Blockchain-Based Implementation of Service Function Chains in Multicloud and Multitenant Environments Over Containerized Networks

Rui Kang[2], Mengfei Zhu[1,2], and Klaus-Tycho Foerster[3],
[1]China Mobile Group Design Institute Co. Ltd., Beijing, China
[2]Kyoto University, Kyoto, Japan
[3]TU Dortmund, Dortmund, Germany

kang.rui.u25@kyoto-u.jp, zhumengfei@cmdi.chinamobile.com, klaus-tycho.foerster@tu-dortmund.de

***Abstract*—To address the growing demand for secure, flexible services, this paper emphasizes the integration of zero-trust security principles within service function chains (SFCs). Zero-trust model reduces security risks associated with the complex and customized nature of contemporary service delivery. SFCs deliver these flexible services, but current implementations often require configuring specific forwarding devices or network plugins, complicating deployment and maintenance. Additionally, these implementations generally lack robust mechanisms for ensuring trustworthiness and traceability, crucial for preventing data tampering and facilitating effective monitoring and rapid responses to security incidents. This paper presents a four-layer architecture, termed SFC-BC, which leverages Kubernetes for managing containerized networks and Hyperledger Fabric for blockchain-based management. Employing distributed databases with synchronization and hash verification, SFC-BC enhances the traceability and reliability of data. We have developed a prototype to demonstrate the practical implementation of SFC-BC, proving that it prevents unauthenticated access to SFCs.This holistic approach facilitates efficient SFC deployment across diverse environments without the intricacies typically involved in network configuration, operation, and management.**

I. Introduction

As demand grows for flexible and sophisticated services, customers seek customized solutions that meet the service level agreements. For example, video streaming services vary based on user requirements, traffic locations, customer preferences, network status, and other policies. Given the complex security needs of these services, and the dynamic, often less-controlled environments they operate in, traditional security models fall short. The increasing personalization of services also escalates the risks to privacy and data integrity, necessitating a security approach. Incorporating zero-trust principles offers a robust security framework that continuously verifies and authenticates all entities within the network. Service providers can deploy these services across multiple data centers [1], [2]. However, assembling service functions (SFs) to configure elastic and reliable services is complex, particularly in multicloud and multitenant environments. Implementing zero-trust in these environments strengthens security by rigorously verifying every access request.

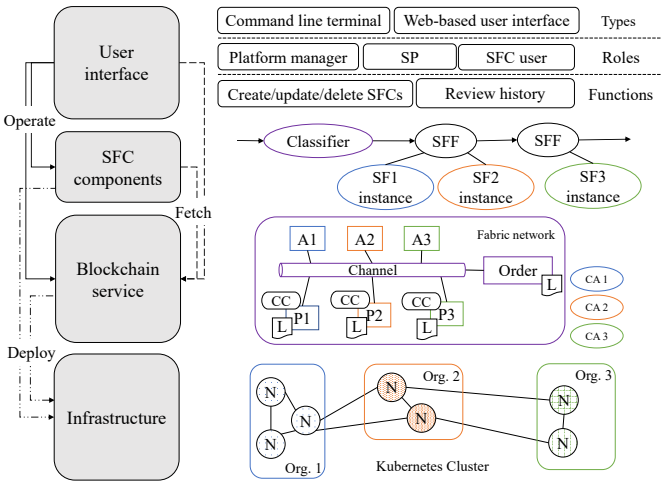
Integrating a micro-service architecture into network application design enhances the delivery of large, complex applications by utilizing small, independent network functions. Network function virtualization (NFV) separates these functions from specific hardware [3], and through NFV, a SFC of ordered, virtualized functions is established [4], serving users' needs economically and flexibly. Each function is containerized, allowing execution within a container rather than on dedicated hardware.

In the zero-trust model, encapsulation and traffic steering technologies of SFC support dynamic security strategies. This model requires that all service provider (SPs), users, and services be authenticated and authorized for each access. Encapsulation verifies packet integrity and origin, while traffic steering routes traffic only to authorized SFs based on current security policies. However, practical implementations of SFC often struggle to align with these zero-trust requirements.

