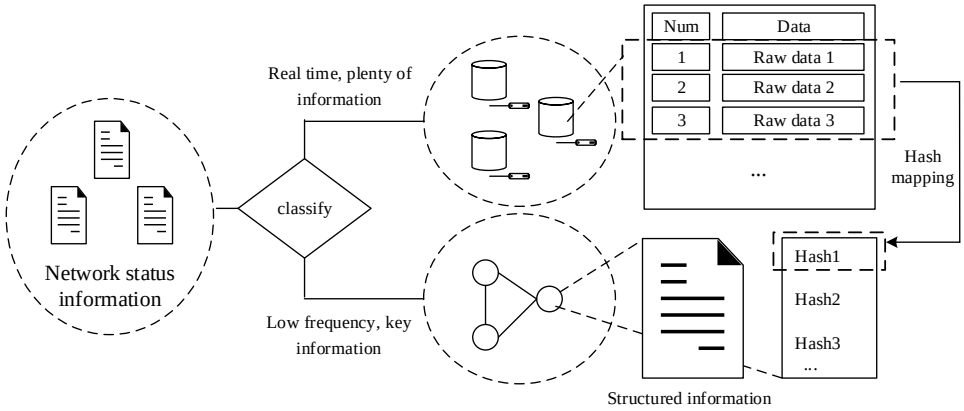

Figure 4. Schematic diagram of differentiated storage

5.2. Hash Mapping Mechanism

Compared with blockchain, distributed databases have better storage performance and are convenient for storing large amounts of data, but they also have the disadvantage of being easily tampered with. If the data in the storage node that the user is connected to is maliciously tampered with, the user is unable to detect it. In this section, a hash mapping mechanism is proposed based on the combination of distributed databases and blockchain technology to verify whether the data in the database has been tampered with. The detailed steps are as follows:

Algorithm 1: Hash Mapping Algorithm
Input: Data table t_1, number of data in each cluster N, start sequence number of this mapping p (initialized as 0) Output: hash value h
Query t to obtain the number of existing data count; Calculate $temp = count\text{-}p$, if $temp < n$ return to 1; end if $hashInput$ = All data in the sequence number interval $[p,p+n]$; Update $p = p+n$; Calculate $hash$ = SHA256 ($hashInput$); Store the $hash$ on the blockchain;

6. EXPERIMENT AND ANALYSIS

6.1. Experiment Environment

The proposed methods have been implemented on the Intel(R) Core(TM) i5-4690, 3.30GHz CPU, 4G RAM over Ubuntu 18.04 LTS system. The Mininet simulator is used to simulate the consistency interaction among multiple controllers. We use Ryu as the controller, and Open vSwitch as switch. The SDN network we generated consists of 20 forwarding nodes and 7 controller nodes. The network is divided into four control areas, distinguished by different colors for the controller nodes. Each control area governs its own forwarding nodes and maintains a local view. Information is synchronized between controller nodes through a data link. In addition, we used Fisco Bcos as the blockchain platform to generate and store blocks containing critical network states. We also utilized a Mysql database to store network states with large data volume. Python and Golang were used as the development tools.

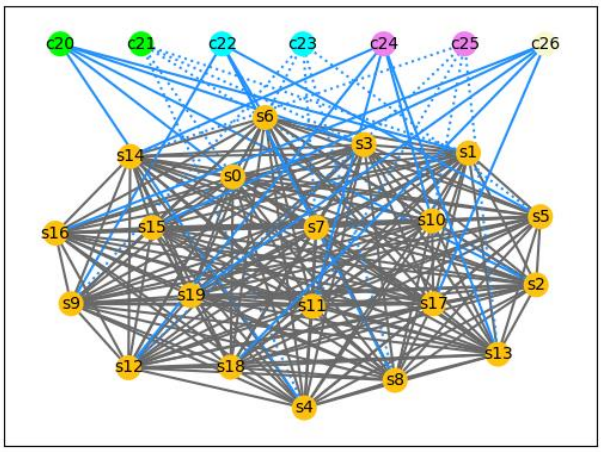

Figure 5. Experimental topology

6.2. Performance of Consensus Slices

To evaluate the correctness and effectiveness of the proposed strategy, we implemented a distributed load balancing application running on top of the controllers. The load distribution among controllers is modelled as the variation of the update rate of each node over time. The average update rate of network events for each node in the topology is randomly selected within the range of [3000, 5000].

