SmartWiFi: Universal and Secure Smart Contract-Enabled WiFi Hotspot

Nikolay Ivanov, Jianzhi Lou, and Qiben Yan

Michigan State University, East Lansing MI 48824, USA
{ivanovn1,loujianz,qyan}@msu.edu

Abstract. WiFi hotspots have been widely used for establishing public WiFi services and enterprise networks. However, WiFi hotspots often suffer from mediocre security, unreliable performance, limited access, and cumbersome authentication procedure. Specifically, public WiFi hotspots can rarely guarantee satisfactory speed and uptime, and their configuration often requires a complicated setup with subscription to a payment aggregator. Moreover, paid hotspots can neither protect clients against low quality or non-service after prepayment, nor do they provide an adequate defense against misuse by the clients. In this paper, we propose SmartWiFi, a universal, secure, and decentralized smart contract-enabled WiFi hotspot that can be deployed in any public or private environment. SmartWiFi provides cross-domain authentication, fully automated accounting and payments, and security assurance for both hotspots and clients without relying on complex authentication and billing infrastructure. SmartWiFi utilizes a novel off-chain transaction scheme called Hash Chain-based Network Connectivity Satisfaction Acknowledgement (Hansa), which enables fast and low-cost provider-client protocol by restricting otherwise unacceptable delays and fees associated with blockchain interaction. In addition, we present DupSet, a dynamic user-perceived speed estimation technique, which can reliably evaluate the quality of Internet connection from the users' perspective. We design and implement SmartWiFi desktop and mobile apps using an Ethereum smart contract. With extensive experimental evaluation, we demonstrate that SmartWiFi exhibits rapid execution with low communication overhead and reduced blockchain fees that are adjustable for balancing delays and costs.

Keywords: WiFi hotspot · Smart contract · Blockchain transaction.

1 Introduction

The number of mobile Internet users have been steadily increasing, corroborating a pressing need for reliable wireless connectivity to be available everywhere, all the time. As opposed to cellular communications, WiFi provides a low-cost solution for wireless Internet access with a miniature infrastructure [14]. During the past two decades, WiFi has become the de facto standard for wireless local area networks (WLAN) and Internet-of-Things (IoT) [19].

The WiFi technology has been used to create hotspots to offer Internet access to users in their proximity. WiFi hotspots are typically seen in such venues as airports, cafes, hotels, etc. Private hotspots are often configured in enterprise, personal, and household networks to serve limited number of WiFi-enabled devices. Both public and private hotspots often require authentication and/or payment. Two or more hotspots belong to the same *authentication domain* if they share the same authentication server and, if applicable, share a payment server. Although a number of technologies have been introduced for cross-domain authentication, such as Passpoint [7] and eduroam [1], WiFi hotspots are still partitioned into a multitude of incompatible domains, which makes seamless WiFi roaming infeasible. In this work, we introduce a practical solution for a universal (i.e., cross-domain) and decentralized hotspot network, which addresses the domain partitioning problem. **Ultimately, we envision a fully automated cross-domain authentication between wireless APs provided by different businesses and private owners, forming a global permissionless decentralized network of free and paid hotspots.** However, in order to achieve this goal, a number of existing hotspots' shortcomings must be addressed.

Motivation. Despite its obvious benefits and popularity, the current WiFi hotspot technology experiences significant shortcomings: **M1)** *Security:* Public WiFi often eliminates password protection or conveys the passwords insecurely. **M2)** *Unreliable performance:* The speed of a WiFi hotspot largely depends on several unpredictable factors, such as the number of connected users or the bandwidth consumption of each individual user. Moreover, the hotspot owners generally have no incentive for upgrading hardware and service. **M3)** *Limited access:* Traditional WiFi hotspots do not offer a universal service for everyone. To be associated with a hotspot service, a user should be ascribed to a certain role or affiliation. The users' access to the service hinges upon their particular subscriptions. **M4)** *Cumbersome procedure or high infrastructure cost:* Connecting to a WiFi hotspot often requires extensive manual effort, such as: searching for SSID, entering payment details, specifying authentication settings, etc. Although WLAN direct IP access or 3GPP IP access enable easy configurations, they both rely on a heavy-cost cellular authentication infrastructure.

In this research, we envision that transferring the point of centralized trust from hotspot and/or client to a decentralized independent party, i.e., blockchain, enhances security of the connection and payment while simplifying the configuration procedure (to address **M1**). SmartWiFi hotspot establishes the dependency between the Quality of Service (QoS) and payment, which creates an incentive for hotspot owners to deliver a high QoS (to address **M2**). The proposed hotspot technology is universal and accessible, i.e., it serves all clients who have means to pay, while also supporting unrestricted free WiFi hotspots (to address **M3**). The simplified configuration procedures offer a full automation of handshake, connection control, and checkout using the enforced execution of smart contract protocols without relying on complex server-based or cloud authentication infrastructure (to address **M4**).

Key Challenges. Designing a universal smart contract-enabled WiFi hotspot involves three major challenges. First, blockchain execution incurs significant processing delays, rendering the execution of many operations impossible within reasonable time limits. Second, blockchain offers very limited data storage. Third, blockchain networks charge considerable fees for executing block-modifying operations, e.g., payment transactions, smart contract deployments, smart contract state transitions, etc.

In this paper, we present SmartWiFi, the *first operational smart contract-enabled WiFi hotspot with automated cross-domain authentication.* SmartWiFi leverages a novel *off-chain protocol* called Hash Chain-based Network Connectivity Satisfaction Acknowledgement (Hansa[1]) to manage secure and reliable connection. An off-chain protocol establishes communication between two entities using blockchain, but executes without any interaction with blockchain, which allows Hansa to enable a fast, low-cost, and low-overhead provider-client interaction with significant reduction of blockchain delays and fees. In addition, we present DupSet, a Dynamic User-Perceived Speed Estimation Technique, which reliably estimates the speed of Internet connection for client-side QoS control. Leveraging these novel techniques, we design and implement SmartWiFi desktop and mobile apps using a smart contract executed by an Ethereum Virtual Machine (EVM). **A video demonstration of the SmartWiFi app is available at https://youtu.be/jrDl204fGso. The source code of the SmartWiFi smart contract is available at https://bit.ly/2X5a4ez**.

This paper makes the following main contributions.

