future internet

MDPI

Article

A Comparison of Blockchain Recovery Time in Static and Mobile IoT-Blockchain Networks

Yue Su [1], Kien Nguyen [1,2,*] and Hiroo Sekiya [1]

[1] Graduate School of Science and Engineering, Chiba University, 1-33, Yayoi-cho, Inage-ku, Chiba-shi 263-8522, Japan
[2] Institute for Advanced Academic Research, Chiba University, 1-33, Yayoi-cho, Inage-ku, Chiba-shi 263-8522, Japan
* Correspondence: nguyen@chiba-u.jp

Abstract: Many IoT-blockchain systems in which blockchain connections run on an infrastructure-based network, such as Wi-Fi or LTE, face a severe problem: the single point of failure (SPoF) (i.e., depending on the availability, an access point of an LTE base station). Using infrastructure-less networks (i.e., ad hoc networks) is an efficient approach to prevent such highly disruptive events. An ad hoc network can automatically restore blockchain communication using an ad hoc routing protocol, even if a node fails. Moreover, an ad hoc routing protocol is more efficient when considering the IoT nodes' mobility. In this paper, we first construct IoT-blockchain systems on emulated and real ad hoc networks with Ethereum and three ad hoc routing protocols (i.e., OLSR, BATMAN, and BABEL). We then evaluate the blockchain recovery time in static and mobile scenarios. The results show that BATMAN achieves the best blockchain recovery performance in all investigated scenarios because BATMAN only determines whether to switch a route by comparing the number of OGM packets received from a different next-hop. More specifically, in the small-scale real IoT-blockchain, BATMAN recovers at least 73.9% and 59.8% better than OLSR and BABEL, respectively. In the medium-scale emulated IoT-blockchain, the recovery time of BATMAN is at least 69% and 60% shorter than OLSR or BABEL, respectively.

Keywords: IoT; blockchain; OLSR; BATMAN; BABEL; recovery time

Citation: Su, Y.; Nguyen, K.; Sekiya, H. A Comparison of Blockchain Recovery Time in Static and Mobile IoT-Blockchain Networks. *Future Internet* **2022**, *14*, 330. https://doi.org/10.3390/fi14110330

Academic Editor: Paolo Bellavista

Received: 10 October 2022
Accepted: 9 November 2022
Published: 14 November 2022

1. Introduction

Blockchain is a distributed database that is based on an encrypted chain block structure, consensus mechanisms, and peer-to-peer communication [1]. Blockchain's characteristics of decentralization, transparency, and immutability have been applied to cryptocurrency [2], mobile communications [3], logistics [4], insurance [5] and in other areas. Recently, there has been increased interest in applying blockchain to the Internet of Things (IoT) [6] since both can support each other. On the one hand, the traditional centralized management approach faces challenges in working effectively in the IoT (e.g., providing security, privacy, etc.). On the other hand, because of its decentralized nature, blockchain provides permanent data preservation and tamper-proof features, which can assist the IoT in solving various security problems. There are several popular blockchain platforms, including Ethereum [7], ConsenSys Quorum [8], IOTA [9], and IBM Blockchain [10]. Among them, Ethereum is open-source and supports smart contracts. Hence, more and more studies have focused on Ethereum to build IoT-blockchain systems. The two most popular types of Ethereum are public blockchain and private blockchain. The public blockchain is open to everyone—users can freely access the blockchain network. The private blockchain is only used within a group or a private organization and only internal personnel can participate. The private Ethereum blockchain is more suitable for building IoT-blockchain systems.

Most existing IoT-blockchain systems are constructed on infrastructure-based underlying networks (e.g., Wi-Fi, LTE, LoRA), where a centralized node, such as an access

point and a base station, manages the communication between IoT devices. Once the control node collapses, the whole IoT-blockchain system inevitably breaks down (i.e., the single point of failure (SPoF) issue). In addition, the IoT-blockchain system faces another problem when considering mobility—if an IoT device moves out of the coverage of the central control node, it can no longer communicate with any node, making blockchain communication ineffective on the mobile node. These problems motivated us to develop an infrastructure-less network for IoT-blockchain systems. The infrastructure-less network (or ad hoc network) can decentralize the underlying network of the IoT-blockchain system and relax the dependence on the control node [11]. Moreover, when an IoT node moves out of its neighbor's communication range, it can still maintain communication in a multi-hop manner. In such networks, the IoT nodes find the intermediate node and forward the packets using ad hoc routing protocols. In previous work [12], we investigated the blockchain recovery time in a simple, static IoT-Ethereum blockchain system with optimized link state routing (OLSR) and "better approach to mobile ad hoc network" (BATMAN) protocols. The promising early results stimulated us to investigate the issue further.

Unlike other studies, we investigate here IoT-blockchain systems with Ethereum and ad hoc routing protocols in static and mobile scenarios. We consider a small-scale system with real IoT devices and an emulated one of larger scale. Then, we evaluate the recovery performance of blockchain communication on the IoT-blockchain systems. In addition to OLSR and BATMAN, we investigate one more ad hoc protocol, named BABEL [13]. We choose BABEL first since it has been proposed more recently than OLSR and BATMAN. Moreover, it has several unique features, such as its design based on several older routing protocols (e.g., DSDV, AODV, HSDV), while still providing link cost estimation as with OLSR. In [14], the authors compared these three routing protocols in a mesh network and found that both BATMAN and BABEL outperformed OLSR in terms of multi-hop performance and route rediscovery latency. Moreover, in [15], the authors compared OLSR, BATMAN, and BABEL in several ad hoc networks considering different network parameters. They found that BABEL performed best in terms of throughput whilst having the smallest overhead. Therefore, BABEL may have the potential to restore blockchain communication quickly. The contributions of the paper include the following:

- We built a real IoT-blockchain system using the private Ethereum blockchain, four Raspberry Pis, and three ad hoc routing protocols. Furthermore, we used the emulator Mininet-WiFi [16] to construct a bigger system with nine more IoT devices than the real one.
- We thoroughly compared the blockchain recovery time of OLSR, BATMAN, and BABEL in static and mobile scenarios.
- The evaluation results show that BATMAN achieved the best performance for blockchain recovery in our system—at least 69% and 59.8% better than OLSR and BABEL, respectively.