Existing SFC implementations primarily attach additional information to manage traffic. Researches in [5]–[10] shows traffic flow matching based on protocol headers, which is unreliable across public networks due to broadcast domains and network address translation. Other studies [11]–[13] differentiate traffic flows in SFCs using tags, protocol headers, and tunneling, resulting in extra packet headers or payloads. In Kubernetes, traffic classification to an SFC is achieved by identifying traffic types via specified container network interface (CNI) [14] or network plugins like OVN4NFV [15] and Nodus [16] which are limited to rerouting SFs across different networks. Most current SFC architectures rely on software-defined networking (SDN) due to the lack of physical switches and SFs that support SFC encapsulation, such as the network service header (NSH) introduced in [17] [18]. A demonstration in [19] highlighted an SFC-supported CNI implementation by the open virtual network [20] based on SDN in Kubernetes. However, these methods fall short in addressing the essential requirements for designing SFCs within zero-trust frameworks, which demand rigorous security protocols and flexible access controls. Addressing these shortcomings is critical to bolster both the security and operational capabilities of SFC architectures.

A blockchain-based approach advances SFC implementations towards zero-trust by transitioning from centralized to distributed management, thereby enhancing trust across tenants and services. This shift facilitates continuous verification of all SFs, ensures robust access control in multicloud environments, and upholds data integrity through strict isolation and dynamic security policies.

Rui Kang and Mengfei Zhu contribute equally to the article.



Note: SP: service provider; SFC: service function chain; SFF: service function forwarder; SF: service function; A: application; L: ledger; CC: chaincode; CA: certification authority; P: peer node; N: physical node.

Fig. 1: Overall structure of SFC-BC.

II. SFC-BC DESIGN

This paper presents the SFC-BC, a framework that leverages blockchain technology to enhance the security and operational integrity of SFCs. This architecture not only secures traceability and trustworthiness across multicloud and multitenant environments but also complies with zero-trust security principles, providing a robust mechanism for continuous verification and decentralized management of SFs. SFC-BC is designed to enable the dynamic integration of SFs from multiple SPs within Kubernetes, focusing on data safety and isolation in a multitenancy and multicloud context. It supports SFCs in containerized networks without requiring specific network plugins, thus cutting deployment and operational costs associated with SDN or extra tags for traffic steering. Unlike previous models, traffic steering in SFC-BC is decoupled from forwarding devices and directly integrated into SFs, reducing load on hardware and enhancing control granularity. The architecture leverages distributed databases to store configurations of SFCs and usage records of SFs, thereby enhancing traceability and enabling comprehensive auditability. It also boosts data reliability and increases tamper-resistance through multi-database synchronization and hash verification. To bolster privacy, user and SP data are isolated at both the data and network layers. Communications between different SFCs are segregated into individual channels (private 'subnets'), restricting access to the members of each channel to only the users of the SFC and the involved SPs. Moreover, sensitive information, such as passwords, is exclusively shared between the user and a specific SP. Within the network cluster, namespaces are utilized to isolate resources, with policies specifically designed to segregate traffic at the IP address or port levels between different namespaces.

Figure 1 illustrates the overall structure of SFC-BC, which consists of four layers:

Infrastructure layer : this base layer is a Kubernetes cluster shared by various organizations such as SPs and the manager of SFC-BC. Each organization can designate specific physical nodes to host their private SFC components and blockchain services, such as SFs and peer nodes. For example, Figure 1 shows three organizations each managing different numbers of worker nodes, allowing them to control their respective components within SFC-BC.

Blockchain service layer : deployed on the physical nodes, this layer stores SFC configurations and historical records. It also supports user interfaces and SFC components for auditing and tracing, manages certification and data security via certificate authorities, and establishes secure connections using transport layer security (TLS). Each organization maintains at least one peer node that connects to a channel to manage ledgers and smart contracts, which record SFC configurations, status updates, and authentication data. Channels are created for each SFC and include the SPs involved and the SFC-BC manager. Orderer nodes sequence transactions within this framework. The blockchain service in SFC-BC is instrumental in storing configurations and operational records of SFCs. Utilizing Hyperledger Fabric, this service validates received data against SFC configurations and previous records from the last hop SF, identifies the addresses of subsequent hops, and logs processing information. This ensures a secure and verifiable flow of information, which is crucial for maintaining the integrity of the network operations.

