IEEE Access
Multidisciplinary : Rapid Review : Open Access Journal

Received 14 September 2022, accepted 23 October 2022, date of publication 26 October 2022, date of current version 4 November 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3217534

RESEARCH ARTICLE

Blockchain-Powered Bandwidth Trading on SDN-Enabled Edge Network

YUSTUS EKO OKTIAN[1], THI-THU-HUONG LE[2], UK JO[3], (Graduate Student Member, IEEE), AND HOWON KIM[3], (Member, IEEE)

[1]Blockchain Platform Research Center, Pusan National University, Busan 609735, South Korea
[2]IoT Research Center, Pusan National University, Busan 609735, South Korea
[3]School of Computer Science and Engineering, Pusan National University, Busan 609735, South Korea

Corresponding author: Howon Kim (howonkim@pusan.ac.kr)

This work was supported in part by the Ministry of Science and ICT (MSIT), South Korea, under the Information Technology Research Center (ITRC) Support Program supervised by the Institute for Information and Communications Technology Planning and Evaluation (IITP), under Grant IITP-2022-2020-0-01797; and in part by IITP Grant through the Korea Government (MSIT) Research on Blockchain Security Technology for the Internet of Things (IoT) Services (50%) under Grant 2018-0-00264.

ABSTRACT Bandwidth trading procedures can be made to incentivize users to sell their needless traffic and indirectly reduce the probability of traffic congestion. However, implementation of bandwidth trading is opex-heavy from Internet Service Provider (ISP) perspective, while on the other hand, users also do not trust network executions from the ISP due to its heavily centralized control. These issues hinder the applicability of bandwidth trading and become our motivation to propose this paper. Our bandwidth-trading framework utilize software-defined networking (SDN) and blockchain. SDN automates the bandwidth trading executions from the ISP side and reduces the opex. Meanwhile, the smart contract is a trusted platform for building a trading marketplace where buyers, sellers, and SDN controllers can negotiate the trading terms. Once the trading is executed, SDN controllers generate proof of trading that must be submitted to the smart contract as proof of provisioning. We implement our works using Ethereum and POX SDN controllers, and the results prove that it can provide a seamless bandwidth trading experience with reasonable overhead. Furthermore, by committing to our framework, bandwidth trading can be executed fairly and securely because all previous provisioning can be cross-checked through the provided proof-of-trading.

INDEX TERMS Bandwidth trading, marketplace, SDN, blockchain.

I. INTRODUCTION

The Internet Service Provider (ISP) is most likely to control the Internet bandwidth statically according to the consumers' subscription plan, and sometimes the Internet deliveries from ISP do not match the numbers that the ISP previously advertised because ISP often aggregates multiple connections from consumers into a single channel. Traffic congestion may occur if many consumers consume bandwidth simultaneously, resulting in performance degradation happening to all consumers sharing the same channel. The ISP may eliminate this issue by upgrading the channel capacity with maximum bandwidth to satisfy all subscribed plans from consumers. However, that solution is expensive and inefficient since cases where all consumers use their bandwidth simultaneously can be considered (in most cases) rare and ephemeral events.

A bandwidth trading scheme can motivate users to shift their traffics during peak hours and indirectly solve the previously mentioned congestion problems. Instead of consuming bandwidth, consumers now have an opportunity to sell parts or all of their bandwidth to nearby consumers. When selling bandwidth, consumers commit that they will not use their bandwidth for a given duration. The ISP can then use this unused bandwidth to serve other consumers sharing the same channel. Later, the consumers will be compensated for their sold bandwidth and earn profits. This trading scheme can potentially satisfy the bandwidth needs of each user without necessarily upgrading the link to all users. Thus, a win-win solution for both consumers and ISPs.

The associate editor coordinating the review of this manuscript and approving it for publication was Thanh Ngoc Dinh.

Despite the previously mentioned benefits, performing such bandwidth trading comes with several challenges. From the ISP perspective, the trading procedures must be easy-to-implement and automated to reduce opex [1]. On the other hand, it is well known that ISP governs the network in a centralized way [2]. As a result, ISP has complete control to manipulate customers' data, and the approvals of bandwidth selling/buying become subject to ISP decisions. When performed maliciously, the trading can be unfair to consumers (e.g., ISP performs censorships on consumers or does not follow the previously agreed rules). Those conditions may impose untrusted relationships from consumers towards ISP [3]. The software-defined networking (SDN) [4] can help ISP to automate day-to-day network processes and indirectly reduce opex. Meanwhile, the blockchain [5] can potentially solve the trust issues between consumers and ISP by facilitating a verifiable and trusted collaborative platform. Therefore, we envision that the combination of SDN and blockchain technology can be used to solve the previously mentioned issues of bandwidth trading.

This paper proposes a novel bandwidth trading mechanism that leverages SDN and blockchain. First, we develop a marketplace, on top of smart contract [6], for buyers and sellers to negotiate on bandwidth trading. Second, we use OpenFlow messages [7] to send Flow Rules to SDN switches that will enforce the bandwidth trading by increasing/decreasing the traffics for buyers/sellers. Third, we design proof of trading mechanism as a cross-checking tool. The SDN controller can create proofs of trading provisioning from OpenFlow messages. Meanwhile, buyers and sellers use their transactions in the blockchain as proofs of bandwidth requests and proofs of selling offers. Fourth, we present a proof of concept implementation of our proposals and further discuss their feasibility through system fairness, security, and performance overhead analysis.

Technically, our research exhibits seamless integration of off-chain and on-chain dataflow from both the SDN and blockchain sides to realize bandwidth trading. We demonstrate an illustrative implementation of the collaboration process among sellers and SDN controllers to fulfill available bandwidth demands from buyers. We also show how the proof of trading can be used to cross-check the trading provisioning from ISP. To our knowledge, research works in realizing bandwidth trading schemes are almost nonexistent. Therefore, our research can also be seen as a preliminary effort to assess the possibility of fair and secure bandwidth trading for both ISP and consumers.

The remainder of this paper is organized as follows. Section II reviews the literature and introduces previous related studies. Section III discuss the problem statement and bandwidth trading scenarios in our paper. Section IV presents our proposed blockchain-based bandwidth trading schemes, while their feasibility, fairness, and security assessments are analyzed in Section V. We then discuss several limitations and possible improvements of our approach in Section VI. Finally, we conclude in Section VII.

II. LITERATURE REVIEW

A. BLOCKCHAIN AND SMART CONTRACTS

Blockchain has gained traction lately because of the popularity of Bitcoin [5] as a truly decentralized peer-to-peer cryptocurrency platform. Generally, a blockchain is an append-only data ledger whose integrity is satisfied with the chain of hashes in the block, and the consensus algorithm guarantees its decentralization. Depending on that consensus algorithm, we can tweak the blockchain network into a public or private setting. For example, Proof-of-work (PoW) [5] allows anyone to join the network, while Practical Byzantine Fault Tolerance (PBFT) [8] only allows a limited number of authenticated nodes to join the network.

The concept of a smart contract was first introduced by Nick Szabo [9], and it is popularized with the integration into the blockchain by Ethereum [6]. Smart contracts allow developers to put distributed and deterministic codes in blockchain networks, creating new concepts of decentralized applications (dapps) [10]. With these smart contracts, researchers can employ blockchain to many use cases aside from cryptocurrency such as decentralized identity [11], non-fungible tokens (NFT) [12], marketplace [13], and crowdsourcing [14].

Our proposal contributes to blockchain integration into bandwidth management in the SDN domain. To our knowledge, only a limited study has been proposed in this area. Our paper represents a preliminary attempt in this direction and widens the use case area for blockchain and smart contracts.

B. TRADING OF NETWORK RESOURCES

There are several examples of network resource management that are achievable via trading.

Customers can request or reserve a given amount of bandwidth to be used in the future, also known as bandwidth-on-demand services. In this case, the available bandwidth reserve are being traded to satisfy customers' demand such as in [15], [16], and [17]. Besides bandwidth reserve, Chase et al. [18] solve the resource distribution problem in cloud computing by trading the amount required for the ISP bandwidth and the number of virtual machines (VM) required.

Yakubu et al. [19] allow consumers to sell parts of their bandwidth to their neighbors, those within the same radio range. In this case, the download capacity is being traded to serve Internet services for others. In [20], network peers are allowed to help the streaming server in case the server is overloaded. Unlike previous studies, the helpers' upload capacity is traded in this scenario.

Ding et al. [21] propose a spectrum trading to eliminate the mismatch between the assigned capacity and the actual traffic in Virtual Optical Network (VON). Different VONs are allowed to trade their spectrum resource at a given period. Similarly, Farshbafan et al. [22] also propose a spectrum trading for device-to-device communication in which spectrums are traded as bandwidth for data transmission.

TABLE 1. Comparisons of our proposal and previous studies in terms of domain, what resources are traded, whether they employ blockchain and smart contract (BC), software-defined networking (SDN), and provide proof of trading (Proof).

Ref	Domain	Resources	BC	SDN	Proof
[17]	Mobile Edge Computing	BW Reserve	✗	✗	✗
[15]	Edge Network	BW Reserve	✓	✓	✗
[16]	Datacenter Network	BW Reserve	✗	✓	✗
[18]	Cloud Network	BW Reserve, VM	✗	✓	✗
[19]	Edge Network	Download Link	✓	✗	✗
[20]	P2P Streaming System	Upload Link	✗	✗	✗
[21]	Virtual Optical Network	Spectrum	✓	✓	✗
[22]	Cellular Network	Spectrum	✗	✗	✗
Ours	Edge Network	BW Reserve, Download Link	✓	✓	✓

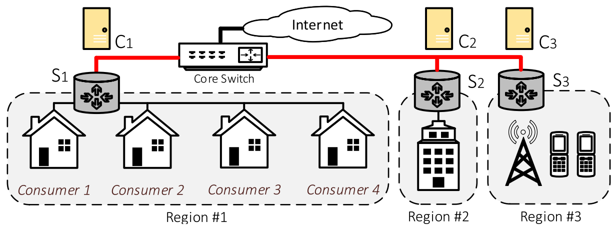


FIGURE 1. An example of SDN-enabled edge networks throughout *n* regions. The SDN controllers (C_n) provision inward/outward traffics in the region, which goes through edge SDN switches (S_n) as gateways to the core switch. The ISP allocates some finite bandwidth to each region, depicted as red lines in the figure.

Our proposal contributes to the trading of bandwidth reserve and download capacity in the edge network, which overlaps the works in [15] and [19].

C. BLOCKCHAIN AND SDN INTEGRATION FOR SECURE AND RELIABLE TRADING

SDN opens programmability on the SDN controllers so that developers can build their customized applications on top of the network, much like how we can create dapps on the blockchain. Due to open APIs, blockchain and SDN can be integrated to make seamless, secure, and reliable trading by following three fundamental principles:

1) Customers negotiate the traded resources in the smart contract; all trading parameters are then recorded in the blockchain.
2) SDN controllers enforce the trading policy to the switches following the agreement previously made in the smart contract.
3) Proof of trading will be generated from the SDN and blockchain sides as validation proofs.

