A Multi-Provider Service Function Chain Paradigm for Flexible Composition

Rui Kang†, Mengfei Zhu*†, Duling Xu‡, and Tong Li‡
†Kyoto University, Kyoto, Japan ‡ Renmin University of China, Beijing, China
*China Mobile Group Design Institute Co. Ltd., Beijing, China

***Abstract*—With the rise of cloud-native applications, service function chains (SFCs) are increasingly composed of functions from multiple service providers (SPs) but often require manual adjustments when replacing or reconfiguring functions from different SPs, limiting flexibility. This paper proposes a blockchain-based paradigm that supports transparent function switching, secure multi-provider coordination, and streamlined communication, simplifying service orchestration across multiple SPs. Demonstrations validate its effectiveness.**

I. Introduction

A service function chain (SFC) represents a sequence of functions that process data flows in a predefined order [1]. Traditionally, SFCs focus on network functions like firewalls and load balancers, operating at the packet level to enforce routing policies. With the adoption of cloud-native architectures, application-oriented SFCs have emerged, locating in the application layer rather than network layer. For example, a data analytics SFC may include ingestion, preprocessing, aggregation, and machine learning; an IoT SFC may handle sensor data collection, event detection, and control signaling. Users assemble these service functions (SFs) from a subscription-based pool provided by different service providers (SPs).

The shift toward multi-provider SFCs highlights the need for flexible service composition. However, existing implementations require manual reconfiguration when switching SFs and often rely on a north-south interaction model, where functions send results back to a central source instead of passing data directly [2], as shown in Fig. 1. This reliance on a central source necessitates updates to data handling, routing, and formatting whenever an SF is replaced, increasing interaction points, reconfiguration efforts, and interface adjustments. Consequently, this back-and-forth communication introduces additional overhead, latency, and complexity to SF switching.

To address these limitations, we propose a service paradigm that simplifies user interaction and supports flexible service composition. The user interacts only once with the entry point of the chain (SF1), unlike the traditional north-south model that requires repeated interactions at each step. Subsequent functions (SF2, SF3, etc.) coordinate autonomously through an "east-west" communication pattern, as shown in the lower part of Fig. 1, thereby reducing user interactions.

The figure illustrates a scenario where SF4 replaces SF2. In a traditional multi-request SFC, this replacement would require manual modifications to the application (upper part of Fig. 1). In contrast, our proposed paradigm enables transparent

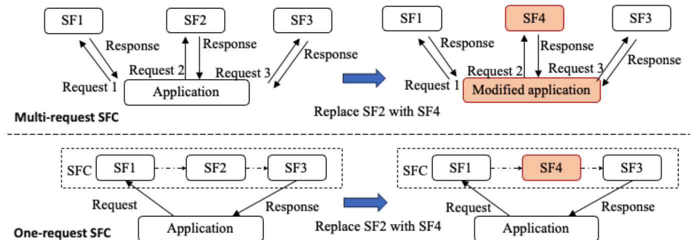


Figure 1: Impact of SF replacement in traditional multi-request SFC and proposed one-request service paradigm.

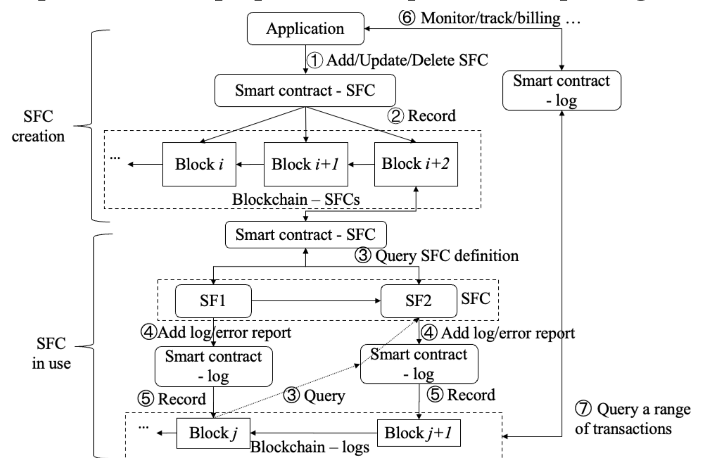


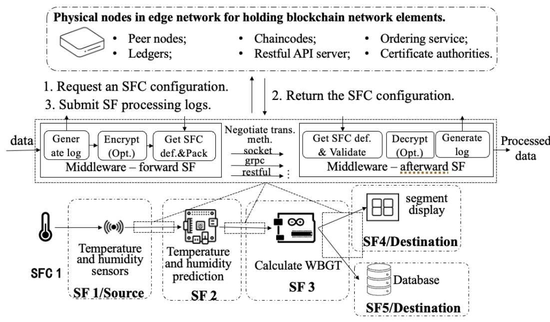
Figure 2: Overall structure.

and flexible service switching without requiring changes to the application or interface. This enables direct communication between functions, reducing user overhead, and ensuring seamless service execution without repeated user intervention.

However, this paradigm shift introduces multiple design and security challenges. One key challenge is the dynamic discovery and validation of subsequent components, which requires to manage interactions and verify the credibility of incoming data from previous hops. Source authentication is crucial to prevent tampered data, as compromised inputs could lead to incorrect billing, unauthorized access, or data leaks. Additionally, accurate resource accounting is necessary to avoid misinterpretation of requests or overcharging by downstream providers, especially when multiple concurrent interactions must be tracked efficiently.