– **Protocol Design.** To build SmartWiFi, we propose Hansa, a novel cryptographic scheme that provides cross-domain authentication and establishes a smart contract-enabled off-chain session arrangement for a hotspot and a client. It provides a fast and low-cost smart contract execution by restricting blockchain transaction delays and fees. We also design DupSet to quantify the QoS of Internet access provided by SmartWiFi hotspots to clients. DupSet allows SmartWiFi clients to perform low-overhead bandwidth estimation to measure the quality of Internet connection.
– **System Implementation.** We implement operational prototypes of SmartWiFi router and client that use Ethereum blockchain as a smart contract platform. Both components are cross-platform, hardware-agnostic, and can be easily deployed into existing infrastructure. In addition, we implement a fully-functional SmartWiFi Android app, demonstrating the feasibility of deploying SmartWiFi on non-rooted mobile platforms.
– **Experimental Evaluation.** We rigorously evaluate the delays, blockchain fees, and communication overhead of SmartWiFi on Ropsten and Mainnet Ethereum networks. We also scrutinize the DupSet technique by juxtaposing its measurements with the results from nine popular bandwidth measurement services. Furthermore, we evaluate the scalability of SmartWiFi

[1] The name is inspired by the Hansa Trade League, which successfully operated under the power of mutual trust for over a century in a turbulent political and economic environment of Medieval Europe.

by demonstrating the stability of the system under the load of more than 100 simultaneous client processes connected to a single SmartWiFi router.

2 Background and Key Insights

2.1 Blockchain and Smart Contracts

Formally, *blockchain* is a distributed abstract data structure (ADS) represented by a list of objects (blocks), which are cryptographically linked in such a way that a modification of any block would require a chain recalculation (validation) of all subsequent blocks in the list. Consequently, any block-modifying operation, except *append*, draws a considerable execution time complexity. The block validation speed is deliberately throttled in the proof-of-work (PoW) consensus protocol employed by Ethereum, Bitcoin, and some other blockchains, making retrospective modifications of these blockchains nearly impossible. Practically, the term blockchain is used to refer to one of many peer-to-peer (P2P) networks that store, synchronize, and cross-validate their respective blockchain data structures. A *smart contract* is a distributed deterministic application, deployed on blockchain, and individually executed by the blockchain participants, with any associated data and results being part of the consensus. Therefore, smart contracts can establish, execute, and unequivocally enforce protocols and agreements between parties.

2.2 Threat Model

We consider a threat model with both malicious clients and malicious hotspots. Malicious clients would attempt to obtain Internet access without payment, which is regarded as *free-rider attack*. They could also try to bring significant performance degradation or complete shutdown of the hotspot. Malicious hotspots, on the other hand, aim to get payment from the clients without providing sufficient QoS.

We assume that hotspots and clients have no knowledge regarding their respective identities, and they have no pre-established trust. Moreover, the blockchain, smart contract, and its underlying cryptography are considered secure and trusted by hotspots and clients, i.e., we do not consider a wide range of attacks towards blockchain [12] and smart contracts [25].

2.3 Overview of Key Insights

Recognizing the shortcomings of existing WiFi hotspots, we bring forth a set of key insights that lead to the design of SmartWiFi.

Off-Chain Interaction. High delays and fees in blockchain networks make it impossible to query the blockchain frequently for trust renewal. The idea of off-chain interaction, in which a smart contract is used by two or more parties as a guarantor of a protocol, but not as an executor of this protocol, has been proposed for fast and cheap payments [21,16]. We extend this idea to WiFi hotspots by limiting the blockchain interaction to only handshake (session initiation) and payment resolution (session conclusion).

Cryptographic Satisfaction Acknowledgement. One of the key design goals of SmartWiFi is to develop a protocol that would deliver a tamper-proof testimony of Internet usage time to the smart contract. The traditional approach is based on connection time and data size measurements performed by the provider itself, which relies on the assumption of its trustworthiness. However, a more comprehensive Internet traffic accounting is needed to ensure proper mutual agreement and non-repudiation. We design such a scheme using periodic cryptographically verifiable acknowledgements sent by the client to the hotspot. Each next acknowledgement testifies the client's satisfaction in the quality of the Internet connection during *a short period of time since the previous acknowledgement*, which we call a *session unit*. Each acknowledgement can be cryptographically verified by the smart contract and exchanged for funds reserved in the contract.

Hash Chain Data Compression. The cryptographically verifiable acknowledgements need to be stored in the smart contract, resulting in fees and consumption of computational time. Our key insight is to represent the set of acknowledgements by a hash chain, which can be generated from one random seed, and the verification of each acknowledgement will only require the head of the hash chain. Therefore, the smart contract only needs to store one hash value, i.e., the hash chain head, to verify a series of acknowledgements. We use hash chain based arrangement instead of signatures to eliminate the need to constantly use the private key by the client, which makes SmartWiFi a safer option for unattended IoT devices.

Dynamic Speed Measurement. The satisfaction acknowledgement based protocol stipulates that the client evaluates the satisfiability of the Internet connection prior to sending an acknowledgement. Aiming for a fully-automated solution, we quantify the quality of the Internet connection using a dynamic speed measurement technique. Existing bandwidth estimation approaches require the transfer of a large amount of data, while we aim for frequent, fast, and low-overhead speed probes. Here, we simulate Internet activities using a set of HTTP servers deployed globally. The concept of measuring the speed of delivering an average web page, rather than consuming the available bandwidth, creates the possibility for frequent and low-overhead speed probes emulating actual user experience.

3 The SmartWiFi System

In this section, we present the design of the SmartWiFi system. Unlike traditional WiFi hotspots, SmartWiFi is a universal infrastructure that supports cross-domain authentication, i.e., anyone can use SmartWiFi as a client or as a hotspot, while the smart contracts authenticate the users by their Eithereum account (generated offline and stored by user). In this work, we use Ethereum

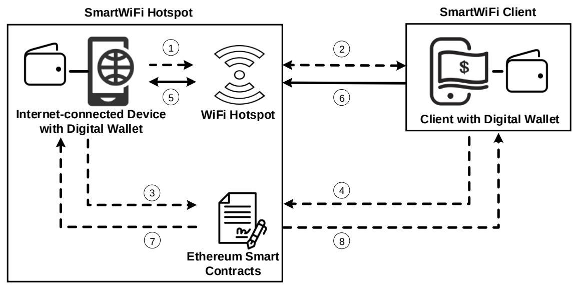


Fig. 1: SmartWiFi workflow: ① an Internet-connected device (router) provides SmartWiFi hotspot service; ② the client connects to the hotspot and sends it a hash chain head and its public address; ③ router provides a grace-period Internet access to the client and stores the public address in the smart contract; ④ client funds the smart contract; ⑤ router activates unrestricted Internet access for the client; ⑥ client periodically sends the router satisfaction acknowledgements (links of hash chain); ⑦ router claims payment from the smart contract using the last acknowledgement; ⑧ client is refunded by smart contract. *The dashed lines represent the* Hansa *protocol communications.*

as the target platform due to its relative maturity and wide popularity. Figure 1 depicts the basic building blocks of the SmartWiFi system, which consists of six major components: SmartWiFi router, the router's Ethereum wallet, the hotspot managed by the router, the client, the client's Ethereum wallet, and the smart contract.