Regardless of domain area, Table 1 shows that existing studies have not fully integrated those three principles when trading network resources. Many of them lack one or more items, while our framework represents a preliminary attempt to satisfy those requirements. In a more detailed comparison, our works overlap the works of Chen et al. [15], which propose PayFlow, and the works of Yakubu et al. [19]. Despite sharing the same research problems, we argue that our proposal is far superior to both studies. Table 2 summarizes our differences.

First, PayFlow only allows SDN controllers to sell bandwidth. Meanwhile, we allow not only SDN controllers but also consumers to sell bandwidth to satisfy the demands. Second, PayFlow does not discuss concurrencies and assumes that only one demand request exists. It is then unclear how their system will react when receiving multiple bandwidth demand requests. Meanwhile, our proposal supports multiple buyers/sellers/controllers to trade their bandwidth. Third, PayFlow does not include smart contract developments. The blockchain is only used as token transfers, while the bandwidth trading provisioning is performed entirely off-chain. Therefore, the integrity guarantee of trading cannot be satisfied in PayFlow. In contrast, we process the trading negotiation on-chain via smart contracts. Fourth, Yakubu et al. do not have any implementation on the bandwidth trading side and only assess the feasibility from the blockchain side. Therefore, their feasibility analysis is incomplete. Meanwhile, we leverage SDN to implement bandwidth trading and integrate them into the blockchain to create a seamless bandwidth trading experience. Finally, we provide a proof-of-trading mechanism that can be used to audit the traded bandwidth resources. Neither works provide this feature.

III. PRELIMINARIES

A. PROBLEM STATEMENT

We envision SDN-enabled edge networks spread out across geographical locations in the form of "regions", as shown in Figure 1. Those regional networks can be in many forms such as residential [23], enterprise [24], or cellular [25]. Regional administrators, through SDN controllers, allocate a finite amount of bandwidth reserve to a region, which will then be shared with all in-region consumers. In this case, a fixed amount of bandwidth from the reserve will be allocated to each consumer.

During day-to-day operations, if needed, a consumer can buy additional bandwidth to the ISP through the SDN controller. This is usually known as a "bandwidth on demand" service. However, because of only a limited bandwidth reserve available, the SDN controller may not be able to accept and provision all of the demand requests.

To alleviate this issue, our proposal allows consumers in the same region to "sell" parts of their bandwidth to fulfill the demand requests. By selling the bandwidth, we do not mean that the consumers provide an off-channel link for buyers to connect to the Internet and become a "re-seller" ISP. Instead, by "selling", we mean that the consumers commit to "not using" parts of their bandwidth temporarily so that the unused bandwidth can be provisioned to fulfill the demand requests. Sellers will then be compensated accordingly for their bandwidth loss. This way, we allow consumers to trade each others' bandwidth so that the bandwidth demand requests may have a higher chance of being accepted.

TABLE 2. The detailed comparison between the proposed framework and the recent progress of trading bandwidth resources in Edge Network from Table 1. Buy, Sell, Close, and Cancel refer to the provided bandwidth trading stages.

Research	Blockchain	Smart Contract	SDN Controller	Buy	Sell	Close	Cancel	Concurrencies	Proof of Trading
Yakubu et. al [19]	Ethereum	✓	✗	✓	✓	✗	✗	✓	✗
PayFlow [15]	IoTA	✗	POX	✓	✗	✗	✗	✗	✗
Proposed	Ethereum	✓	POX	✓	✓	✓	✓	✓	✓

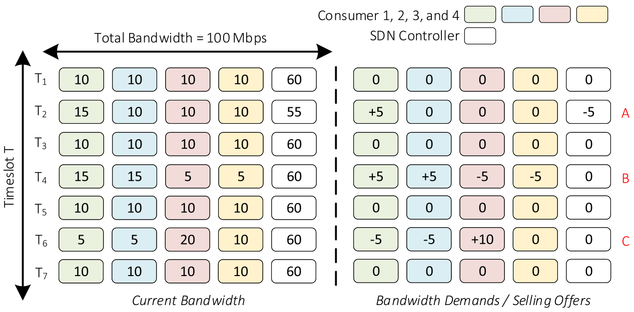


FIGURE 2. The bandwidth trading scenario depicting the states of current bandwidth (left) and demand/offer requests (right) from consumers and the SDN controller over a given period of timeslots. Pluses indicate bandwidth demands, while minuses refer to selling offers. A, B, and C indicate three cases of bandwidth trading.

B. TIMESLOT ALLOCATION

We design the bandwidth trading in a pre-order fashion in which all bandwidth demands and selling offers must be made before the actual trading time. For this reason, we introduce a timeslot $\mathcal{T}$ mechanism, which denoted as $\mathcal{T} = \{T_1, T_2, T_3, \ldots, T_n, \ldots, T_N\}$ with N is the total number of available timeslots. One timeslot refers to one period of time (e.g., one hour, half-hour, or 15 minutes). We present the following simulation scenario to describe bandwidth allocation per timeslot as illustrated in Figure 2.

We assume that the total shared bandwidth in the region is 100 Mbps. Four consumers (u_1, u_2, u_3, u_4) are in the region; each is assigned a default 10 Mbps bandwidth to access the Internet. Hence, the SDN controller has 60 Mbps bandwidth available in the reserve pool by default. Throughout this scenario, three bandwidth trading happens.

Case A on T_2 (One Buyer, One Seller): u_1 wants to buy 5 Mbps of additional bandwidth, so before T_2 ends, u_1 makes a bandwidth demand request. Unfortunately, no other consumers are willing to sell their bandwidth, so the SDN controller provides 5 Mpbs of bandwidth for u_1 from the reserve. During T_2, the SDN controller will temporarily increase the bandwidth capacity of u_1 from 10 to 15 Mbps, while the reserve is reduced from 60 to 55 Mbps.

Case B on T_4 (Two Buyers, Two Sellers): In this second case, two concurrent demands exist, u_1 and u_2 previously requested a 5 Mbps bandwidth in T_4. Then, u_3 and u_4 are willing to sell parts of their bandwidth to fulfill each demand. During T_4, the SDN controller will increase the bandwidth capacity of u_1 and u_2 from 10 to 15 Mbps temporarily, while reducing the bandwidth of u_3 and u_4 from 10 to 5 Mbps.

Case C on T_6 (One Buyer, Two Sellers): In this final scenario, we show two sellers collaborating to fulfill a demand request. Prior to T_6, u_3 requested a 10 Mbps bandwidth demand. Then, u_1 offered a 5 Mbps bandwidth to u_3's request. On the other hand, u_2 wants to sell all bandwidth to fulfill u_3 demand. However, because a 5 Mbps offer already exists from u_1, this 10 Mbps offer cannot be processed. Instead, u_2 must give a 5 Mbps offer to fulfill the demand. During T_6, the SDN controller will increase u_3's bandwidth capacity to 20 Mbps, while reducing the bandwidth of u_1 and u_2 to 5 Mbps each.

No trading: When there is no bandwidth trading, the default values are applied in each timeslot as shown in T_1, T_3, T_5, and T_7. Similarly, the bandwidth capacities are restored to their default values after each trading completed as shown in T_3, T_5, and T_7.

C. TRADING TABLE

Based on the previously mentioned scenario, we need to provide two items to facilitate reliable bandwidth trading. First, the controllers must know how much bandwidth they should provide to each consumer at each given timeslot. Second, we must record trading information on-chain to reap the integrity guarantee from the blockchain. However, because we expect $N \approx \infty$, maintaining $\mathcal{T}$ on-chain become costly. Instead of storing the amount of bandwidth that each consumer has at each timeslot (left side of Figure 2), we store the demand/offer requests at each timeslot (right side of Figure 2). This way, when there is no demand/offer, we do not store any value on the blockchain, resulting in zero values and saving many costs (Ethereum initiates integers as zeros by default). The amount of bandwidth the controller must provide to each consumer can still be calculated by adding or subtracting the current consumer bandwidth with the demand or offer at each timeslot.

IV. SYSTEM MODEL

Our proposed framework is shown in Figure 3, and Table 3 presents the description of important notations and variables in this paper.

A. REGISTRATION

We describe all preparations that must be made before performing the bandwidth trading.

1) HEADQUARTER SETUP

During network startup, headquarter administrators h first run the web server and the blockchain network. They then

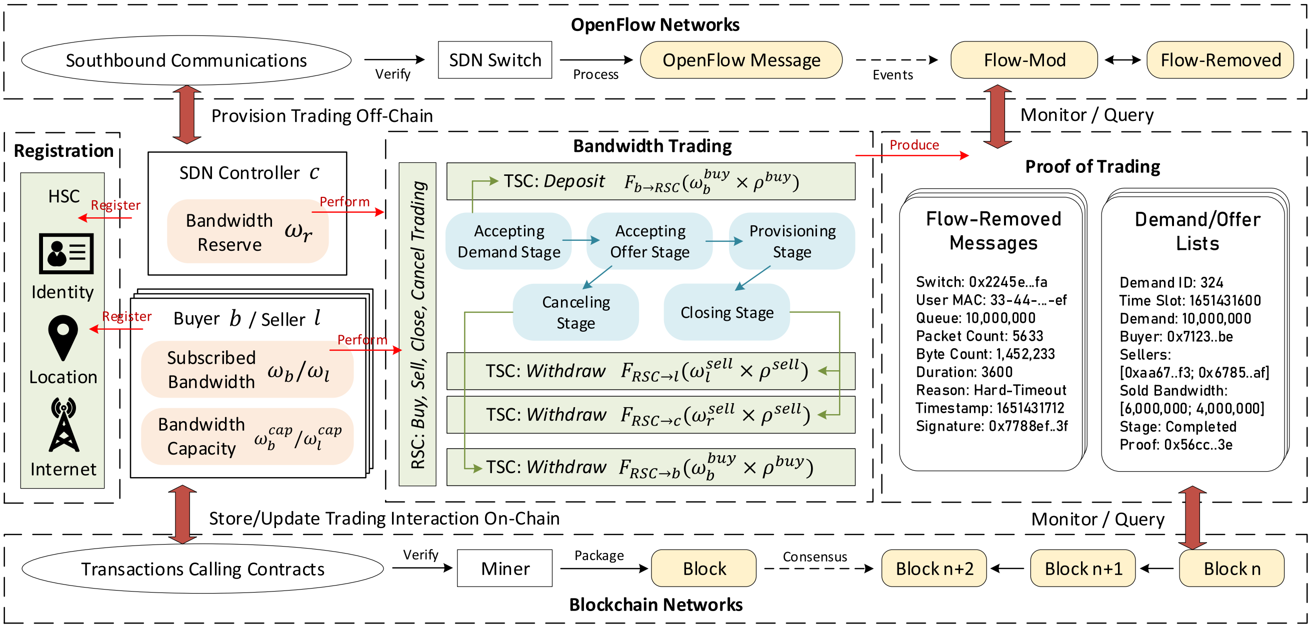


FIGURE 3. **The architecture of BLOCKBAND, which includes three main smart contracts. Headquarter Smart Contract (HSC) handles the registration of consumers and network devices, Regional Smart Contract (RSC) performs day-to-day operations such as bandwidth reserve allocation and bandwidth trading, and Token Smart Contract (TSC) manages the token distributions and payment-related services.**

