QoSChain: Provisioning Inter-AS QoS in Software-Defined Networks with Blockchain

Murat Karakus* , Evrim Guler and Suleyman Uludag

***Abstract*—Data flows of various applications, such as video conferencing, telesurgery, online-gaming, etc., require particular treatment regarding Quality of Service (QoS) because of their sensitivity to such parameters as high bandwidth, low delay, and so on. However, provisioning of QoS-based routing services across Autonomous Systems (ASes) poses gruelling challenges due to the distributed nature and business confidentially hesitations. This study presents a novel blockchain-aided QoS-enabled inter-AS routing framework, *QoSChain (QC)*, by blending merits of Software-Defined Networking (SDN) and Blockchain technologies. *QC* framework introduces a coordination framework with mutually distrusting participants that eliminates centralized mediators in provisioning QoS among SDN-enabled ASes. Also, *QC* reduces the time to set up an end-to-end (E2E) path and the number of messages exchanged and processed by SDN controllers for a QoS flow. Furthermore, it alleviates the concerns of network operators regarding potentially sensitive information disclosure resulting from the required information sharing for QoS-based cross-AS paths by masking details of any confidential information. Experimental results verify the feasibility of the proposed framework on E2E QoS-based routing among SDN networks. This work aims at being a useful primer for researchers in providing insights on how blockchain can be deployed in QoS-enabled routing.**

***Index Terms*—Software-Defined Networking, Blockchain, SDN, Routing, QoS, Inter-domain routing, Inter Autonomous Systems.**

I. Introduction

THE meteoric rise in global Internet traffic is fueled by diverse applications, such as video-conferencing, Video-on-Demand (VoD), Voice-over-IP (VoIP), online gaming, etc. with different rigid requirements, such as high bandwidth, low delay, and so on. Providing these operational thresholds is typically referred to as *Quality-of-Service (QoS)* as an umbrella term. The primary goal of QoS is to provide prioritization regarding certain characteristics including, but not limited to, bandwidth, delay, jitter, and so on for network traffic.

While intra-AS QoS routing problem is important, supporting QoS at inter-AS level is arguably more crucial yet more daunting for network administrators owing to the volume and velocity of the Internet traffic spanning over multiple ASes [1] under strict business confidentiality constraints. The provisioning of QoS-enabled inter-AS routing today primarily suffers from the following problems:

- **Multiple AS Hop Issue.** Networks can guarantee QoS-enabled routing only through their first-hop neighbor domains through Service Level Agreements (SLAs) and the direct connections.
- **Information Sharing Issues.** An agile and scalable communication framework for inter-AS routing needs information sharing among the ASes at some level. However, sharing such a detailed information imposes certain issues as well: *(1) Reluctance for Information Sharing.* Network operators are hesitant to share details (bandwidth, delay, etc.) about their networks with others to preserve the confidentiality of sensitive information. *(2) QoS Signaling Overhead.* Inter-AS communication for QoS-based routing inflicts a large message exchange overhead considering flow request frequency for both network-originated and transit flows.

As the aforementioned problems have pointed out, uniform and consistent implementation of E2E QoS requirements may not be assured across domains. However, inter-AS communication is mandatory for a successful E2E QoS provisioning. Therefore, an agile, reliable, secure, and adaptive coordination mechanism is necessary to mitigate the above-mentioned problems of QoS-based inter-AS routing.

A. Software Defined Networking (SDN) and Blockchain for Inter-AS QoS Routing

This subsection discusses the motivation for Software Defined Networking (SDN) and Blockchain (BC) technologies to provide a feasible solution to the inter-AS QoS-based routing.

SDN [2] is an emerging architecture in which the control and data planes are decoupled, network intelligence and state are logically centralized, and the underlying network infrastructure is abstracted from the applications. With SDN architecture, per-flow routing becomes viable through more scalable, simpler and less time-consuming mechanisms compared to traditional architectures [3]. OpenFlow enables network operators to use various routing algorithms (rather than the typical shortest path) within the controller to generate forwarding tables that govern different isolated flows in the data plane. Also, dynamic routing of flows becomes feasible by controllers due to the decoupling of control and forwarding functions of devices. These abilities, both per-flow and dynamic routing, allow network administrators to come up with more QoS-enabled routing mechanisms for their networks. Moreover, SDN allows network managers to monitor network dynamics and collect up-to-date global network state statistics through counters at very low levels such as per-flow rule, per-packet, per-port, per-table, per-queue, and per-meter bases.

*Corresponding author

M. Karakus is with the Department of Computer Technologies, Bayburt University, Bayburt 69000, Turkey, (e-mail: muratkarakus@bayburt.edu.tr).

E. Guler is with the Department of Computer Engineering, Bartin University, Bartin 74110, Turkey (e-mail: evrimguler@bartin.edu.tr).

S. Uludag is with the Department of Computer Science, Engin, & Phy., University of Michigan - Flint, Flint, Michigan, USA (email: uludag@umich.edu).

BC [4] can offer effective capabilities to enhance QoS-based routing experiences of network operators at inter-AS level. BC brings a distributed environment to manage transactions through consensus protocols. Thanks to the distributed environments and the decentralization feature that BC provides, it can help ASes overcome their hesitations with regards to security and privacy resulting from the necessity to share proprietary information with third-parties in centralized structures such as Broker concept and SDX project in SDN networks [3], and provide QoS for inter-AS routing in a distributed manner without any dependency on an external entity. Also, with BC's transparency, all participants in the BC are able to obtain, confirm, and monitor transaction activities over the network. Such functionality can help BC-enabled frameworks in collaborative network environments so that service and resource providers can track down and screen transactions to read QoS-related data while setting up an E2E path for a service flow. Furthermore, BC's tamper-proof feature may also be beneficial to ensure the integrity of QoS-related data and reconcile disagreements among ASes in case of conflicts across large-scale distrusting environments by utilizing unalterable transaction ledgers.

B. Contributions

This study introduces a novel BC-employed QoS-enabled inter-AS routing framework, *QoSChain (QC)*, by integrating the BC technology in SDN networks. The main contributions of this work can be stated in five-fold as follows: First, this study introduces a coordination framework that brings together parties with conflicts of interest regarding SLAs, security and privacy risks, and so on for inter-AS QoS-based routing. Second, this study eliminates the need for centralized third-party entities in QoS-supported inter-AS routing models by exploiting BC's decentralized feature. Third, this framework requires less QoS signaling messages to setup a path due to the nature of the proposed architecture as explained later. Furthermore, the framework helps mitigate the privacy/security concerns of ASes when cooperating for inter-AS routing by masking details of pathlets from other networks. Finally, *QC* framework also presents a new use case, thereby, potential research directions for researchers from both academia and industry in which the BC can be exploited to provide QoS for inter-AS routing in SDN networks.

To the best of our knowledge, this is the first comprehensive study exploiting the merits of BC to establish a novel coordination framework for QoS-enabled inter-AS routing in SDN networks. This work aims at being a useful primer for researchers to provide insights on how BC can be exploited in QoS-oriented routing applications in networking.

This paper's organization is as follows: In Section II, SDN and BC are lightly explained to make this paper self-inclusive. Section III presents routing/QoS-related similar studies in SDN and BC. A network model to apply the proposed routing framework is presented in Section IV. Section V introduces the design of *QC* framework along with the basic terms, processes, and functionalities. Section VI provides the experiment results before the concluding remarks in Section VIII.

II. Overview Of SDN and Blockchain

As shown in Fig. 1, Open Networking Foundation (ONF) vertically splits SDN architecture into three main planes [5].
Data Plane. The bottom-most is the data plane, consisting of network devices such as routers, physical/virtual switches, access points etc. These devices are accessible and managed through Controller-Data Plane Interfaces (C-DPIs), such as OpenFlow protocol [6], by SDN controller(s).

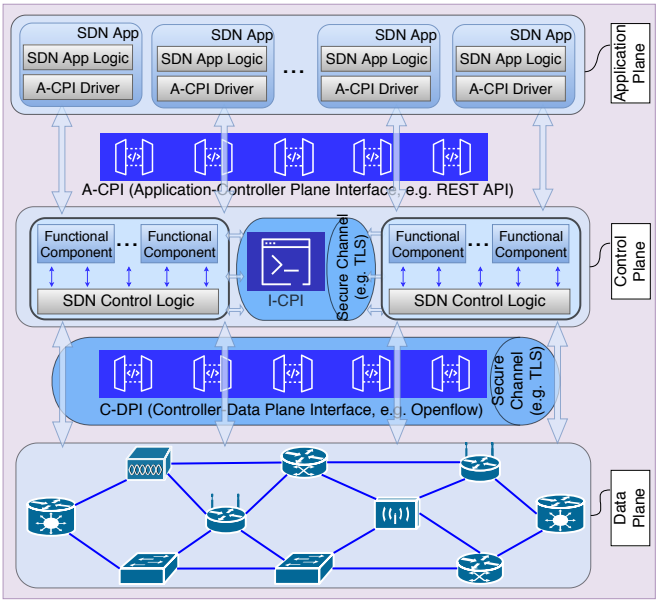


Fig. 1: An overview of an SDN architecture with its main planes: Data plane, control plane, and application plane.

