Düzce University
Journal of Science & Technology

Research Article

Implementation of Blockchain-Assisted Source Routing for Traffic Management in Software-Defined Networks

Murat KARAKUŞ [a,*]

[a] *Department of Software Engineering, Faculty of Engineering, Ankara University, Ankara, TÜRKİYE*
* *Corresponding author's e-mail address:* mrtkarakus@ankara.edu.tr
DOI: 10.29130/dubited.1209656

ABSTRACT

The control and infrastructure layers are split into Software-Defined Networks (SDNs). With the control and infrastructure planes split, new network applications may be developed with more simplicity and greater independence. On the other hand, the disadvantages of SDN create a slew of questions. In large-scale networks, such as Wide Area Networks (WANs) covering huge areas, more propagation delays substantially contribute to network convergence time. In addition, traditional SDN restricts network design flexibility due to the influence of controller location on network performance in large-scale networks. SDN-based source routing (SR) has emerged as a viable solution to the issues above, where the packet header field is used to specify a packet's route. This study presents an SR-based End-to-End (E2E) traffic management framework called *SoRBlock*. In *SoRBlock*, inter-domain routing uses blockchain technology, while intra-domain routing relies on the SR technique in SDNs. The simulation results show that the proposed SR-based *SoRBlock* framework outperforms the traditional hierarchical routing approach, *HRA*, in SDN networks by lowering path setup time (PST) and the number of controller messages. While the same (i.e., identical origin and target) service requests were used for all runs in the simulations, the proposed *SoRBlock* architecture presents almost three times less total PST between 45*ms* and 65*ms* than the *HRA* method between 130*ms* and 200*ms* due to the *HRA* approach's increased node-controller and controller-controller latencies. On the other hand, *SoRBlock* shows two times less PST ([75*ms* – 90*ms*]) than HRA ([150*ms* – 175*ms*]) when different service requests (i.e., different origin and target) were used. Concerning Controller Messages Processed (CMP), the *HRA* deals nearly 50% more controller messages between 7 and 15 than the *SoRBlock* between 3 and 10 when the number of domains varies, while the CMP in the *SoRBlock* scheme ([10 - 17]) approaches that in the *HRA* framework ([15 - 20]) regarding the ratio while the count of nodes rises in domains.

Keywords: *Source Routing, Blockchain, Software Defined Networks, Traffic Management*

Yazılım Tanımlı Ağlarda Trafik Yönetimi İçin Blokzincir Destekli Kaynak Yönlendirmesinin Uygulanması

ÖZ

Kontrol ve veri katmanları, Yazılım Tanımlı Ağlarda (YTA) bölünmüştür. Kontrol ve veri düzlemlerinin ayrılmasıyla, yeni ağ uygulamaları daha basit ve bağımsız bir şekilde geliştirilebilir. Öte yandan, Yazılım Tanımlı Ağların dezavantajları birçok sorun oluşturmaktadır. Geniş Alan Ağları (WAN'lar) gibi büyük ölçekli ağlarda, daha fazla yayılma gecikmesi, ağ yakınsama süresine önemli ölçüde katkıda bulunmaktadır. Ek olarak, geleneksel YTA, büyük ölçekli ağlarda denetleyici konumunun ağ performansı üzerindeki etkisi nedeniyle ağ tasarım esnekliğini kısıtlar. YTA-bazlı kaynak yönlendirmesi, paket başlık alanının bir paketin ağ üzerindeki yolunu belirtmek için kullanıldığı ve yukarıdaki sorunlara uygulanabilir bir çözüm olarak ortaya çıkmıştır. Bu çalışma, *SoRBlock* adlı kaynak yönlendirme tabanlı uçtan uca trafik yönetimi çerçevesini sunmaktadır.

Received: 24/11/2022, Revised: 12/04/2023, Accepted: 13/04/2023

SoRBlock'ta, ağlar arası yönlendirme, blokzincir teknolojisini kullanırken, ağ içi yönlendirme, YTA-bazlı kaynak yönlendirme tekniğine dayanmaktadır. Simülasyon sonuçları, önerilen kaynak yönlendirme tabanlı *SoRBlock* çerçevesinin, yol kurulum süresini (Path Setup Time - PST) ve işlenen denetleyici mesajlarının (Controller Messages Processed - CMP) sayısını azaltarak YTA ağlarında geleneksel hiyerarşik yönlendirme yaklaşımı olan *HRA*'dan daha iyi performans gösterdiğini göstermektedir. Önerilen *SoRBlock* mimarisi 45*ms* ve 65*ms* aralığında olmak üzere, tüm simülasyon çalıştırmalarında aynı (aynı kaynak ve hedef düğüm) hizmet isteklerinin kullanıldığı senaryoda, *HRA* yaklaşımının artan düğüm - denetleyici ve denetleyici - denetleyici gecikmelerinden dolayı *HRA* yönteminden 130*ms* ve 200*ms* aralığında olmak üzere neredeyse üç kat daha az toplam PST sunmaktadır. Öte yandan *SoRBlock* ([75ms – 90ms]), farklı hizmet istekleri (farklı kaynak ve hedef) kullanıldığında *HRA*'dan ([150ms – 175ms]) iki kat daha az PST göstermektedir. İşlenen Denetleyici Mesajları (CMP) bakımından, etki alanı (domain) sayısı arttığında *HRA* ([7 - 15]), *SoRBlock*'tan ([3 - 10]) yaklaşık %50 daha fazla denetleyici mesajı işlerken, *SoRBlock* çerçevesinde ki CMP ([10 - 17]), *HRA* çerçevesinde ([15 - 20]) CMP'ye, etki alanlarındaki düğüm sayısı artarken oran olarak yaklaşmaktadır.

Anahtar Kelimeler: *Kaynak Yönlendirme, Blokzincir, Yazılım Tanımlı Ağlar, Trafik Yönetimi*

I. INTRODUCTION

The centralized operations and administration in Software Defined Networking (SDN) [1] leads to scalability challenges owing to the recent per-flow forwarding state blast in networks caused by a growth in the count of nodes along with intelligent, flexible, and policy-driven service requests. One challenge is that WANs can suffer from extended propagation delays due to the controller's requirement to deploy flow rules on each data plane device along the flow's route. Long propagation delays may result in lengthy path setup times in large-scale SDN networks with many geographically dispersed switches. Route setup delay may not affect elephants and/or ordinary flows. However, the Quality of Service (QoS) of delay-aware flows might be considerably degraded [2] from such lengthy path setup times. Another scalability challenge in SDN networks is the message processing load by data plane devices and SDN controllers. SDN switches may struggle to process OpenFlow messages because of their limited processing capability. Because SDN devices can hold a certain amount of flow rules in flow tables, the "match-and-action" concept may affect the granularity with which flows are controlled. A similar problem with resource capacity affects SDN controllers while processing service request setup status messages. They only have a slight processor and storage capacity installed on their computers. The controller message load to process increases as the network expands, resulting in a network computation bottleneck. Finally, SDN switches have limited storage capabilities to maintain flow rules. SDN devices often keep flow tables and rules in TCAM-based memories [3]. However, the TCAM memories are expensive, power-costly, and restricted storage resources. In order to handle SDN architecture's scalability challenges, it is necessary to diminish the number of network incidents handled in the control layer and only process needed network incidents [4].

Increasing the scalability of SDN by lowering the number of flow rules in the switches has recently been deemed a potential strategy, i.e., easing the information change among the controller and data layer nodes [5]. At the origin node, Source Routing (SR) lays out the whole route from source to target. SDN networks' centralized management allows the controller to keep track of the network's global topology. This means that various objectives may be met by using different enhanced SR strategies. Similar to SDN-based destination routing, source switch-based route encapsulation aims to save flow information in the packet header. This means that all that would be required of each intermediary switch along the line is to read the correct egress ports and send the packets. To provide fine-grained traffic management, a limited and finite amount of flow rules may be exploited by all flows without requiring any interaction from the controller during route creation.

A potential solution to these issues mentioned above (i.e., path setup time latency and augmented controller message processing) might be utilizing an SR approach in routing functionalities, i.e., putting the route information into the packet headers [6], [7]. This allows putting a limited number of flow-independent forwarding rules into the network devices, which considerably diminishes the forwarding state and improves the control plane scalability by reducing the workload on controllers to add flow entries into data layer nodes.

This work exploits and builds on the idea proposed in our previous study [8]. This study integrates a *Port IDs and TTL (PIDTTL)* based SR scheme for intra-domain traffic management into our blockchain (BC)-supported QoS-concentrated cross-network routing framework [8] in order to enable QoS-based end-to-end (E2E) routing in SDNs. This study aims to design and implement an E2E routing framework (*SoRBlock*) in SDNs, where inter-domain level routing exploits BC technology and intra-domain level routing leverages the SR scheme. The performance of the *SoRBlock* framework is quantified regarding *Path Setup Time* and the number of *Controller Messages Processed* (by controllers) metrics. *SoRBlock* is also compared against a QoS-aware routing strategy in SDNs, *Hierarchical Routing Approach (HRA)*, presented in our earlier research [9] to evaluate the performance gains. This study is certainly not the first to conceive SR. However, it is the first to effectively couple the SR concept, SDN, and BC to implement and demonstrate the collaboration of combined SDN, BC, and SR schemes on inter-domain routing applicability.