TABLE 3. **List of notations used in the paper.**

Notation	Description
h / r	Headquarter admin / Regional admin
s / c	SDN switches / SDN controllers
u / l / b	Consumers / Sellers / Buyers
HSC	Headquarter smart contract
RSC	Regional smart contract
TSC	Token smart contract
α / μ	Blockchain address / MAC address
ω / ω^{cap}	Consumer bandwidth / bandwidth capacity
ω_r	Available bandwidth reserve
ω^{buy} / ω^{sell}	Bandwidth to buy / Bandwidth to sell
ρ^{buy} / ρ^{sell}	Buying price rate / Selling price rate
$\mathcal{T}$	Timeslots
$\mathcal{D}$ / $\mathcal{O}$	List of bandwidth demands / List of selling offers
g	Current stage of bandwidth demand
p	Proof of bandwidth demand provisioning
SK / PK	A secret key / public key pair
$SIGN_{SK}(J)$	Generates a digital signature for data J using private key SK
$VER_{\alpha}(J, K)$	Verifies whether the holder of blockchain address α signs data J and generates the digital signature K
$H(J)$	Generates a hash of data J
$X \parallel Y$	A concatenation of data X with data Y
$F_{a\rightarrow b}(c)$	Token transfers from a to b with c amount

create a new private SK_h and public key pair PK_h along with a blockchain address α_h. Once created, they can deploy the Headquarter Smart Contract HSC. This action will mark α_h as the owner of HSC and indirectly register h's identity in the blockchain.

The headquarter admins then begin adding SDN controllers c and SDN switches s that they have to HSC. Similar to h's registration, h first creates SK_c, PK_c, α_c, SK_s, PK_s, and α_s. They then only upload α_c and α_s to HSC, while keeping the rest in the secure storage of each entity.

Users u interested in the Internet service that h provide can subscribe to an Internet access plan to h's web server. They must first creates SK_u, PK_u and α_u. After that, they send α_u to the web server, which will be relayed to HSC.

2) REGIONAL SETUP

When creating a new region, regional administrators r first create SK_r, PK_r, and α_r. They then deploy the Regional Smart Contract RSC in the blockchain network. This deployment will indirectly assign α_r as the owner of RSC. After that, r report this newly created RSC to the headquarter by uploading α_{RSC} to HSC. Thus, we have a map between HSC and all deployed RSC.

Regional admins can set the SDN controller α_c and switch α_s that will be responsible to manage the region in RSC. Once configured, they also need to update the operational location of the controller and switch in HSC. This way, anyone can know where the controller and switch are currently provisioned. For simplicity, we assume that only one controller and switch can be assigned to one region at any time.

During the network startup, r configure u's network device to be workable in their region by allowing their MAC address μ_u to use the Internet. For residential and enterprise cases, r record μ_u of the modem placed in the users' domains. Meanwhile, r log the μ_u of the users' smartphones for the cellular case. Those μ_u are stored in c's local database, and tied to their corresponding α_u. After that, r officially assign α_u to the region by saving their information in RSC.

Algorithm 1 Accepting Demand Procedure in *RSC*

Input: $\omega_b^{buy}, T_n, \alpha_b, t_{now}$
Output: d

1: **if** $t_{now} > T_n$ **then** abort **end if**
2: **if** $\omega_b^{buy} + \omega_b > \omega_b^{cap}$ **then** abort **end if**
3: **if** $F_{b \rightarrow RSC}(\omega_b^{buy} \times \rho^{buy})$ **then**
4: set $g =$ `AcceptingOffer`
5: $d = \omega_b^{buy} \parallel T_n \parallel g \parallel \alpha_b$
6: save d in $\mathcal{D}$ **if and only if** $d \notin \mathcal{D}$, where $\mathcal{D}$ is a list of demands, $\mathcal{D} = \{1, 2, 3, \ldots, d, \ldots, D\}$ with D as the total number of demands.
7: **end if**

Then, r also update the location of consumer to *HSC*. One consumer can only be assigned to one region at any given time.

3) BANDWIDTH RESERVE ALLOCATION

Regional admins initially set the bandwidth reserve ω_r currently available in *RSC*. This ω_r value will be updated over time as consumers enter or exit the region.

When r add new user α_u in *RSC*, r also assign the consumable bandwidth for that user ω_u. Some amount of bandwidth then must be extracted from the bandwidth reserve pool to provision this user. In particular, *RSC* update ω_r as $\omega_r \leftarrow \omega_r - \omega_u$. If the resulting $\omega_r < 0$, we should reject u's assignment to this region because we cannot facilitate enough bandwidth for u. Furthermore, the assigned consumer bandwidth ω_u should not exceed the capacity of the physical link ω_u^{cap} that the consumer has.

In contrast, if r removes existing user α_u in *RSC*, the bandwidth reserve previously assigned to that user is restored to the pool. Specifically, *RSC* update ω_r as $\omega_r \leftarrow \omega_r + \omega_u$.

B. BANDWIDTH TRADING

We describe the bandwidth trading protocol that is performed in each region. The parameters and algorithms presented here are given as basic procedures. It can be further modified to match the use cases in each region when necessary.

1) STAGES OF TRADING

The overall trading is divided into five stages. First, consumers (acting as buyers) request a bandwidth demand. Other in-region consumers (willing to sell parts of their bandwidth) compete to fulfill the demand. The demand is considered valid when it finds enough selling offers to match the requested bandwidth. Otherwise, the demand will be canceled. Finally, the controller starts provisioning the trading of valid demands and then submits proof of trading when it ends.

Stage 1: *Accepting Demand Stage*

Buyers b initiate bandwidth demand requests to *RSC* by specifying how much bandwidth to buy ω_b^{buy} and the timeslot T_n, indicating what time they need the demand. The rest of the procedures is summarized in Algorithm 1.

Algorithm 2 Accepting Offer Procedure in *RSC*

Input: $\omega_l^{sell}, \omega_r^{sell}, d, \alpha_l, \alpha_c, t_{now}$
Output: o

1: **if** $d \notin \mathcal{D}$ **then** abort **end if**
2: **if** $t_{now} > T_n - t_{end}$ **then** abort **end if**
3: **for** l **do**
4: **if** $\omega_l^{sell} > \omega_l$ **then** abort **end if**
5: **if** $(\sum_{o \in \mathcal{O}} \omega_l^{sell} + \sum_{o \in \mathcal{O}} \omega_r^{sell}) + \omega_l^{sell} > \omega_b^{buy}$ **then** abort **end if**
6: $o = \omega_l^{sell} \parallel T_n \parallel \alpha_b \parallel \alpha_l$
7: **end for**
8: **for** c **do**
9: **if** $t_{now} < t_{start}$ **then** abort **end if**
10: **if** $\omega_r^{sell} > \omega_r$ **then** abort **end if**
11: **if** $(\sum_{o \in \mathcal{O}} \omega_l^{sell} + \sum_{o \in \mathcal{O}} \omega_r^{sell}) + \omega_r^{sell} > \omega_b^{buy}$ **then** abort **end if**
12: $o = \omega_r^{sell} \parallel T_n \parallel \alpha_b \parallel \alpha_c$
13: **end for**
14: save o in $\mathcal{O}$ **if and only if** $o \notin \mathcal{O}$, where $\mathcal{O}$ is a list of offers, $\mathcal{O} = \{1, 2, 3, \ldots, o, \ldots, O\}$ with O as the total number of offers.
15: **if** $\omega_b^{buy} = \sum_{o \in \mathcal{O}} \omega_l^{sell} + \sum_{o \in \mathcal{O}} \omega_r^{sell}$ **then** update $g =$ `Provisioning` **end if**

Buyers must order the demand ahead of time, so *RSC* makes sure that the request is made before the scheduled T_n (line 1). t_{now} is the current timestamp. *RSC* also validates if buyers request too much bandwidth; they cannot request bandwidth that exceeds the bandwidth capacity of their physical link ω_b^{cap} (line 2). Otherwise, exceeding bandwidth will be wasted. All invalid requests will be rejected.

Buyers must first make a deposit before *RSC* can accept their request (line 3). This way, we can ensure buyers have money to pay the sellers. The deposit amount depends on the buying price ρ^{buy}, which is rated per Kbps. More bandwidth to buy means more deposit is needed. *RSC* then create a stage indicator g and set it to `AcceptingOffer` stage. After that, it is saved together with ω_b^{buy}, T_n, and α_b to the blockchain (line 4-7).

Stage 2: *Accepting Offer Stage*

Sellers l and SDN controllers c can all simultaneously give selling offers to a given demand d. For sellers, they can mention how much bandwidth to sell ω_l^{sell} to *RSC*. For controller, c specify bandwidth reserve to sell ω_r^{sell} to *RSC*. The rest of the procedures is summarized in Algorithm 2.

RSC performs validations on the selling offer. First, *RSC* ensures that the demand exists (line 1). Request to non-existing demand is rejected. Second, *RSC* checks whether the offer is made within a valid time offer (line 2 and 9). All offers must be processed before a time limit t_{end}. Furthermore, our policy prioritizes offers from sellers rather than the controller. We give more chances (time window)

Algorithm 3 Provisioning Procedure in c

Input: $\mathcal{D}$, $\mathcal{O}$, t_{now}, T_n
Output: `FLOW-MOD` messages

1: **Before** T_n (at $T_n - t_{end} \leq t_{now} < T_n$):
2: form $\mathcal{D}'$, all $d \in \mathcal{D}$ that is scheduled at T_n
3: form $\mathcal{O}'$, all $o \in \mathcal{O}$ that satisfy $d' \in \mathcal{D}'$
4: **On** T_n (at $T_n \leq t_{now} < T_{n+1}$):
5: **for** $d' \in \mathcal{D}'$ **do**
6: $\alpha'_b, \omega_b^{buy'} \leftarrow d'$
7: get μ_b, ω_b from local database using α'_b
8: create M_b, `FLOW-MOD` messages with `ENQUEUE` for μ_b to increase $\omega_b \leftarrow \omega_b + \omega_b^{buy'}$
9: **end for**
10: **for** $o' \in \mathcal{O}'$ **do**
11: $\alpha'_l, \omega_l^{sell'}, \omega_r^{sell'} \leftarrow o'$
12: $\omega_r \leftarrow \omega_r - \omega_r^{sell'}$
13: get μ_l, ω_l from local database using α'_l
14: create M_l, `FLOW-MOD` messages with `ENQUEUE` for μ_l to reduce $\omega_l \leftarrow \omega_l - \omega_l^{sell'}$
15: **end for**
16: send M_b and M_l to s
17: **After** T_n ($t_{now} \geq T_{n+1}$):
18: **for** $d' \in \mathcal{D}'$ **do**
19: $\alpha'_b, \omega_b^{buy'} \leftarrow d'$
20: get μ_b, ω_b from local database using α'_b
21: create M'_b, `FLOW-MOD` messages with `ENQUEUE` for μ_b to decrease $\omega_b \leftarrow \omega_b - \omega_b^{buy'}$
22: **end for**
23: **for** $o' \in \mathcal{O}'$ **do**
24: $\alpha'_l, \omega_l^{sell'}, \omega_r^{sell'} \leftarrow o'$
25: $\omega_r \leftarrow \omega_r + \omega_r^{sell'}$
26: get μ_l, ω_l from local database using α'_l
27: create M'_l, `FLOW-MOD` messages with `ENQUEUE` for μ_l to increase $\omega_l \leftarrow \omega_l + \omega_l^{sell'}$
28: **end for**
29: send M'_b and M'_l to s

for sellers to give an offer at any time as long as it is not expired, while the controller can only give an offer after some delay t_{start}. Third, *RSC* ensures the bandwidth to sell does not exceed the sellers or reserve capacity (line 4 and 10). Fourth, several offers may contribute to a given demand, *RSC* must confirm that all offers from ω_l^{sell} or ω_r^{sell} do not exceed the demand (line 5 and 11).