We compared the results of the adjustable consensus slice mechanism proposed in this paper with those of the non-adjustable strategy. The initial synchronization period of the adaptive strategy was set to 10 seconds. The consistency levels of these strategies over time are shown in Figure 7, where a lower consistency level is better. In the case of the proposed adaptive strategy (ACS), the consistency level is more stable under intense changes in network load. This is because the synchronization period of the proposed strategy can be adaptively adjusted based on network changes to achieve a certain level of network consistency, whereas the synchronization period of the non-adaptive strategy is fixed.

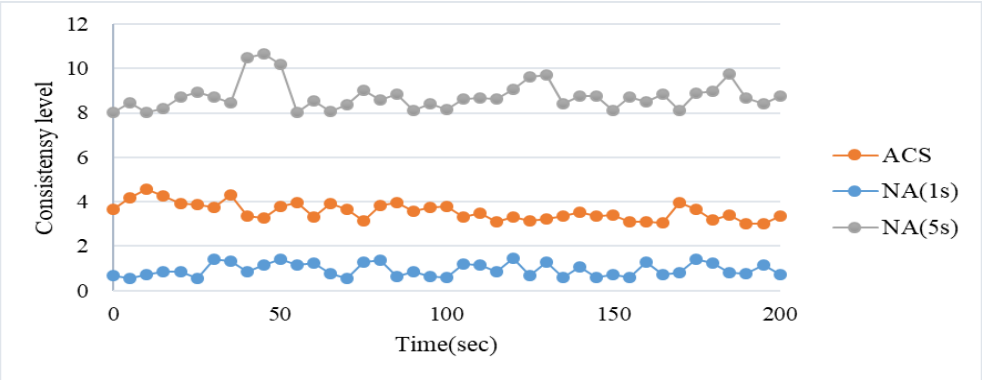

Figure 7. Consistency level comparison

The communication overhead brought by network state synchronization is shown in Figure 8. Compared with NA (1s) and NA (5s), the communication overhead of AS is much lower. This is because AS is based on an elastic synchronization period based on the current level of consistency. In the AS synchronization strategy, the controllers are divided into two roles: primary and secondary controllers, and synchronization communication is only carried out between primary controllers. This approach avoids frequent communication between controllers and significantly reduces synchronization overhead. It can be seen that AS has lower overall communication overhead than non-adaptive strategies.

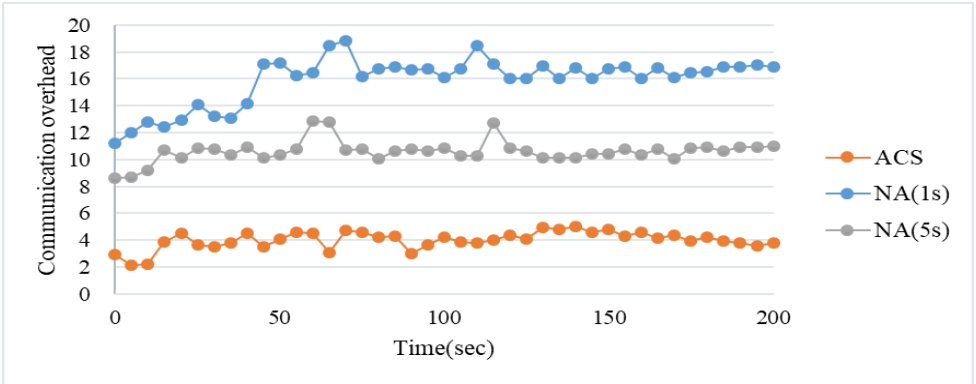

Figure 8. Communication overhead comparison

6.3. Comparison of Storage Overhead

The total latency of differential storage includes four steps: storing in the database, querying the database, computing the hash value, and putting the hash value on the chain. The time spent on each step of the differential storage scheme is shown in Table 2. During the experiment, each group was tested 10 times and the average value was taken. As shown in Table 2, the time spent on storing data packets in the database is the most significant. Compared with other steps, the time for querying the database and computing the hash value is relatively short and is not greatly affected by the number of data packets. The time spent on putting the hash value on the chain fluctuates around 20ms and is not affected by the number of data packets. This is because only one transaction is generated when storing the hash value in the blockchain. That is to say, the consensus only needs to be reached once among the nodes of the blockchain. The total time consumption of differential storage always falls between that of the blockchain and the database. Like the other two schemes, the storage time of this scheme also increases with the number of data packets, but the rate of increase is much slower than that of the blockchain. This advantage becomes more obvious as the number of synchronized data packets increases.