Control Plane. An SDN control plane comprises a set of software-based SDN controller(s) to provide control functionality in order to supervise the network forwarding behavior through C-DPI. It has interfaces to enable communication among controllers in a control plane (Intermediate-Controller Plane Interface, i.e. I-CPI [7], optionally secured using the TLS), between controllers and network devices (C-DPI), and also between controllers and applications (Application-Controller Plane Interface, i.e., A-CPI).
Application Plane. An SDN application plane consists of one or more end-user applications (security, visualization etc.) that interact with controller(s) to utilize an abstract view of the network for their internal decision making process. These applications communicate with controller(s) via an open A-CPI (e.g., REST API). An SDN application comprises an SDN App Logic and A-CPI Driver.

BC provides a decentralized structure to maintain a distributed database by participants without a pre-established or assumed trust relationship and with no centralized trusted third party. Each process, called a transaction, is written into the blocks and added to the chain in order to create the general BC structure. Fig. 2 shows a general data structure of a block in a BC. The structure of a block mainly consists of a block header including various data to define the block and a block body containing the list and number of transactions within the block. The structure may change depending on the BC

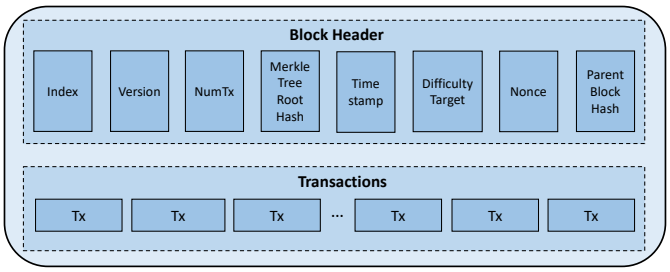


Fig. 2: A representation of block data structure in BC.

use cases and consensus protocols. The block header contains various fields, such as the version or identification number, the encrypted value of the transactions in the Merkle tree structure, a timestamp, a difficulty variable, a varying variable in each computation, and the hashed value of the previous/parent block [4]. Users on the network have a private key and a public key to perform transactions. Authentication of users' access within the network is provided by private keys. Each user checks the transactions published on the BC and does not accept blocks that contain invalid transactions. The hashed value of the previous block is also checked. In this way, all blocks created up to the first block, called Genesis block, are verified. None of the participants in the system can modify previous blocks by manipulating the data in their network. This provides the BC a tamper-proof structure.

III. Related Work

There are many studies introducing applications of BC technology in the literature. BC is exploited in various use cases such as finance, IoT, edge computing, agriculture, supply chains, energy markets, and healthcare systems.

There are research efforts considering BC in their routing or QoS frameworks in the literature [8]–[13]. However, these studies do not propose any QoS-based routing framework for SDN ASes. The work presented in [14] is close to our proposal in a way that it provides latency aware routing among SDN-based ASes. The proposed architecture periodically measures bilateral Round-Trip-Times (RTT) between each pair of connected ASes and stores them in a distributed decentralized BC network where each AS holds its share. However, *QC* differs from their work in the way that we consider bandwidth as QoS parameter in addition to delay parameter. Also, while they consider only delay among ASes at coarse AS-level, *QC* exploits delay among edge nodes inside ASes in a finer granularity. This yields more accuracy regarding QoS constraints of a service request while establishing an E2E path.

Routing as a use case of BC technology is not well-studied while there is no BC-enabled multi-QoS-based inter-AS routing in SDN space in the literature yet. In this sense, to the best of our knowledge, this work, the journal version of our preliminary conference paper [15], is the first attempt in the literature by presenting a comprehensive framework serving inter-AS traffic in SDN networks using a novel BC technology to ensure a distributed coordination mechanism with reduced computational and communications overhead.

IV. System Model

A. Inter-AS QoS-based Path Problem (IA-QbP)

Given a set of ASes with $N_i(V_i, E_i)$ network topologies as depicted in Fig. 3a, let V_i, E_i, and ${}^*V_i$ denote the sets of all nodes, links, and border nodes, respectively, in network N_i where ${}^*V_i \subseteq V_i$ $(1 \leq i \leq n)$, let $N(V, E)$ denote an overlay network topology abstracted from the ASes N_i, as represented in Fig. 3b, where $V = \bigcup_{i=1}^{n} {}^*V_i$ and E are the set of all border nodes of the ASes and (logical) links among the those border nodes, respectively. Assume the number of QoS constraints (e.g., delay, bandwidth, etc.) is denoted by m and each link has an m-dimensional link weight vector, consisting of m non-negative QoS weights $(w_i(u, v), i = 1, ..., m$ where $u, v \in V$, $(u, v) \in E)$ as components. An inter-AS QoS-based path problem (IA-QbP) can be expressed as a (multi) constrained path problem as follows:

Definition 1: **Inter-AS QoS-based Path Problem (IA-QbP).** Consider a network $N(V, E)$ abstracted from networks $N_i(V_i, E_i)$. Each link $(u, v) \in E$ is specified by a link weight vector of m additive QoS weights $w_i(u, v) \geq 0$, $i = 1, ..., m$. Given m constraints L_i, $i = 1, ..., m$, the problem is to find a path P from a source node s to a destination node d (where $s \in {}^*V_i$, $d \in {}^*V_j$ and ${}^*V_i \neq {}^*V_j$) such that $w_i(P) \stackrel{def}{=} \sum_{(u,v)\in P} w_i(u, v) \leq L_i$ for $i = 1, ..., m$.
A path satisfying all m constraints is referred to as a (QoS-based) feasible path. There may be multiple paths in the graph $N(V, E)$ satisfying the constraints. According to Definition 1, any of these paths is a solution to the IA-QbP problem.

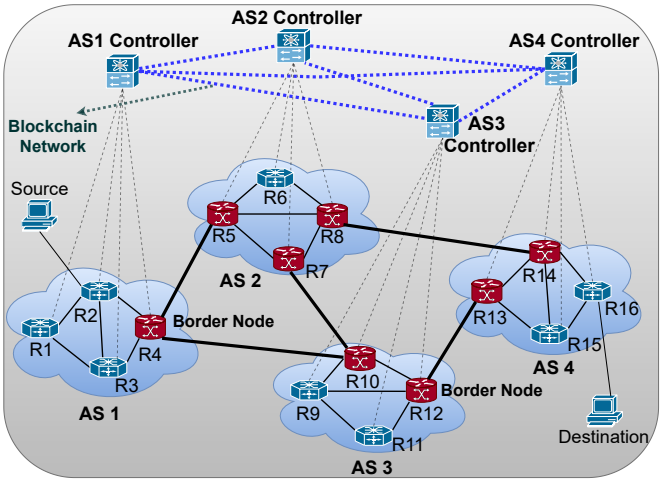
B. Autonomous Systems (ASes)

An AS links geographically dispersed communication devices and networks whose IP prefixes are assigned to an Internet Service Provider (ISP). Each AS is given a unique Autonomous System Number (ASN) by the Internet Assigned Numbers Authority (IANA).

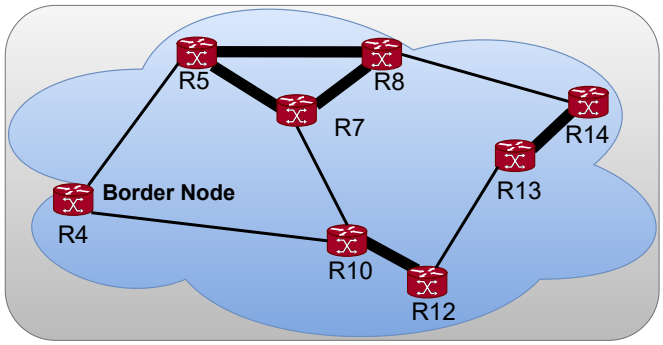
In *QC* framework, there is a *private* BC network among the AS controllers (i.e., BC nodes) to maintain transactions and blocks reflecting network states. ASes do not necessarily (physically) connect to all other networks. Fig. 3a exemplifies a network model consisting of 4 SDN ASes controlled by controllers. Network devices are either a border node (i.e., brown cylindrical objects) connecting to another border node in a different AS through an inter-AS link or a core network device (i.e., blue hexagonal objects) without an inter-AS link. Black dashed lines show the control paths (links) between controllers and network devices. Blue dashed lines represent the logical links among controllers in the BC network.

C. Blockchain-Adapted SDN Controller

AS controllers represent the nodes participating in the BC. We have extended our SDN controller with new controller modules and network applications on the top, adapted for BC capabilities in the *QC* framework. Controller modules implement networking functions that are of common use to a majority of applications such as discovering and exposing network states events (topology, devices, flows, etc.), enabling



(a) A representative network model consisting of 4 SDN ASes.



(b) An overlay topology abstracted from the ASes in Fig. 3a.

Fig. 3: An illustration of SDN AS networks model and its abstracted overlay network.

controller communication with network devices, collecting network resources-related statistics and so on.