RSC then saves ω_l^{sell} or ω_r^{sell} together with T_n, α_b, and (α_l or α_c) to the blockchain (line 6, 12, and 14). Finally, when all offers fulfil the demand, *RSC* set g to `Provisioning` stage (line 15).

Stage 3: *Provisioning Stage*

The provisioning stage is performed off-chain and divided into three parts: before, on, and after T_n as summarized in Algorithm 3.

Algorithm 4 Closing Procedure in *RSC*

Input: p, d, t_{now}
Output: updated d

1: **if** $t_{now} < T_{n+1}$ **then** abort **end if**
2: form $\mathcal{O}'$, all $o \in \mathcal{O}$ that satisfied d
3: **for** $o' \in \mathcal{O}'$ **do**
4: $\omega_l^{sell'}, \omega_r^{sell'} \leftarrow o'$
5: $F_{RSC \to l}(\omega_l^{sell'} \times \rho^{sell})$
6: $F_{RSC \to c}(\omega_r^{sell'} \times \rho^{sell})$
7: **end for**
8: $g \leftarrow d$
9: update $g =$ `Closed`
10: update $d \leftarrow d \parallel p$

Algorithm 5 Canceling Procedure in *RSC*

Input: d, t_{now}
Output: updated d

1: **if** $t_{now} < T_n$ **then** abort **end if**
2: $\omega_b^{buy}, g \leftarrow d$
3: **if** $g \neq$ `AcceptingOffer` **then** abort **end if**
4: **if** $F_{RSC \to b}(\omega_b^{buy} \times \rho^{buy})$ **then**
5: update $g =$ `Canceled`
6: **end if**

Before trading, the controllers get all demands scheduled at T_n (line 2). They also gather all corresponding offers that satisfy those demands (line 3).

During the trading period, the controllers retrieve consumers' MAC address μ by querying the local database based on given α (line 7, 13, 20, and 26). The controllers then form `FLOW-MOD` messages to increase buyers' bandwidth according to how much they buy (line 5-9). Similarly, they also form `FLOW-MOD` messages to decrease sellers' bandwidth based on how much they sell (line 10-15). Those messages are then delivered to the corresponding switches where the consumers are located (line 16). If the controllers contribute to selling demands, controllers must also reduce the bandwidth reserve based on how much they sell (line 12).

Once the trading duration is over, the controllers form `FLOW-MOD` messages to revert the bandwidth state of buyers and sellers (line 18-29). The controller reset the bandwidth reserve to the original values (line 25).

Stage 4: *Closing Stage*

After the trading completes, the controllers must provide proof p of its provisioning and close the demands. The proofs are formed off-chain using hashes of `FLOW-REMOVED` messages related to the previously transmitted `FLOW-MOD` messages in the previous stage. p is then included in the closing request submitted to *RSC*. The rest of the procedure is summarized in Algorithm 4.

RSC validates whether the request is made in the valid time window after the trading ends; premature closing requests are rejected (line 1). After that, *RSC* gets all previous offers that satisfied the to-be-closed demand and then transfers the rewards to their submitters (can be consumers or controllers). The rewards are calculated based on the selling price per Kbps ρ^{sell}. The more bandwidth the sellers sold, the more rewards they received (line 2-7). After a successful reward transfer, *RSC* officially sets the demand stage as `Closed` and saves the submitted proofs to the blockchain (line 8-10).

Stage 5: *Canceling Stage*

When demands cannot get enough offers during the offering window, the controllers can cancel those demands in *RSC*, as shown in Algorithm 5. The canceling request must be made within the valid time window (line 1) and in the right stage (line 3). At this time, the stage of demand can be in either `Provisioning` or `AcceptingOffer`. The former indicates that offers successfully fulfill a demand, while the latter indicates otherwise. Only unsatisfied demands can be canceled. If the request is valid, *RSC* returns the buyers' deposit (line 4) and sets the stage for this demand to `Canceled` (line 5).

2) INCENTIVES

We develop Token Smart Contract *TSC* in the form of ERC20 smart contract [26] to facilitate the incentives for bandwidth trading. Only one instance of *TSC* will be deployed in the blockchain network, and its usage is shared among *RSC* in all regions.

Consumers can mint the token by exchanging liquid assets such as US dollars or other valuable currencies in the ISP. The token then can be used to purchase various ISP-related services, for example, Internet service bills, bandwidth demand, public IP, cloud storage, and hosting. Consumers can also burn the token to obtain the liquid assets back.

During the trading, consumers must pay a deposit based on the buying price rate ρ^{buy}. Meanwhile, sellers obtain rewards for their offers based on the selling price rate ρ^{sell}. Ideally, the ISP will configure $\rho^{buy} > \rho^{sell}$ to earn profits. The bigger the gap, the more tokens the ISP will get. Furthermore, the more bandwidth (per Kbps) is traded, the more profits the ISP will earn. The total rewards that ISP can get for a given demand can be calculated as follows.

$$R = \left(\omega_b^{buy} \times \rho^{buy} - \sum_{o' \in \mathcal{O}'} \omega_l^{sell} \times \rho^{sell}\right) + \omega_r^{sell} \times \rho^{sell} \tag{1}$$

$\mathcal{O}'$ is the list of consumer offers that satisfy the given demand. Note that ISP can get more profits if ISP also offers to sell bandwidth reserve ω_r^{sell} in the given demand.

C. PROOF OF TRADING

The proofs of bandwidth trading provisioning are collected off-chain and on-chain in the form of `FLOW-REMOVED` messages for the former case and a list of demands/offers for the latter case.

TABLE 4. List of metadata stored on-chain.

Category	Contract	Metadata
Identity	HSC	$\alpha_b, \alpha_l, \alpha_s, \alpha_c, \alpha_{RSC}$
Location	HSC	$\{\alpha_b, \alpha_{RSC}\}, \{\alpha_l, \alpha_{RSC}\}, \{\alpha_c, \alpha_{RSC}\}, \{\alpha_s, \alpha_{RSC}\}$
Bandwidth	RSC	$\omega_b, \omega_b^{cap}, \omega_l, \omega_l^{cap}, \omega_r$
Trading	RSC	$\omega_b^{buy}, \omega_l^{sell}, \omega_r^{sell}, T_n, p$
Incentive	TSC	$\omega_b^{buy} \times \rho^{buy}, \omega_l^{sell} \times \rho^{sell}, \omega_r^{sell} \times \rho^{sell}$

1) OFF-CHAIN PROOF

Proving that controllers have provisioned trading correctly: When receiving `FLOW-MOD` messages from controllers, SDN switches create new Flow Table entries (Flow Rules) and begin updating their statistics whenever they route packets using those rules. Those statistics are later sent back to the controllers when the rules expire via `FLOW-REMOVED` messages. From these reported messages, we can check whether the controller previously sent `FLOW-MOD` messages with valid parameters (e.g., MAC address, Queue value, timeout duration) correspond to the bandwidth demand/offer requests.

The switch must include the current timestamp t_1^{expire} to record the time when the Flow Rule expires and sign it together with the `FLOW-REMOVED` message M_1 as follows.

$$\begin{aligned} C_1 &= SIGN_{SK_s}(M_1 \parallel t_1^{expire}) \\ X_1 &= C_1 \parallel M_1 \parallel t_1^{expire} \end{aligned} \tag{2}$$

The switch then sends X_1 to the controllers.

The expiry timestamp can be used to estimate when the Flow Rule was inserted by subtracting t_1^{expire} with `HARD-TIMEOUT` duration

$$t_1^{start} \approx t_1^{expire} - \texttt{HARD-TIMEOUT} \tag{3}$$

The trading is considered valid if $|t_1^{start} - T_n| \approx 0$, with error tolerance level in seconds. A smaller gap indicates that the `FLOW-MOD` messages are sent right after the trading start, which is the ideal case. The difference cannot be zero because we must consider the network delay for transmitting `FLOW-MOD` messages from controllers to switches and the asynchronous time between the switch's clock and the controllers' clock.

The controllers may receive several `FLOW-REMOVED` messages for one trading (e.g., if we have multiple sellers for one demand). Once all expected messages are received, the controllers hash them together.

$$p = H(X_1 \parallel X_2 \parallel X_3 \parallel \ldots \parallel X_P) \tag{4}$$

X_1, X_2, X_3 refers to the first three `FLOW-REMOVED` messages. P is the total number of received messages. The controllers must save these messages in a secure permanent storage off-chain. They will be used to prove their behaviors when challenged by an untrusted party in the future. Meanwhile, the hash p is submitted to *RSC* during the `Closing` stage of bandwidth trading.

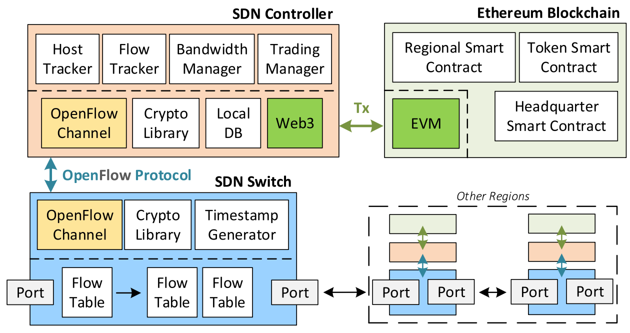


FIGURE 4. **The implementation of** BLOCKBAND **in our testbed.**

2) ON-CHAIN PROOF

Proving that trading interactions happen: We can leverage the on-chain metadata in the smart contracts (c.f., Table 4) to prove that trading interaction happens between buyers, sellers, and controllers.