SmartWiFi is enabled by three main ingredients: the Hansa protocol, the DupSet speed measurement, and the smart contract. The Hansa protocol establishes and maintains an Internet connection, and it includes two major sessions: handshake and service. Payment and refund are processed after the client-router connection terminates. While handshake, payment, and refund require interaction with blockchain, the service session is executed off-chain. DupSet is a speed measurement technique that allows the clients to quantify their QoS satisfaction and continuously monitor the Internet access quality of SmartWiFi. The smart contract is designed to process the payment and refund.

3.1 SmartWiFi Setup

SmartWiFi uses a smart contract to serve as an intermediate trust layer to hold/release the payment and enforce fair behavior between the router and the client. SmartWiFi also uses the router's firewall policy to control the clients' access privilege. The hotspot initiates SmartWiFi service after the router performs the following steps: (1) SmartWiFi router deploys several reusable smart contracts, the number of which equals the maximum number of concurrently-served clients; (2) the router establishes a two-way communication channel with every user; (3) the router activates a default firewall policy that allows every client to have a restricted access to required services, such as the blockchain

API. We define the *service period* as the period the client is connected to the Internet via a SmartWiFi router, and the *service unit* as the minimum service period that the client will be charged for.

3.2 Hansa Handshake Session

In Hansa handshake session, the router and the client establish a relationship regulated and protected by the smart contract. Hansa protocol begins when the client connects to the hotspot and establishes a TCP connection with the SmartWiFi router. The router replies with a greeting message, and the client generates a hash chain $\varUpsilon$ using a random secret seed $\varUpsilon_0$. The length of the hash chain, denoted as $|\varUpsilon|$, is calculated as $|\varUpsilon| = \frac{T}{\eta}$, where T is the length of the service period, and η is the length of the service unit. For instance, in our prototype, the length of the service period is 3,600 seconds, and the length of the service unit is 60 seconds, i.e., one Hansa session serves a connection up to 1 hour in length with per-minute acknowledgements.

The client keeps the seed of the hash chain in secret and sends the head of the hash chain $\varUpsilon_{head}$ and the public address A_{pub} to the router. The router then prepares the smart contract by storing the public key of the client's public address and the head of the hash chain in the smart contract. After that, the router replies to the client with the address of the smart contract. Before executing prepayment, the client verifies the bytecode of the smart contract and the price per service unit ξ, which is hard-coded in the smart contract. Then, the client prepays the smart contract with the amount of cryptocurrency $\varXi$ that corresponds to the cost of the entire service period, i.e., $\varXi = \xi \times \frac{T}{\eta}$. Once the prepayment is processed by the blockchain, the client and router enter the Hansa service session. If the price is unacceptable, the client terminates the connection.

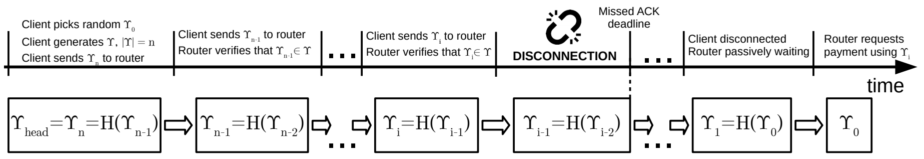

Client picks random Υ_0
Client generates Υ, $|\Upsilon| = n$
Client sends Υ_n to router
Client sends Υ_{n-1} to router
Router verifies that $\Upsilon_{n-1} \in \Upsilon$
Client sends Υ_i to router
Router verifies that $\Upsilon_i \in \Upsilon$
Missed ACK deadline
DISCONNECTION
Client disconnected
Router passively waiting
Router requests payment using Υ_i
time
$\Upsilon_{head}=\Upsilon_n=H(\Upsilon_{n-1})$
$\Upsilon_{n-1}=H(\Upsilon_{n-2})$
$\Upsilon_i=H(\Upsilon_{i-1})$
$\Upsilon_{i-1}=H(\Upsilon_{i-2})$
$\Upsilon_1=H(\Upsilon_0)$
Υ_0

Fig. 2: Hansa timeline with respect to the hash chain $\varUpsilon$. In this scenario, the client disconnects after releasing acknowledgement $\varUpsilon_i$, and the acknowledgement $\varUpsilon_{i-2}$ was not released. When the service session timer expires, the router uses the last available acknowledgement $\varUpsilon_i$ to request payment from the smart contract.

3.3 Hansa Service Session

The Hansa service session begins after the router verifies that the client has funded the smart contract. Then follows the grace period, which seamlessly switches into an unrestricted Internet access. Meanwhile, the router sends the client a short message signifying the beginning of a service session, and both the client and router start service session timers.

Table 1: Summary of smart contract features required for executing Hansa.

Feature	Type	Access Control	Security Measure
Υ_{head} (hash chain head)	variable	`rw-r--r--`	timer
A_{pub} (public address)	variable	`rw-r--r--`	timer
ξ (price)	constant	`rw-r--r--`	read only
T (session length)	constant	`r--r--r--`	read only
τ (refund delay)	constant	`r--r--r--`	read only
η (session unit length)	constant	`r--r--r--`	read only
balance check	function	`r-xr-xr-x`	read only
Υ_{head}-accessor	function	`r-xr-xr-x`	read only
Υ_{head}-mutator	function	`r-xr--r--`	timer
A_{pub}-accessor	function	`r-xr-xr-x`	read only
A_{pub}-mutator	function	`r-xr--r--`	timer
prepay (fund)	p-function	`r-xr-xr-x`	none
checkout	function	`r-xr--r--`	Υ-check
refund	function	`r--r-xr--`	delay (τ)

Satisfaction Acknowledgement. Traditional paid WiFi hotspots charge users ahead of the service, and if the QoS is unacceptable, requesting refund is often challenging. We use cryptographic satisfaction acknowledgements to allow the client to control its service session and payment. The first service unit of a service session is regarded as a free trial, during which the client confirms that the Internet connection is active and starts measuring speed (described in Section 3.4). Before the end of each service unit, the client confirms a satisfactory QoS by sending to the router a *satisfaction acknowledgement* (the next hash in the hash chain), as shown in Figure 2. The router verifies that the acknowledgement is the valid hash on the hash chain, replies with an *acknowledgement response*, and extends the connection for another service unit.