As Fig. 4 depicts the modules of our SDN controller with new and existing modules, whose relationships are summarized as follows: The main BC-enabler module is the **Blockchain Manager (BM)** and its sub-components. This module conducts all BC-related functions in an AS controller. The *Validator Agent* is responsible for validating incoming blocks from the other controllers based on the block validation rules in the context of BC. The *Hashing Agent* is responsible for hashing transactions and blocks to be sent to the BC network. While the *Transaction Agent* creates transactions, the *Block Agent* is responsible for block formation in the controller depending on the BC use case and consensus protocol. The *Consensus Protocol Handler* implements the consensus algorithm (`Clique` algorithm explained below) to achieve agreement on agreed transactions and blocks. The **Resource Monitoring Manager (RMM)** continuously monitors network resources such as bandwidth, delay, jitter, etc. In case of any changes, it notifies the *BM* module (i.e., *Transaction Agent*) to create the corresponding transaction(s).

The controller architecture has also new applications. Regarding routing application, the **Global Routing Agent (GRA)** implements the inter-AS routing functionality when

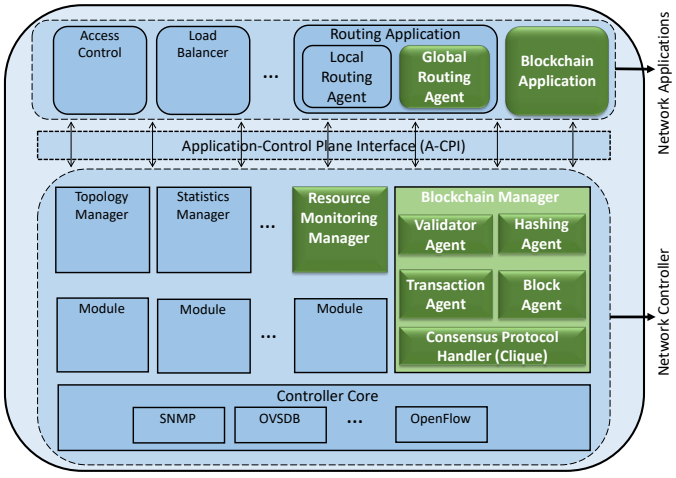


Fig. 4: Overview of an BC-adapted SDN controller with new controller modules and network applications shown in blocks with green color in white text.

an inter-AS service request comes to the controller. This agent takes the ingress and egress fields of new transactions in the BC ledger as input parameters to find an E2E path for the incoming service request and invokes **Blockchain Application (BA)**. **BA** is also responsible for sending/receiving blocks to/from the BC network and handling service/pathlet requests in coordination with the *BM*.

D. Pathlet

In this study, we define a pathlet as a distinct path between network border node pairs (entering/exiting nodes) in an AS. The endpoints of a pathlet are called *ingress* and *egress* nodes.

Definition 2: **Pathlet.** Given an AS network, N_i, with a topology represented as $N_i(V_i, E_i)$, a path $P =< v_i^1, ..., v_i^t >$ is called a pathlet if $(\forall v_i^j \text{ in } P) \in V_i$ and $v_i^1, v_i^t \in {}^*V_i$, and an edge e_i^l from v_i^j to v_i^{j+1}, $(\forall v_i^j, v_i^{j+1} \in P, \forall e_i^l \in E_i)$.

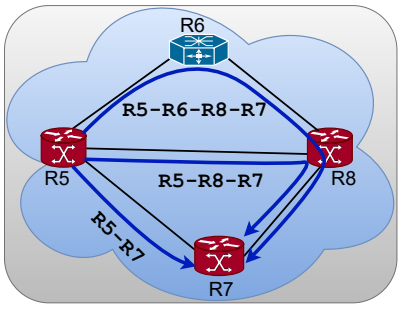


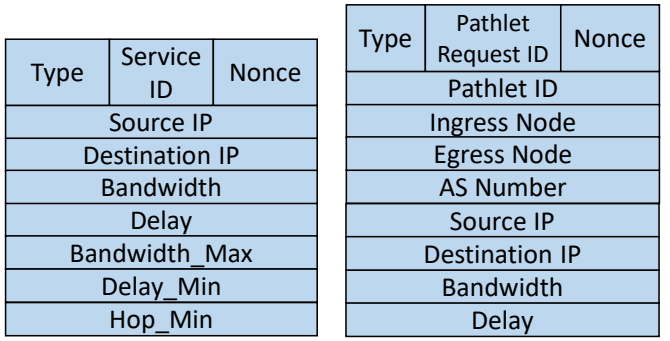
Fig. 5: Pathlets between network devices `R5` and `R7` in `AS2` network.

For example, as shown in Fig. 5, paths, shown in blue lines with arrows, `R5-R7`, `R5-R8-R7`, and `R5-R6-R8-R7` (assuming links are bidirectional) are pathlets between border nodes `R5` and `R7` in `AS2` according to Definition 2. Distinct pathlets for all border node pairs are computed by AS controllers for their networks. As explained later, transactions are created from pathlets along with their QoS values by BC nodes.

E. Service Request

Routing is one of the functions implemented in networks to provide intra- and inter-domain connectivity. In this regard, in

QC framework, a **service request** refers to the provisioning of *connectivity* with certain QoS parameters, e.g., bandwidth and delay, between users (hosts) in the same or different networks.



(a) *S_Req* data structure. (b) *P_Req* data structure.

(c) *P_Res* data structure. (d) *S_Res* data structure.

Fig. 6: Message data structures in *QC* framework.

Fig. 6 exhibits the data structures of *Service Request* (*S_Req*), *Pathlet Request* (*P_Req*), *Pathlet Response* (*P_Res*), and *Service Response* (*S_Res*) messages in *QC* framework.

Fig. 6a shows the data fields of an *S_Req* message:

- `Type`: This field indicates whether the message is an *S_Req*, a *P_Req*, a *P_Res* or an *S_Res* message.
- `Service ID`: A persistent service identifier.
- `Nonce`: Random number appended to the `Service ID` to logically form a unique *Service Request ID*.
- `Source IP` and `Destination IP`: IP addresses of the source and destination hosts for the service.
- `Bandwidth` and `Delay`: The requested bandwidth and delay values over the E2E path for the service.
- `Bandwidth_Max`, `Delay_Min`, and `Hop_Min`: These parameters indicate the path preference order in case of multiple feasible paths, respectively.

The *S_Req* message is created at a user computer by a respective application and sent to the source-AS controller.

Fig. 6b shows the data fields of a *P_Req* message:

- `Pathlet Request ID`: A pathlet request identifier.
- `Nonce`: Random number appended to the `Pathlet Request ID` to form a unique *Pathlet Request ID*.
- `Pathlet ID`: The unique ID of the pathlet requested from an AS controller for the service.
- `Ingress Node` and `Egress Node`: The start and end nodes of the pathlet requested from an AS.
- `AS Number`: Unique Autonomous System number of the AS that the pathlet is requested from.
- `Source IP` and `Destination IP`: IP addresses of the source and destination hosts for the service.
- `Bandwidth` and `Delay`: The requested bandwidth and delay values of the pathlet for the service.

The *P_Req* messages are created at the source-AS controller by the *BA* module and sent to the AS controllers over the final E2E path selected for the service.

Fig. 6c shows the data fields of a *P_Res* message:

- `Pathlet Response ID`: A pathlet response identifier.
- `Nonce`: Random number appended to the `Pathlet Response ID` to form a unique *Pathlet Response ID*.
- `Pathlet_Request ID`: Combination of the `Pathlet Request ID` and `Nonce` fields of the *P_Req* message.
- `Pathlet ID`: The unique ID of pathlet requested from the AS controller for the service.
- `AS Number`: Unique Autonomous System Number of the AS that the pathlet is requested from.
- `Response`: The decision (ACCEPT or REJECT) of the AS controller on providing the requested pathlet.

The *P_Res* messages are created by the *BA* module at all AS controllers over the final E2E path selected for the service and sent back to the source-AS controller.

Fig. 6d shows the data fields of an *S_Res* message:

- `Service Response ID`: A service response identifier.
- `Nonce`: Random number appended to the `Service Response ID` to form a unique *Service Response ID*.
- `Service_Request ID`: The combination of the `Service ID` and `Nonce` fields of the *S_Req* message.
- `Response`: The decision (ACCEPT or REJECT) by source-AS controller on providing the requested service after evaluating the incoming *P_Res* messages.

The *S_Res* message is created by the *BA* module at the source-AS controller for the service and sent to the user.

V. BC-ENABLED QOS-BASED INTER-AS ROUTING FRAMEWORK: *QoSChain*

In this section, we describe and present the design of *QC* framework along with the basic terms, processes, and functionalities used to perform BC and networking operations in the proposed framework throughout the paper. Also, the underlying design angles of the proposed framework in Subsection V-B, the consensus protocol used in the framework in Subsection V-C, and workflow of the framework in Subsection V-D are presented, respectively.

A. Blockchain Model

We present the BC-related operations and entities to explain the BC model utilized by AS controllers in this subsection.