The remainder of this paper is organized as follows. Section 2 describes related work. Section 3 presents the background to IoT-blockchains and ad hoc routing protocols. In Section 4, we describe the methodology. In Section 5, the evaluation results are presented. Finally, Section 6 concludes the paper.

2. Related Work

There is increasing interest in applying blockchain to IoT. In [17], the authors introduced a repeatable and testable IoT blockchain application platform named PlaTIBART. They showed how PlaTIBART can be used to develop and analyze fault-tolerant IoT blockchain applications. In [18], the authors first illustrated how a combination of blockchain and IoT can simplify and bring benefits to modern supply chains. They then derived six research propositions and described how blockchain technology affects the scalability, security, information flow, traceability, and interoperability of IoT. The authors of [19] proposed a blockchain-based decentralized IoT self-counting voting system framework to solve fairness problems in the self-counting voting system, such as adaptability and

abortion problems caused by malicious voters. In [20], the authors focused on introducing blockchain as a decentralized technology into the transportation systems in smart cities. The vehicles can work together without the permission of a central control node. The data transfer between vehicles uses a peer-to-peer network where each node communicates directly with every other node and is verified by its associated endpoint node. Blockchain is also applied in the healthcare IoT. For example, remote patient monitoring technology faces severe privacy and security risks when transmitting data and recording data transactions. The security and privacy issues associated with these medical data may lead to delays in progressing patients' treatment. To solve this problem, the authors of [21] proposed the use of blockchain to provide secure management and analysis of medical data. However, blockchain is computationally expensive and requires high bandwidth and additional computing power. Therefore, the authors proposed a new framework for a revised blockchain model suitable for IoT devices.

When researchers have explored the combination of blockchain and IoT, they have paid little attention to the underlying network. Research has been primarily based on the most common underlying networks, such as infrastructure-based networks, which are prone to single point of failure problems. Utilizing an ad hoc network (i.e., using ad hoc routing) for the IoT-blockchain system can prevent this risk. Ad hoc routing protocols have previously been thoroughly investigated. However, they have rarely been studied with the blockchain. In [22], the authors provided an overview of ad hoc routing protocols that have been proposed in the literature. They presented performance comparison results for all routing protocols. The authors of [23] investigated and compared the performance of several routing protocols, including AODV, PAODV, CBRP, DSR and DSDV. They simulated various workloads and scenarios and showed that, although CBRP helps to reduce routing request packets, its overhead is higher than DSR due to its periodic hello messages. Compared with DSR and CBRP, AODV had the shortest end-to-end packet delay. PAODV showed a slight improvement over AODV. In [24], the authors implemented and analysed a testbed considering OLSR and BATMAN's link quality window size (LQWS) parameters. They evaluated the influence of mobility on a mobile ad hoc network's throughput. The authors also evaluated the performance of the testbed in terms of throughput, round-trip time (RTT), jitter, and packet loss. They found that TCP throughput improved when LQWS was reduced. In [25], the authors studied vehicular networks, where MANET routing protocols are typically designed and analyzed with 2-D scenarios. Since there is no guarantee of how these will support a 3-D topology, the authors evaluated and compared MANET routing protocols, including AODV, DSDV, and DSR and location-based routing. They found that topology-based protocols achieved acceptable transmission rates and path expansion performance, while location-based protocols achieved higher data rates. In our work, we choose OLSR, BATMAN, and BABEL because OLSR and BATMAN have been investigated and compared in different scenarios and networks. For example, the authors of [26] evaluated OLSR and BATMAN when low-power nodes transmitted VoIP data streams. In [27], the authors assessed the performance of OLSR and BATMAN by moving devices up and down different floors in a school building. As a novel routing protocol of the same type as OLSR and BATMAN, BABEL continues to demonstrate characteristics of some old routing protocols.

Most closely related studies have not researched blockchain communication recovery in the ad hoc IoT-Blockchain system. In [28], the authors focused on combining IoT and blockchain for smart home applications. However, they only investigated an infrastructure-based network and considered static scenarios. In [29], combining IoT and blockchain for UAVs in a real deploymentwas investigated and a key management mechanism based on blockchain technology was proposed. Although the authors of [11] investigated the recovery time in an ad hoc IoT-blockchain system, they only focused on a single ad hoc routing protocol: OLSR. Moreover, the evaluation was conducted on a small-scale network. To highlight the differences between previous studies and ours, we summarize them in Table 1. An earlier version of this study was published in [12], which only considered

the evaluation of two routing protocols (i.e., OLSR and BATMAN) in a static four-node emulated network. Here, we first extend the assessment with the new routing protocol, BABEL. Second, we deploy IoT-blockchain systems on actual IoT nodes with real blockchain and routing protocols. In addition, we investigate the performance in a larger network size for emulation. Finally, compared to the previous investigation, we also evaluate mobile scenarios that may occur in several IoT-blockchain applications.

Table 1. Comparison of this study with previous studies.