Hansa allows the client to pause the connection, which can happen automatically as a result of a speed probe, or can be triggered manually by the user. If the router does not receive the next acknowledgement on time, it deactivates the Internet access for the client. During the service period, the client can resume acknowledging the service, which will reactivate the Internet access, with a maximum reactivation delay η. The service session concludes when either the timer reaches the value T, or when the client-router connection breaks.

3.4 DupSet Speed Measurement

We present DupSet, a bandwidth estimation solution that allows SmartWiFi clients to quantify hotspots' QoS. SmartWiFi is designed to operate in a flexible range of speeds and with different number of mobile or stationary users, so the bandwidth estimation should be frequent and with low overhead. Traditional speed evaluation methods include four metrics: capacity, available bandwidth, TCP throughput, and bulk transfer capacity (BTC) [22]. Although these techniques can provide very accurate results, they are not suitable for SmartWiFi since they require lengthy probes and transfer of large amounts of data.

The core of DupSet is a metric called *user-perceived speed*, represented by the transmission component of the throughput when loading an average web page. Measuring the transmission component, instead of the entire end-to-end communication, allows to achieve transparency with respect to different bandwidth uses, such as video streaming services or VPN traffic. DupSet draws probes from pre-selected servers. Unlike many traditional bandwidth services, such as M-Lab [17] and Ookla [20], the DupSet servers do not require to deliver high computational and throughput performance. Each probe calculates a statistical summary[2] of readings from all the reachable servers from the list. Then, the current DupSet reading is calculated using a simple moving average[3].

Each DupSet server is an HTTP server with two payload files with random information available for download. The size of the first file (P_1 bytes) is much greater than the size of the second file (P_2 bytes). The client loads both the files and calculates the difference between delays of downloading the first and the second file, which extracts the *transmission delay* from the total end-to-end delay. Then, the user-perceived speed reading (in bytes/second) for i^{th} server is determined as $Speed_i = \frac{EPF(P_1 - P_2)}{\Delta D_i}$, where EPF is the *Effective Payload Function* defined as follows:

$$EPF(x) = \begin{cases} 0, & \text{if } x \leq 0; \\ 0, & \text{if request failure;} \\ x, & \text{otherwise.} \end{cases}$$

ΔD_i is the time in seconds needed to load the file from the server i. The EPF function filters out unreliable results and ignores results from inaccessible DupSet servers, so when one or several DupSet servers are unavailable or provide unreliable readings, the accuracy of the DupSet result is not affected.

3.5 SmartWiFi Smart Contract

The SmartWiFi smart contract provides an overarching trust layer between the router and the client to exchange data and payments. The SmartWiFi smart contract has the following components: a) state variables; b) state changing functions; c) cryptocurrency balance; and d) *payable function (p-function)* for incoming payments. The functions that do not submit transactions (*pure* and *view* functions) are called anonymously, whereas the calls to state changing functions are signed by a specific user (using the account's private key).

The minimal set of the SmartWiFi smart contract features is summarized in Table 1, which includes constants, variables, functions, and one payable function. The access control to each feature is represented in the Unix-style symbolic access mode format, where the first triple refers to the router's privilege, the second triple is for user's privilege, and the third one is for others. The security column describes protective measures employed for each feature.

[2] We experimentally found that the third quartile statistic achieves a better measurement accuracy compared to mean, median, and maximum.

[3] We empirically determine that simple moving average over 6 periods (SMA-6) delivers stable and reliable results.

Algorithm 1 Smart contract payment routine

```
INPUT: $\Upsilon_{head}$, $\Upsilon_i$, $\xi$, $t$, $T$, $\eta$, $A_{pub}$
OUTPUT: none
1: if $\Upsilon_i \in \Upsilon$ and $caller = Router$ and $Timestamp \geq t + T$ then
2:     $RouterBalance \leftarrow i \times \xi$
3:     $RefundAmount \leftarrow (\frac{T}{\eta} - i) \times \xi$
4:     $TransferFunds(A_{pub}, RefundAmount)$
5: end if
6: return
```

The price ξ, session length T, refund delay τ, and service unit length η are set as constants to reduce execution delays and fees. The smart contract has two variables for hash chain head Υ_{head} and client public address A_{pub}; they can only be set by the router using their mutators. The accessor and balance check functions are called without fees since they do not modify the blockchain. Both the mutators use timers to prevent the modification of the values they set. The values are protected using a timer for at least the duration of a Hansa session, including handshake, service session, and checkout. The timer plays two important roles: first, it prevents a malicious modification of Υ_{head} and A_{pub} by the router; second, it facilitates the reuse of the smart contract, thereby reducing blockchain fees and delays. The prepay function funds the smart contract. The checkout and refund functions include additional security checks as depicted in Algorithms 1 and 2, which will be described next.

3.6 Payment and Refund

The fair payment and refund procedures are automatically enforced by the SmartWiFi smart contract. The smart contract holds the amount of cryptocurrency Ξ, sufficient for funding one Hansa session. The router is prohibited from claiming its payment until the blockchain timestamp reaches the value $t + T$, where t is the saved timestamp at the beginning of the service session. Algorithm 1 shows how the payment is executed. The inputs include: the hash chain head Υ_{head}, last retrieved acknowledgement Υ_i, price ξ (stored as a constant in smart contract), timestamp t (saved during handshake), service session length T (constant), session unit length η, and the user's public address A_{pub}. The router obtains the payment based on the depth of Υ_i, and the remaining funds are transferred back to the client as a refund.

The execution of lines 2-4 in Algorithm 1 can only be triggered by the router. In case when the router does not request any payment, the client may never receive any refund, for which case we design an additional refund routine, described in Algorithm 2. The execution of the actual refund (lines 2–3) is permitted only by the client after the pre-determined refund delay τ, which prevents refund before payment described in Section 4.