The following outline is the body of this paper: The next section presents studies exploiting SR in SDN. Section III briefly discusses SDN, SR, and BC to render this study self-sufficient. Section IV introduces the details of SR-based intra-domain and BC-enhanced inter-domain routing strategies in SDN in this study. Section V presents the experimental numbers before the final notes in the Conclusion section.

II. RELATED WORK

A packet's complete routing path may be determined solely by the origin using SR. Because of its simple and lower-state routing packets, it has already received substantial attention. SDN research also leverages the SR concept. SDN-based SR was suggested by [6] for scalable service chaining in the data center. The packet header contains the route information. Intermediary data plane nodes' ports are represented as straight integers to eliminate the requirement for additional bytes for route information.

On the other hand, SDN-based data centers concentrate on route installation rather than path discovery. Furthermore, it does not provide QoS. A novel forwarding method for OpenFlow-based systems is discussed by [10], who emphasizes the controller scalability and performance difficulties in SDNs. The equation for the number-encoded SR route is derived in this study. This dramatically minimizes the number of routing states sent to forwarding nodes by pushing the route rule to the ingress node. Using OpenFlow and NOX controller extensions, StEERING [11] attempts to set up a route that travels through certain middle-boxes. Splitting a single flow table into many small tables is the approach to limiting "rule explosion". StEERING utilizes Graph Theory to provide a solution to the route planning conundrum. StEERING's main shortcoming is the absence of QoS support. It uses middle-boxes procedures to find the best route. With SlickFlow [12], the emphasis is on resolving problems at the source through SR. The controller's fault recovery program sends the route information to the entry nodes in special headers. Each part of the journey includes all hops and other routes to the next hop.

In order to virtualize the data center network, SecondNet [13] uses SR through MPLS port switching. SecondNet uses shorter pathways and, as a result, does not encounter some of the limits associated with service chaining that apply to linking pairs of virtual machines. Improved flexibility and performance are the goals of [14]'s source-routed fabric. A source-routed fabric in leaf-spine data center topologies is discussed in the paper, which estimates the throughput advantages of the proposed fabric over existing forwarding methods. For data centers, the architecture described by [7] is comparable to the port switching procedure developed by [15] and the SR techniques shown in [14]. Wireless networks may also benefit from the use of SR. One of the most used SR schemes in MANET and Ad-Hoc networks is Dynamic Source Routing (DSR) [16]. A packet's route data may be included in its header, allowing intermediary nodes to avoid maintaining a routing table in DSR. Finally, Segment Routing [17] is an alternate SR-based method proposed by the IETF to provide simplification, improved traffic engineering, and quick reroute capabilities.

New traffic engineering activities and services demand low-overhead routing and forwarding disruptions over-complicated network topologies. Source routing (SR) reduces network states to provide expressiveness and agility. M-PolKA [18], a topology-agnostic multipath source routing strategy and orchestration framework for reliable communications, examines exceptional characteristics from Residue Number System (RNS) polynomial arithmetic. Current routing protocols do not consider the communication characteristics of different applications. SDN was considered a solution, but it has scalability issues. Source Routing over IPv6 (SRv6), a decentralized source routing protocol, is expected to scale better. The proposed system in [19], Acar, uses SRv6 and adapts routing based on application bandwidth requirements and network link utilization. SDN has scalability issues in WANs due to the separation of control and data planes, which causes increased response time and overhead. The authors in [20] proposed the Source-Path Routing Model (SPRM) framework that addresses scalability, performance, and link failure issues by combining proactive and reactive approaches. They employ multiple pre-calculated paths and dynamic network state information (NSI) to assign the best path for an incoming flow. They also utilize multiple paths as a backup to each other in case of link failures. [21] proposes an SDN-based approach for quickly recovering Time Sensitive Networks (TSN) from failure events using SR and a stateless data plane. The authors employ a TSN failure recovery routing heuristic to minimize link congestion and TSN subgraphs to quickly reschedule flows in problematic areas. The authors in [22] introduce SRCV, a mechanism for verifying control-data plane consistency in P4-based SDN at runtime. SRCV uses active probe traffic with source routing labels, collects matching flow rule information, and compares it with control plane flow rules information through symbolic execution. Table 1 summarizes the related works studying SR and SDN.

Table 1. *Summary of Related Works*

WORK	SUMMARY
[6]	A scalable service chaining at which the NF-path path is encoded into the packet header
[7]	A forwarding method compactly encoding a packet's network route in its address fields
[10]	Focus on SR to solve SDN-based WAN convergence and controller placement issues
[11]	A framework for dynamically routing traffic through any sequence of middleboxes
[12]	An SR-based fast failure recovery by adding alternative path data into the packet header.
[13]	Data center network virtualization architecture using port-switching-based SR
[14]	SR-based clean abstraction and efficient implementation for future network fabrics
[15]	An SR approach for increased scalability and robustness in multi-tenant datacenters
[16]	Dynamic Source Routing (DSR) for Mobile Ad Hoc Networks for IPv4
[17]	Defining SR functions that are required in the IS-IS protocol
[18]	Topology-agnostic multipath source routing strategy and orchestration framework
[19]	A routing by considering application bandwidth requirements and link utilization
[20]	SPRM framework to address scalability, performance, and link failure issues in SDN
[21]	An SDN-based approach for ultra-fast TSN recovery using SR and stateless data plane
[22]	An SR-based control-data plane runtime consistency verification mechanism using P4
SoRBlock	An SR-based E2E traffic management framework in which inter-domain routing uses BC, while intra-domain routing relies on the SR technique in SDNs

SR is seen as a surrogate to the traditional hop-based forwarding strategy in SDN, as seen by the concepts described in this section. Despite this, there are a variety of implementation models to choose from. Despite this, other implementation forms exist. Supplementing these notions, this study demonstrates the applicability of SR strategies in actual SDN environments. However, this study differs from the works mentioned above in different ways. First, QoS information is considered in the path selection process to support QoS provisioning. Second, BC is exploited to reduce further QoS-related signaling overhead messages handling by network controllers to enhance control plane scalability in SDN networks. Third, SR-based intra-domain and BC-enhanced inter-domain routing approaches cooperate to propose an E2E traffic management scheme. Therefore, this study is the first to exploit the confederacy of the SR, BC, and SDN concepts to manage E2E traffic to the best of the author's knowledge.

III. OVERVIEW OF SR, SDN, AND BC

This section briefly explains SR, SDN, and BC to make this paper self-inclusive.

A. SOURCE ROUTING (SR)

It is possible to set up a complete route from a single origin node to a single destination in SR. It requires the insertion of the set of labels into the packet header, which the switch fabric will handle. The process of SR can be facilitated by deploying a network controller in SDN, which will be responsible for the path computation and the population of path insertion entries into the edge switches. Reducing the amount of forwarding state may be accomplished via SR [23]. By reducing the number of states, significant reductions in switch TCAM can be achieved, resulting in more cost-effective switching equipment. In theory, switches could send packets with minimum flow-independent forwarding rules when employing SR because the path is encoded in the packet header. Each packet's route is represented as a list of labels that roughly correspond to the ports of the data plane nodes it must traverse. The switches can free up much TCAM capacity by not keeping forwarding entries to all network destinations at the L2 or L3 levels.

Because network topology may not match the route given by the source node, the usage of SR is restricted in practical networks. For this problem to exist, the source node must be deprived of a global topological perspective to establish a reliable route. As a result of the SDN controller's global topology perspective, it can accomplish globally optimal resource allocation and efficiency, including E2E routes. This means that in the scope of SDN, it is rather probable that a route will be found for SR. SDN uses SR in a variety of ways, including in data center networks [6], WANs [10], fault tolerance/recovery [13], and so on.

B. SOFTWARE-DEFINED NETWORKING (SDN)

The control plane and forwarding planes are intended to be separated by SDN. Network engineers and technicians benefit from this isolation, allowing them to use network resources more effectively and expedite service delivery. SDN also makes it simpler to modify the properties of entire networks via programmability. Network administration is more superficial because it is split from the data plane. SDN design [1] allows network administrators to simply and swiftly control and modify their networks' resources using self-written, flexible, and proprietary-free applications. Furthermore, unlike traditional networking, controllers in SDN possess a global view of the entire network since the network is logically centralized. As a result, they can dynamically improve resource allocation and flow management.