Underlying Network	Application	Evaluation Environment	Network State	Characteristic	Ref.
Infrastructure-based	Combining IoT and blockchain for smart home	Simulation	Static	Propose a blockchain-based smart home gateway architecture	[28]
Ad hoc	Combining IoT and blockchain for UAVs	Real	Mobile	Propose a blockchain-based distributed key management scheme for FANET	[29]
Ad hoc	Combining IoT and blockchain	Real	Static	Evaluate the recovery time of OLSR in a small network	[11]
Ad hoc	Combining blockchain and IoT	Emulation and Real	Static and mobile	Evaluate communication recovery performance of OLSR, BATMAN, BABEL	This study

3. Background of IoT-Blockchain with Ad Hoc Network

3.1. Ethereum Blockchain

In this investigation, the IoT-blockchain systems include blockchain applications and several IoT devices. Some examples of IoT-blockchains considered in this research include smart home (for static scenarios) and smart community (for mobile scenarios) [30]. In the former, the blockchain deployment is straightforward and has been investigated elsewhere. In the latter, the smart community utilizes wireless sensor networks, mobile networks, and other communication technologies to integrate security, property management and other systems to provide a safe, comfortable and convenient modern living environment for community residents. The protection of community residents' identity information faces security problems, such as privacy leakage and identity fraud. The high reliability, traceability, and difficulty of tampering with blockchain can protect the security of residents' identity information. A smart contract may trigger transactions when the resident's privacy information is updated or uploaded. In such a case, a single point of failure should be avoided. Moreover, if a failure does happen, the IoT-blockchain should be automatically recovered. In this investigation, we use Ethereum, which has a set of virtual machines (EVM) that can execute a Turing-complete scripting language and smart contract. A smart contract is a piece of code written on the blockchain, which can be automatically executed once an event triggers the terms in the contract. In other words, it can be triggered when conditions are met without human control. In the Ethereum blockchain, there are normal nodes and miner nodes. The miner nodes prove the correctness of a transaction and record it on the blockchain. Then, all the nodes will work together to maintain and store the transaction data through a consensus mechanism. The best-known and most widely used consensus mechanism is Proof of Work (PoW). In PoW, the miner nodes compute a

mathematical problem to find a random number (called nonce) that satisfies the block-out requirement. If the miner finds a nonce value that meets the need, it can successfully generate a new block on the blockchain and receive a corresponding reward. This process is known as mining. In an IoT-blockchain network, when a node completes a transaction, it uploads it to its transaction pool (txpool) to queue it for verification. It also broadcasts the transaction to its peer nodes via P2P communication. Its peers continue to broadcast transactions to their peers so that all nodes in the entire IoT-blockchain network can receive the transaction and deposit it into their txpools. The new blocks generated when a miner node verifies a successful transaction are also broadcast to all nodes so that all nodes are openly and transparently aware of all transaction information and block information across the blockchain.

Although blockchain brings its decentralized nature to the IoT, many IoT-blockchain systems still operate on a centralized underlying network. All IoT devices must communicate via a central node. The IoT-blockchain transmission and communication must go through the central node. However, once this central node encounters any failure or attack, the messages transmitted by any node will be interrupted and cannot guarantee even the most basic mutual communication. The IoT-blockchain system will lose its robustness and security based on such an underlying network. As mentioned, the underlying ad hoc network can avoid the problem with its decentralized features. In such a context, ad hoc routing protocols are also essential, enabling IoT devices to deliver packets to other nodes based on routing information. Even if there is a problem with the current routing path, the routing protocol will provide other viable routing options. The performance of ad hoc routing protocols in the underlying network also affects the upper-layer blockchain applications, so it is crucial to understand and evaluate the performance of different routing protocols in the IoT-blockchain network.

3.2. Ad Hoc Network and Routing Protocols

This section presents the three routing protocols investigated in this study: OLSR, BATMAN, and BABEL.

3.2.1. OLSR

OLSR adopts a multi-point relaying (MPR) mechanism based on the traditional link state algorithm to reduce the protocol overhead [31] In OLSR, nodes exchange control packets, including HELLO packets and topology control (TC) packets. OLSR performs distributed computing to establish a network topology. The node must know the information about its neighbor's link and whether the link connection is bidirectional. So OLSR will periodically broadcast the HELLO message (the time interval for sending the HELLO message in our configuration is four seconds) through the four-way handshake to sense and detect neighbors and determine the bi-directionality of the link between them. The node only needs to propagate the HELLO message to its one-hop node. When mastering the link and neighbor information, the node selects a part of its neighbor nodes as its own MPR and the node becomes an MPR selector (MS). The MPR is responsible for periodically distributing TC packets to the network, while other non-MPR nodes do not play this role. The MPR nodes will cover all the two-hop nodes of the MS nodes, so the TC packets forwarded by the MPR nodes will finally be received by all nodes in the network. Thus, they can know the topology information of the network. The important parameters of OLSR are listed in Table 2.

OLSR has an expected transmission count (ETX) indicator, which is closely related to the link's quality in the network and is calculated based on the link quality and neighbor link quality. First, OLSR judges the link quality between nodes based on the loss of OLSR packets a node receives from its neighbors. The link quality (LQ) indicates how good a given link between a node and its neighbors is in the direction from the neighbor to the node. For example, when two out of five packets are lost on the way from a node's neighbor to this node, three out of five packets are sent successfully by the neighbor. As a result, the

probability of successfully transmitting packets from the neighbor to this node becomes 40%, and this probability is the LQ. The neighbor link quality (NLQ), on the other hand, expresses how good a given link between a node and its neighbors is in the direction from that node to the neighbor. When judging the successful transmission rate of packets on the bidirectional connection between the node and its neighbors, it can be obtained by LQ × NLQ. ETX indicates how many transmissions are required to transmit the packet to the node or its neighbors, which can be calculated by $\frac{1}{LQ \times NLQ}$.