Blockchain Nodes: In *QC* framework, each BC node corresponds to an AS controller and runs its own BC instance. Therefore, we will use a BC node and AS controller interchangeably. A BC node has an IP address for reachability and a pair of public and private keys for cryptographic operations.

Peering: AS controllers exchange their public keys to form a peering relationships. They bind the public keys with BC node identifiers, which are Autonomous System Number (ASN), in order to distinguish the BC nodes. BC nodes broadcast their public keys along with their digitally signed networking

information, such as IP, ASN, list of border nodes, etc., and request the same data from other nodes.

Transaction: A transaction is a single operation of data transfer on a BC network or a record of such transaction in the BC. Transactions update the state recorded on a BC. They can contain various data depending on the use cases.

Fig. 7: Transaction data structure of pathlets in *QC* framework.

In *QC* framework, transactions are created from pathlets along with their QoS values by BC nodes. Hence, distinct pathlets of all border node pairs are computed by AS controllers for their networks. Fig. 7 shows the data structure of a transaction generated by an AS controller in the *QC* framework. A transaction includes:

- `Tx ID`: Unique ID of the transaction.
- `Signature`: The digital signature of the transaction.
- `ASN`: Unique Autonomous System Number of the AS.
- `Pathlet ID`: Unique ID of a distinct pathlet in an AS.
- `Ingress Node` and `Egress Node`: The start and end nodes of a pathlet in an AS.
- `Max Bandwidth`: The max bandwidth available in a pathlet in an AS.
- `Min Delay`: The min delay in a pathlet in an AS.

In *QC* framework, nodes validate a transaction using a set of rules. These rules confirm that (i) the transaction is digitally signed, (ii) none of the fields in the transaction data structure is empty, (iii) a pathlet's QoS-related fields of a transaction are positive values in the transaction data structure, (iv) a valid ASN is available in the ASN field of the transaction data structure, and (v) ingress and egress node IDs in the transaction belong to the AS creating the transaction.

Tx ID	Signature	ASN	Pathlet ID	Ingress Node	Egress Node	Max Band-width	Min Delay
AS2_1	0xa54be...	AS2	R5_R7_1	R5	R7	15	8
AS2_2	0x19a7d...	AS2	R5_R7_2	R5	R7	5	12
AS2_3	0xb8f8e...	AS2	R5_R7_3	R5	R7	10	15
AS2_4	0x5c91a...	AS2	R5_R7_2	R5	R7	20	12
AS2_5	0x42ae7...	AS2	R5_R7_3	R5	R7	20	15
AS2_6	0xacd42...	AS2	R5_R4_1	R5	R4	40	30
...	...	...	...	...	...	...	...

TABLE I: An example set of transactions by `AS2` controller for pathlets between `R5` and `R7` in `AS2` from Fig. 3a.

Table I presents an example set of transactions created by `AS2` controller for pathlets between `R5` and `R7` border devices in `AS2` network shown in Fig. 3a. Transactions for the other border devices pairs such as `R5 - R8` and `R7 - R8` are also created and propagated in the BC network in the same way. When an AS joins the BC network, it starts creating initial transactions (`AS2_1`, `AS2_2`, and `AS2_3` for pathlets `R5-R7`, `R5-R8-R7`, `R5-R6-R8-R7`, respectively) for the pathlets among the border network devices. They are then propagated to the network or respective BC nodes. Once there is a QoS-related state change in the network, e.g., bandwidth update in a link, which affects the state of some pathlets, then the controller creates update (new) transactions (`AS2_4` and `AS2_5` for `AS2_2` and `AS2_3`, respectively, assuming bandwidth increased by 15 Mbps in link between `R7` and `R8`) reflecting the state change on the respective pathlets. The transaction with `AS_6` ID is created for the inter-connecting link from `R5` to `R4` in `AS2` and `AS1`, respectively. Full network device list over a pathlet is only known by the controller created the corresponding transaction (`AS2` controller in this case). If a pathlet has more than one transaction in controller's BC ledger, then the controller will exploit the most recent one of them while computing an E2E path for a service request.

Block: A block contains transactions batched into a particular data structure by a verifying node after validation of their structures and signatures. A block includes a hash pointer as an interface to the former block, which is one of the measures that guarantee the cryptographic security of the BC.

In *QC*, the primary node of the current block proposal epoch creates the new block. The primary role is transferred from the list of nodes at each epoch to the next node. A new block is created by the primary node following the procedures below:

- All new transactions are gathered from its transaction pool.
- Transactions are reviewed according to the transaction validation rules. The invalid transactions are rejected.
- The primary node monitors the conformance to block generation limits, such as time limits, if available.
- A block is generated including the valid transactions and signed using the primary node's private key.
- The new block is propagated to the other nodes in the BC.

If a continues to generate invalid blocks, it can be banned or excluded from the list of peering nodes. However, this mechanism is out of scope in this work.

B. BC-Driven Design Aspects of QC *Framework*

This subsection explains the BC-driven design principles, aligned with the AS needs and adopted in *QC* framework and realized by means of `Clique`, along with their advantages in QoS-based inter-AS routing as shown in Table II.

BC-Driven Design Aspects	Advantages
Short block generation time	• Network State Accuracy • More transaction throughput
Convenient block proposal	• Less computation cost • Less energy consumption
Democratic block proposal	• Shared block-proposing burden

TABLE II: The design aspects adopted in *QC* framework.

P1. Short block generation time. Rapid reflection of pathlet state updates (so as to let the other nodes use them quickly) is crucial while setting up an accurate and reliable QoS-based E2E path from the point of the whole coordination framework. Moreover, short block generation time can result in a possible increase in transaction throughput.

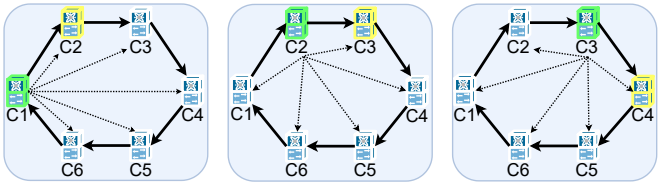
P2. Convenient block proposal. Loading controllers with computation-intensive tasks for block proposal would throttle their performance. Also, a computation-friendly block proposal mechanism would reduce energy consumption.

P3. Democratic/Fair block proposal. Implementing a democratic/fair block proposal mechanism stimulates an equal

sharing of the burden stemming from the blockchain tasks. This in turn prevents the overloading of the controllers.

C. `Clique` Consensus Protocol

We use an implementation of Proof-of-Authority (PoA) consensus algorithm, called `Clique` [16] . `Clique` consensus mechanism has several prominent merits aligning with the design aspects explained in Subsection V-B and, thus, encouraged us to exploit it in *QC* framework as per other consensus models requiring hardware/computational resources, such as Proof-of-Work (PoW), and an existing "share", such as Proof-of-Stake (PoS) [17].



(a) Epoch `i-1`. (b) Epoch `i`. (c) Epoch `i+1`.

Fig. 8: The process of primary (green) and secondary (yellow) BC nodes selection in `Clique`.

One advantage of `Clique` is that the time interval to generate blocks is predictable and shorter while it can deviate in PoW and PoS mechanisms. Also, it provides a high transaction rate in proportion to the other consensus mechanisms. Blocks are created by authorized network nodes in a sequence at specified epochs. This increases the speed of transaction confirmations. Another advantage is that PoA consensus does not require nodes to devote computing resources to solve complex mathematical tasks as opposed to other popular mechanisms such as PoW consensus. This feature comes into prominence for controllers in the *QC* framework because network-related tasks and messages already challenge BC nodes, i.e. controllers, in terms of computational resources in daily operations of the networks. Therefore, there is no need for high-performance equipment. Furthermore, `Clique` determines at most $N-(N/2+1)$ primary and secondary BC nodes to process transactions, propose a block, and broadcast the proposed block to the BC network for each epoch in a round-robin fashion. Fig. 8 presents the process of primary (green) and secondary (yellow) BC nodes selection to propose and propagate a block to the network in case of $N = 6$ BC nodes (i.e., controllers) in three consecutive steps. The secondary node takes over the responsibility in case the primary node fails to propose the block. This sequential process democratizes the block proposal task among the nodes and apportions the burden resulting from block generation and propagation among the BC participants.

D. *Workflow of* QC *Framework*

This subsection briefly explains with an example how *QC* framework works and handles service requests in an SDN network as illustrated in Fig. 9.

When a QoS-based inter-AS service request comes from a user to an AS controller in the form of an *S_Req* message (step 1), the controller starts finding an E2E path considering the QoS parameters and priorities stated in the message using its BC ledger (step 2). The E2E path consists of pathlets from a border node of source-AS to a border node of destination-AS over an overlay network, as pictured in Fig. 3b, abstracted by exploiting the transactions for pathlets from BC.