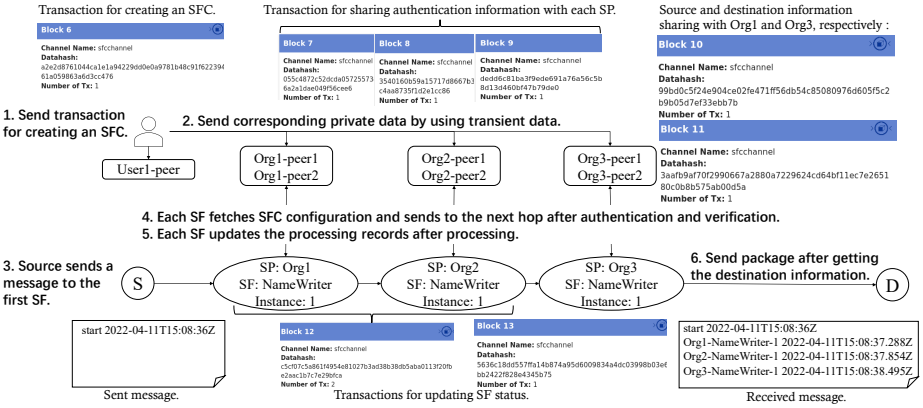
SFC components : these components retrieve SFC configurations from the blockchain service, recognizing the current SF position, verifying packet/frame integrity, and logging operational data. The classifier, managed by the SFC-BC manager, optionally directs traffic to the initial SFC entry based on security or performance needs. SFFs function as standard forwarding devices, not relying on specific SFC encapsulations [13], [17], [18]. SFs execute required functions and coordinate with blockchain applications to transmit data to subsequent SFs.

Each SF checks the trustworthiness of incoming data. If validated, the data are processed and passed to the next hop and recorded on the blockchain. Unvalidated data are discarded, and the failure is logged. Validation policies, configurable per SF, include *none* (no validation), *hash* (verification based on data hash values), *address* (verification based on addresses and ports). The combination of *hash* and *address* is also possible.

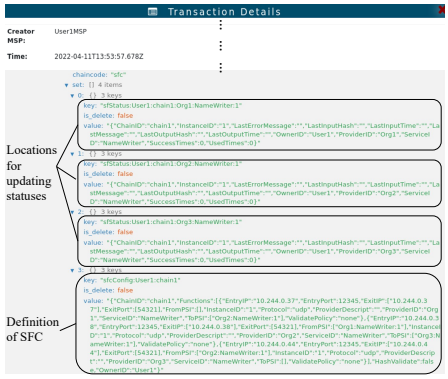
User interface : this interface facilitates the management of SFC configurations and the querying of historical records. Both command-line and web-based interfaces are available for SFC-BC managers, SPs, and users to manage and review SFC and SF records.

III. DEMONSTRATIONS AND EVALUATIONS

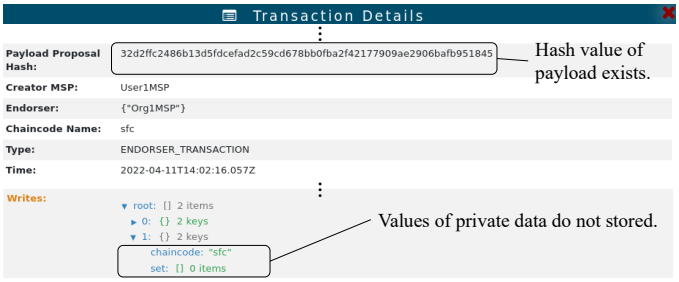
We demonstrate the functions and evaluate the performances of SFC-BC in this section. We create Kubernetes cluster (ver-



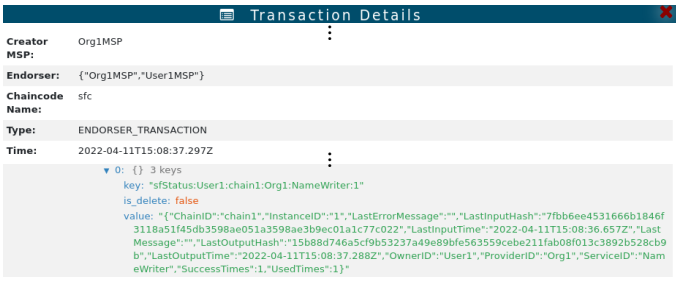
(a) Sketch and block information during functionality test. S: source. D: destination.



(b) Transaction of creating SFC.



(c) Example of transaction to share private data.



(d) Example of transaction to submit SF status.

Fig. 2: Sketch and screenshots of transactions for functionality test.

sion 1.23.4) for demonstrations. The version of Hyperledger Fabric is 2.4.3.

A. Functionality demonstration

We show the functionality of SFC-BC by a short SFC including three SFs from three SPs. Each SF receives the messages in UDP datagrams and appends its name and current timestamp, and then sends the messages to the next hop. The sketch of demonstration and screenshots of results are shown in Fig. 2.