Table 2. OLSR parameters.

Parameter	Value	Meaning
HELLO_INTERVAL	Default: 2 s	Interval for hello packets
REFRESH_INTERVAL	Default: 2 s	Interval for nodes to keep track of the latest connectivity change
TC_INTERVAL	Default: 5 s	Interval for transmitting TC packets
NEIGHB_HOLD_TIME	Default: 6 s	Holding time of neighboring information
TOP_HOLD_TIME	Default: 15 s	Holding time of topology information

3.2.2. BATMAN

BATMAN is designed to let all nodes in the ad hoc network master the information of the best end-to-end path between other nodes [32]. Each node only perceives and maintains the best next-hop information for all other destination nodes. There is only one type of broadcast message: the origination message (OGM) in BATMAN. Every node broadcasts the OGM packets to inform neighboring nodes of its existence. These neighbors rebroadcast the OGM packets according to specific rules to inform their neighbors of the presence of the original originator. Therefore, the network will be gradually flooded with originator information. The OGM data packet is broadcast and forwarded by UDP; its data packet is 52 bytes and it contains at least the originator's address, the node address which transmits the packet, the time to live (TTL), and the sequence number. The OGM packets will be broadcast periodically (the default interval for sending OGM packets is one second). When an OGM packet is received by the node at least once, or exceeds the TTL value, it will not be sent again.

A unique data structure in BATMAN is called a sliding window, the size of which can be adjusted. When each node receives the OGM packet broadcast or it is forwarded by its next-hop node, it will store the sequence number of the OGM packet in the sliding window. When some OGM packet sequence numbers are stored within the size range of the sliding window, they are all regarded as newly received OGM packets. However, when the sequence numbers of some OGM packets are not within the sliding window range, they are regarded as old packets. The node will receive OGM packets from different next hops through various links. According to the neighbor ranking mechanism in BATMAN, the node will compare the number of newly received current OGM packets to know the link from which it receives the largest quantity of new OGM packets' sequence numbers as the best link. The one-hop node on this link is the best next hop for this node to reach other destination nodes. The main parameters of BATMAN are shown in Table 3.

Table 3. BATMAN parameters.

Parameter	Value	Meaning
OGM_INTERVAL	Default: 1 s	Interval for sending OGM packets
PURGE_TIMEOUT	Default: 200 s	Time for removing the node in BATMAN's database
WINDOW_SIZE	Proposed in RFC: 8	Size of the sliding window

3.2.3. BABEL

Babel is a loop-avoiding distance vector routing protocol for IPv6 and IPv4 and is based on the Bellman–Ford algorithm [13]. BABEL inherits the old routing protocols, such as DSDV and AODV, and also introduces an ETX indicator, such as OLSR. Such a design makes BABEL robust and efficient to use in unstable networks and able to select the appropriate link more intelligently. The characteristics of BABEL are as follows: First, it will not directly switch the link when the link quality changes, but will continue to use the originally used route, which can minimize the impact of route changes and switch. This impact is that route fluctuations may occur when nodes constantly switch routes between the destination node. Therefore, BABEL allows nodes to maintain the original routing path when facing multiple routes with similar link quality to the initial route and avoid switching back and forth on numerous routes. The second feature is that, when BABEL detects a failure in the original route, it will compulsorily request routing information and can quickly synchronize the information of the new route after the route is updated. BABEL achieves the goal of avoiding loops mainly by applying appropriate feasibility conditions. A feasibility condition (FC) means a route is feasible only if its metric is less than any previous route updates.

A BABEL node detects its neighbors by exchanging two kinds of information: HELLO messages and I Heard You (IHU) messages. Each node broadcasts a HELLO message (the default time to send a HELLO message is four seconds) to detect its neighbor nodes. When the node knows the existence of a neighbor, the node will also periodically send an IHU message (the default sending IHU time is twelve seconds) to the neighbor to tell it that it has received the HELLO message. When a node and its neighbor know the existence of each other, the node can also estimate the receiving cost of the link and share it with the neighbor node. In a network configured with BABEL, the node needs to periodically transmit the updated route message to all other nodes, making all nodes understand the whole network's topology and the receiving cost of every link. Some of the important parameters of BABEL are shown in Table 4.

Table 4. BABEL parameters.

Parameter	Value	Meaning
HELLO_INTERVAL	Default: 4 s (for wireless network)	Interval for sending hello packets
IHU_TIMEOUT	Default: 12 s	Interval for advertising IHU packets
UPDATE_INTERVAL	Default: 16 s	Interval for advertising or withdraws routes

4. Methodology

This section first introduces the construction of IoT-blockchain systems. We then present the evaluation methodology.

4.1. Constructing IoT-Blockchain System

4.1.1. Ad Hoc Network

We describe the construction of the IoT-blockchain system from the bottom layer to the top. In the real system, we connect four Raspberry Pis (RPs), each of which operates in ad hoc mode. We configure the ad hoc network by modifying the system file (i.e., in the /etc/network directory) of RPis. We can add ad hoc network-related content, such as ESSID, channel, IP address, etc., to the file. In addition to the real system, we also build a bigger one using an emulator. We leverage the network emulator Mininet-WiFi, which can emulate Wi-Fi nodes while supporting real IP, transport, and application layers. First, we create nine nodes to simulate IoT devices, which operate according to the IEEE 802.11 g standard. None of them are connected to any access points and their wireless interfaces are set to the ad hoc mode. We also adjust the signal range propagated by each node so that

some nodes in this ad hoc network need the help of relay nodes to communicate with other nodes that are not within the same signal range. At this stage, we complete the physical connection for the system.