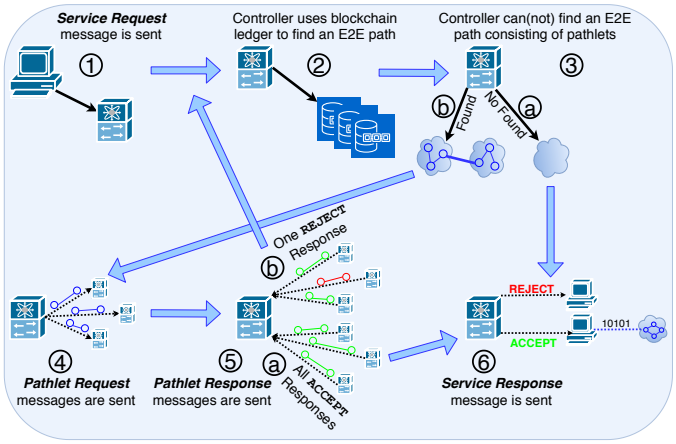


Fig. 9: E2E path finding process in *QC* framework.

The source-AS controller uses `Ingress Node`, `Egress Node`, `Max Bandwidth`, and `Min Delay` fields of transactions, as shown in Table I, to find the E2E path. If the source-AS controller cannot find an E2E path meeting the conditions specified in the *S_Req* message (step 3a), then the source-AS controller sends an *S_Res* message including the REJECT response back to the user/client to state that the service request cannot be provided. When an E2e path is found satisfying the QoS requirements of the service request (step 3b), the source-AS controller asks each AS controller that has a pathlet over the E2E path (step 4) whether they can provide the requested QoS values over the corresponding pathlet by sending *P_Req* messages. The purpose of this inquiry is to confirm that the corresponding pathlet can still provide QoS values advertised in the respective transaction. Because (transaction) updates on the BC are not real-time as the state updates on the pathlets of networks, a pathlet's QoS values may be different than the advertised ones due to certain reasons such as intra-AS traffic and/or another inter-AS traffic. After receiving the *P_Req* messages, respective AS controllers answer to the source-AS controller by sending back their *P_Res* messages stating whether they can provide (ACCEPT or REJECT response) the corresponding pathlet with requested QoS parameters.

Finally, if all respective AS controllers send an ACCEPT response in *P_Res* messages to the source-AS controller (Step 5a), the source-AS controller sends an *S_Res* message including an ACCEPT response back to the user (Step 6) to state that service request can be satisfied and, therefore, can start in the network. The underlying networking-related arrangements such as adding flow entries in flow tables of the corresponding network devices and reserving the network resources for the promised pathlet are handled by the corresponding AS controllers according to the data in the *S_Req* and *P_Req* messages if they accept providing the service/pathlet. Therefore, a service flow is not investigated by any AS controller over

the E2E path again when it enters the AS network. To the contrary, if any of the AS controller over the E2E path sends a REJECT response in *P_Res* messages (Step 5b) to the source-AS controller, the source-AS controller starts finding another E2E path with the same properties and conditions using its BC ledger as explained earlier.

VI. EVALUATION

In this section, we evaluate and compare the performance of *QC* framework against two common QoS-based routing strategies in SDN networks [3], [18]: *Hierarchical Routing Approach (HRA)*, proposed in our previous study [19], and *Distributed Routing Approach (DRA)*, reflecting the current operational mode of the networks. We assume that *DRA* always operates on Border Gateway Protocol (BGP) routes, hence it uses shortest AS_PATH criterion at the AS level in case it detects two similar paths with nearly the same local preference, weight and locally originated or aggregate addresses. We employ E2E Flow Setup Time (FST), Messages Exchanged and Processed (MEP), and Requests Serviced (RS) metrics for the analysis. We also use Random, NSFNET, and US Backbone network topologies at AS-level to conduct a topology-wise sensitivity analysis on performance of the *QC* and the other approaches. We use the same topologies, randomly generated, in intra-AS settings of Random, NSFNET, and US Backbone networks as the number of switches vary in the simulations.

A. Experimental Setup

Mininet emulator and Ryu controller have been used to build and simulate the SDN topology as well as to measure the values of the corresponding parameters as shown in Table III. We have built and used a custom BC network with various numbers of AS controllers and different size of BC transactions. The proposed framework with the original number of pathlet transactions is denoted as *QC*, extended 250K transactions as *QC_250K* and 500K transactions as *QC_500K* in simulations as shown in the figures. When the original pathlet size is less than the aforementioned transaction numbers, we have generated update transactions to extend the size of pathlet transactions in the BC. In the experiments, we have considered only inter-AS service requests with bandwidth parameter (1 Mbps) and provided enough bandwidth (100 Gbps) in physical links so that there is no service request rejection due to network resource limitations. Also, we have used a random topology generator defined by Erdos-Renyi random graph model [20] to randomly create the networks with a degree of connectivity of 0.5. Furthermore, we have averaged 30 runs for each experiment to achieve and exceed 95% statistical significance.

The total number of ASes, switches, and controllers vary from 6 to 10, 36 to 240, and 6 to 10, respectively, in the simulation settings. m in the formulas represents the miscellaneous messages (e.g., the request of E2E global path, respective ingress/egress node information, finding a feasible path, and decision data for a request, etc.) sent among controllers in *DRA* and *HRA*.

Symbol	Definition
$\mathcal{S}$	$\mathcal{S} = \{s^j \mid s^j$ is a switch, $1 \leq j \leq n\}$ (Set of SDN switches from all the ASes)
C	$C = \{^k c \mid ^k c$ is a controller, $1 \leq k \leq n\}$ (Set of SDN controllers from all the ASes)
$\mathcal{S}_i$	$\mathcal{S}_i = \{s_i^j \mid s_i^j$ is a switch, $s^j \in \mathcal{S}\}$ (Set of switches in i-th AS)
C_i	$C_i = \{^k c_i \mid ^k c_i$ is a controller, $^k c \in C\}$ (Set of controllers from i-th AS)
${}_f\mathcal{S}$	${}_f\mathcal{S} = \{{}_f s^j \mid {}_f s^j$ is a switch, $s^j \in \mathcal{S}\}$ (Set of switches over the E2E path for flow f)
${}^k_f\mathcal{S}$	${}^k_f\mathcal{S} = \{{}^k_f s^j \mid {}^k_f s^j$ is a switch, ${}_f s^j \in {}_f\mathcal{S}\}$ (Set of switches connected to k-th controller over the E2E path for flow f
${}_f C$	${}_f C = \{{}^k_f c \mid {}^k_f c$ is a controller, $^k c \in C\}$ (Set of controllers over the E2E path for flow f)
${}_f p$	1st packet of flow f
ρ	OpenFlow `packet_in` message
θ	OpenFlow `flow_mod` message
${}_f\overrightarrow{y}$	A source ($\rightarrow$) network device (y) for flow f
${}_f\overleftarrow{y}$	A destination ($\leftarrow$) network device (y) for flow f
BR	A (super) controller called "Broker"
B^i	Set of border network devices in i-th AS
$T(x)^y_{proc}$	Time to process a message (x) at a network device (y)
$T(x)^{y\rightarrow z}_{prop}$	Time to propagate a message (x) from a network device (y) to a network device (z)
$T(path_L)^y_{comp}$	Time to compute a local path (intra-AS) at controller (y)
$T(path_{E2E_{BC}})^y_{comp}$	Time to compute an E2E path for the service request using BC at controller (y)
$T(path_{E2E})^{BR}_{comp}$	Time to compute an E2E path for the service request at BR

TABLE III: Notation table.

B. Experimental Results

In this subsection, we present the results of experiments conducted with respect to FST, MEP, and RS metrics to evaluate the performance of *QC* framework in comparison to *DRA* and *HRA* frameworks.

1) E2E Flow Setup Time (FST): E2E Flow Setup Time (FST) refers to the time to install corresponding flow rule entries enabling the service/flow request in flow-tables of switches over the E2E path. Thus, there are various steps taking time in determining the FST metric. It is typically determined by (i) networking/topology-based delays (e.g., RTT (Round-Trip-Time) between controller and switches, and controller-to-controller message/packet propagation time) and (ii) switch/controller-based delays (e.g., switch and/or controller packet/message processing time, and path computation time in a controller). If any of these delays are high, then resulting flow setup latency becomes high too. This, in turn, may result in a congestion in both control and data plane levels, and a longer fail-over time in the network. Therefore, FST is one of the important metrics to measure the performance of routing frameworks in SDN networks because it is a good indicator of the scalability dimension.