Algorithm 2 Smart contract refund routine

```
INPUT: $\Upsilon_{head}$, $\xi$, $t$, $T$, $\eta$, $\tau$, $A_{pub}$
OUTPUT: none
1: if $caller.address = A_{pub}$ and $Timestamp \geq t + T + \tau$ then
2:     $RefundAmount \leftarrow \frac{T}{\eta} \times \xi$
3:     $TransferFunds(A_{pub}, RefundAmount)$
4: end if
5: return
```

4 Security Analysis

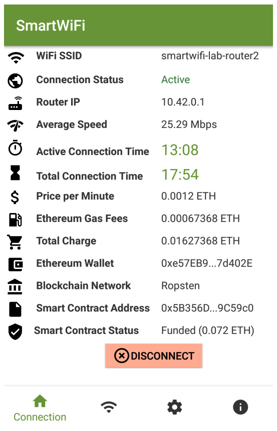
The security threats of SmartWiFi come either from malicious clients or from malicious hotspots/routers. In this section, we analyze the security of SmartWiFi.

Non-Service by Malicious Hotspot. The goal of the client is to have a satisfying Internet connection for the money paid. If the high-quality service is not provided, full or partial refund should be guaranteed. A malicious router might refuse a service, i.e., to receive a payment without providing a quality connection. To counteract such a behavior, the SmartWiFi client uses DupSet to assess the quality of Internet connection before sending each subsequent acknowledgement, while the SmartWiFi smart contract guarantees a full or partial refund.

Refund before Payment. The router expects to be fairly paid after the connection period is over. The goal of the client, who prepaid the smart contract with one whole period worth of money, is to receive a refund for all service units that do not result in satisfaction acknowledgement. *Refund before payment* indicates the case when a malicious client claims no service received and asks for a full refund. In SmartWiFi, this threat is prevented by the refund delay τ for the router to claim payment, during which the refund is impossible.

Handshake Flooding. The handshake of SmartWiFi is prone to denial-of-service attacks. The goal of the adversary in the handshake flooding attack is to render the router unavailable or degrade its performance. This can be achieved by initiating multiple incomplete handshakes, in which the attacker, pretending to be a valid client, forces the router to submit values to the smart contract, for which the blockchain charges fees. In SmartWiFi, this attack is prevented by checking the balance of the client before preparing the smart contract for that client. SmartWiFi router also curbs the number of clients to serve: once the number of requests exceeds the maximum, SmartWiFi starts dropping requests.

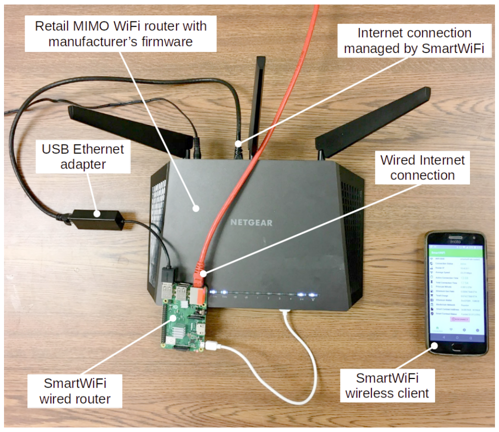
Free-Rider Attack (Non-Payment). The existence of the free trial in SmartWiFi allows any user who funds the contract to gain one service unit of Internet connectivity without providing an acknowledgement. A dedicated attacker may use multiple client devices to interchangeably connect to the router, use the free trial period (1 minute in this paper), disconnect without providing any acknowledgements, tunnel the traffic to the same outlet, and then request a full refund. We define such connection misuse as *traffic hopping*, which is a special



SmartWiFi

WiFi SSID	smartwifi-lab-router2
Connection Status	Active
Router IP	10.42.0.1
Average Speed	25.29 Mbps
Active Connection Time	13:08
Total Connection Time	17:54
Price per Minute	0.0012 ETH
Ethereum Gas Fees	0.00067368 ETH
Total Charge	0.01627368 ETH
Ethereum Wallet	0xe57EB9...7d402E
Blockchain Network	Ropsten
Smart Contract Address	0x5B356D...9C59c0
Smart Contract Status	Funded (0.072 ETH)

(a)


Retail MIMO WiFi router with manufacturer's firmware
Internet connection managed by SmartWiFi
USB Ethernet adapter
Wired Internet connection
NETGEAR
SmartWiFi wired router
SmartWiFi wireless client

(b)

Fig. 3: SmartWiFi prototype: (a) The connection page of the SmartWiFi Android app; (b) SmartWiFi configuration with a wired Internet connection, Raspberry Pi as a SmartWiFi router, retail WiFi router with factory software, and Android smartphone as a SmartWiFi client.

case of the free-rider attack. In SmartWiFi, we prevent such threats by relying on the accruing blockchain fees. As the creation of new malicious nodes (i.e., Sybil nodes) will require the attacker to transfer funds into multiple accounts and pay fees for each funding transaction, such fees, after being summed up from multiple accounts, will nullify the benefits of the free riding. After the free trial, the router expects to receive regular satisfaction acknowledgements. Each acknowledgement from a client is expected to arrive before a strict deadline, otherwise, the Internet connection will be terminated by the router.

5 Implementation

We implement a fully-functional SmartWiFi prototype on a Netgear router and Raspberry Pi clients for testing the general functionality and performance of the system. In addition, we implement an Android SmartWiFi client app, as shown in Figure 3(a), for testing the performance of SmartWiFi on mobile devices. The client app can be easily ported to iOS. We use Java 11 and Web3j for implementing the software of the router, the desktop/IoT client, and the Android client. We use Infura API [6] to interact with Ethereum blockchain.

Figure 3(b) shows one possible configuration of SmartWiFi, in which SmartWiFi router software is installed on Raspberry Pi with two Ethernet interfaces: one for Internet connection, another for delivering the Internet to the WiFi router. The retail WiFi router runs its original software; the configuration of this router includes TCP port 5566 forwarding in order to allow connected devices to access the SmartWiFi router. The client in this configuration is an ordinary Android smartphone without rooting. This configuration ensures SmartWiFi's compatibility with legacy systems, i.e., we can easily deploy SmartWiFi by plugging in a device running SmartWiFi router software.

We implement a prototype SmartWiFi Ethereum smart contract using Solidity programming language. In our prototype and evaluation, we use both Mainnet and Ropsten testnet for executing the smart contracts. Furthermore, we build an IoT testbed with five Raspberry Pi clients simultaneously connected to a single-antenna all-in-one SmartWiFi router (AMD A4 Micro-6400T, 4GB RAM, Xubuntu 18.04). This setup demonstrates that SmartWiFi *can be easily adapted to support a diverse variety of IoT configurations*.