First of all, we make sure that entities are registered. Second, we verify that they reside in the same region by querying the location mapping. Third, we can check the bandwidth information of involved entities to spot abnormalities, for example, buying or selling excessive bandwidth. Fourth, we can check the trading logs from the lists of demands and offers to see who initiates the demand request, who the sellers are, how much bandwidth they buy/sell, when the trading occurs, and what is the hash p to prove the FLOW-REMOVED messages. Finally, we validate whether the rewards have been distributed correctly.

Because sending data to the smart contracts requires senders to sign the transactions, once recorded in the blockchain, each involved party cannot repudiate their contributions to the trading.

3) COMPLEMENTARY PROOFS

Proving that consumers exist in the region: In daily network operational, u's Internet access may trigger PACKET-IN messages from s. Inside those messages, μ_u will be recorded, and c can then extract α_u from μ_u (recall that the mapping between α_u and μ_u is stored in the local database during the user assignment). After that, c can look up the policy in *RSC*, whether to allow or reject Internet access for α_u. This way, only authenticated consumers can use services in the region.

Proving off-chain communications: All entities can leverage their registered address in *HSC* to authenticate themselves off-chain by signing particular messages M with the associated secret key. For example, $S_{owner} = SIGN_{SK_{owner}}(M)$. The recipient can verify if the sender is truly *owner* by making sure that the resulting signature S_{owner} is valid, which is $VER_{\alpha_{owner}}(M, S_{owner})$ equals True. This way, the integrity of the off-chain communications can be preserved, and we also know that the off-chain sender is the same entity as the one on the on-chain.

TABLE 5. **The sizes of all deployed smart contracts in** BLOCKBAND**. We assume the smart contract size limit is 24 KB.**

Name	Type	Used For	Size (in KB)	% Limit
Registry	Contract	*HSC*	9.56	39.83
ConsumerStorage	Contract	*HSC*	2.08	8.67
ControllerStorage	Contract	*HSC*	2.08	8.67
SwitchStorage	Contract	*HSC*	2.08	8.67
IRegistry	Interface	*HSC*	0	0.00
IRegistryStorage	Interface	*HSC*	0	0.00
ERC20	Contract	*TSC*	6.79	28.29
IERC20	Interface	*TSC*	0	0.00
Regional	Contract	*RSC*	19.82	82.58
BandwidthManager	Contract	*RSC*	1.25	5.21
ConsumerManager	Contract	*RSC*	5.46	22.75
DeviceManager	Contract	*RSC*	1.63	6.79
TradingManager	Contract	*RSC*	9.6	40.00
TradingParam	Contract	*RSC*	0.06	0.25
IRegional	Interface	*RSC*	0	0.00
SafeMath	Library	All	0.08	0.33

V. EXPERIMENTAL RESULTS

Figure 4 shows BLOCKBAND's software architecture operating in one region. The SDN switch communicates with the SDN controller through the OpenFlow protocol. Meanwhile, the SDN controller accesses the smart contracts by sending transactions (Tx) to the Ethereum blockchain network. The experiment is performed in hardware with the following specification: Intel Core i7-10700K CPU @ 3.80 GHz and Samsung DIMM @ 2667MHz RAM.

A. ON-CHAIN EVALUATION

We first evaluate our smart contract implementations as one way to analyze blockchain-related performance.

Setup: We build a docker container utilizing 1 core of CPU and 1 GB of RAM to run Ganache [27], a simulated local Ethereum testbed. The smart contract is written in Solidity language and is deployed to the Ganache using Truffle JS [28]. We divide the implementation of *HSC* and *RSC* into multiple child smart contracts for (i) improving code readability, (ii) reducing byte size per contract, and (iii) allowing us to upgrade the contract when necessary. The list of all deployed contracts is shown in Table 5.

1) CONTRACT SIZE

Ethereum network prohibits developers from deploying smart contracts beyond 24 KB [29]. Therefore, a feasible contract must stay below that bound. Table 5 shows that all of our contracts are within that boundary and, therefore, should be possible to be deployed in the Ethereum network. Interfaces cannot have implemented methods, so they do not affect contract size, resulting in zero values.

2) GAS CONSUMPTION

All Ethereum smart contract executions that modify the blockchain network state are subject to a unit called "gas". The more complex the smart contract methods become, the more gas is required to execute them. Table 6 shows gas consumptions of all writable methods in our proposal. Read functions are free and do not require gas; hence, we do not include them in the list.

TABLE 6. List of writable smart contract methods and their gas consumption in BLOCKBAND. We assume the block limit of 30 million gas. The estimated throughput in transactions per second (TPS) is calculated based on the block interval in Mainnet, Kovan, and Klaytn networks.

Case / Description	Callers	Contracts	Methods	Gas Usage	% Limit	Throughput (TPS)		
						Mainnet	Kovan	Klaytn
Case I: Initiation in headquarters								
Deploy main registry contract	h	Registry	Deploy	2,178,371	7.26	1.06	3.44	13.77
Deploy child contract for controller storage	h	CS	Deploy	535,473	1.78	4.31	14.01	56.03
Deploy child contract for switch storage	h	SS	Deploy	535,473	1.78	4.31	14.01	56.03
Deploy child contract for consumer storage	h	US	Deploy	535,461	1.78	4.31	14.01	56.03
Link controller storage to main contract	h	Registry	SetControllerStorage	43,412	0.14	53.16	172.76	691.05
Link switch storage to main contract	h	Registry	SetSwitchStorage	43,454	0.14	53.11	172.60	690.39
Link consumer storage to main contract	h	Registry	SetConsumerStorage	43,410	0.14	53.16	172.77	691.09
Deploy main ERC20 contract	h	ERC20	Deploy	1,618,405	5.39	1.43	4.63	18.54
			Total	**5,533,459**	**18.44**	**0.42**	**1.36**	**5.42**
Case II: Initiation in regionals								
Deploy main regional contract	r	Regional	Deploy	4,492,269	14.97	0.51	1.67	6.68
Deploy child contract to manage bandwidth	r	BM	Deploy	370,432	1.23	6.23	20.25	80.99
Deploy child contract to manage device	r	DM	Deploy	435,450	1.45	5.30	17.22	68.89
Deploy child contract to manage consumer	r	CM	Deploy	1,275,504	4.25	1.81	5.88	23.52
Deploy child contract to manage trading	r	TM	Deploy	2,285,658	7.62	1.01	3.28	13.13
Link bandwidth manager to main contract	r	Regional	SetBandwidthManager	43,472	0.14	53.08	172.52	690.10
Link device manager to main contract	r	Regional	SetDeviceManager	43,505	0.15	53.04	172.39	689.58
Link consumer manager to main contract	r	Regional	SetConsumerManager	43,506	0.15	53.04	172.39	689.56
Link trading manager to main contract	r	Regional	SetTradingManager	43,528	0.15	53.02	172.30	689.21
Link regional to registry contract	h	Registry	AddRegional	44,597	0.15	51.75	168.17	672.69
			Total	**9,077,921**	**30.26**	**0.25**	**0.83**	**3.30**
Case III: Registering new SDN controller								
Add new SDN controller to registry	h	Registry	AddController	50,436	0.17	45.75	148.70	594.81
Set operational location of SDN controller	h	Registry	SetControllerLocation	41,017	0.14	56.26	182.85	731.40
Assign SDN controller to regional	r	Regional	SetController	57,121	0.19	40.40	131.30	525.20
			Total	**148,574**	**0.50**	**15.53**	**50.48**	**201.92**
Case IV: Registering new SDN switch								
Add new SDN switch to registry	h	Registry	AddSwitch	50,347	0.17	45.84	148.97	595.86
Set operational location of SDN switch	h	Registry	SetSwitchLocation	41,008	0.14	56.27	182.89	731.56
Assign SDN switch to regional	r	Regional	SetSwitch	56,991	0.19	40.49	131.60	526.40
			Total	**148,346**	**0.49**	**15.56**	**50.56**	**202.23**
Case V: Registering new consumer								
Add new consumer to registry	h	Registry	AddConsumer	50,426	0.17	45.76	148.73	594.93
Set operational location of consumer	h	Registry	SetConsumerLocation	40,920	0.14	56.40	183.28	733.14
Mint some tokens to consumer	h	ERC20	Mint	54,461	0.18	42.37	137.71	550.85
Assign consumer to regional	c	Regional	AddConsumer	119,945	0.40	19.24	62.53	250.11
			Total	**265,752**	**0.89**	**8.68**	**28.22**	**112.89**
Case VI: Buying bandwidth demand								
Request a bandwidth demand	u	Regional	BuyBandwidth	133,825	0.45	17.24	56.04	224.17
Deposit token as rewards for sellers	u	ERC20	Deposit	81,212	0.27	28.42	92.35	369.40
			Total	**215,037**	**0.72**	**10.73**	**34.88**	**139.51**
Case VII: Selling of consumer bandwidth								
Give a selling offer to a bandwidth demand	u	Regional	SellBandwidth	151,031	0.50	15.28	49.66	198.63
Add allowance to withdraw the deposit	u	ERC20	IncreaseAllowance	47,248	0.16	48.84	158.74	634.95
			Total	**198,279**	**0.66**	**11.64**	**37.83**	**151.30**
Case VIII: Selling of bandwidth reserve								
Give a selling offer to a bandwidth demand	c	Regional	SellReserve	135,473	0.45	17.03	55.36	221.45
Add allowance to withdraw the deposit	c	ERC20	IncreaseAllowance	47,248	0.16	48.84	158.74	634.95
			Total	**182,721**	**0.61**	**12.63**	**41.05**	**164.18**
Case IX: Finalizing bandwidth trading								
Close a bandwidth trading	c	Regional	CloseTrading	48,254	0.16	47.82	155.43	621.71
Withdraw the deposit to seller's account	c	ERC20	Withdraw	67,285	0.22	34.30	111.47	445.86
Remove allowance to withdraw the deposit	c	ERC20	DecreaseAllowance	47,257	0.16	48.83	158.71	634.83
			Total	**162,796**	**0.54**	**14.18**	**46.07**	**184.28**
Case X: Aborting bandwidth trading								
Cancel a bandwidth trading	c	Regional	CancelTrading	43,511	0.15	53.04	172.37	689.48
Withdraw the deposit to buyer's account	c	ERC20	Withdraw	67,285	0.22	34.30	111.47	445.86
			Total	**110,796**	**0.37**	**20.83**	**67.69**	**270.77**

h: Headquarter Admin, r: Regional Admin, c: SDN Controller, u: Consumer
CS: Controller Storage, SS: Switch Storage, US: Consumer Storage
BM: Bandwidth Manager, DM: Device Manager, CM: Consumer Manager, TM: Trading Manager

As shown in the table, all our implemented methods are below the Ethereum gas limit standard of 30 million per block [30]. This indicates that running them in Ethereum networks is feasible. The rest of the gas consumption can then be analyzed case by case as follows.