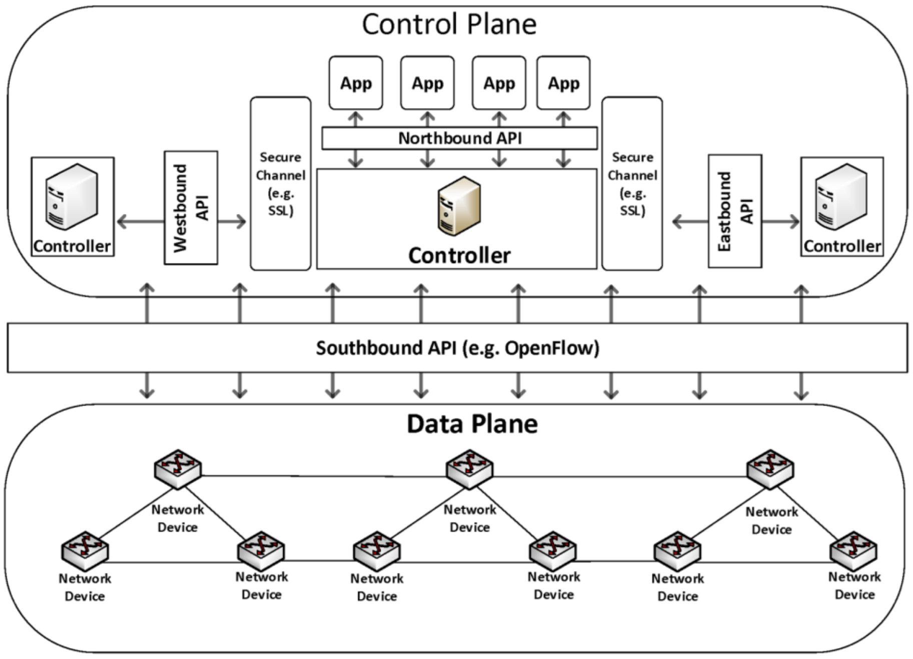


Figure 1. *An outline of SDN with fundamental components.*

Furthermore, the control and forwarding layers of SDN are interconnected through interfaces. These interfaces let devices communicate with others in the network. East-West APIs [24] aim to exchange information between controllers, which may be from the same or different organizations. Northbound APIs make network application and controller(s) interaction possible for network services. On the other hand, southbound APIs, as shown in Figure 1, enable access between the controller(s) and forwarding layer nodes such as routers, physical switches, or virtual switches. OpenFlow [25] is the primary southbound protocol for interaction between the control and data layers.

C. BLOCKCHAIN (BC)

BC technology is a democratic system where all participants working and participating in a network can follow the processes without a centralized authority. In another aspect, the system is a distributed database that records all transactions performed. Thanks to its decentralized structure, all operations performed without a centralized agent are executed with specific protocols and trust mechanisms. The overall BC structure is created by writing each activity, referred to as a transaction, into the blocks and adding it to the chain.

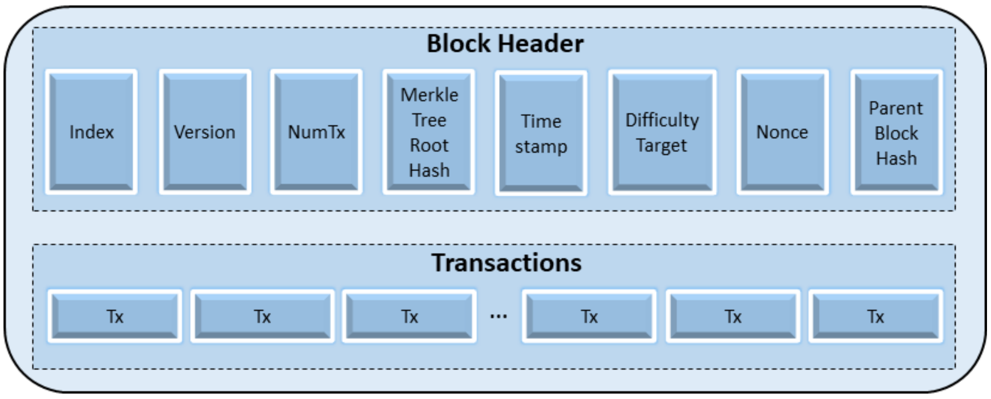


Figure 2*. A visual depiction of a block's data structure in the BC.*

A generic block data format in a BC is shown in Figure 2. A block's structure is primarily composed of: (i) a block header, which contains different data describing the block, and (ii) a block body, which contains the batch and count of transactions included inside the block. The construction of the BC might alter based on the applications and consensus methods that utilize it. Data such as the version or identifier, encrypted transaction values in a Merkle tree, a timestamp, and difficulty variables are included in the block header. The hashed value of the preceding/parent block is also included. The network's users use private and public keys to conduct transactional activities. Users' network access is protected by private keys, which act as a kind of identification.

IV. MOTIVATION

Combining SR, SDN, and BC can give networks a safe and effective routing solution. This method is not just tamper-proof and decentralized; it also gives users more say over how their data goes throughout the network. The optimal and reliable path for data to travel through a network may be determined using SR. SDNs allow the route to be dynamically altered based on current network circumstances, allowing optimal data delivery. BC may be used to make the network more secure and resistant to manipulation by storing and verifying routing information.

Altogether, SR, SDN, and BC can provide a complete answer for decreasing routing path setup times. This is especially relevant in networks that provide time-sensitive activities, such as online stock trading or live video conferencing. Combining these three technologies can be an effective strategy for easing the burden on routing controllers. Maintaining and updating the routing table is the controller's responsibility in conventional routing systems, which can burden the controller's resources and negatively impact performance. Organizations can relieve pressure on the controller by shifting part of the work to SR, SDNs, and BC.

The controller does not have to spend as much time maintaining and updating the flow tables and rules if SR is used to indicate the exact path that data should follow via the network. In order to ensure that data is delivered fast and effectively while minimizing the need for manual intervention and updates from the controller, SDNs allow for the routing path to be dynamically altered based on real-time network circumstances. BC can offer a decentralized and tamper-resistant solution to store and validate routing information to cut down on the time and effort needed for path construction and verification and the need for the controller to maintain centralized flow tables and rules.

Blending these three technologies can be especially helpful in sectors like banking, healthcare, and logistics, where safe and dependable transport is essential. When applied to a healthcare network, for instance, SR can guarantee the safe and timely transfer of patient information to the treating physician. Using BC can increase security by making routing data more difficult to alter and easier to verify. In addition, this method can give more programmability in routing, which opens the door to individualized routing solutions that meet the unique requirements of each business. This can boost efficiency, save money, and enhance safety and dependability.

Overall, SR, SDN, and BC form a powerful routing solution. Organizations may build a fast, secure, and reliable network by combining the benefits of these three technologies, lessening the time spent on path creation and verification and lightening the controller's workload.

V. SoRBlock: AN END-TO-END TRAFFIC MANAGEMENT FRAMEWORK

This section presents the technical details of the proposed E2E routing framework, \emph{SoRBlock}, in two parts. First, a BC-based inter-domain level routing is presented in subsection IV-A.

Subsequently, the intra-domain level routing exploiting the SDN-based SR approach to form E2E connectivity is presented in subsection IV-B.

A. BLOCKCHAIN-BASED INTER-DOMAIN LEVEL ROUTING

This subsection presents an overview of the BC-based inter-domain level routing strategy in line with the framework discussed in [8] and used in this study. Readers can refer to [8] for more details.

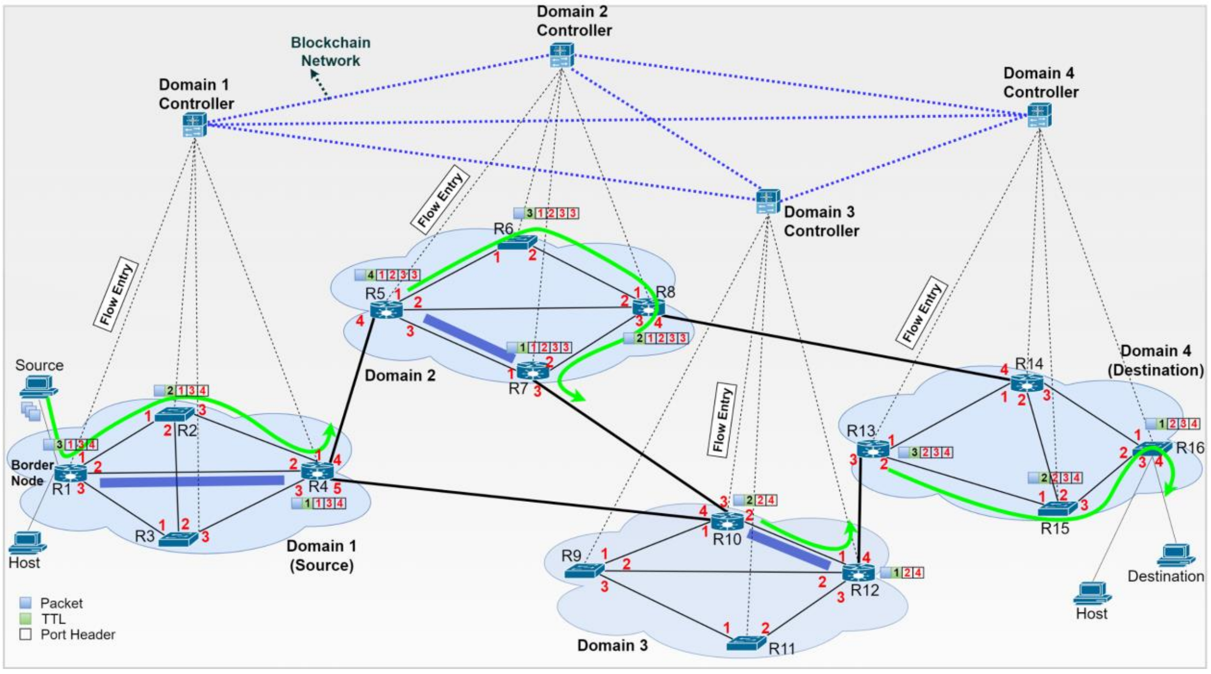


Figure 3. *A representation of the proposed traffic management framework, SoRBlock, with BC-assisted inter-domain routing and SR-based intra-domain routing schemes for 4 SDN domains.*