6 Evaluation

We thoroughly evaluate the performance of the SmartWiFi prototype by scrutinizing the following system parameters under different circumstances: blockchain-related delays, Ethereum gas fees, smart contract storage, the accuracy of DupSet speed probes, the scalability of the system, and the communication overhead. In Ethereum, all blockchain-modifying transactions require the caller to pay fees measured in the unit named *gas*, which is convertible into Ether using a dynamic variable called *gas price*. In our evaluation, the service session lasts for one hour ($T = 3,600$ seconds), and the service unit is one minute ($\eta = 60$ seconds).

6.1 Delays

In this section, we evaluate the blockchain-related delays of SmartWiFi sessions in both Mainnet and Ropsten Ethereum networks. We add the Ropsten testnet for comparison to demonstrate the performance stability of SmartWiFi under Ethereum networks with different amounts of mining hash power. Thus, we show that if the parameters of the blockchain change in the future, it will not significantly affect the performance of SmartWiFi. For each type of blockchain-related delays, ten measurements have been taken. The average delays (with standard deviations) for Ropsten and Mainnet are presented in Table 2, from which we observe similar delays in both the networks.

The connection initiation phase, in which no blockchain interaction occurs, takes a few seconds on average; after this phase the user can start accessing the Internet. The handshake phase, whose average delay is below one minute for both Ropsten and Mainnet, initiates the payment arrangement. The code check phase, which requires only a non-modifying blockchain operation, also takes a few seconds in delay. The smart contract funding phase is essentially a cryptocurrency transaction, which requires more time than a read-only blockchain request. Similarly, payment and refund routines, although demanding additional calculations and checks, demonstrate delays just a little longer than a simple Ether transfer. In summary, to connect to a SmartWiFi router and start Internet access, *the client only experiences a few seconds of connection initiation delay*, which is completely acceptable.

The delay of blockchain execution in the Ethereum network can be further reduced by increasing the gas price offered for a transaction [13]. However, such a

Table 2: Comparison of blockchain-related average delays (in seconds) with relatively high gas price (100 GWei for Ropsten and 5 GWei for Mainnet).

Delay Type	**Ropsten Testnet**		**Ethereum Mainnet**	
	d_{avg}	σ	d_{avg}	σ
Connection initiation	3.965	0.177	4.161	0.202
Handshake	39.093	18.504	53.161	16.432
Bytecode verification	4.268	0.376	4.291	0.360
Funding	23.629	17.711	25.449	14.519
Payment	30.729	23.208	31.512	17.304
Refund	33.194	23.640	37.521	23.006

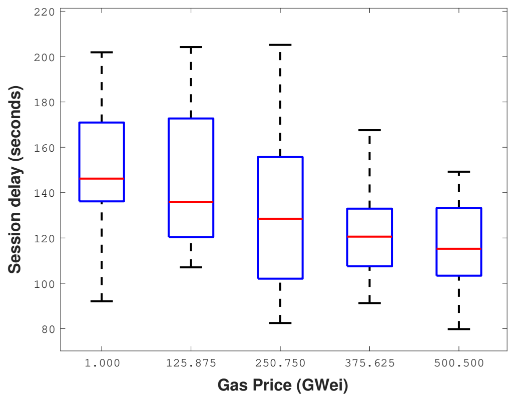


Fig. 4: Full session delays with different gas prices in Ropsten network. The graph has a logarithmic Gas price axis, and it shows that while it is empirically true that offering more gas increases the chance of faster transaction, the speed improvement is insignificant.

performance optimization is not guaranteed [24]. First, the Ethereum blockchain protocol does not enforce the prioritization of incoming transactions, leaving this decision to the discretion of miners. Second, since Ethereum is a decentralized network, the increase of transaction execution speed adopts a best-effort approach. Here, we conduct an empirical testing to evaluate the delays with respect to different gas prices, the result of which, presented in Figure 4, demonstrates a slight but consistent reduction of total SmartWiFi session delays as the gas price increases, which shows the possibility of reducing delays by offering a higher gas price. However, given the increasing cost, the delay reduction may not be worthwhile.

6.2 Fees

In this section, we measure the gas fees per transaction when a public function of the SmartWiFi smart contract is called. In order to exclude the possibility

Table 3: Gas fees for different functions of the SmartWiFi smart contract. Since SmartWiFi uses an off-chain execution protocol with infrequent smart contract transactions, the resulting fee overhead drops significantly.

Function	**Transaction fee with recommended gas price [2]**	
	Gas	**Approx. USD**
A_{pub}-accessor	0	0
A_{pub}-mutator	28,366	0.11
Υ_{head}-accessor	0	0
Υ_{head}-mutator	33,684	0.13
Balance check	0	0
Payment	50,076	0.20
Refund	42,266	0.17
Fund contract	21,040	0.08
Download contract bytecode	0	0

of variable fees, we take every measurement twice, and confirm that the cost remains the same for both measurements. The summary of gas fees is presented in Table 3. The address accessor, hash chain head accessor, balance check, and bytecode download are read-only blockchain operations, which do not incur any fees. However, the mutators and payable functions require the caller to pay fees. The fees in Table 3 are calculated for one 60-minute service session, with 1-minute service units.

Ethereum allows the issuer of a transaction to offer an arbitrary gas price to prioritize the transaction. Similar to the delay-measuring experiment in Figure 4, we record fees over 10 measurements on Ropsten network for a more realistic 10 gas prices equally spread across the interval between 0.5 and 5.0 GWei. The cumulative fee (for both router and client) is less than \$0.4, even with the highest gas price and ETH market price. Since the highest gas price is used rarely in production systems, the fee overhead is expected to be significantly lower then the maximum.

It is important to note that the cryptocurrency market price variations have little effect on SmartWiFi fee overhead. Ethereum is a dynamic self-regulating system, so when the market price of Ether goes up, the users can afford less, and they offer smaller fees for transactions, which results in lower average gas price, and vice versa [13]. The curve of resulting fee in USD is thus smoothed and flattened. Therefore, regardless of any cryptocurrency price fluctuations, SmartWiFi blockchain *fees paid in USD will remain approximately the same.*

6.3 Smart Contract Storage

Table 4 shows a comparison between data stored in the SmartWiFi smart contract with and without hash chain compression, from which we can see that the hash chain in Hansa stores about 17 times less data in the smart contract, effectively reducing per-session delays. Moreover, it also reduces per-session fees from \$8 to about 40¢ in USD equivalent, which corroborates the feasibility of SmartWiFi in terms of low cost.