After approving and committing the chaincode, eight blocks (blocks 6-13) in the blockchain are created during one transmission by the creation and usage of the SFC. Block 6 contains a transaction to create an SFC and initialize the records of SF statuses, whose details are shown in Fig. 2(a). Blocks 7-11 contain the transactions for sharing the private data (authentication and source&destination information) between the user and a specific SP. As an example which shares the user name and password between the user (*User1*) and SP (*Org1*) in Fig. 2(b), the hash value of payload is recorded in the blockchain without specific values, which protects the sensitive information of the user from being disclosed. At the same time, other SPs cannot get the sensitive information from the public ledger. Blocks 12 and 13 contain three transactions for submitting the updates of SF statues. Two transactions are included in block 12 since the transactions arrive within the batch timeout set in Fabric. An example of submitting the SF status is shown in Fig. 2(c). In the example, the SF whose PSIID is *Org:NameWriter:1* in *chain1* owned by *User1* is submitted by *Org1*. In this test, the message sent by the source is received by the destination after passing and processed by three SFs.

B. Test for unauthenticated usages and messages with forged source

If the user of an SFC sets a wrong authentication information with an SP, the error message is submitted and stored in blockchain when a message is received by an SF provided by the SP as shown in Fig. 3. At the same time, the SF stops processing the traffic and refuses to send to the next hop.

SFC-BC enables the validation of traffic from the previous hops by setting a field in the SFC configurations, which is used to defend the attacks and usages from the forged sources. When an SF received messages from an unregistered source or the hash value of input data does not match the hash value of the data output from the previous hop, the error message is submitted and stored in blockchain as shown in Fig. 3. At the same time, the SF stops processing the traffic and refuses to send to the next hop.

When the transmission rate is relatively high so that the updating cannot be submitted to the blockchain before the next submission, the keys of SF status records can be attached with random prefixes or suffixes to avoid dirty writes which lead *MVCC_READ_CONFLICT* errors in Fabric. In this demonstration, we only use one key for each record.

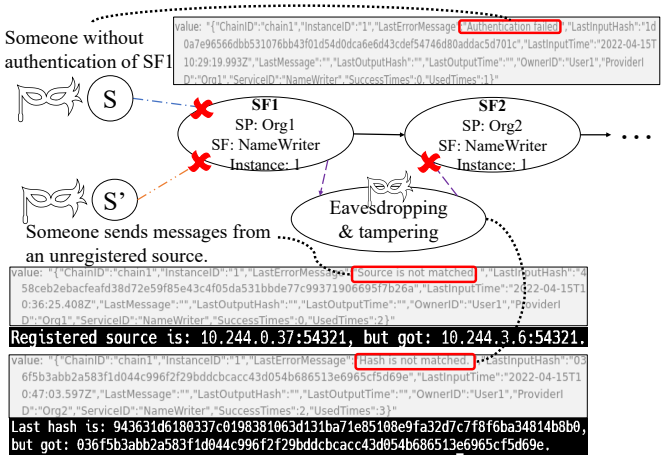


Fig. 3: Errors are recorded into blockchain, when authentication fails, traffic source is not authenticated, and the hash verification fails. S: source. D: destination.

The hash validation introduces extra connections with the *peer* nodes for submitting and querying the input hash values and the output hash values from the last hops, respectively. Compared with the situations without validation and the validation which only validate the source addresses, ports, and protocols of incoming packets, the hash validation has higher processing latency.

The hash validation mainly plays two roles in SFC-BC. Firstly, it works for the security of SFs and data cooperating with the network policies set by Kubernetes, which refuse the connections from untrustful sources and namespaces. Secondly, if more than one SFC is created in one channel, which is not recommended, hash values can be used to distinguish the packets from different SFCs.

IV. Conclusion

This paper proposed SFC-BC, an innovative architecture for implementing SFCs in Kubernetes environments, leveraging the capabilities of Hyperledger Fabric for achieveing zero-trust. Designed for multicloud and multitenant contexts, SFC-BC provides a trustful, traceable, and comprehensive solution that alleviates the complexities associated with configuring and interconnecting diverse underlying technologies. The architecture encompasses several layers, each dedicated to distinct aspects of service delivery and management. The infrastructure layer facilitates the deployment of containerized SFC and blockchain elements, offering isolated computing and network resources. The blockchain service layer is instrumental in storing and managing configurations and operational records of SFCs, enabling precise validation and data processing throughout the network. Traffic steering is managed within the SFC component layer, optimizing data flow and routing for enhanced service delivery. Additionally, the user interface layer supplies intuitive graphical and command-line tools that empower users to efficiently manage and operate SFCs. Our prototype demonstrations validate the functionality of SFC-BC, supporting complex SFC configurations and robust authentication and validation mechanisms.