Figure 3 exhibits the proposed traffic management framework, *SoRBlock*, with BC-assisted inter-domain routing and SR-based intra-domain routing schemes. Figure 3 embodies the *SoRBlock* with 4 SDN domains with devices managed by corresponding controllers. Infrastructure nodes are two types: (i) A border device (rounded devices) connected to a border device in a different domain by an interconnecting link and (ii) a core network node (diagonal devices) without an interconnecting link. Black dashed lines represent the links among controllers and infrastructure nodes. There is also a BC network among controllers. Blue dashed lines show the connections in the BC amongst controllers. The thick blue lines show the pathlets, which compromise the E2E path, over the E2E path computed by the source-domain (Domain 1 in Figure 3) controller in domains. The green lines represent the SR-based intra-domain paths computed by controllers of domains over the E2E path. The red numbers are the port ID numbers of the network nodes.

The controller used in the *SoRBlock* framework has new modules and standard modules such as topology manager, statistics manager, etc. The Blockchain Manager (BM) and its sub-components are the primary BC enabler module. This component is a domain controller performing all BC-related operations. The Validator Agent is capable of proving new blocks from other controllers under BC's block validation criteria. The Hashing Agent is in charge of hashing transactions and blocks before transmitting to the BC. The Transaction/Block Agent is responsible for implementing the transactions/blocks that comprise the BC. The Consensus Protocol Handler conducts the BC consensus mechanism. The Resource Monitoring Manager (RMM) continually tracks network resources, such as bandwidth, latency, and jitter, to detect network modifications. It alerts the BM module of any updates and instructs it to construct the related transaction(s). Additionally, the controller design has novel applications. The Global Routing Agent (GRA) performs the cross-domain routing capabilities when a controller receives an inter-domain service request. GRA uses transactions' ingress and egress fields to determine an E2E route for the arriving service request. The Blockchain Application (BA) is in charge of transmitting and receiving blocks and managing messages related to service requests.

In the framework, a pathlet is represented as: Given $G_i = (V_i, E_i)$, let ${}^*V_i$ denote the set of border devices in the network N_i where ${}^*V_i \subseteq V_i$, a path $P = \langle v_i^1, \ldots, v_i^n \rangle$ is named a pathlet if $\left(\forall\, v_i^j \in P\right) \in V_i$ and $v_i^1, v_i^n \in {}^*V_i$. A pathlet is a path that connects two border node pairs (entry and leaving nodes) in a domain. The nodes at the beginning and end of a pathlet are referred to as *ingress* and *egress* nodes. For example, as shown in Figure 3, paths R10-R12, R10-R9-R12, and R10-R9-R11-R10 (assuming duplex links) are pathlets between border devices R10 and R10 in Domain 3.

Each BC node represents a domain controller and operates its BC replica using the inter-domain level routing mechanism. Within the context of inter-domain routing, BC nodes generate transactions from pathlets and their associated QoS values. As a result, domain controllers calculate separate pathlets for all network border node pairings. A transaction's data structure consists of the following elements: Tx ID, Signature, Domain Number, Pathlet ID, Ingress Node, Egress Node, Max Bandwidth, and Min Delay. When a domain is added to the BC network, it begins producing the first transactions for the pathlets between the edge network nodes. Depending on the BC's consensus procedure, they are subsequently broadcast as blocks to the network or the corresponding BC nodes. Once a network variation affecting QoS, such as a bandwidth adjustment on a link, impacts the state of a pathlet, the controller produces a new transaction indicating the pathlet's variation. Controllers use the most up-to-date transaction for pathlets with the same IDs by checking the transaction IDs while finding an E2E route for service. The logic behind this procedure is that the most up-to-date transaction reflects the recent state of the respective pathlet concerning QoS parameters in the network. As in any BC application, a block has two parts in SoRBlock: block header and block body. The block header includes Block ID, Previous Block Hash, Merkle Root Hash, Primary-ID, and Timestamp data fields. The block body keeps pathlet transactions that domain controllers use to compute inter-domain E2E paths for service requests.

The inter-domain level routing is conducted after particular requested service-related messages are exchanged among domains. When a user/client sends a QoS-based inter-domain service request to a (source) domain controller through the *S_Req* message, the controller begins searching for an E2E *inter-domain level* route using its BC ledger, considering the QoS metrics and priority specified in the message. The *inter-domain level* E2E route is composed of pathlets connecting an edge device of its network (i.e., source-domain) to an edge device of the destination-domain through an overlay network, which is abstracted by using the BC's pathlet transactions. The path consisting of the thick blue pathlets, R1-R4-R5-R7-R10-R12-R13, is an example *inter-domain level* E2E path in Figure 3. *Reject* reply is sent back to the user if the source-domain controller cannot locate an *inter-domain level* E2E route that meets the service request requirements indicated in the *S_Req* message created by the BA module. The user/client is notified of this by the *S_Res* message. Each domain controller with a pathlet on the *inter-domain level* E2E path (domain controllers 2, 3, and 4 in Figure 3) is queried by the origin-domain controller to arrange the specified QoS values after getting back the *P_Req* messages generated by the BA module. *S_Res* message with an *Accept* answer is sent to the user/client by the source-domain controller, which states that the service request may be met and that the network will commence if all relevant domain controllers give *Accept* replies to the source-domain controller. A *Reject* response in *P_Res* messages from any domain controller across an E2E route causes the source-domain controller to search for a new intra-domain level E2E path with the same attributes and criteria.

B. SOURCE ROUTING-BASED INTRA-DOMAIN LEVEL ROUTING

There are three primary strategies identified and exploited for SR [6], [14], [15]. The most basic approach to constructing SR is to use the Label Sequence and Pointer solution. The series of (switch port) labels is encoded in packet headers, and the reference to the following label is encoded in a distinct header field. The source/destination MAC address and IP(v6) address are good candidates for preserving labels (without additional transmission costs). The Label Sequence with TTL method utilizes TTL rather than a separate entry for the pointer to the following label (opposite the previous approach). Finally, the Switch IDs technique assumes unique identifier switch IDs and defines the route as a series of switch IDs. This necessitates a fixed matching table in each network device containing the IDs of surrounding devices. Because a switch can identify its ID in the chain, there is no requirement for a pointer. These implementations reserve their intrinsic advantages and disadvantages.

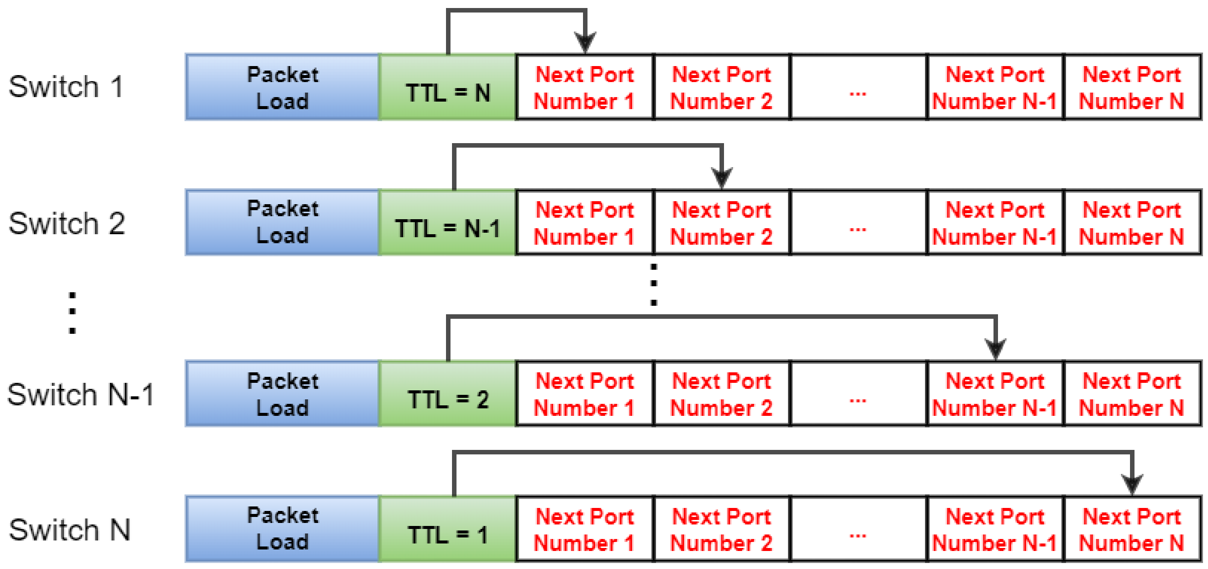


Figure 4. *Packet header ports and TTL values in PIDTTL-based SR implementation.*