Then in the IoT-blockchain system, we introduce three different ad hoc routing protocols. First, we use the configuration file (i.e., olsrd.conf) on each node to configure OLSR. In the file, we add options representing properties of OLSR in our systems. We mainly set the sending time interval of HELLO and TC packets, the hold time for this control information, and so on. We store this configuration file in the /etc directory and then use OLSR's debug command to check if OLSR is successfully enabled. For the BATMAN configuration, we choose to install the BATMAN IV version, named BATMAN ADV. This BATMAN version runs on layer2 as a kernel module running on the Linux system. It uses raw Ethernet frames to transmit routing information. To configure and debug this kernel module, BATMAN ADV introduces a tool called batctl. In the configuration, we mainly use the batctl tool and several commands to configure BATMAN ADV on the wireless interface of each node. During the configuration process, the batctl tool will create a virtual interface bat0 for each node and assign a new IP address to this interface. This interface is a BATMAN ADV instance. The configuration of BABEL requires compiling various script files in the BABEL folder. Then, we install BABEL's default configuration file with one command. Most importantly, we must stop the network manager when configuring BABEL on each node interface, as this may cause BABEL to publish all cached route entries. Finally, we can enable BABEL with a command on each node's wireless interface.

4.1.2. Blockchain Deployment

After establishing the underlying network in real and emulated IoT-blockchain systems, we deploy the Ethereum private blockchain to it. We select the private blockchain because, compared to the public blockchain, the number of devices that join the private blockchain is limited. Devices process less workload (i.e., processing transactions and blocks), which means that each device consumes less energy and resources. We first install Geth, the classic client for Ethereum, and then create folders for each device to store its information. In this folder, we make a genesis block file to define and create the first block in the blockchain. We also create an account for each device stored in the folder. Then, we can enable the device in the blockchain and enter the console through a series of options (e.g., device IP address, blockchain network id, the port number for devices' P2P communication, etc.). When each device successfully enters the blockchain console, we query the details of the devices enode in the blockchain, through which we can connect devices. We choose a stable connection, that is, store the enode information of other devices in each device. When the device starts, it will automatically obtain a blockchain connection with other devices, eliminating the need to manually connect while the device is running. Once each device has a blockchain connection, we deploy a smart contract. The main content of this smart contract is to initialize a random value and recursively multiply it by a fixed value. To make the smart contract work, we have the device trigger the function in the smart contract by passing a transaction file containing the fixed value. Every time the device sends a transaction file, its terminal interface will display the transaction's hash value and the geth log will record more detailed transaction information and transaction processes in the device account folder.

4.2. Evaluation Methodology

4.2.1. Network Performance Evaluation

When creating the IoT-blockchain system, it is important to investigate the network performance to confirm the correctness, as well as the capacity, of the systems. Similar to other studies, we also focus on testing the throughput and RTT. For the former, we use iperf3, while ping is adopted for RTT measurements. For the throughput or RTT tests, we select two devices: the sender and the receiver in either the real or emulated IoT-blockchain

systems. Then we conduct the measurement multiple times and collect and calculate the results.

4.2.2. Recovery Evaluation

We then focus on evaluating blockchain recovery under different scenarios. We define the recovery time as the time for the routing protocol to resume routing plus the time for blockchain transactions to continue synchronization in the IoT-blockchain system. When the system is static in both real and emulated IoT-blockchain systems, we fail the devices by the following methods: First, we use the tcptraceroute tool to query the routing table to find the relay nodes between the sender and the receiver. We find that OLSR and BABEL always choose a fixed routing path, while BATMAN switches routes due to the change in the number of OGM packets received. Therefore, when using OLSR and BABEL, we let the interface of a fixed relay node close to emulate the failure. When using BATMAN, we shut down a relay node that is frequently selected. As a result, the path between the sender and the receiver will lose connectivity and the ability to forward packets. In this case, the ad hoc routing protocols will re-update the routes to maintain the new network topology. Each routing protocol has a different process for finding new routes, as described below.

OLSR needs to know and maintain the topology and routing changes of the whole network. When there is a problem with the relay node between the sender and the receiver, the MPR node selected by the sender can no longer perform its duty of delivering network topology information (i.e., TC packets). Therefore, the sender needs a new MPR node to be responsible for forwarding packets. As mentioned in the previous content, in the static IoT-blockchain system we shut down the sender's MPR. The sender needs to detect that it has lost connectivity with its neighbor node by sending multiple HELLO messages and then spending some time identifying a new MPR node. The movement of the receiver in a mobile IoT-blockchain system also causes a shift and re-selection of the sender's MPR. Finally, all nodes need to be aware of the new changes in the network topology before they completely resume routing. Based on the recovery process of OLSR, the recovery time with OLSR can be calculated as Equation (1) below:

$$T_{recovery} = T_{MPR\_lost} + T_{NEIGHB\_HOLD\_TIME} + T_{pure\_blockchain}, \quad (1)$$

where T_{MPR_lost} means the time for the sender to realize its current MPR has lost; it is equal to three or four times the HELLO_INTERVAL. $T_{pure\_blockchain}$ means the recovery time of the blockchain itself.

BABEL is more flexible than OLSR in that it can mandatorily request routing information proactively when it knows that there is a problem with the routing path it has been choosing. But BABEL also needs some time to recover because, when a node detects its neighbors to build a route, it needs to send HELLO and IHU packets. BABEL also sends packets about updating neighbors when it has identified a new neighbor. Equation (2) represents the recovery time when using BABEL in the IoT-blockchain system. It is expressed as:

$$T_{recovery} = T_{neighbor\_update} + T_{pure\,blockchain}, \quad (2)$$

where $T_{neighbor\_update}$ means the time for the sender to update its new neighbor. It equals three times the HELLO_INTERVAL. Several IHU packets and one updated packet are included in the time it takes for BABEL to realize that there is a problem with the current neighbor node and find a new relaying neighbor node.