Case I and II: The contract deployment cases are expected to be the most expensive of all cases. During deployment, we need to store the bytes of smart contracts in the blockchain. Storing data in the blockchain is the smart contract's most expensive operation. From Table 5, we can see that *RSC* implementation has more bytes than *HSC* and *TSC*. Consequently, *RSC* consumes a lot of gas when deployed. The Storage and Manager contracts are the child contracts for *HSC* and *RSC*. When deploying child contracts, we must link them to the main contract using the SetXStorage(·) and SetXManager(·) methods; replace *X* with the name of child contracts. These additional method calls increase the overall gas usage during the initiation cases.

Case III, IV, and V: The gas consumptions for registering controllers and switches produce a small gap since they perform similar tasks: adding their addresses in the contract and setting their respective operational locations. On the contrary, consumer registrations take more gas because we include additional processing such as minting the tokens (used later as deposits for requesting bandwidth demands) and storing the bandwidth information (e.g., bandwidth capacity link and eligible bandwidth that the consumer can use in the region).

Case VI, VII and VIII: Buying a bandwidth demand generates a considerable amount of gas because we must save the request log in the blockchain for auditing. Similarly, consumers and controllers also store their selling information in the contract. Those savings require some amount of gas usage.

Selling scenarios produces a marginally lower gas consumption than buying cases, with an 8% difference in selling consumer bandwidth cases and a 16% gap in selling bandwidth reserve cases. The latter is slightly cheaper than the former because we only have one controller per region while having multiple consumers per region. Therefore, we use arrays to track data for many consumers. Implementing arrays result in more gas usage.

Finally, it is also worth noting that buyers and sellers consume a slightly similar amount of total gas usage when buying and selling bandwidth. This indicates an excellent balance, where no party should feel at a disadvantage over the others because they all contribute about the same amount of work on-chain.

Case IX and X: Closing bandwidth trading includes storing a 32-byte hash proof of the `FLOW-REMOVED` messages to the smart contract, which results in more gas usage compared to the canceling scenario.

3) TRANSACTION THROUGHPUT

The blockchain throughput can be calculated as how many transactions the network can process per second (TPS). This metric depends on two factors: (i) how many transactions can be included in the block, which corresponds to the gas limit per block g_{limit} (in the Ethereum case), and (ii) how long it takes to generate one block (a.k.a., block interval $b_{interval}$). With the results of the gas usage g_{usage} per method from our experiments, we can estimate the projected TPS using the following formula

$$tps = (g_{limit}/g_{usage})/b_{interval} \tag{5}$$

Block intervals vary among different blockchain consensus algorithms. We consider three networks to measure the throughput: Mainnet (Ethereum main network using PoW [31]), Kovan Testnet (Ethereum test network using PoA [32]), and Klaytn (private Ethereum network using PBFT [8]). Mainnet process one block every 13 seconds [33], Kovan Testnet can do it in four seconds [34], while Klaytn can form a block within a second [35]. The lower the block interval, the higher the throughput becomes. We summarize the throughput results in Table 6. We assume that one block only contains transactions from the same methods.

The initiation cases (Case I and II) are very slow due to the enormous gas required to process those deployment methods. Fortunately, those initiations only happen once in a lifetime. Therefore, even the slowest value (i.e., 0.25 TPS value in Case II) is still acceptable; we can still process it in about 4 minutes. The registration cases (Case III to V) are relatively more frequent than initiation cases, but it happens only once per registered instance. After all required entities are registered, calls to these methods are reduced drastically. In contrast, we will frequently execute the bandwidth trading operations (Case VI to X). Therefore, they need to be executed as fast as possible.

We employed some restrictions in the bandwidth trading protocol to limit the possibility of entities abusing the system by submitting too many requests. First, one buyer can only request a demand once per timeslot. With a one-hour timeslot, we can provision at most 36,000 user requests per timeslot using Mainnet (assuming all requests are buying requests). This corresponds to at most 125,000 and 500,000 requests in Kovan and Klaytn, respectively.

Second, one seller can only offer a sale once per demand. Furthermore, consumers and controllers can sell only a finite amount of bandwidth depending on their current bandwidth capacity. Once they sell all of their bandwidth, they cannot make more offers; thus, flooding the network with selling requests is impossible. Assuming that one seller satisfies one demand request, we can provision at most 42,000 selling requests in a given timeslot for Mainnet, which corresponds to 136,000 and 545,000 requests for Kovan and Klaytn.

Third, the closing and canceling of bandwidth only happen once per demand. Therefore, these scenarios share the same arguments as in the buying case. We can provision at most 51,000 closing and 75,000 canceling of demand in a given timeslot for Mainnet, with 165,000 and 243,000 for Kovan, 663,000, and 975,000 for Klaytn.

Moreover, the fact that we process the trading in a "pre-order" fashion further alleviates the expected throughput

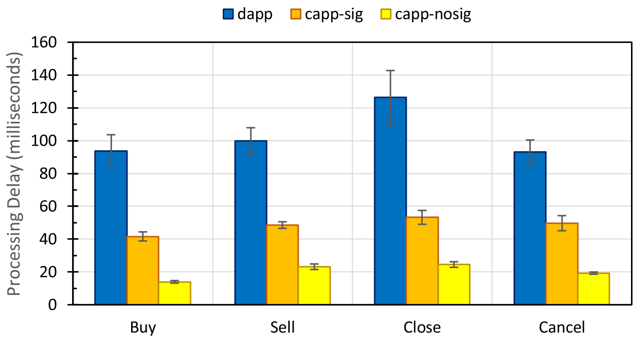


FIGURE 5. **The processing delay of performing bandwidth trading in buy, sell, close, and cancel scenarios when implemented as a decentralized application (dapp), centralized application with ECDSA signature (capp-sig), and centralized application without signature (capp-nosig).**

burden during peak hours. For example, many users are expected to request bandwidth during peak hours. However, because buyers must pre-order the demand, the buying request to the desired demand must be performed before the peak hours. Similarly, the selling offer also must be made beforehand. Therefore, the trading requests should spread in non-peak hours instead of being saturated in peak hours.

In production cases, transactions in one block may originate from multiple methods. Therefore, we should consider the given throughput from Table 6 as upper bound values. The lower bound values are zero because the inclusion of the submitted transactions to the block is subject to the miner's decision. If the miner is unwilling to include the transaction in the block (e.g., censorship), then our method will never be executed.

4) PROCESSING DELAYS PER STAGE

Setup: We build a docker container using 1 CPU and 1 GB of RAM to implement our bandwidth trading applications. Specifically, we develop a decentralized application (dapp) and centralized applications (capp) for comparisons; they are all implemented in Node JS. The dapp is connected to the Ganache network using Web3 JS [36]. Meanwhile, we use Express JS [37] to implement the conventional REST API-based server for capp. Because client requests in dapp are submitted to the blockchain network in the form of signed transactions, we build a digital signature variant of capp as a comparison to further analyze the overhead of our dapp. The signature in capp is implemented using the same Web3 JS library as in dapp. After that, we run procedures for buyers, sellers, or SDN controllers to request the buy, sell, close, or cancel operations to our dapp and capp. We run our simulations for 50 iterations and measure the processing delay for each scenario as shown in Figure 5.

Results: The processing delays of centralized applications without any signature (capp-nosig) become our basis for comparisons. They represent how we usually build a web-based application in a centralized environment. Adding a digital signature to capp increases the delay by up to 2.7× higher on average, while the dapp implementation slows the process even more by up to 8× slower on average compared to capp.

While dapp is the slowest among all, it is the most secure implementation with the highest integrity. Trading requests in dapp must be made through signed transactions. Afterward, the blockchain nodes verify the transactions before submitting blocks to the network. Once submitted, they must wait for the Ethereum Virtual Machine (EVM) to process the transactions and wait for the block consensus before sending responses to clients confirming that the trading has been accepted. Those procedures contribute to the high integrity of dapp implementation while also increasing the processing delay.

It is worth noting that the EVM process and consensus take about 5× more delays in our experiments. Thus, we can confirm that they are the main bottleneck in our system. Furthermore, all of our experiments are performed locally in a simulated testbed. Hence, they produce the ideal scenarios. In production cases, the real-world network latency can further increase the processing delay.

B. OFF-CHAIN EVALUATION

In this second part of our evaluation, we discuss parts of our proposal that does not relate to the blockchain.

1) PROVISIONING OF BANDWIDTH TRADING IN SDN TESTBED

Setup: We build a virtual machine (VM) with the specification of 4 cores of CPU and 4 GB of RAM to run Mininet [38] as our SDN testbed. The Open vSwitch is used as an SDN switch, and we leverage `ovs-vsctl` [39] to configure rate limiting on the switch to simulate our bandwidth trading. Furthermore, we create two applications on POX [40] SDN controller. One is to implement a minimal Layer 2 switching (L2 Switch), which will reactively install Flow Rules with `ENQUEUE` property to limit default traffic. Another application (`BLOCKBAND`) is used to implement bandwidth trading, which installs Flow Rules proactively according to the demand/offer lists.

We simulate scenarios from Figure 2 in our testbed by arranging four hosts (i.e., Consumer 1-4) connected to a single edge switch (because they should exist in the same region). We then build a dummy host to run `iperf` [41] from outside the region towards all consumers to monitor their bandwidth capacity. We run the experiment for 700 seconds with one timeslot equal to 100 seconds, resulting in 7 timeslots.

Results: Figure 6 shows the bandwidth measurement from `iperf` for each hosts every 10 seconds. We confirm that our deployed applications can dynamically provision bandwidth trading. Furthermore, the average bandwidth throughput on each timeslot (c.f. Table 7) shows that our approach can adjust the bandwidth capacity for each consumer according to the demands/offers list. All three bandwidth trading scenarios from Figure 2 can be performed correctly. While it is expected that the `iperf` bandwidth measurements fluctuates at some

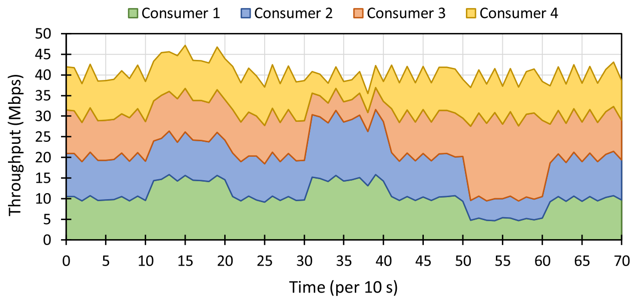


FIGURE 6. **The** `iperf` **throughput results based on the trading scenarios in Figure 2. The timeslot is set to 100 seconds (s).**

TABLE 7. **The average** `iperf` **throughput measurements (in bit per seconds) per timeslot T_n from Figure 6, following the scenarios in Figure 2. Bold numbers indicate the results of bandwidth trading.**

T_n	Consumer 1	Consumer 2	Consumer 3	Consumer 4
T_1	10,031,887	10,026,001	10,041,493	10,012,323
T_2	**14,816,290**	**9,916,125**	**9,967,082**	**9,975,748**
T_3	9,957,897	9,939,863	9,958,115	9,961,053
T_4	**14,718,219**	**14,747,759**	**5,023,272**	**5,030,176**
T_5	10,118,744	10,162,216	10,097,512	10,099,451
T_6	**5,028,051**	**5,021,792**	**19,395,607**	**9,964,691**
T_7	9,982,702	10,027,142	10,057,809	10,008,602

point, the trends from Figure 6 is close enough to represent scenarios in Figure 2.