Table 2 Time required for each step in a differentiated network states storage solution

Number of synchronized packets	Time of database storage (ms)	Time of database query (ms)	Time of Hash (ms)	Time of store blockchain(ms)	Total time (ms)
10	34.7	0.787	0.044	20	55.531
20	80	1.756	0.075	25.8	107.631
30	111.5	0.704	0.116	21.8	134.12
40	152.5	0.83	0.177	23	176.507
50	214.1	0.808	0.21	24.8	239.918
60	238.9	1.157	0.217	18.9	259.174
70	266.5	2.061	0.252	22	290.813
80	304.8	1.028	0.317	22.2	328.345
90	316.7	1.065	0.36	23.2	342.325
100	349.9	0.742	0.388	23.2	374.23

7. CONCLUSIONS

We innovatively propose a consensus slicing mechanism to address the problem of network state synchronization among multiple controllers. This solution dynamically formulates synchronization strategies based on network application requirements and real-time network status, minimizing consistency differences between controllers and reducing the performance overhead of network state synchronization. In terms of specific implementation, we utilize a storage method that combines blockchain with distributed databases to store the global network view, improving information storage and query efficiency while ensuring the tamper-proofing of state information. Finally, we validate the effectiveness of the proposed consensus slicing mechanism through experiments and provide the time overhead of the proposed storage method.

REFERENCES

[1] Poularakis K, Qin Q, Ma L, et al. Learning the optimal synchronization rates in distributed SDN control architectures[C]//IEEE INFOCOM 2019-IEEE Conference on Computer Communications. IEEE, 2019: 1099-1107.
[2] Zhang B, Wang X, Huang M. Adaptive consistency strategy of multiple controllers in SDN[J]. IEEE Access, 2018, 6: 78640-78649.
[3] Bannour F, Souihi S, Mellouk A. Adaptive distributed SDN controllers: Application to content-centric delivery networks[J]. Future Generation Computer Systems, 2020, 113: 78-93.
[4] Muqaddas A S, Giaccone P, Bianco A, et al. Inter-controller traffic to support consistency in ONOS clusters[J]. IEEE Transactions on Network and Service Management, 2017, 14(4): 1018-1031.
[5] Yu T, Hong Y, Cui H, et al. A survey of Multi-controllers Consistency on SDN[C]//2018 4th International Conference on Universal Village (UV). IEEE, 2018: 1-6.
[6] Hu T, Guo Z, Yi P, et al. Multi-controller based software-defined networking: A survey[J]. IEEE access, 2018, 6: 15980-15996.
[7] Foerster K T, Schmid S, Vissicchio S. Survey of consistent software-defined network updates[J]. IEEE Communications Surveys & Tutorials, 2018, 21(2): 1435-1461.
[8] Zhang T, Giaccone P, Bianco A, et al. The role of the inter-controller consensus in the placement of distributed SDN controllers[J]. Computer Communications, 2017, 113: 1-13.
[9] Poularakis K, Qin Q, Ma L, et al. Learning the optimal synchronization rates in distributed SDN control architectures[C]//IEEE INFOCOM 2019-IEEE Conference on Computer Communications. IEEE, 2019: 1099-1107.
[10] Aslan M, Matrawy A. A clustering-based consistency adaptation strategy for distributed SDN controllers[C]//2018 4th IEEE Conference on Network Softwarization and Workshops (netsoft). IEEE, 2018: 441-448.
[11] Bannour F, Souihi S, Mellouk A. Adaptive state consistency for distributed onos controllers[C]//2018 IEEE Global Communications Conference (Globecom). IEEE, 2018: 1-7.
[12] Backman J, Yrjölä S, Valtanen K, et al. Blockchain network slice broker in 5G: Slice leasing in factory of the future use case[C]//2017 Internet of Things Business Models, Users, and Networks. IEEE, 2017: 1-8.
[13] Addad R A, Taleb T, Flinck H, et al. Network slice mobility in next generation mobile systems: Challenges and potential solutions[J]. IEEE Network, 2020, 34(1): 84-93.

AUTHOR

Shanqing Jiang is currently pursuing the Ph.D. degree in the School of Cyber Science and Engineering, Southeast University. His research interests include network resilience, SDN architecture, and application of blockchain.