This study employs the *PIDTTL* implementation due to efficient header space usage. The source and/or destination MAC addresses are used to encode the corresponding list of data plane device output ports. MAC address fields, in particular, are filled with a list of output ports that each node along the route will use to direct each packet. The TTL field is also utilized as a reference to the following port number in the MAC address. Figure 4 shows the *Next Port Number* (i.e., node output ports) and *TTL* (Time-to-Live) values of packet headers when packets arrive at network nodes in *PIDTTL*-based SR implementation.

- ***Time-To-Live (TTL)*:** It is an eight-bit header showing how many hops the packet has left. The target node must know that it is the final hop to handle the payload in SR. As a result, the *TTL* was created with this purpose in mind, and its value is set at the origin node and then dropped by one with each successive hop.
- ***Next Port Number*:** It is a four-bit field that contains the packet's subsequent output port on a node over the computed intra-domain path. The node reads the corresponding port number using the current value in the *TTL* field of the packet header.

After *Accept* responses in *P_Res* messages, SR-based *intra-domain level* routing is conducted by domain controllers. These controllers add a flow rule in flow tables of only respective ingress border nodes of their domains for the service request packets. The relevant flow entry modifies each packet header to contain port numbers and TTL value that the packet traverse through inside the domain for the SR, as explained in subsection IV-B. In the example of Figure 3, after controller 2 sends a *P_Res* message to the source domain (i.e., domain 1) controller, the domain 2 controller inserts a flow entry in the flow table of R5 for the packets of the requested service to traverse through the domain 2.

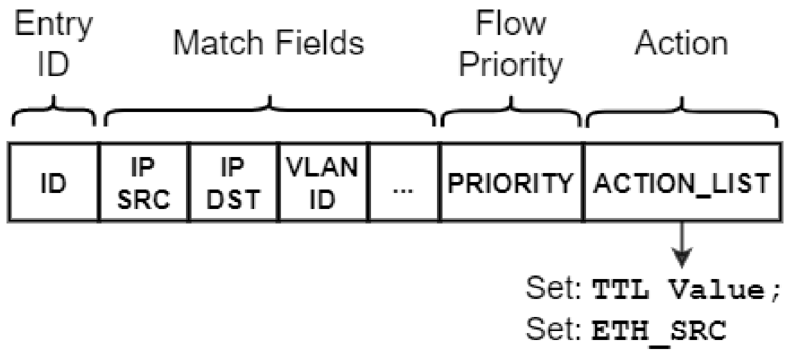


Figure 5. *Flow entry data structure inserted in flow tables of the ingress network nodes of domains over the E2E path.*

Figure 5 represents the data structure of flow entries added in the flow tables of the ingress network nodes of domains over the E2E path. The flow entry added in R5 by domain controller 2 modifies the header of each packet matching the flow entry to include an initial TTL value matching the hop count and port ID sequence as 4 and $\langle 1, 2, 3, 3 \rangle$, respectively, in the case of Figure 3. The TTL value is decremented as usual by each node that the packet traverse through inside the domain points to the subsequent port number encoded in the MAC address. This process is applied in each domain over the E2E inter-domain level routing path computed by the source domain controller. When the packets pass to the next domain over the E2E path, the ingress node of the domain that the packets traverse through modifies the headers of the packets accordingly by means of the corresponding flow entry inserted by the relevant domain controller. For example, the flow entry added in R10 by controller 3 modifies each packet header to include the initial TTL value and port sequence as 2 and $\langle 2, 4 \rangle$, respectively, when the packets come from R7 to R10 in the case of Figure 3.

VI. EXPERIMENTAL RESULTS

The performance of *SoRBlock* is tested by analyzing its *Path Setup Time (PST)* and *Controller Messages Processed (CMP)* parameters to those of the hierarchy-based routing approach, *Hierarchical Routing Approach (HRA)*, presented in our previous research [9].

In the experiments, Mininet simulator and Ryu controller were employed to create and model SDN networking devices and measure the results for the respective parameters listed in Table 2 during the experiments. MATLAB software was used to carry out numerical computations. A custom BC network built in Java has been used with different counts of network elements (i.e., data plane nodes and controllers) and two different transaction sizes (original pathlet transactions number as *SoRBlock_Org* and 500K transactions as *SoRBlock_500K*).

Table 2. *List of parameters and notations used in the equations.*

SYMBOL	**DEFINITION**
${}^k_fS = \{{}^k_fs^j \mid {}^k_fs^j\ is\ a\ switch, {}_fs^j \in {}_fS\}$	List of data plane nodes connected to k-th controller on the E2E path for flow f
${}_fC = \{{}^k_fc \mid {}^k_fc\ is\ a\ controller, {}^kc \in C\}$	List of controllers on the E2E route for flow f
$T(x)^y_{proc}$	Duration to process a message (x) at a network device (y)
$T(x)^{y \to z}_{prop}$	Duration to propagate a message (x) from a network device (y) to a network device (z)
$T(path_L)^y_{comp}$	Duration to find an intra-domain route at controller (y)
$T(path_{E2E_{BC}})^y_{comp}$	Duration to find an E2E route for the service demand

	using BC at controller (y)
$T(path_{E2E})^{BR}_{comp}$	Time to find an E2E route for the service demand at BR
${}_f p$	1st packet of flow f
ρ	*packet_in* message from a switch to a domain controller
θ	*flow_mod* message from a domain controller to a switch
m	Various messages sent among domain controllers and BR
${}_f\overrightarrow{h}$	Source-host
${}_f\overrightarrow{s}$	Source-switch
${}_f\overrightarrow{c}$	Source-domain controller
${}_f\overrightarrow{N}$	Source-domain
${}_f\overleftarrow{N}$	Destination-domain
BR	A (super) controller called "Broker"
B^i	List of edge network nodes in *i*-th domain

Table 3 tabulates the parameters and their corresponding values used in the simulations. |S| and $|C|$represents the total number of nodes and controllers used in the simulations, respectively. Their values vary from 25 to 100 nodes and 5 to 10 controllers depending on the topologies, as shown in the figures. The edge nodes within a topology vary between 2 and 4 nodes.

Table 3. *Parameters and values used in the simulations*

PARAMETER	**VALUE**
$|S|$	25 – 100
$|C|$	5 – 10
$T(x)^{y}_{proc}$	0.1ms – 1ms
$T(x)^{y\to z}_{prop}$	8ms – 45ms
$T(path_L)^{y}_{comp}$	1ms – 3ms
$T(path_{E2E_{BC}})^{y}_{comp}$	6ms – 15ms
$T(path_{E2E})^{BR}_{comp}$	6ms – 15ms
$|B^i|$	2– 4
Link Bandwidth	100 Gbps
Request bandwidth	1 Mbps
RAM	12 GB
CPU	Intel(R) Core(TM) i7-5500U CPU @ 2.40GHz 2.40 GHz
OS	Ubuntu 14.04 in Oracle VirtualBox, Windows 11 Pro (Host)
Runs	20
Connectivity Degree	0.5
BC Transaction Size	Original and 500K

The experiments have considered inter-ISP service requests (i.e., the origin and target nodes of a service request are in different domains) with bandwidth resources (1 Mbps). Sufficient bandwidth (100 Gbps) in links is supplied to prevent service denial because of resource shortage to evaluate the real impact of these three options. Also, the Erdos-Renyi model [26] is used to haphazardly create networks with 0.5 connectivity degrees for infrastructure nodes and domain controllers. Additionally, each experiment was performed twenty times to surpass 95% statistical significance using the same and distinct service demands situations in each run. Lastly, all trials were conducted on an Intel Core i7-5500 machine with 12GB RAM running Ubuntu 14.04 in Oracle VirtualBox.