References

[1] S. Paul, R. Jain, M. Samaka, and J. Pan, "Application delivery in multi-cloud environments using software defined networking," *Computer Networks*, vol. 68, pp. 166–186, 2014.

[2] D. Bhamare, R. Jain, M. Samaka, and A. Erbad, "A survey on service function chaining," *Journal of Network and Computer Applications*, vol. 75, pp. 138–155, 2016.

[3] J. Martins, M. Ahmed, C. Raiciu, V. Olteanu, M. Honda, R. Bifulco, and F. Huici, "ClickOS and the art of network function virtualization," in *11th USENIX Symp. on Networked Syst. Design and Implementation (NSDI 14)*, 2014, pp. 459–473.

[4] J. M. Halpern and C. Pignataro, "Service Function Chaining (SFC) Architecture," RFC 7665, Oct. 2015. [Online]. Available: https://www.rfc-editor.org/info/rfc7665

[5] J. Blendin, J. Rückert, N. Leymann, G. Schyguda, and D. Hausheer, "Position paper: Software-defined network service chaining," in *2014 Third European Workshop on Software Defined Networks*, 2014, pp. 109–114.

[6] B. Martini, F. Paganelli, A. Mohammed, M. Gharbaoui, A. Sgambelluri, and P. Castoldi, "SDN controller for context-aware data delivery in dynamic service chaining," in *Proceedings of the 2015 1st IEEE Conference on Network Softwarization (NetSoft)*. IEEE, 2015, pp. 1–5.

[7] Y. Zhang, N. Beheshti, L. Beliveau, G. Lefebvre, R. Manghirmalani, R. Mishra, R. Patneyt, M. Shirazipour, R. Subrahmaniam, C. Truchan, and M. Tatipamula, "Steering: A software-defined networking for inline service chaining," in *2013 21st IEEE International Conference on Network Protocols (ICNP)*, 2013, pp. 1–10.

[8] A. Abujoda and P. Papadimitriou, "Midas: Middlebox discovery and selection for on-path flow processing," in *2015 7th International Conference on Communication Systems and Networks*, 2015, pp. 1–8.

[9] A. Csoma, B. Sonkoly, L. Csikor, F. Németh, A. Gulyás, W. Tavernier, and S. Sahhaf, "Escape: Extensible service chain prototyping environment using mininet, click, netconf and pox," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 4, pp. 125–126, 2014.

[10] F. Callegati, W. Cerroni, C. Contoli, and G. Santandrea, "Dynamic chaining of virtual network functions in cloud-based edge networks," in *Proceedings of the 2015 1st IEEE Conference on Network Softwarization (NetSoft)*. IEEE, 2015, pp. 1–5.

[11] Z. Qazi, C.-C. Tu, R. Miao, L. Chiang, V. Sekar, and M. Yu, "Practical and incremental convergence between SDN and middleboxes," *Open Network Summit, Santa Clara, CA*, 2013.

[12] S. K. Fayazbakhsh, L. Chiang, V. Sekar, M. Yu, and J. C. Mogul, "Enforcing Network-Wide policies in the presence of dynamic middlebox actions using FlowTags," in *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*, 2014, pp. 543–546.

[13] R. Kang, F. He, T. Sato, and E. Oki, "Demonstration of network service header based service function chain application with function allocation model," in *NOMS 2020 - 2020 IEEE/IFIP Network Operations and Management Symposium*, 2020, pp. 1–2.

[14] The CNI Authors, "CNI," https://www.cni.dev.

[15] OPNFV, "opnfv/ovn4nfv-k8s-plugin," https://github.com/opnfv/ovn4nfv-k8s-plugin.

[16] Akraino, "akraino-edge-stack/icn-nodus," https://github.com/akraino-edge-stack/icn-nodus.

[17] P. Quinn, U. Elzur, and C. Pignataro, "Network Service Header (NSH)," RFC 8300, Jan. 2018. [Online]. Available: https://www.rfc-editor.org/info/rfc8300

[18] A. M. Medhat, T. Taleb, A. Elmangoush, G. A. Carella, S. Covaci, and T. Magedanz, "Service function chaining in next generation networks: State of the art and research challenges," *IEEE Communications Magazine*, vol. 55, no. 2, pp. 216–223, 2017.

[19] R. Kang, M. Zhu, and E. Oki, "Implementation of service function chain deployment with allocation models in Kubernetes," in *2022 IEEE Conference on Computer Communications (INFOCOM) Workshops*, May 2022.

[20] The OVN community, "OVN, Open Virtual Network :: OVN project documentation website," https://www.ovn.org/en/, accessed Oct. 12, 2021.