It is also worth noting that we perform the bandwidth trading seamlessly, and there are no packet drops in the `iperf` traffics throughout our testings. We achieve this condition by requiring L2 Switch applications to first install default 10 Mbps Flow Rules with a lower priority. Meanwhile, `BLOCKBAND` applications later install higher priority buying/selling Flow Rules according to the demands/offers list. This way, we make sure that the switch always has Flow Rules to route `iperf` traffics and prevent `PACKET-IN` generation while testing (except for the first packet of `iperf` traffics at the beginning). This feature highlights that our proposal can be performed without interrupting any running applications from the consumers' end.

2) ADDITIONAL OVERHEAD IN SDN SWITCH

Setup: The vanilla Open vSwitch does not have modules to perform cryptography and timestamp clock. Therefore, for rapid prototyping, instead of modifying the switch code base, we develop an internal TCP proxy server that intercepts the messages coming from Open vSwitch and relays them to the SDN controller. The proxy must sign `PACKET-IN`, `FLOW-REMOVED` messages, as well as QoS configuration messages from `ovs-vsctl` for proofs of trading. The proxy also adds a timestamp to the original message. We run our simulation for 100 iterations in three scenarios: (i) no signature in which we only apply timestamp, (ii) HMAC signature + timestamp, and (iii) ECDSA signature + timestamp. Figure 7 summarizes our results.

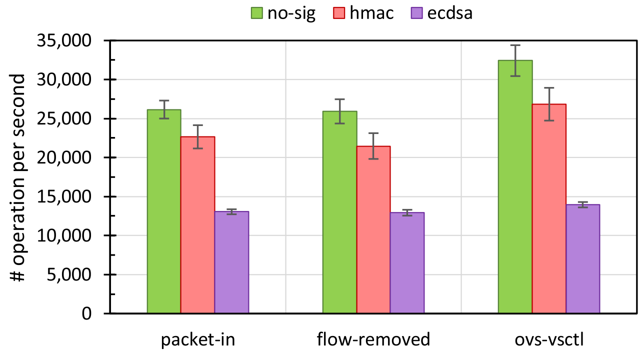


FIGURE 7. **The expected additional overhead in SDN switch when performing** `PACKET-IN`, `FLOW-REMOVED`, **and** `ovs-vsctl` **processes to generate Proof of Trading. The results are measured in three scenarios: without signature (no-sig), using HMAC, or ECDSA algorithm.**

Results: It is evident that performing cryptographic signatures on transmitted OpenFlow or Open vSwitch messages improves the messages' integrity but reduces the overall performance. From our test, we can measure the performance drop of up to 17% if using HMAC and 56% when using the ECDSA algorithm. ECDSA provides a stronger non-repudiation guarantee than HMAC with the cost of more complex processing. The choice of which algorithm to pick can differ depending on each use case, and our result can give insights into such trade-offs.

While the performance overhead seems high, it is essential to point out that `PACKET-IN`, `FLOW-REMOVED`, and `ovs-vsctl` messages are generated only for short durations at a given timeslot. The `PACKET-IN` is generated only at the beginning of the timeslot for authentication, while `FLOW-REMOVED` and `ovs-vsctl` are used only a few times at the closing stages. Therefore, we argue that those overheads should not heavily impact the switch's day-to-day operations.

C. SECURITY AND FAIRNESS

In the final part of our evaluation, we assess the security and fairness of our bandwidth trading.

Assumptions: We assume buyers, sellers, and SDN controllers can lie about their info by providing fake inputs to the system when performing bandwidth trading. They can also repudiate their sent messages and deny their involvement in the trading. Furthermore, we assume that the blockchain network is always secure such that there is no 51% or eclipse attack on the network and that the smart contract execution is always deterministic.

The followings are several design decisions made in the paper to guarantee the security and fairness of our proposed bandwidth trading.

Sellers cannot fabricate fake bandwidth information. For example, sellers cannot create fake selling requests to sell 10 Mbps if they only own 5 Mbps. This is because all of the consumers' available bandwidth is recorded in the smart contract, and the contract will accept or reject selling offers according to recorded data.

Buyers cannot request bandwidth demand without deposits. We mandate that all buyers deposit their tokens during the demand requests to prove that they have the money to reward sellers. This deposit will be locked in the smart contract and cannot be spent for other uses outside bandwidth tradings. This way, we can guarantee that we will have enough funds to compensate the sellers fairly.

Sellers and buyers cannot cancel their requests. We made our bandwidth trading irrevocable, meaning consumers cannot cancel their demands/offers once submitted to the blockchain. This policy ensures that consumers cannot create ''fake promises'' by temporarily committing to buying/selling bandwidth and canceling their requests near the trading deadline. Such actions will be unfair to other consumers and may prevent demands and offers from reaching equilibrium if abused by adversaries.

SDN controllers must submit proof of trading to close the demands and get incentives. SDN controllers must reveal the hash of `FLOW-REMOVED` messages from SDN switches to prove that they have provisioned the trading correctly. The controllers can retrieve the trading incentives only after the proof is submitted. This way, we enforce accountability to the SDN controllers. If we find SDN controller behavior frauds, we can use the previously submitted proof of trading to judge the controllers.

All of the trading interactions must be negotiated on-chain. Aside from the bandwidth provisioning through OpenFlow messages, all other trading operations must be performed on-chain. All demands and offers are made through signed blockchain transactions. Thus, participants cannot repudiate their involvement in the tradings. Furthermore, the smart contract runs the trading settlement through the EVM, which has a high integrity factor such that no party can censor/manipulate the trading process.

VI. DISCUSSIONS AND FUTURE WORKS

This section discusses the limitations of our proposal, which also indicate the area of improvement for future works.

A. DATA LEAKAGE

While blockchain maintains privacy-preserving property by anonymizing all involved entities, blockchain nodes may still gain useful insights from the publicly stored data. In our case, all information from Table 4 will be visible to nodes. They can then get an approximation of how many consumers are in a region, how much bandwidth is reserved for spotting rich/poor neighborhoods, and even deduce whether consumers are currently at home/office by inspecting the demand requests. Solutions to this problem include applying a permissioned blockchain that can restrict access to the blockchain network only to authorized entities. Therefore, adversaries will not be included as blockchain nodes. Other solutions include improving the privacy-preserving aspect of the blockchain network itself, which is considered an ongoing research agenda.

B. LOG CAN BE MISSING WHEN SDN SWITCHES FAIL

If the switches fail during the provisioning, they may lose Flow Table states. Therefore, `FLOW-REMOVED` messages will not be generated, and the controllers cannot generate proof-of-tradings. To reduce the impact of this issue, the controllers can divide the schedule into smaller timeslot durations (e.g., in order of minutes instead of hours). Therefore, minimizing the impact of information loss. Additionally, controllers should also include the proof-of-switch-failures as complementary proof-of-trading when such failures happen during the provisioning of bandwidth trading. This way, validators can be notified that the proof for this demand is unavailable because of a legitimate switch failure.

C. REQUIRE MODIFICATIONS ON SDN SWITCHES AND SDN CONTROLLERS HARDWARE

The signatures and expiry timestamp on `FLOW-REMOVED` messages allow customers to judge whether ISP performs bandwidth trading correctly. For this reason, we need alterations on the SDN switch hardware to add custom modules to sign OpenFlow messages before sending them to the controller. Furthermore, reliable timestamp generators should also be present in the switch, for example, using Trusted Execution Environment (TEE). On the other hand, the SDN controller needs to provide additional storages to store those signatures and messages for proof of trading. The longer the controller stores those messages, the greater the required storage.

D. EXTERNAL FACTORS

Our proposal depends heavily on bandwidth limitation and other economic or phycological factors to drive the usability of bandwidth trading. When the bandwidth is unlimited, all customers in a given region will become satisfied with the bandwidth provided, and there is no reason for customers to do the trading. Furthermore, sellers must sacrifice their Internet usage for the sake of others having more bandwidths with given incentives. Those actions are trivial to achieve when the sellers are away (e.g., currently not using the Internet because of sleeping or out of the house/office). However, it becomes subjective when consumers experience terrible Internet service due to their sellings. Some consumers may think that selling the bandwidth is not worth the degradation of the Internet service. Therefore, while our study proves the applicability to performing reliable bandwidth trading from a technical perspective, further study is required to analyze the economic and phycological aspects.

VII. CONCLUSION

This paper proposed a blockchain-based framework to provide trading of bandwidth resources in SDN-enabled edge networks. Buyers can request bandwidth demands in given timeslots. Other consumers and SDN controllers can then compete to fulfill those demands by offering sellings of their bandwidth. SDN controllers provisioned the trading by sending FLOW-MOD messages to increase/decrease buyer/seller traffic. The corresponding FLOW-REMOVED messages will be saved as proof-of-tradings. We implemented and comprehensively analyzed our framework based on the POX SDN controller and Ethereum platforms. The results proved that the system could provide seamless off-chain and on-chain bandwidth trading provisioning with reasonable overhead.

While our study showed the applicability of performing reliable bandwidth trading from a technical perspective, further work is required to improve the blockchain privacy issues, provide proof of switch failures, and analyze the willingness of consumers to perform trading. The use of clusters of SDN controllers can also be investigated in the future to further enhance the system scalability from the SDN side.

ACKNOWLEDGMENT
The authors would like to thank the anonymous reviewers for their valuable comments.

REFERENCES

[1] K. Casier, S. Verbrugge, R. Meersman, J. Van Ooteghem, D. Colle, M. Pickavet, P. and Demeester, "A fair cost allocation scheme for capex and opex for a network service provider," in *Proc. 5th Conf. Telecommun. Techno-Economics*, 2006, pp. 1–13.

[2] J. Arkko, "The influence of internet architecture on centralised versus distributed internet services," *J. Cyber Policy*, vol. 5, no. 1, pp. 30–45, Jan. 2020.

[3] P. Thaichon and T. N. Quach, "The relationship between service quality, satisfaction, trust, value, commitment and loyalty of internet service providers' customers," *J. Global Scholars Marketing Sci.*, vol. 25, no. 4, pp. 295–313, Oct. 2015.