The FST of a service/flow request in *QC* framework (FST_{QC}) can be given as in Eq. (1) without loss of generality.

$$
\begin{aligned}
FST_{QC} &= \lambda(S\_Req, S\_Req, {}_f\overrightarrow{h}, {}_f\overrightarrow{s}, {}_f\overrightarrow{c}) + T(path_{E2E_{BC}})^{{}_f\overrightarrow{c}}_{comp} \\
&+ \max_{\{\forall {}^k_f c \in {}_f C\}} \Big\{ \gamma(P\_Req, {}_f\overrightarrow{c}, {}^k_f c) + \gamma(P\_Res, {}^k_f c, {}_f\overrightarrow{c}) \\
&+ \max_{\{\forall {}^k_f s^j \in {}^k_f \mathcal{S}\}} \Big\{ T(\theta)^{{}^k_f c \rightarrow {}^k_f s^j}_{prop} \Big\} + T(P\_Res)^{{}_f\overrightarrow{c}}_{proc} \Big\} \\
&+ \lambda(S\_Res, S\_Res, {}_f\overrightarrow{c}, {}_f\overrightarrow{s}, {}_f\overrightarrow{h})
\end{aligned}
\tag{1}
$$

The processing and propagation times of corresponding *S_Req* and *S_Res* messages at/among ${}_f\overrightarrow{h}$, ${}_f\overrightarrow{s}$, and ${}_f\overrightarrow{c}$ are calculated using Eq. (2). Eq. (1) finds the max. of processing and propagation times of *P_Req*, *P_Res* using Eq. (3). As reflected in Eq. (1), it also involves `flow_mod` messages at/among the controllers and switches over the E2E path.

$$\lambda(x_1, x_2, y_1, y_2, y_3) = T(x_1)^{y_1}_{proc} + T(x_1)^{y_1 \rightarrow y_2}_{prop} + T(x_1)^{y_2}_{proc} + T(x_2)^{y_2 \rightarrow y_3}_{prop} + T(x_2)^{y_3}_{proc} \quad (2)$$

$$\gamma(x, y_1, y_2) = T(x)^{y_1 \rightarrow y_2}_{prop} + T(x)^{y_2}_{proc} \quad (3)$$

Similarly, the FST of a service/flow request in the *HRA* framework (FST_{HRA}) can be characterized as in Eq. (4).

$$FST_{HRA} = \delta({}_f p, \rho, {}_f\overrightarrow{h}, {}_f\overrightarrow{s}, {}_f\overrightarrow{c}) + T(m)^{{}_f\overrightarrow{c} \rightarrow BR}_{prop} + \max\left\{\beta(m, path_L, {}_f\overrightarrow{c}), \beta(m, path_L, {}_f\overleftarrow{c})\right\} + T(path_{E2E})^{BR}_{comp} + \max_{\{\forall {}^k_f c \in {}_f C\}}\left\{\beta(m, path_L, {}^k_f c) + \max_{\{\forall {}^k_f s^j \in {}^k_f S\}}\left\{T(\theta)^{{}^k_f c \rightarrow {}^k_f s^j}_{prop}\right\} + T(m)^{BR}_{proc}\right\} \quad (4)$$

As reflected in Eq. (4), it primarily involves: (i) processing and propagation times for the 1st packet of flow f and corresponding `packet_in` message at/among ${}_f\overrightarrow{h}$, ${}_f\overrightarrow{s}$, and ${}_f\overrightarrow{c}$ as formulated in Eq. (5), (ii) maximum of (a) propagation time of local path request messages from *BR* to ${}_f\overrightarrow{c}$ plus local path computation time at ${}_f\overrightarrow{c}$ and (b) the same procedures for ${}_f\overleftarrow{c}$ as formulated in Eq. (6), (iii) computation time to find a feasible E2E path for the service request at *BR*, and (iv) maximum of processing and propagation times of path request/setup messages and `flow_mod` messages at/among *BR*, controllers, and switches over the E2E path.

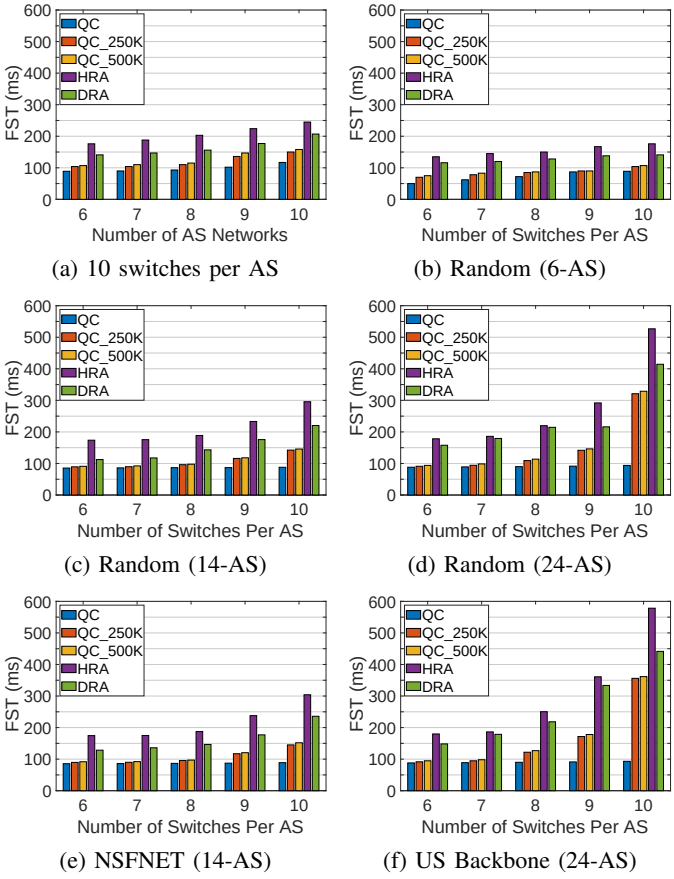
$$\delta(x_1, x_2, y_1, y_2, y_3) = \lambda(x_1, x_2, y_1, y_2, y_3) + T(x_2)^{y_2}_{proc} \quad (5)$$

$$\beta(x, P, y) = T(x)^{BR \rightarrow y}_{prop} + T(P)^{y}_{comp} + T(x)^{y \rightarrow BR}_{prop} \quad (6)$$

As for the FST of a service/flow request in *DRA* (FST_{DRA}), it can be shown as in Eq. 7.

$$FST_{DRA} = \delta({}_f p, \rho, {}_f\overrightarrow{h}, {}_f\overrightarrow{s}, {}_f\overrightarrow{c}) + \sum_{\substack{\forall {}^k_f c \in {}_f C \\ 1 \le k \le |{}_f C|}}\left(T(path_L)^{{}^k_f c}_{comp} + T(m)^{{}^k_f c \rightarrow {}^{k+1}_f c}_{prop}\right) + \max\left\{\sum_{\substack{\forall {}^k_f c \in {}_f C \\ |{}_f C| \le k \le 1}}\left(T(m)^{{}^k_f c}_{proc} + T(m)^{{}^k_f c \rightarrow {}^{k-1}_f c}_{prop}\right), \max_{\{\forall {}^k_f c \in {}_f C\}}\left\{\max_{\{\forall {}^k_f s^j \in {}^k_f S\}}\left\{T(\theta)^{{}^k_f c \rightarrow {}^k_f s^j}_{prop}\right\}\right\}\right\} \quad (7)$$

As reflected in Eq. 7, it consists of (i) processing and propagation times for the 1st packet of flow f and corresponding `packet_in` message at/among ${}_f\overrightarrow{h}$, ${}_f\overrightarrow{s}$, and ${}_f\overrightarrow{c}$ as formulated in Eq. (5) , (ii) local path computation time at a controller and propagation time for decision message to the next controller over the E2E path, (iii) maximum of processing and propagation times of path reservation messages and `flow_mod` messages at/among controllers and switches over the E2E path. We note that k represents an index in the respective sets, not an ID. For example, assuming ${}_f C = \{A, E, B, R\}$ is a set of controllers over the E2E path for flow f, then, ${}^1_f c = A$, ${}^2_f c = E$, ${}^3_f c = B$, ${}^4_f c = R$.



(a) 10 switches per AS (b) Random (6-AS)

(c) Random (14-AS) (d) Random (24-AS)

(e) NSFNET (14-AS) (f) US Backbone (24-AS)

Fig. 10: Flow Setup Time (FST) of *QC*, *DRA*, and *HRA*.

Fig. 10a shows the FST values while varying AS numbers with 10 switches per AS network and different service requests (i.e., different source and destinations). The *QC* has better FST values than *DRA* and *HRA* across the board, as intuitively expected from the information already maintained by the BC infrastructure and due to higher computations of DRA and HRA emanating from more ASes. The *DRA* outperforms *HRA* regarding FST in the simulations because the Broker communicates with all AS controllers over the E2E path in *HRA* whereas BGP-based shortest path is used in *DRA* by means of BGP next-hop attribute values towards the destination AS to setup an E2E path.

Figs. 10b - 10f show the total FST values in *QC*, *QC_250K*, *QC_500K*, *DRA*, and *HRA* cases while varying the number of switches per AS in 6-, 14-, 24-AS Random, NSFNET, and US Backbone network topologies by using different service requests (i.e., each request has different source-destination pairs where source and destinations are from different ASes). As it can be seen in the figures, the *QC* framework outperforms the *DRA* and *HRA* approaches in terms of the FST as the number of switches per AS increases because more switch-controller and controller-controller communication delays are involved in the flow setup processes in *DRA* and *HRA*. This is because the

QC approach takes advantages of using available transactions, reflecting QoS-related states of the ASes, to determine the routing path with minimal number of control messages than both the *DRA* and *HRA* approaches. The increase ratio of number of distinct pathlets in an AS is not necessarily linear as the number of switches per AS increases. The increase on the number of distinct pathlet in an AS results in an increase, in turn, in the number of transactions in the BC that the AS needs to search and read to compute an E2E path. The spikes in point 9 and particularly 10 of *QC_250K* and *QC_500K* in Fig. 10d and Fig. 10f are results of the aforementioned situation along with more ASes in the topology.