Table 4: Data stored in the smart contract per session ($T = 3,600, \eta = 60$).

Data Unit	**Stored data per session**	
	With hash chain	**Without hash chain**
Acknowledgement data	32 bytes	1,920 bytes
Client identity	20 bytes	20 bytes
Auxiliary data	64 bytes	64 bytes
Total	116 bytes	2,004 bytes

6.4 DupSet Measurement and Overhead

In this section, we evaluate the feasibility of DupSet by comparing our estimations with the average readings obtained from nine popular Internet speed measurement services, specifically: Bandwidth Place, DSLReports, Fast.com, Google Fiber, Internet Health Test, M-Lab, Ookla, Speed-Of.Me, and Xfinity. We test ten different SmartWiFi router Internet connections belonging to different speed tiers, and evaluate average speeds of each of these connections by taking six speed test probes at each of the nine services listed above. The six speed test probes consist of three probes per service before running the DupSet simulation, and three probes per service right after the DupSet simulation.

In our prototype setup, we deploy ten DupSet servers in different geographic locations. In order to achieve further diversity in measurements, we use servers provided by two different cloud services, DigitalOcean [5] and Vultr [15]. For each Internet connection, we run 60 probes of DupSet for measuring the transmission speed component from each of the servers based on the payload of 10 kilobytes. The fastest reading from the ten servers represents the speed result of a probe.

The experiment confirms that the low-overhead DupSet estimations correlate with the high-overhead traditional Internet speed readings. Figure 5 shows that DupSet speed measurement results accurately reflect the Internet connection speed tier, which quantifies the QoS of user service. The spiked increase in the gap between the two readings at high speeds demonstrates the core difference between traditional bandwidth measurements and user-perceived speed estimation: a drastic increase in available bandwidth after a certain threshold does not trigger a proportional boost in loading web pages. In a high-speed Internet, the performance bottleneck moves from the client to the server.

The overall maximum communication overhead of DupSet probes depends on the number of DupSet servers and the size of payload on any of these servers. In our prototype, we empirically select a 10-kilobyte DupSet payload and 10 DupSet servers, resulting in 100-kilobyte maximum overhead per probe, or approximately 6 Mb of overhead per one-hour session. Through this experiment, we demonstrate that DupSet *probes reflect accurate user-perceived speed with low overhead.*

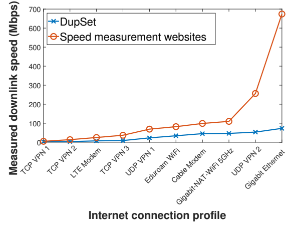


Fig. 5: Correlation between traditional Internet speed measurement (average result from nine websites) and DupSet probes over 10 different Internet connection profiles.

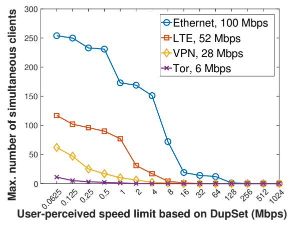


Fig. 6: Maximum number of clients simultaneously served by the router for 15 minutes under different Internet connectivities, with random web surfing simulation in the background.

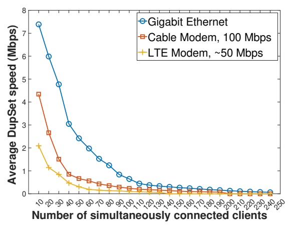


Fig. 7: Average DupSet readings for different types of Internet connection profiles with different number of clients simultaneously served by a SmartWiFi router.

A SmartWiFi client uses DupSet to control minimum expected speed. Since different users may have different minimum speed requirements at different times (e.g, watching stream video needs higher speed than reading e-mail), it is required from the users to explicitly specify their expectations in the client settings. In the SmartWiFi Android app, for example, we let the user choose between 5 discrete options.

6.5 Scalability

SmartWiFi is designed to scale to multiple clients connecting to a single router. We evaluate the performance of the system under the load of different numbers of users. For each client, we perform a background web surfing simulation that picks and loads a random website from the Alexa Top 10K list [11] every 10 seconds. Figure 6 shows the number of clients one router could serve without disconnection. As we can see, this capacity depends on the bandwidth of the Internet connection of the router and the maximum expected Internet speed set by the client. The experiment shows that when the router has a high-bandwidth Internet connection, and clients do not request high speed, SmartWiFi is capable of serving hundreds of clients simultaneously[4].

Figure 7 shows average DupSet readings for different Internet connections with different number of simultaneously served clients under a background web surfing simulation. The graph shows the number of clients one SmartWiFi router can serve based on its Internet connection bandwidth and average speed expectations. For example, it will be overly ambitious for a SmartWiFi router with 100 Mbps connection to serve 40 users whose average speed expectation

[4] The growing number of users incurs higher rate of physical layer packet collisions. One way to mitigate this is to use MIMO WiFi access point hardware.

is 2 Mbps. However, if the expectation is reduced to 1 Mbps, serving 40 users simultaneously will likely be a realistic projection.

6.6 SmartWiFi Communication Overhead

The *communication overhead* includes the client-router TCP traffic and the Infura blockchain API communication. We measure overhead by capturing network traffic and calculating a cumulative one-hour session TCP payload using Wireshark. Each session's average result is based on 10 measurements. The results in Table 5 demonstrate that the overhead of off-chain communication is low compared to the results for blockchain-related calls.

Table 5: Session communication overhead for different SmartWiFi calls over 10 measurements. The *local* calls represent off-chain communication between the hotspot and the client, including handshake E_h, connection initiation E_c, connection status check E_s, and acknowledgement E_a. *Blockchain* (B/C) calls use Infura API [6].

Procedure Call	**Avg TCP Payload (bytes)**	$\boldsymbol{\sigma}$
Local: E_h	580	20
Local: E_c	412	0
Local: E_s	274	9
Local: E_a	334	8
B/C: download bytecode	69,797	32,633
B/C: A_{pub}-accessor	51,597	38,224
B/C: A_{pub}-mutator	51,279	34,389
B/C: Υ_{head}-accessor	50,472	33,205
B/C: Υ_{head}-mutator	60,606	36,489
B/C: balance check	64,487	46,150
B/C: payment	45,326	28,016
B/C: refund	59,500	41,856
B/C: fund contract	56,834	39,542

7 Related Work

Traditional WiFi Hotspot Solutions. Current non-blockchain WiFi hotspot solutions are represented either by manual setups, or cloud-managed subscription-based proprietary products, such as Cisco Meraki [3], Aruba [4], Ruckus [9], etc. However, none of these approaches addresses the set of objectives achieved by SmartWiFi, namely: a) enhancing hotspot security against malicious routers *and* clients; b) providing universal authentication and billing; and c) making payment based on service quality.