BATMAN detects its neighbors and builds a network route only by OMG packets. BATMAN does not require the sender to know the topology change of the whole network, but only the sender needs to know its best next-hop to the receiver. The sender selects the best next-hop by comparing the number of OGM packets received from different next-hop nodes. So when the current best next-hop has a problem, and the sender gets no more OGM packets from it, the sender will choose the new best next-hop once the number of OGM packets received from another next-hop exceeds that of the OGM packets received from the

current best next-hop which has a problem. More specifically, BATMAN implements route translation based on the number of OGM packets in the sliding window range. BATMAN switches routes immediately when the number of OGM packets received from another next-hop is even one more than that of the OGM packets obtained from the current selected best next-hop. Equation (3) shows the recovery time of BATMAN.

$$T_{recovery} = T_{switch\_next\_hop} + T_{pure\_blockchain}, \quad (3)$$

where $T_{switch\_next\_hop}$ means the time for the sender to switch the best next-hop to the receiver. It changes into the range between $OGM\_INTERVAL$ and $WINDOW\_SIZE \times OGM\_INTERVAL$.

When there is no blockchain connection, we can first have ICMP packets transmitted between them by a sender pinging the receiver. This is followed in a manner that causes temporary communication failure between the receiver and the sender mentioned above. Using the Wireshark tool, we can see that the sender keeps sending ICMP request packets but it cannot receive ICMP response packets from the receiver during communication failure. However, when the routing protocol resumes communication, the sender can receive ICMP response packets again. So, we consider that the time interval between when the sender does not receive the ICMP response packet and when this packet is received again is ad hoc routing recovery time. In addition to the time it takes for the ad hoc routing protocol to recover the route, the blockchain itself also takes time to recover. We obtain the recovery time for the entire IoT-blockchain system as follows: We let the sender send one transaction per second and the transaction is quickly broadcast to the receiver device. The synchronization of transactions between them can be said to be timely. However, when the IoT-blockchain system fails, as described above, some transactions sent by the sender cannot be broadcast to the receiver promptly. At this point, the number of transactions queued in the sender's txpool continues to increase, while the number of transactions queued in the receiver's txpool remains unchanged. Once the routing protocol updates the routing table, the device can learn the new topology of the network, then the sender and receiver can communicate again. Once routing and blockchain are restored, then the IoT-blockchain system's communication resumes. At this point, the sender re-broadcasts all previously unsynchronized transactions in its txpool to the receiver and the receiver conveniently resumes synchronization of transactions with the sender. We consider the interval between the time when the receiver is unable to synchronize transactions and the time when transactions are resumed again as the recovery time. We calculate the recovery time based on the detail information and corresponding timestamps in the geth log of the receiver. For better understanding, we describe the method of obtaining the recovery time in the IoT-blockchain system in the flowchart in Figure 1.

In [33], the authors define the workflow when broadcasting transactions between a blockchain node and its peer nodes. When a sender node sends a transaction file that can trigger a smart contract, the transaction is submitted to the sender's txpool and queued for validation. At this point, the sender's geth log will record "Submitted transaction", indicating that the sender has uploaded a transaction to its txpool and is waiting to be validated and broadcast. When the transaction is successfully verified, the sender will broadcast it to all peer nodes. When the peer node receives the broadcast transaction, the transaction is still stored in its txpool and queued for validation. At this point, the peer enters the "Promoted queued transaction" phase, and this message is recorded in the peer's geth log. According to the above workflow, we focus on the timestamp corresponding to the "Promoted queued transaction" message in the receiver's geth log. Based on the change in the number of transactions over time, we can calculate the time when the transaction resumes synchronization, which is when the IoT-blockchain restores blockchain communication.