To address these issues, we incorporate blockchain as a decentralized, tamper-proof mechanism to support secure, traceability, and east-west traffic steering. By leveraging Hyper-

This work is supported by the National Natural Science Foundation of China (No.62202473, No. 62441230).
Rui Kang and Mengfei Zhu contributed equally. Corresponding author: Tong Li (tong.li@ruc.edu.cn).

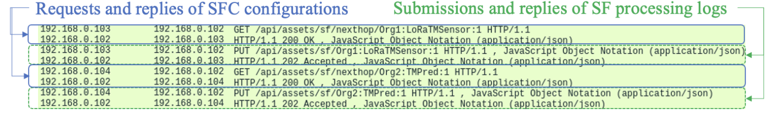


(a) Demonstration setup.

Record for SF processing log

set: [] 1 item
0: {} 3 keys
key: "sfStatus:User1:chain1:Org1:LoRaTMSensor:1"
is_delete: false
value: "{"ChainID":"chain1","InstanceID":"1","LastErrorMessage":"","LastInputHash":"ced906381c4de9fe1a0ec0d440f838ce05f0f14d562700c38af72342d9b93765","LastInputTime":"2022-05-30 16:06:51","LastMessage":"","LastOutputHash":"158157d25c0e1546f8c1ee87e2ade5380d29bd4f9cb5f886a8f2d3230feb3cdc","LastOutputTime":"2022-05-30 16:06:51","OwnerID":"User1","ProviderID":"Org1","ServiceID":"LoRaTMSensor","SuccessTimes":1,"UsedTimes":1}"

(b) Example of SF processing log stored in blockchain.



(c) Examples of communications through RESTful API.

Figure 3: Settings and results in demonstration.

ledger Fabric, the system records SFC configurations and execution logs, enabling functions to verify the origin and integrity of data without relying on a central authority. This distributed ledger facilitates direct interactions between functions, ensuring that east-west traffic flows seamlessly along the chain without unnecessary detours to central sources.

II. System Design

A. Overall structure

To realize the proposed service paradigm, we design the system as shown in Fig. 2. The user decomposes the requested service into SFs, selects providers from a subscription-based pool, and assembles them into an SFC. A request to create, modify, or delete the SFC configuration is submitted to the blockchain via a smart contract for managing configurations (①). The smart contract records configuration changes on the blockchain to ensure the latest state and track history (②).

During service execution, middleware at each SF manages traffic between SFs. The source sends data only to the first SF, eliminating the need for user interaction with subsequent SFs or manual error handling. Each SF checks the SFC definition and latest log to verify the source and determine the next hop (③). It then generates logs for usage and errors, which are recorded on the blockchain via smart contracts (④⑤) to monitor operations and support billing (⑥⑦).

B. SFC definition

An SFC configuration stores metadata (identifiers and settings) and descriptions of SFs, including their identifiers, settings, network addresses, and ports. The SF logs include the following information: SF ID, usage and success counts, error and debugging messages, and hash values with timestamps of input and output data. For security, authentication credentials between a specific SP and a user are exchanged using private and transient data to ensure confidentiality.

C. Middleware

To parse and manage traffic between SFs, middleware is attached to each SF. Based on the SF definition in the subscription pool, the middleware listens on the corresponding port by creating a socket or launching a proxy server. When traffic arrives, the middleware parses the data according to the interface specifications defined by the SF. It retrieves the SFC definition to which it belongs and validates the credibility of the previous hop using the SFC configuration or the hash value recorded in the latest log. If the previous hop is deemed untrustworthy, the middleware drops the traffic and logs the event to the blockchain. Otherwise, the data passes through an optional decryption module and is then forwarded to the SF.

Once the SF processes the data, the middleware intercepts the output. The output may pass through an optional encryption module before being packaged and sent according to the SFC definition stored in the blockchain, including the interface format, next-hop SF, and its IP address. Finally, the middleware records the operation and the hash value of the sent data in the blockchain.

III. Demonstration

We design an SFC with five SFs from different SPs to collect and process temperature and moisture data, as shown in Fig.3(a). The demonstrated SFC is configured to issue alerts for heatstroke risks in the next time slot, based on the wet-bulb globe temperature (WBGT). The SFC configuration is generated and uploaded to the blockchain. The source, SF1, retrieves the configuration from the blockchain via a RESTful API to obtain the necessary settings, as shown in Fig.3(c), and then sends the temperature and humidity data to the next hops. Once processing is completed, a log is submitted to the blockchain through the RESTful API, as shown in Figs. 3(b) and 3(c). The intermediate SFs, SF2 and SF3, receive the data, retrieve configurations and logs from the blockchain, verify authorization information, validate the correctness and reliability of the data source, process the data, send it to the next hops, and submit their processing logs. The final SFs, SF4 and SF5, execute their assigned actions according to the settings in the SFC configuration.

IV. Conclusion

We proposed a service paradigm for SFCs that uses an east-west communication model, where users interact with the SFC only once. Middleware handles SF communication, error handling, and logic processing, simplifying development and enabling seamless switching of functions without modifications. The demonstration shows that the middleware ensures secure data flow, source authentication, and accurate resource accounting, addressing key design and security challenges.

References

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

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