A. PATH SETUP TIME

The *Path Setup Time (PST)* is the time required to deploy the appropriate flow rule records that enable SR by altering the header information in the flow tables of domain entrance nodes along the E2E path. Thus, several stages contribute to the *PST* measure. It is typically characterized by (i) networking/topology-related latency, such as the RTT between the controller and the infrastructure nodes and the time required for controller-to-controller message/packet transmission, and (ii) node/controller-related latency, such as node and/or controller packet/message process time and controller route calculation time. If either of these latencies is excessive, the associated path setup latency increases, resulting in a lengthier flow rule installation, removal, or modification duration in nodes' flow tables. Subsequently, it may lead to congestion at both the control and data layer levels and a lengthy network recovery time. Thus, *PST* is a crucial parameter for evaluating the efficiency of routing architectures in SDN networks, as it contributes to the overall network scalability. Table 2 tabulates the parameters and notations used in *PST* and *CMP* metrics of *SoRBlock* and *HRA* frameworks.

$$
\begin{aligned}
PST_{SoRBlock} = {} & T(S\_Req)_{proc}^{{}_{f}\vec{h}} + T(S\_Req)_{prop}^{{}_{f}\vec{h} \to {}_{f}\vec{s}} + T(S\_Req)_{proc}^{{}_{f}\vec{s}} + T(S\_Req)_{prop}^{{}_{f}\vec{s} \to {}_{f}\vec{c}} \\
& + T(S\_Req)_{proc}^{{}_{f}\vec{c}} + T(path_{E2E_{BC}})_{comp}^{{}_{f}\vec{c}} \\
& + \max_{\forall {}^{k}_{f}c \,\in\, {}_{f}C} \Bigg\{ T(P\_Req)_{prop}^{{}_{f}\vec{c} \to {}^{k}_{f}c} + T(P\_Req)_{proc}^{{}^{k}_{f}c} + T(P\_Res)_{proc}^{{}^{k}_{f}c} \\
& + T(P\_Res)_{prop}^{{}^{k}_{f}c \to {}_{f}\vec{c}} + \max_{\forall {}^{k}_{f}s^{j} \,\in\, {}^{k}_{f}S} \left\{ T(\theta)_{prop}^{{}^{k}_{f}c \to {}^{k}_{f}s^{j}} \right\} + T(P\_Res)_{proc}^{{}_{f}\vec{c}} \Bigg\} \\
& + T(S\_Res)_{proc}^{{}_{f}\vec{c}} + T(S\_Res)_{prop}^{{}_{f}\vec{c} \to {}_{f}\vec{s}} + T(S\_Res)_{proc}^{{}_{f}\vec{s}} + T(S\_Res)_{prop}^{{}_{f}\vec{s} \to {}_{f}\vec{h}} \\
& + T(S\_Res)_{proc}^{{}_{f}\vec{h}}
\end{aligned}
\tag{1}
$$

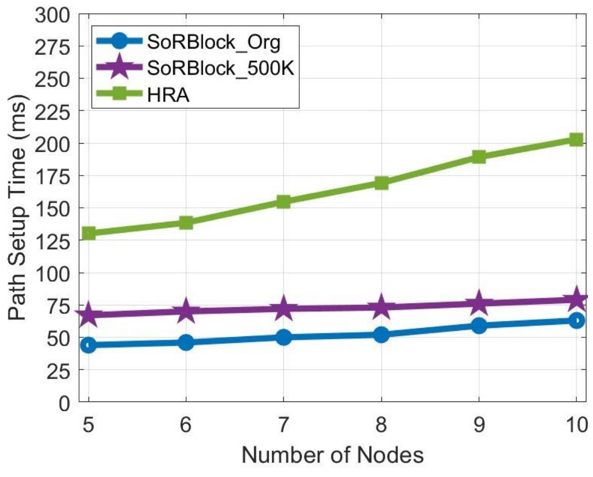
Equation (1) presents the *PST* formula of a flow for a service request in the proposed *SoRBlock* framework. It primarily includes: (i) process and propagation duration of *S_Req* and *S_Res* messages at/among ${}_{f}\vec{h}$, ${}_{f}\vec{s}$, ${}_{f}\vec{c}$ (ii) computation duration for ${}_{f}\vec{h}$ to find an available E2E route for the service demand using BC, and (iii) max of processing and propagation duration of *P_Req*, *P_Res*, and *flow_mod* messages at/among the controllers and data plane nodes on the E2E route.

$$
\begin{aligned}
PST_{HRA} = {} & T({}_{f}p)_{proc}^{{}_{f}\vec{h}} + T({}_{f}p)_{prop}^{{}_{f}\vec{h} \to {}_{f}\vec{s}} + T({}_{f}p)_{proc}^{{}_{f}\vec{s}} + T(\rho)_{proc}^{{}_{f}\vec{s}} + T(\rho)_{prop}^{{}_{f}\vec{s} \to {}_{f}\vec{c}} + T(\rho)_{proc}^{{}_{f}\vec{c}} \\
& + T(m)_{prop}^{{}_{f}\vec{c} \to BR} \max\Bigg\{ \bigg(T(m)_{prop}^{BR \to {}_{f}\vec{c}} + T(path_{L})_{comp}^{{}_{f}\vec{c}} \\
& + T(m)_{prop}^{{}_{f}\vec{c} \to BR} \bigg), \bigg(T(m)_{prop}^{BR \to {}_{f}\overleftarrow{c}} + T(path_{L})_{comp}^{{}_{f}\overleftarrow{c}} + T(m)_{prop}^{{}_{f}\overleftarrow{c} \to BR} \bigg) \Bigg\} \\
& + T(path_{E2E})_{comp}^{BR} \\
& + \max_{\forall {}^{k}_{f}c \,\in\, {}_{f}C} \Bigg\{ T(m)_{prop}^{BR \to {}^{k}_{f}c} + T(path_{L})_{comp}^{{}^{k}_{f}c} + T(m)_{prop}^{{}^{k}_{f}c \to BR} \\
& + \max_{\forall {}^{k}_{f}s^{j} \,\in\, {}^{k}_{f}S} \left\{ T(\theta)_{prop}^{{}^{k}_{f}c \to {}^{k}_{f}s^{j}} \right\} + T(m)_{proc}^{BR} \Bigg\}
\end{aligned}
\tag{2}
$$

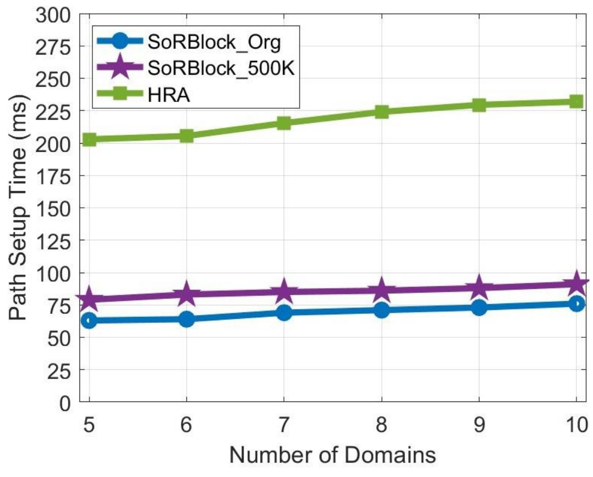
Equation (2) gives the *PST* formula of a flow for a service request in the *HRA* framework. It contains: (i) process and propagation duration for the 1st packet of flow f and respective *packet_in* message at/among ${}_{f}\vec{h}$, ${}_{f}\vec{s}$, ${}_{f}\vec{c}$ (ii) max of (a) propagation duration of intra-domain route demand messages from BR to ${}_{f}\vec{c}$ + intra-domain route calculation duration at ${}_{f}\vec{c}$ and (b) the same operations for ${}_{f}\overleftarrow{c}$,

(iii) calculation duration to find an available E2E route for the service demand at BR, and (iv) max of processing and propagation duration of route demand messages and *flow_mod* messages at/among BR, controllers, and data plane nodes on the E2E route.

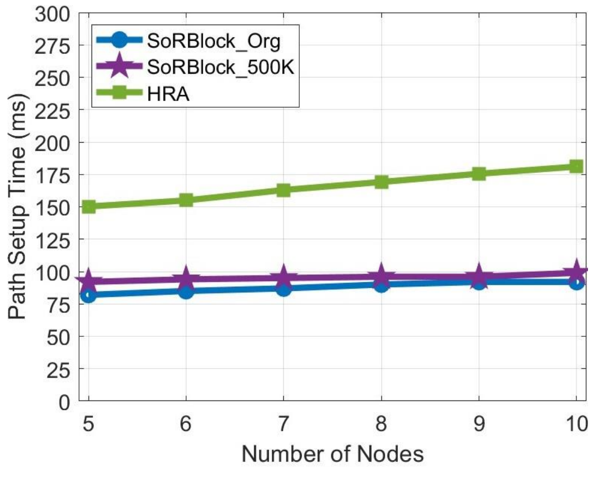
Figure 6a and Figure 6c show the total path setup times in *SoRBlock* with the original (*SoRBlock_Org*) and 500K (*SoRBlock_500K*) transactions sizes and the *HRA* scheme while changing the count of infrastructure nodes in domains. Intra-domain nodes are randomly connected, while inter-domain connections mimic NSFNET topology. While the same (i.e., exact origin and target) service requests were used for all runs in Figure 6a, different service requests (i.e., different origin and target) were used in Figure 6c. As shown in Figure 6a, the presented SR architecture surpasses the *HRA* method regarding the augmentation ratio in total *PST*, owing to the *HRA* approach's increased node-controller and controller-controller latencies in both Figure 6a and Figure 6c. That is because the proposed model utilizes available transactions to identify the E2E route with fewer control messages than the *HRA* method and restricts the deployment of flow rules to the domains' entrance nodes.