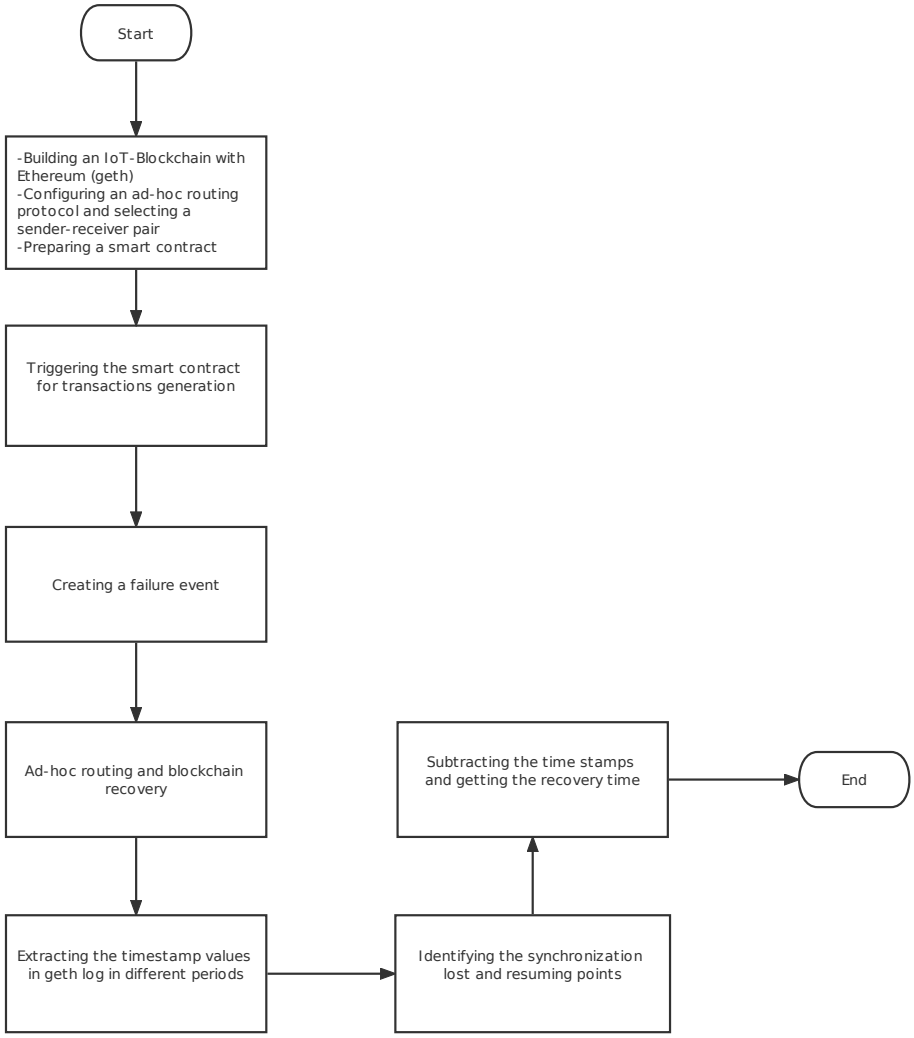


Figure 1. Method of obtaining IoT-blockchain's recovery time.

5. Evaluation

5.1. Real IoT-Blockchain System

5.1.1. Static Scenario

The real IoT-blockchain system includes four RPis deployed on the fifth floor of Building 1 of the Faculty of Engineering, Chiba University, Japan. Each node runs on the Ubuntu Mate 20.04 with the Wi-Fi card in ad hoc mode. The detailed information of each node and ad hoc routing protocol are shown in Table 5. Figure 2a presents the system's connection diagram and the distance between nodes. The distances between device1 and device2 and device3 and device4 are all 8 m. While the distances between device1 and device4, and device2 and device3 are 41 m. The black dashed lines between the devices indicate the ad hoc connection. We let device2 and device4 join the blockchain network; the red dotted line in the figure indicates the blockchain connection between them. We have device2 as the sender and device4 as the receiver of transactions. We find that, when using all three routing protocols, device2 always chooses the same path to send packets to device4 (i.e., device2-device1-device4). When evaluating the recovery time, we shut down the relay node between the sender and receiver (i.e., device1) (e.g., by using a specific Linux command).

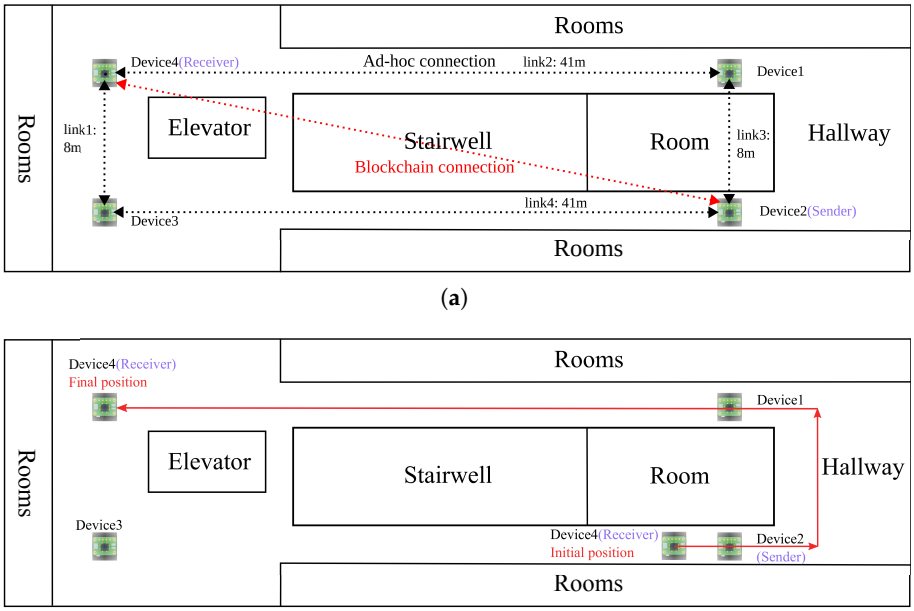


Figure 2. Real IoT-blockchain system evaluation. (**a**) Static scenario; (**b**) Mobile scenario.

Table 5. RPi, Ethereum, and routing configurations.

Raspberry Pi	model 4B
OS	Ubuntu Mate 20.04 LTS
Linux kernel verison	5.4.0
CPU	Quad core Cortex-A72@1.5 GHz
Ethereum	Geth 1.10.9-stable-eae3b194
OLSR	olsrd 0.9.9
BATMAN	batman IV
BABEL	babeld 1.12.1

We first evaluate the network performance of the static IoT-blockchain system. We measure TCP throughput, RTT on different links, and RTT from the sender to the receiver. Each throughput or RTT experiment is repeated ten times. Each RTT experiment is conducted with 50 ICMP packets via ping. Since OLSR, BATMAN, BABEL share the same performance, we present the typical results in Figure 3, in which Figure 3a,b show the throughput and RTT, respectively. At each condition, the y-axis shows the maximum, minimum, and average. Note that, as shown in Figure 2a, we label the name of each link and the two paths from the sender to the receiver. They are path1 (i.e., device2-device1-device4) and path2 (i.e., device2-device3-device4). The results show the consistency between RTT and throughput and the dependence of throughput on distance.