2) Messages Exchanged and Processed (MEP): Network controllers may have to cope with more flow requests, which generate more Messages Exchanged and Processed (MEP) among network devices and other controllers to set up QoS-based E2E paths, as the number of nodes grows. These could be any messages needed to setup a QoS-based E2E path for a service request and be called network and communication overhead. These message exchange and handling procedures can make the controllers as bottleneck due to its limited computation resources (i.e., CPU and memory). Hence, minimizing the number of MEP to set up an E2E path for a service/flow request at/by respective decision-maker controllers is important for the controllers, which makes MEP another crucial metric to evaluate the routing performance and the scalability of the entire network system in SDN networks.

The number of MEP to set up an E2E path for a service/flow request in *QC* framework (MEP_{QC}), *HRA* framework (MEP_{HRA}), and *DRA* framework (MEP_{DRA}) are given in Eqs. (8)-(10), respectively.

$$MEP_{QC} = 2 \cdot |_f C| + \sum_{\forall {}^k_f c \in_f C} \left| {}^k_f \mathcal{S} \right| \tag{8}$$

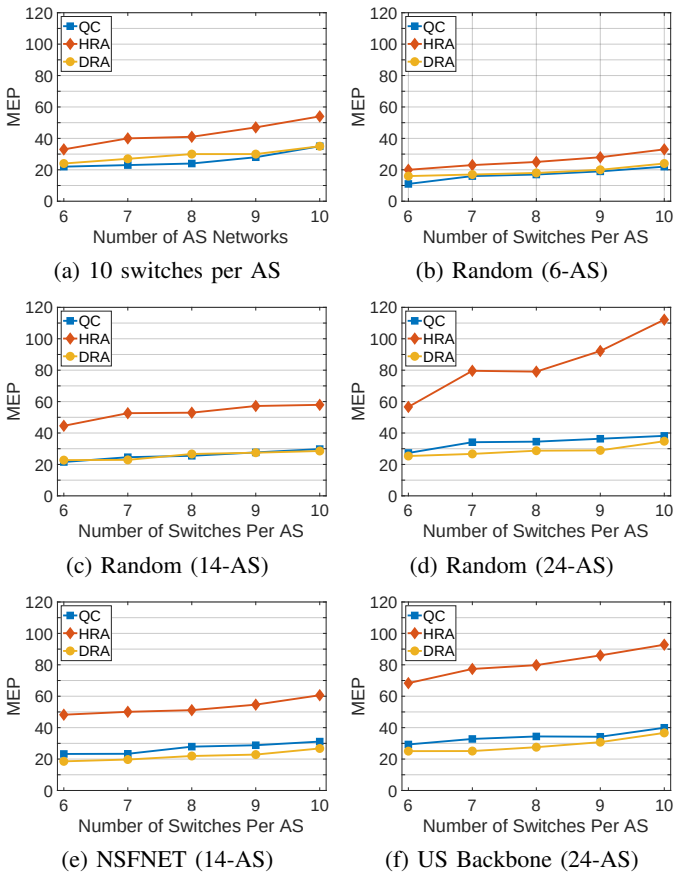
As shown in Eq. (8), MEP_{QC} mainly involves: (i) an *S_Req* message sent from source host ($_f \overrightarrow{h}$) to source controller ($_f \overrightarrow{c}$), (ii) *P_Req* messages sent from $_f \overrightarrow{c}$ to the other controllers over the E2E path found for the service request, (iii) *P_Res* messages sent from the controllers over the E2E path found for the service request to the $_f \overrightarrow{c}$ in response to the *P_Req* messages, (iv) an *S_Res* message sent from $_f \overrightarrow{c}$ to $_f \overrightarrow{h}$, and (v) `flow_mod` messages sent from the controllers to the switches in their networks over the E2E path.

$$MEP_{HRA} = 4 + \left|B^{f\,\overrightarrow{N}}\right| + \left|B^{f\,\overleftarrow{N}}\right| + 2 \cdot |_f C| + \sum_{\forall {}^k_f c \in_f C} \left| {}^k_f \mathcal{S} \right| \tag{9}$$

$_f \overrightarrow{N}$ and $_f \overleftarrow{N}$ are source and destination ASes over the E2E path for flow f, respectively. Similarly, in Eq. (9), MEP_{HRA} mainly involves: (i) a `packet_in` message sent from source-switch ($_f \overrightarrow{s}$) to $_f \overrightarrow{c}$, (ii) an E2E path request message sent from $_f \overrightarrow{c}$ to the *BR*, (iii) two computation request messages from *BR* to the $_f \overrightarrow{c}$ and destination-controller ($_f \overleftarrow{c}$) (for local paths advertisements between each border node and source/destination switch pairs), (iv) messages including pathlet advertisements for each border node and source/destination switch pairs from $_f \overrightarrow{c}$ and $_f \overleftarrow{c}$ to *BR*, (v) messages (including respective border node pairs in each network) sent from the *BR* to the controllers over the E2E path, (vi) confirmation messages (for the requested pathlets in each network) sent from the controllers over the E2E path to *BR*, and (vii) `flow_mod` messages sent from the controllers over the E2E path to the switches in their networks over the E2E path.

$$MEP_{DRA} = 1 + 2 \cdot |_f C| + \sum_{\forall {}^k_f c \in_f C} \left| {}^k_f \mathcal{S} \right| \tag{10}$$

MEP_{DRA} as in Eq. 10 mainly involves: (i) a `packet_in` message sent from $_f \overrightarrow{s}$ to $_f \overrightarrow{c}$, (ii) decision and notification (for path reservation) messages sent among the controllers throughout an E2E path for the requested path, and (iii) `flow_mod` messages sent from the controllers to the switches in their networks over the E2E path.



(a) 10 switches per AS (b) Random (6-AS)

(c) Random (14-AS) (d) Random (24-AS)

(e) NSFNET (14-AS) (f) US Backbone (24-AS)

Fig. 11: Messages Exchanged and Processed (MEP) of *QC*, *DRA*, and *HRA*.

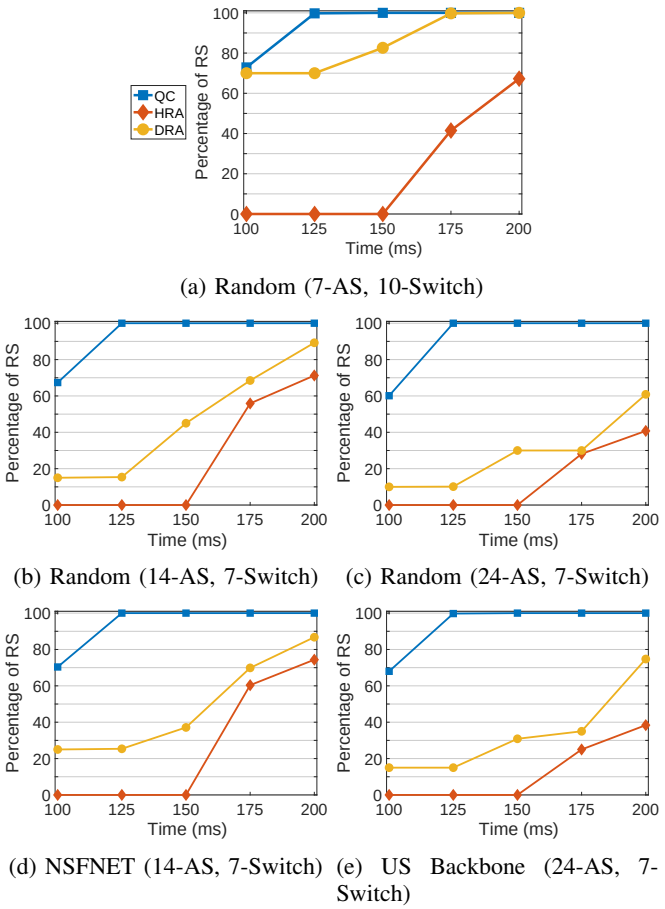
Fig. 11 shows the MEP values in *QC*, *DRA*, and *HRA* models. In Fig. 11a, AS numbers vary with the fixed number of switches (10) per AS. In Figs. 11b - 11f, the number of switches per AS vary with 6-, 14-, 24-AS Random, NSFNET, and US Backbone networks, respectively.

In Fig. 11a, the *HRA* requires as high as 50% more MEP than the *QC* and *DRA* throughout all AS cases. This is because the *HRA* uses more number of border devices in ASes as reflected in Eq. 9.