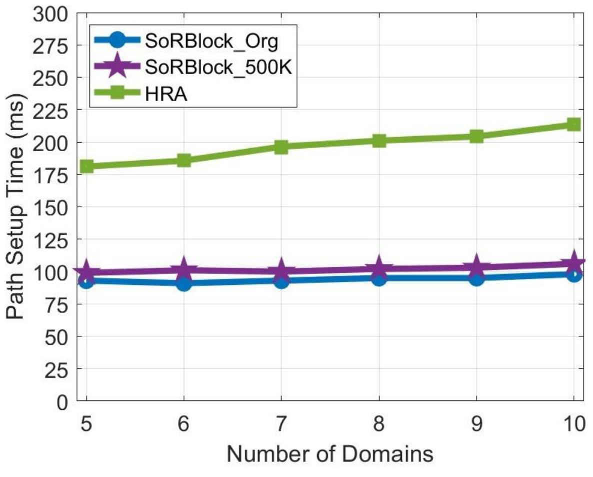
***(a)** Random connectivity in intra-domains - NSFNET in inter-domains - Same requests.*



***(b)** Random connectivity in inter-domains - USNET in intra-domains - Same requests.*



***(c)** Random connectivity in intra-domains - NSFNET in inter-domains - Different requests.*



***(d)** Random connectivity in inter-domains - USNET in intra-domains - Different requests.*

***Figure 6.** PST performances of the SoRBlock and HRA frameworks.*

Additionally, Figure 6b and Figure 6d illustrate the overall path setup duration in the *SoRBlock* using the original and 500K transaction sizes in the BC and *HRA* strategies, respectively, while altering the number of domains. Intra-domain nodes are linked in a manner similar to that of the USNET, while inter-domain connections are formed at random. In Figure 6b, the same (i.e., exact origin and target) service requests for each iteration are used, while in Figure 6d, unique service requests (i.e., distinct origin and target) are employed. The overall path setup time grows as the number of domains in the findings rises for all *SoRBlock_Org* and *SoRBlock_500K* arrangements and *HRA*. This occurs as a consequence of the increased number of domains, which results in a massive load of pathlets and hence transactions arriving from each domain, resulting in increased path computation times and latency. In Figure 6d, SR-based schemes surpass the *HRA* method about twice as much as the *HRA* strategy in *Path Setup Time*, whereas SR-based schemes easily quadruple the *HRA* performance in Figure 6b. On the other hand, Figure 6d further validates the superior performance of the *SoRBlock* scheme while calculating E2E paths from the current transactions. That is because the *SoRBlock*'s routing approach is quicker than the *HRA* scheme.

B. CONTROLLER MESSAGES PROCESSED

As the network increases in size relative to the nodes (e.g., end-points, data plane devices, controllers, etc.), the controllers may need to deal with increased flow demands, and the associated messages exchanged and processed between network nodes and the others in order to establish a QoS-based E2E path across multiple domains. However, due to the restricted computing resources available to the controllers, such as the CPU and RAM, these message exchange and processing tasks might result in the controllers being a bottleneck. The controllers must minimize the number of messages exchanged and processed to establish an E2E route for service request packets. Thus, *Controller Messages Processed (CMP)* is another key indicator for assessing the performance of routing schemes in SDN networks, as it contributes to the overall network's scalability. The *CMP* measure indicates the number of messages processed to establish an E2E route for packets of a service request at/by corresponding decision-maker controllers, i.e., controllers on the E2E route and Broker (if available), in this research.

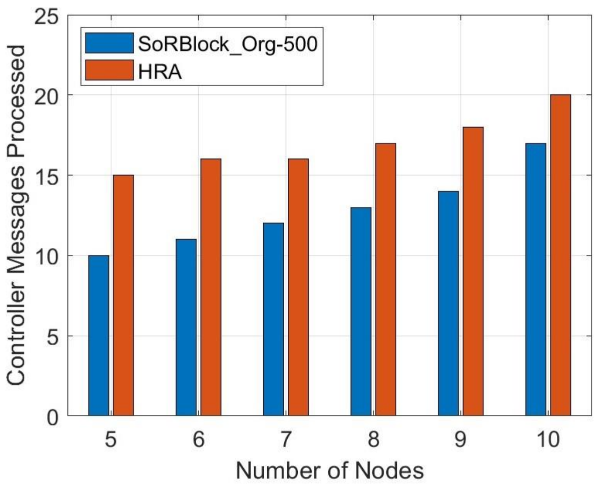
$$CMP_{SoRBlock} = 2x\left|{}_fC\right| + \sum_{\forall {}_f^kc \,\in\, {}_fC} \left|{}_f^kS\right| \tag{3}$$

Equation (3) shows the counts of *CMP* to arrange an E2E route for a flow of service demand in the *SoRBlock* framework. It primarily includes: (i) an *S_Req* message from $\underset{f\ h}{\rightarrow}$ to $\underset{f\ c}{\rightarrow}$, (ii) *P_Req* messages from $\underset{f\ c}{\rightarrow}$ to the other controllers on the E2E route computed for the service demand, (iii) *P_Res* messages from the controllers on the E2E route computed for the service demand to the $\underset{f\ c}{\rightarrow}$ in return for the *P_Req* messages, (iv) an *S_Res* message from $\underset{f\ c}{\rightarrow}$ to $\underset{f\ h}{\rightarrow}$, and (v) *flow_mod* messages from the controllers to the data plane nodes in their domains on the E2E route.

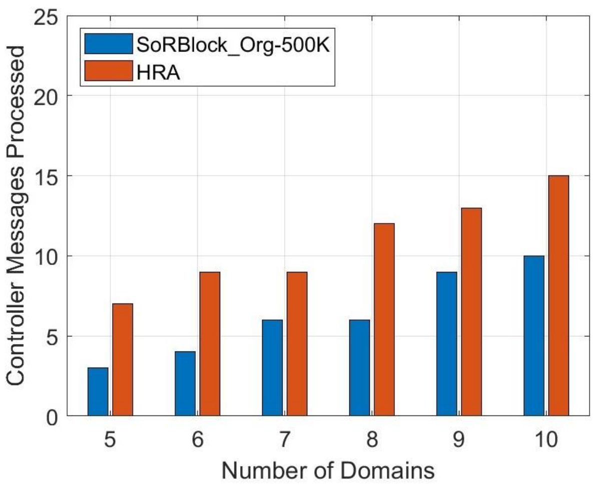
$$CMP_{HRA} = 4 + \left|B^{f\overrightarrow{N}}\right| + \left|B^{f\overleftarrow{N}}\right| + 2x\left|{}_fC\right| + \sum_{\forall {}_f^kc \,\in\, {}_fC} \left|{}_f^kS\right| \tag{4}$$

Likewise, Equation (4) presents the number of *CMP* to install an E2E route for a flow of service demand in the *HRA* framework. It basically contains: (i) a *packet_in* message from $\underset{f\ s}{\rightarrow}$ to $\underset{f\ c}{\rightarrow}$, (ii) an E2E route demand message from $\underset{f\ c}{\rightarrow}$ to the BR, (iii) two computing demand messages from BR to the $\underset{f\ c}{\rightarrow}$ and destination-controller (for intra-domain route announcements among every edge device and source/destination node pairs), (iv) messages including pathlet announcements for every edge device and source/destination node pairs from $\underset{f\ c}{\rightarrow}$and $\underset{f\ c}{\leftarrow}$ to BR, (v) messages (including corresponding edge device pairs in every domain) from the BR to the controllers on the E2E route, (vi) approval messages (for the demanded pathlets in every domain) from the controllers on the E2E route to BR, and (vii) *flow_mod* messages from the controllers on the E2E route to the data plane nodes in their domains on the E2E route.

Figure 7 shows the controller messages processed in the proposed framework with the original (*SoRBlock_Org*) and 500K (*SoRBlock_500K*) transaction sizes and *HRA* scheme while changing the count of infrastructure nodes in each domain. Intra-domain nodes are randomly connected, while inter-domain connections mimic NSFNET topology in Figure 7a. Intra-domain nodes mimic USNET topology, while inter-domain connections are randomly connected under different service requests (i.e., distinct origin and targets) in Figure 7b.