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

[5] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," Decentralized Bus. Rev., White Paper, 2008, p. 21260. [Online]. Available: https://www.debr.io/article/21260-bitcoin-a-peer-to-peer-electronic-cash-system

[6] V. Buterin, "A next-generation smart contract and decentralized application platform," White Paper, 2014, vol. 3, no. 37. [Online]. Available: https://nft2x.com/wp-content/uploads/2021/03/EthereumWP.pdf

[7] (2022). Open Networking Foundation. *Openflow Switch Specification*. Accessed: Jun. 6, 2022. [Online]. Available: https://opennetworking.org/wp-content/uploads/2013/04/openflow-spec-v1.0.0.pdf

[8] M. Castro and B. Liskov, "Practical Byzantine fault tolerance," in *Proc. OSDI*, vol. 99, 1999, pp. 173–186.

[9] N. Szabo, "Formalizing and securing relationships on public networks," *1st Monday*, vol. 2, no. 9, Sep. 1997. [Online]. Available: https://firstmonday.org/ojs/index.php/fm/article/view/548

[10] W. Cai, Z. Wang, J. B. Ernst, Z. Hong, C. Feng, and V. C. Leung, "Decentralized applications: The blockchain-empowered software system," *IEEE Access*, vol. 6, pp. 53019–53033, 2018.

[11] Y. Chen, C. Liu, Y. Wang, and Y. Wang, "A self-sovereign decentralized identity platform based on blockchain," in *Proc. IEEE Symp. Comput. Commun. (ISCC)*, Sep. 2021, pp. 1–7.

[12] U. W. Chohan, "Non-fungible tokens: Blockchains, scarcity, and value," Crit. Blockchain Res. Initiative (CBRI) Work. Papers, Tech. Rep., 2021. [Online]. Available: https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3822743

[13] M. Koscina, M. Lombard-Platet, and C. Negri-Ribalta, "A blockchain-based marketplace platform for circular economy," in *Proc. 36th Annu. ACM Symp. Appl. Comput.*, Mar. 2021, pp. 1746–1749.

[14] L. Sun, Q. Yang, X. Chen, and Z. Chen, "RC-chain: Reputation-based crowdsourcing blockchain for vehicular networks," *J. Netw. Comput. Appl.*, vol. 176, Feb. 2021, Art. no. 102956.

[15] D. Chen, Z. Zhang, A. Krishnan, and B. Krishnamachari, "PayFlow: Micropayments for bandwidth reservations in software defined networks," in *Proc. IEEE Conf. Comput. Commun. Workshops (INFOCOM WKSHPS)*, Apr. 2019, pp. 26–31.

[16] W. Li, D. Guo, K. Li, H. Qi, and J. Zhang, "IDaaS: Inter-datacenter network as a service," *IEEE Trans. Parallel Distrib. Syst.*, vol. 29, no. 7, pp. 1515–1529, Jul. 2018.

[17] T. H. T. Le, N. H. Tran, T. LeAnh, T. Z. Oo, K. Kim, S. Ren, and C. S. Hong, "Auction mechanism for dynamic bandwidth allocation in multi-tenant edge computing," *IEEE Trans. Veh. Technol.*, vol. 69, no. 12, pp. 15162–15176, Dec. 2020.

[18] J. Chase, R. Kaewpuang, W. Yonggang, and D. Niyato, "Joint virtual machine and bandwidth allocation in software defined network (SDN) and cloud computing environments," in *Proc. IEEE Int. Conf. Commun. (ICC)*, Jun. 2014, pp. 2969–2974.

[19] B. M. Yakubu, M. M. Ahmad, A. B. Sulaiman, A. S. Kazaure, M. I. Khan, and N. Javaid, "Blockchain based smart marketplace for secure internet bandwidth trading," in *Proc. 1st Int. Conf. Multidisciplinary Eng. Appl. Sci. (ICMEAS)*, Jul. 2021, pp. 1–6.

[20] S. Mostafavi, "Design space of online bandwidth trading markets for P2P streaming systems," *J. Commun. Technol., Electron. Comput. Sci.*, vol. 26, pp. 1–9, Dec. 2019.

[21] S. Ding, G. Shen, K. X. Pan, S. K. Bose, Q. Zhang, and B. Mukherjee, "Blockchain-assisted spectrum trading between elastic virtual optical networks," *IEEE Netw.*, vol. 34, no. 6, pp. 205–211, Nov. 2020.

[22] M. K. Farshbafan, M. H. Bahonar, and F. Khaiehraveni, "Spectrum trading for device-to-device communication in cellular networks using incomplete information bandwidth-auction game," in *Proc. 27th Iranian Conf. Electr. Eng. (ICEE)*, Apr. 2019, pp. 1441–1447.

[23] N. Merayo, D. de Pintos, J. C. Aguado, I. de Miguel, R. J. Durán, P. Fernández, R. M. Lorenzo, and E. J. Abril, "Experimental validation of an SDN residential network management proposal over a GPON testbed," *Opt. Switching Netw.*, vol. 42, Nov. 2021, Art. no. 100631.

[24] M. Zaher, A. H. Alawadi, and S. Molnár, "Sieve: A flow scheduling framework in SDN based data center networks," *Comput. Commun.*, vol. 171, pp. 99–111, Apr. 2021.

[25] N. Xia, P. Tsai, T. C. Ling, and C. Yang, "A service-aware transport layer framework for SDN-enabled cellular core networks," *IET Commun.*, vol. 16, no. 11, pp. 1279–1289, Jul. 2021.

[26] F. Vogelsteller and V. Buterin. (2015). *EIP-20: Token Standard*. Accessed: Jun. 6, 2022. [Online]. Available: https://eips.ethereum.org/EIPS/eip-20

[27] (2022). Truffle Suite. *Ganache: One Click Blockchain*. Accessed: Jun. 13, 2022. [Online]. Available: https://trufflesuite.com/ganache/

[28] (2022). Truffle Suite. *Truffle: Smart Contracts Made Sweeter*. Accessed: Jun. 13, 2022. [Online]. Available: https://trufflesuite.com/truffle/

[29] N. Mudge. (2020). *Ethereum's Maximum Contract Size Limit is Solved With the Diamond Standard*. Accessed: Apr. 22, 2022. [Online]. Available: https://dev.to/mudgen/ethereum-s-maximum-contract-size-limit-is-solved-with-the-diamond-standard-2189

[30] (2022). *Gas and Fees*. Accessed: Jun. 6, 2022. [Online]. Available: https://ethereum.org/en/developers/docs/gas/

[31] Y. Sompolinsky and A. Zohar, "Secure high-rate transaction processing in Bitcoin," in *Financial Cryptography and Data Security* (Lecture Notes in Computer Science), vol. 8975, R. Böhme and T. Okamoto, Eds. Berlin, Germany: Springer, 2015. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-662-47854-7_32, doi: 10.1007/978-3-662-47854-7_32.

[32] (2021). OpenEthereum. *Aura—Authority Round—Wiki*. Accessed: Jun. 6, 2022. [Online]. Available: https://openethereum.github.io/Aura

[33] (2022). Etherscan. *Ethereum Average Block Time Chart*. Accessed: Jun. 6, 2022. [Online]. Available: https://etherscan.io/chart/blocktime

[34] (2022). Etherscan. *Kovan Testnet Explorer*. Accessed: Jun. 6, 2022. [Online]. Available: https://kovan.etherscan.io/

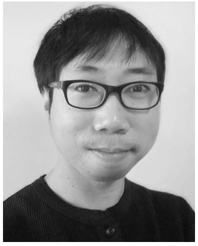
[35] (2022). Klaytn. *Klaytn Overview*. Accessed: Jun. 6, 2022. [Online]. Available: https://docs.klaytn.foundation/klaytn

[36] (2022). ChainSafe. *Web3.js—Ethereum Javascript API*. Accessed: Jun. 6, 2022. [Online]. Available: https://github.com/ChainSafe/web3.js

[37] (2022). OpenJS Foundation. *Fast, Unopinionated, Minimalist Web Framework for Node.js*. Accessed: Jun. 6, 2022. [Online]. Available: https://expressjs.com/

[38] R. L. S. de Oliveira, C. M. Schweitzer, A. A. Shinoda, and L. R. Prete, "Using mininet for emulation and prototyping software-defined networks," in *Proc. IEEE Colombian Conf. Commun. Comput. (COLCOM)*, Jun. 2014, pp. 1–6.

[39] (2022). Open vSwitch. *Open Vswitch Manual*. Accessed: May 24, 2022. [Online]. Available: http://www.openvswitch.org/support/dist-docs/ovs-vsctl.8.txt
[40] (2022). McCauley. *Pox Manual Current Documentation*. Accessed: Jun. 6, 2022. [Online]. Available: https://noxrepo.github.io/pox-doc/html/
[41] J. Dugan, S. Elliott, B. A. Mah, J. Poskanzer, and K. Prabhu. (2022). *IPerf—The Ultimate Speed Test Tool for TCP, UDP and SCTP*. Accessed: Jun. 6, 2022. [Online]. Available: https://iperf.fr/iperf-doc.php



YUSTUS EKO OKTIAN received the bachelor's degree in electrical engineering from Petra Christian University, Indonesia, in 2013, and the master's and doctoral degrees in computer engineering from Dongseo University, South Korea, in 2016 and 2021, respectively. He is currently a Postdoctoral Researcher with Pusan National University, South Korea. His research interests include network security, distributed computing, blockchain, Internet of Things (IoT), and software-defined networking (SDN).

THI-THU-HUONG LE received the bachelor's degree from the Hung Yen University of Technology and Education (HYUTE), Vietnam, in 2007, the master's degree from the Hanoi University of Science and Technology (HUST), in 2013, and the Ph.D. degree from Pusan National University (PNU), South Korea, in 2020. She has three years of experience as a Postdoctoral Researcher at PNU, since 2020. She had seven years of experience as a Lecturer at HYUTE. She has participated in machine learning projects, such as NILM, IDS, AI Industry4.0, AI security, and deep learning-based CFD. She is a Research Professor with the IoT Research Center, PNU. Her research interests include machine learning, deep learning, data analysis, XAI, and signal processing.

UK JO (Graduate Student Member, IEEE) received the bachelor's degree from the Kwangwoon University, South Korea. He is currently pursing the Ph.D. degree with Pusan National University. He has conducted various projects, such as blockchain platform, smart city platform development, and crypto implementation. His current research interests include blockchain and security.

HOWON KIM (Member, IEEE) received the Ph.D. degree from the Pohang University of Science and Technology (POSTECH), in 1999. He has been at the Electronics and Telecommunications Research Institute (ETRI), since 1998. He visited the Chair for the Communication Security Group (COSY) in Ruhr-University Bochum, Germany, as a Postdoctoral Researcher, from July 2003 to June 2004. He is currently a Professor at the Department of Computer Engineering, the Chief of the Energy IoT Research Center, and the Chief of the Blockchain Platform Research Center. His research interests include blockchain platform, cryptography, deep learning, and security chip design.