Blockchain Solutions. The most relevant work to SmartWiFi is a use case in OPPay [23], a peer-to-peer opportunistic data service system. However, the OPPay-based solution is impractical for a WiFi hotspot, as it incurs high fees and does not offer QoS measurement for sustaining a reliable service. A commercial project WinQ [10] has been in development since 2016. Advertised as a blockchain-enabled mobile WiFi hotspot, the solution was intended to operate on its own blockchain called QLC Chain [8]. We installed both the Android and iOS apps to discover that the system is activated only on testnet blockchain, which was practically unavailable.

Dynamic Speed Evaluation. QDASH was proposed for dynamic speed measurement [18], which is based on the assumption that the user traffic is available to the client connection handler. This requirement makes QDASH and its derivatives unsuitable for use by SmartWiFi clients. Xylophone [26] observes the behavior of TCP ACK and RST packets for speed measurement. Although the technique accurately estimates the bandwidth, it requires extended permissions for the client to capture TCP packets, which are usually not available on Android and iOS without rooting/jailbreaking.

8 Conclusion

In this paper, we proposed SmartWiFi, a smart contract-enabled WiFi hotspot system, which provides universal accessibility, cross-domain authentication, association of QoS and payment, and security enhancement. SmartWiFi utilizes a novel cryptographic mechanism, Hansa, to establish connection. Hansa provides low-cost off-chain execution by restricting otherwise unacceptable smart contract fees, and significantly reduces delays associated with smart contract interaction. To validate the feasibility of SmartWiFi system, we designed and implemented a SmartWiFi prototype using an Ethereum smart contract. The experimental results show that SmartWiFi exhibits low operational delays, minimum communication overhead, and small blockchain fees. We demonstrated that SmartWiFi is a scalable, secure, and efficient WiFi hotspot solution, which can be easily deployed in a variety of systems with minimal intervention. The limited adoption of cryptocurrencies and the volatility of their market prices can be further addressed through the use of stablecoin tokens, which we leave for future work.

Acknowledgement

We would like to thank the anonymous reviewers for providing valuable feedback on our work. This work was supported in part by National Science Foundation grants CNS1950171 and CNS-1949753.

References

1. eduroam - World Wide Education Roaming for Research & Education. https://www.eduroam.org/, accessed: 2020-05-10
2. ETH Gas Station. https://ethgasstation.info/, accessed: 2020-05-17
3. Cisco meraki for sp public wifi. http://marketo.meraki.com/rs/010-KNZ-501/images/Meraki_for_SP_Public_WiFi.pdf (2019), accessed: 2020-04-03

4. Cloud managed networking. https://www.arubanetworks.com/solutions/cloud-managed/ (2019), accessed: 2020-04-10
5. Digitalocean. https://www.digitalocean.com (2019), accessed: 2020-04-03
6. Infura: Scalable blockchain infrastructure. https://github.com/INFURA (2019), accessed: 2020-04-03
7. Passpoint. https://www.wi-fi.org/discover-wi-fi/passpoint (2019), accessed: 2020-04-03
8. Qlc chain. https://medium.com/qlc-chain/chain/home (2019), accessed: 2020-04-03
9. Ruckus cloud wi-fi. https://www.ruckuswireless.com/products/system-management-control/cloud-wifi (2019), accessed: 2020-04-03
10. Winq. https://winq.net/ (2019), accessed: 2020-04-03
11. Amazon Web Services, I.: Alexa top sites. https://docs.aws.amazon.com/AlexaTopSites/latest/MakingRequestsChapter.html (2019), accessed: 2020-04-03
12. Bonneau, J., Miller, A., Clark, J., Narayanan, A., Kroll, J.A., Felten, E.W.: Sok: Research perspectives and challenges for bitcoin and cryptocurrencies. In: 2015 IEEE Symposium on Security and Privacy. pp. 104–121 (2015)
13. Buterin, V.: A next-generation smart contract and decentralized application platform. white paper (2014)
14. Chiang, M.: Networked Life: 20 Questions and Answers, chap. How WiFi is different from cellular, pp. 406–409. Cambridge University Press (2012)
15. Corporation, V.H.: Vultr. https://www.vultr.com (2019), accessed: 2020-04-03
16. Eberhardt, J., Tai, S.: On or off the blockchain? insights on off-chaining computation and data. In: European Conference on Service-Oriented and Cloud Computing. pp. 3–15. Springer (2017)
17. M-Lab: Measurement lab speed test. https://speed.measurementlab.net (2019), accessed: 2020-04-03
18. Mok, R.K., Luo, X., Chan, E.W., Chang, R.K.: Qdash: a qoe-aware dash system. In: Proceedings of the 3rd Multimedia Systems Conference. pp. 11–22 (2012)
19. Molisch, A.F.: Wireless Communications. Second Edition, chap. 1, p. 14. John Wiley & Sons (2011)
20. Ookla, L.: Ookla lab speed test. https://www.speedtest.net (2019), accessed: 2020-04-03
21. Poon, J., Dryja, T.: The bitcoin lightning network: Scalable off-chain instant payments (2016)
22. Prasad, R., Dovrolis, C., Murray, M., Claffy, K.: Bandwidth estimation: metrics, measurement techniques, and tools. IEEE network **17**(6), 27–35 (2003)
23. Shi, F., Qin, Z., McCann, J.A.: Oppay: Design and implementation of a payment system for opportunistic data services. In: 2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS). pp. 1618–1628 (2017)
24. Signer, C.: Gas Cost Analysis for Ethereum Smart Contracts. Master's thesis, ETH Zurich, Department of Computer Science (2018)
25. Tsankov, P., Dan, A., Drachsler-Cohen, D., Gervais, A., Buenzli, F., Vechev, M.: Securify: Practical security analysis of smart contracts. In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security. pp. 67–82 (2018)
26. Xing, X., Dang, J., Mishra, S., Liu, X.: A highly scalable bandwidth estimation of commercial hotspot access points. In: 2011 Proceedings IEEE INFOCOM. pp. 1143–1151 (2011)