Figs. 11b - 11f present that the *QC* and *DRA* frameworks display a better performance than the *HRA* approach with respect to varying the number of switches per AS under different network topologies. In the figures, the MEP of the *QC* and *DRA* presents almost identical values while the number of switches per AS increases. One reason for this fact is that the E2E path for all frameworks may not necessarily be the same in case of the same request (i.e., total number of switches and controllers may have different values). Another reason is that an AS always finds a feasible path by asking only its default BGP next-hop AS to the destination in the *DRA*,i.e, using the shortest path at AS level. Therefore, an AS do not need to communicate and exchange any messages with another AS other than only its next-hop AS to the destination (less message communication and exchange). It is worth noting that while MEP of *QC* is similar to that of *DRA*, although *DRA* operates on the shortest BGP path at AS-level, *QC* significantly reduces the FST as shown in Fig. 10. Moreover, the MEP values are getting higher in Figs. 11b - 11f when increasing number of switches over the E2E paths as intra-AS switch number grows in each setting.

3) Requests Serviced (RS): Allocating network control instances for each service/flow request in flow tables of switches over the E2E path may influence the number of requests serviced (RS) for the entire network in *QC*, *HRA*, and *DRA* frameworks due to their FST. When the FST is higher, the frameworks can establish the service request in a longer time period. Also, we note that the major goal of the RS metric is to provide another validation for the results of the FST parameter to corroborate those observations from a different perspective. Therefore, the number of RS per unit of time is another important metric for the scalability of routing frameworks in case of QoS-based E2E path setup.

Fig. 12 shows the percentage of RS in case of 800 service requests while varying the time range (ms) for 7-, 14-, 24-AS Random, NSFNET, and US Backbone networks. As time increases, the *QC* framework can easily handle all service requests. However, overall, the percentage of RS for *DRA* framework decreases while the intensity of ASes increases. As expected from Fig. 10 FST values, Fig. 12 confirms that the *HRA* cannot handle the service requests until time is 150 ms. When time exceeds 150 ms, the *HRA* starts servicing the requests as shown in Figs. 12a - 12e. In other words, the *HRA* needs higher FST to find the E2E paths. While the HRA and DRA performances get closer to each other compared to smaller time values there continues to be a noticeable difference and DRA dominates HRA at all times. As a result of RS metric, the *QC* frameworks can establish all service requests in shorter time whereas the *HRA* and *DRA* frameworks cannot handle all service requests as shown in Figs. 12b - 12e. This is because the *QC* framework takes the advantages of BC transactions, already established by the AS controllers, to find QoS-based E2E path over AS networks.



(a) Random (7-AS, 10-Switch)

(b) Random (14-AS, 7-Switch)

(c) Random (24-AS, 7-Switch)

(d) NSFNET (14-AS, 7-Switch)

(e) US Backbone (24-AS, 7-Switch)

Fig. 12: Requests Serviced (RS) of *QC*, *DRA*, and *HRA*.

VII. Discussions

Just like any other emerging and promising technology, BC is facing challenges as well. These issues indicate possible future research directions. The following provides a summary of these areas where there is room for improvement:

- *Block Generation Time*: This time can help the entire model achieve higher transaction throughput and keep the BC ledger up-to-date regarding pathlet states. On the other hand, this interval can lead to more computation on network controllers and use more link resources on the networks.
- *Consensus Protocol Scalability*: Loading AS controllers with computation-intensive tasks, such as solving cryptographic problems, to generate a new block may limit their network-related performances and, hence, necessitates computational power. Therefore, utilizing a more lightweight yet efficient consensus protocol would relieve controllers' load.
- *BC Ledger Update Time*: It is possible that the reflected state change of the pathlet in the transaction may not hold until it is reflected in the BC ledger because another update on the state of the corresponding pathlet may happen in the network. Therefore, some of the transactions may not reflect the real up-to-date pathlet states on the BC.
- *Transaction Amount*: Network updates are reflected in new transactions and propagated to all participating networks in the proposed model. Therefore, storage may become a constraint in the very long run. This situation can also affect the network overhead that controllers deal with.

VIII. Conclusions and Future Works

This study presents an emerging possible use case and research direction for the BC technology. We have explored the viability and feasibility of a novel BC-based SDN for QoS-based inter-AS routing. Preliminary results show promising performance in terms of flow setup time and the percentage of requests serviced. The *QC* shows a comparable message overhead to the shortest-path based distributed approach although the latter operates on the shortest BGP paths in terms of hop count at the AS-level. Our experiments also display a faster convergence by *QC* than the other hierarchical and distributed QoS routing approaches in SDN.

As for the future work, while the experimental results of QC have shown acceptable levels of performance for various metrics, we will work on our techniques to improve their performance further. Also, we will examine the robustness of the proposed framework by analyzing various security attacks. Finally, we will look into the economic dimension to extend the current work into a practical framework by means of smart contracts.

Acknowledgement

This work is supported by the Scientific and Technological Research Council of Turkey (TUBITAK) under Grant No. 120E448.

References

[1] C. Labovitz, S. Iekel-Johnson, D. McPherson, J. Oberheide, and F. Jahanian, "Internet Inter-Domain Traffic," *SIGCOMM Comput. Commun. Rev.*, vol. 41, no. 4, pp. –, Aug. 2010.

[2] "Software-Defined Networking: The New Norm for Networks," Open Networking Foundation (ONF), Tech. Rep., April 2012.

[3] 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.

[4] D. Yaga, P. Mell, N. Roby, and K. Scarfone, "NISTIR 8202 Blockchain Technology Overview," *National Institute of Standards and Technology*.

[5] "SDN architecture," Open Networking Foundation (ONF), Tech. Rep., June 2014.

[6] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: Enabling innovation in campus networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, Mar. 2008.

[7] P. Lin, J. Bi, S. Wolff, Y. Wang, A. Xu, Z. Chen, H. Hu, and Y. Lin, "A West-East Bridge Based SDN Inter-Domain Testbed," *Communications Magazine, IEEE*, vol. 53, no. 2, pp. 190–197, Feb 2015.

[8] P. Wang, X. Liu, J. Chen, Y. Zhan, and Z. Jin, "QoS-Aware Service Composition Using Blockchain-Based Smart Contracts," in *ICSE*, New York, NY, USA, 2018, p. 296–297.

[9] E. Ak and B. Canberk, "BCDN: A proof of concept model for blockchain-aided CDN orchestration and routing," *Computer Networks*, vol. 161, pp. 162 – 171, 2019.

[10] J. Yang, S. He, Y. Xu, L. Chen, and J. Ren, "A Trusted Routing Scheme using Blockchain and Reinforcement Learning for Wireless Sensor Networks," *Sensors*, vol. 19, no. 4, p. 970, 2019.

[11] M. Saad, A. Anwar, A. Ahmad, H. Alasmary, M. Yuksel, and A. Mohaisen, "RouteChain: Towards Blockchain-based Secure and Efficient BGP Routing," in *ICBC*, May 2019, pp. 210–218.

[12] G. Ramezan and C. Leung, "A Blockchain-Based Contractual Routing Protocol for the Internet of Things Using Smart Contracts," *Wireless Communications and Mobile Computing*, vol. 2018, 2018.

[13] P. Kamboj and S. Pal, "QoS in Software Defined IoT Network Using Blockchain Based Smart Contract: Poster Abstract," in *SenSys*, New York, NY, USA, 2019, p. 430–431.

[14] A. Arins, "Blockchain based Inter-domain Latency Aware Routing Proposal in Software Defined Network," in *AIEEE*, 2018, pp. 1–2.

[15] M. Karakus and E. Guler, "Routingchain: A proof-of-concept model for a blockchain-enabled qos-based inter-as routing in sdn," in *BlackSeaCom*, 2020, pp. 1–6.

[16] S. De Angelis, "Assessing Security and Performances of Consensus Algorithms for Permissioned Blockchains," *arXiv preprint arXiv:1805.03490*, 2018.

[17] Y. Xiao, N. Zhang, W. Lou, and Y. T. Hou, "A survey of distributed consensus protocols for blockchain networks," *arXiv preprint arXiv:1904.04098*, 2019.

[18] 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.

[19] M. Karakus and A. Durresi, "A Scalable Inter-AS QoS Routing Architecture in Software Defined Network (SDN)," in *AINA*, March 2015, pp. 148–154.

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

Murat Karakus received the Ph.D. degree in the Computer Science from Purdue School of Science in Indianapolis, USA, in 2018. He is currently working as an Assistant Professor at Bayburt University in Turkey. His current research interests include new network architectures, e.g. SDN, QoS, scalability, routing, Blockchain technology, economics analysis of network architectures and designs.

Evrim Guler received the Ph.D. degree in Computer Science from Georgia State University, Atlanta, GA in 2019. Currently, he is an Assistant Professor in the Department of Computer Engineering at Bartin Universiy, Turkey, where he leads the Distance Education and Research Center. His research interests include smart systems and network modeling.

Suleyman Uludag received the Ph.D. degree in Computer Science from DePaul University, Chicago, IL, in 2007. He is currently an Associate Professor of Computer Science at the University of Michigan - Flint. He received the Fulbright U.S. Scholar Program Core Award in 2012 and 2018. His research interests include security, privacy, and optimization of data collection particularly as applied to the Smart Grid and Intelligent Transportation Systems.