(a) *Random connectivity in intra-domains - NSFNET in inter-domains.*



(b) *Random connectivity in inter-domains - USNET in intra-domains.*

Figure 7. *CMP in the SoRBlock and HRA frameworks.*

Figure 7a and Figure 7b present that the proposed SR-based framework has an improved performance than the *HRA* scheme regarding various numbers of domains and nodes. Particularly, the *HRA* deals with nearly 50% more controller messages than the proposed framework when the number of domains varies in Figure 7b. This is because of the count of border nodes in domains. Interestingly, the number of controller messages processed in the *SoRBlock* scheme approaches that in the *HRA* framework with respect to ratio. At the same time, the count of nodes rises in domains, as illustrated in Figure 7a, because more nodes are utilized on the E2E path. Furthermore, the number of controller messages is higher in Figure 7a due to the random service request model used and the growing count of nodes on the E2E paths as the intra-domain node number increases.

VII. CONCLUSION

This study has presented an E2E traffic management framework (*SoRBlock*) in SDNs, where inter-domain level routing exploits BC technology and intra-domain level routing leverages the SR approach. The experimental results have illustrated that the proposed *SoRBlock* architecture outperforms the traditional hierarchical routing scheme (*HRA*) by reducing path setup time and controller messages processed in SDN networks. The simulation findings demonstrate that by reducing path setup time (PST) and the quantity of controller messages, the proposed SR-based *SoRBlock* architecture outperforms the conventional hierarchical routing strategy, *HRA*, in SDN networks. The proposed *SoRBlock* architecture presents ([45ms - 65ms]) almost three times less total PST than the *HRA* method ([130ms - 200ms]), even though the same (i.e., exact origin and target) service requests were used for all runs. This is because the *HRA* approach has higher node-controller and controller-controller latencies. On the other hand, when different service requests (i.e., different origin and target) were employed, *SoRBlock* displayed PST that was two times lower ([75*ms* - 90*ms*]) than *HRA* ([150*ms* - 175*ms*]). When the number of domains changes, the *HRA* processes approximately 50% more controller messages than the *SoRBlock* ([3 - 10]), although the ratio of CMP in the *SoRBlock* scheme ([10 - 17]) approaches that in the *HRA* framework ([15 - 20]) as the number of nodes increases in domains.

This study is in its infancy stage and needs improvements and research from various aspects because it demonstrates a new potential application for BC and a new research avenue. In future work, it is planned to work on the scalability and security performances of *SoRBlock* by analyzing and evaluating metrics such as transaction/block throughput and delay as well as possible attacks in the BC in addition to improving *PST* and *CMP* metrics performances.

ACKNOWLEDGMENT: This work is supported by the Scientific & Technological Research Council of Türkiye (TUBITAK) under Grant No. 120E448.

VIII. REFERENCES

[1] D. Kreutz, F. M. V. Ramos, P. Esteves Verissimo, C. Esteve Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-Defined Networking: A Comprehensive Survey," *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, Jan. 2015.

[2] M. Karakus and A. Durresi, "Quality of Service (QoS) in Software Defined Networking (SDN): A survey," *Journal of Network and Computer Applications*, vol. 80. pp. 200–218, 2017.

[3] A. Ghiasian, "Impact of TCAM Size on Power Efficiency in a Network of OpenFlow Switches," *IET Networks*, vol. 9, no. 6, pp. 367–371, Nov. 2020.

[4] M. Karakus and A. Durresi, "A Survey: Control Plane Scalability Issues and Approaches in Software-Defined Networking (SDN)," *Computer Networks*, vol. 112, pp. 279-293, 2017.

[5] P. L. Ventre *et al.*, "Segment Routing: A Comprehensive Survey of Research Activities, Standardization Efforts, and Implementation Results," *IEEE Communications Surveys & Tutorials,* vol. 23, no. 1, pp. 182–221, 2021.

[6] A. Abujoda, H. R. Kouchaksaraei, and P. Papadimitriou, "SDN-based Source Routing for Scalable Service Chaining in Datacenters," in *Wired/Wireless Internet Communications: 14th IFIP WG 6.2 International Conference (WWIC 2016),* Thessaloniki, Greece, 2016, pp. 66–77.

[7] A. Hari, T. V Lakshman, and G. Wilfong, "Path Switching: Reduced-State Flow Handling in SDN using Path Information," in *Proceedings of the 11th ACM Conference on Emerging Networking Experiments and Technologies*, Heidelberg, Germany, 2015, pp. 1–7.

[8] M. Karakus and E. Guler, "RoutingChain: A Proof-of-Concept Model for a Blockchain-Enabled QoS-Based Inter-AS Routing in SDN," in *2020 IEEE International Black Sea Conference on Communications and Networking (BlackSeaCom),* Odessa, Ukraine, 2020, pp. 1-6.

[9] M. Karakus and A. Durresi, "A Scalable Inter-AS QoS Routing Architecture in Software Defined Network (SDN)," in *Proceedings - International Conference on Advanced Information Networking and Applications (AINA 2015),* Gwangju, Korea (South), 2015, pp. 148–154.

[10] M. Soliman, B. Nandy, I. Lambadaris, and P. Ashwood-Smith, "Exploring Source Routed Forwarding in SDN-based WANs," in *2014 IEEE International Conference on Communications (ICC)*, Sydney, Australia, 2014, pp. 3070–3075.

[11] Y. Zhang *et al.*, “StEERING: A Software-Defined Networking for Inline Service Chaining,” in *2013 21st IEEE International Conference on Network Protocols (ICNP)*, Goettingen, Germany, 2013, pp. 1–10.

[12] R. M. Ramos, M. Martinello, and C. Esteve Rothenberg, “SlickFlow: Resilient Source Routing in Data Center Networks Unlocked by OpenFlow,” in *38th Annual IEEE Conference on Local Computer Networks*, Sydney, NSW, Australia, 2013, pp. 606–613.

[13] C. Guo *et al.*, “Secondnet: A Data Center Network Virtualization Architecture with Bandwidth Guarantees,” in *Proceedings of the 6th International Conference*, Philadelphia, Pennsylvania, USA, 2010, pp. 1–12.

[14] S. A. Jyothi, M. Dong, and P. B. Godfrey, “Towards a Flexible Data Center Fabric with Source Routing,” in *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research* (SOSR ’15). Santa Clara, CA, USA, 2015, pp. 1-8.

[15] K. Papadopoulos and P. Papadimitriou, “Leveraging on Source Routing for Scalability and Robustness in Datacenters,” in *2019 IEEE 2nd 5G World Forum (5GWF)*, Dresden, Germany, 2019, pp. 148–153.

[16] *The Dynamic Source Routing Protocol (DSR) for Mobile Ad Hoc Networks for IPv4*, IETF RFC 4728, 2007.

[17] *Segment Routing with IS-IS Routing Protocol*, Draft-previdi-filsfils-isis-segment-routing-02, IETF (Work in Progress), 2013.

[18] R. S. Guimarães *et al.*, “M-PolKA: Multipath Polynomial Key-Based Source Routing for Reliable Communications,” *IEEE Transactions on Network and Service Management*, vol. 19, no. 3, pp. 2639–2651, 2022.

[19] T. Sugiura, K. Takahashi, K. Ichikawa, and H. Iida, “Acar: An Application-Aware Network Routing System using SRv6,” in *2022 IEEE 19th Annual Consumer Communications & Networking Conference (CCNC)*, Las Vegas, NV, USA, 2022, pp. 751–752.

[20] S. Komajwar and T. Korkmaz, “SPRM: Source Path Routing Model and Link Failure Handling in Software-Defined Networks,” *IEEE Transactions on Network and Service Management*, vol. 18, no. 3, pp. 2873–2887, 2021.

[21] G. N. Kumar, K. Katsalis, P. Papadimitriou, P. Pop, and G. Carle, “Failure Handling for Time-Sensitive Networks using SDN and Source Routing,” in *2021 IEEE 7th International Conference on Network Softwarization (NetSoft)*, Tokyo, Japan, 2021, pp. 226–234.

[22] J. Xia, P. Cui, Z. Li, and J. Lan, “SRCV: A Source Routing based Consistency Verification Mechanism in SDN,” in *2021 3rd International Conference on Advances in Computer Technology, Information Science and Communication (CTISC)*, Shanghai, China, 2021, pp. 77–81.

[23] Q. Dong, J. Li, Y. Ma, and S. Han, “A Path Allocation Method Based on Source Routing in SDN Traffic Engineering,” in *2019 IEEE International Conference on Smart Cloud (SmartCloud)*, Tokyo, Japan, 2019, pp. 163–168.

[24] P. Lin *et al.*, “A West-East Bridge based SDN Inter-Domain Testbed,” *Communications Magazine, IEEE*, vol. 53, no. 2, pp. 190–197, Feb. 2015.

[25] N. McKeown *et al.*, “OpenFlow: Enabling Innovation in Campus Networks,” *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, Mar. 2008.

[26] A. Erdős P. and Rényi, “On the Strength of Connectedness of a Random Graph,” *Acta Mathematica Academiae Scientiarum Hungarica*, vol. 12, no. 1, pp. 261–267, 1964.