We then evaluate the recovery performance of three ad hoc routing protocols in the IoT-blockchain system. We conduct ten experiments to obtain the average, maximum, and minimum values of the blockchain recovery time when using OLSR, BABEL, and BATMAN. We collect from the geth logs of the receiver how the number of transactions in its txpools varies over time. Then we can calculate the recovery time of communication in the IoT-blockchain system. The results are shown in Figure 4a, where the protocols are shown in the x-axis. We can see that BATMAN has the lowest recovery time. This is because when the relay node (i.e., device1) failed, the sender no longer received OGM packets broadcast or forwarded from device1. However, the sender also receives OGM packets directly from another node (i.e., device3). Once the number of OGM packets received from device3 exceeds that obtained from device1, the sender immediately switches to a new route. To get a closer observation of the recovery process, we plot the time consumed by each process's

component in Figure 4b. In the figure, we use stacked bars to display the average values of different parts of the recovery time when using each routing protocol. For OLSR, T_{MPR_lost} and $T_{NEIGHB\_HOLD\_TIME}$ accounted for a total of 60.6% and $T_{pure\_blockchain}$ for 39.4%. For BABEL, $T_{neighbor\_update}$ accounted for 83.3% and $T_{pure\_blockchain}$ for 16.7%. For BATMAN, $T_{switch\_next\_hop}$ accounted for 69.2% and $T_{pure\_blockchain}$ accounted for 30.8%.

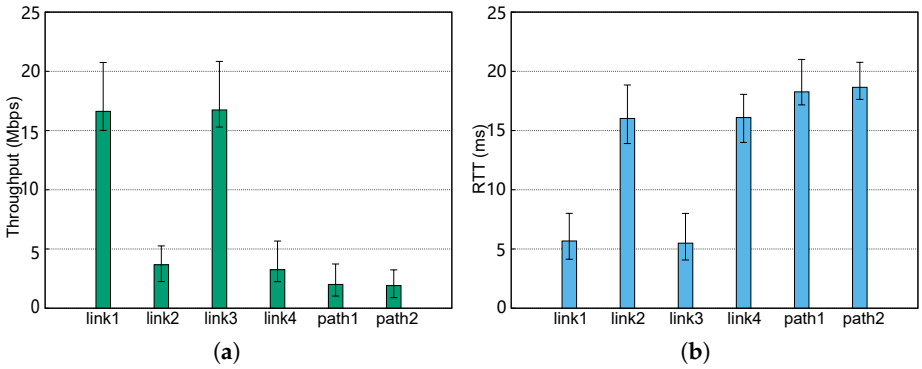


Figure 3. Throughput and RTT of real IoT-Blockchain system. (**a**) Throughput; (**b**) RTT.

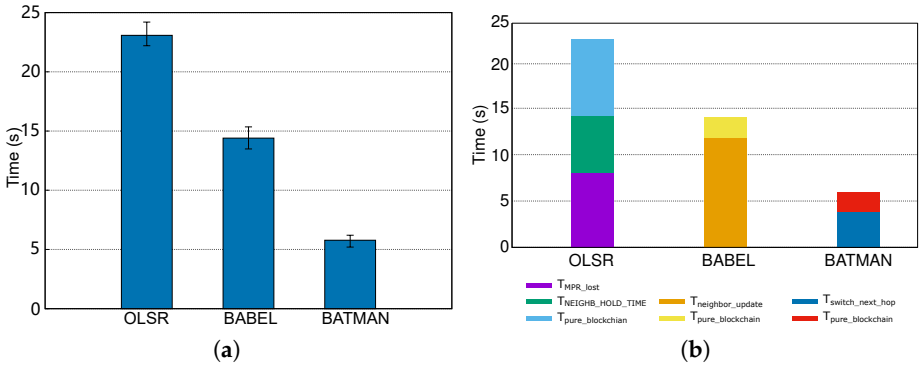


Figure 4. Recovery time of real IoT-blockchain in static scenario. (**a**) Recovery time; (**b**) Average recovery time with components.

5.1.2. Mobile Scenario

In the mobile scenario, we put the receiver on a chair and move it around the floor. The red arrow in Figure 2b shows the moving trajectory. The receiver starts moving at the initial position and stays at the final one marked in the figure. When the receiver is within the communication range of the sender, they can communicate directly. Communication is lost when the receiver moves out of range of the sender's coverage signal. Then, the ad hoc routing protocols on the devices need to find a new path. It is obvious that the routing protocols will select device1 as a relay node for the new route, thus restarting the blockchain communication between the sender and the receiver.

We use the method in the previous section to check and calculate the blockchain recovery time in the IoT-blockchain system. We find the information and timestamp in the receiver's geth log corresponding to when some transactions are broadcast simultaneously. The recovery time results with the three ad hoc routing protocols are shown in Figure 5a. Similar to the static scenario, the results show that BATMAN has better recovery performance than OLSR and BABEL. It is 73.9% faster than OLSR and 61.8% faster than BABEL. During its recovery, the receiver with BATMAN only needs to compare the number of OGM packets received from different relay nodes to select a new route quickly. The values of different components of each recovery time with each ad hoc routing protocol are shown in Figure 5b. For OLSR, T_{MPR_lost} and $T_{NEIGHB\_HOLD\_TIME}$ are about 54.5% of the total recovery time, while $T_{pure\_blockchain}$ occupies 45.5%. For BABEL, $T_{neighbor\_update}$ and

$T_{pure\_blockchain}$ are about 68.4% and 31.6% of the total recovery, respectively. In BATMAN, $T_{switch\_next\_hop}$ consumes 59.8% of the time, and $T_{pure\_blockchain}$ about 40.2%.

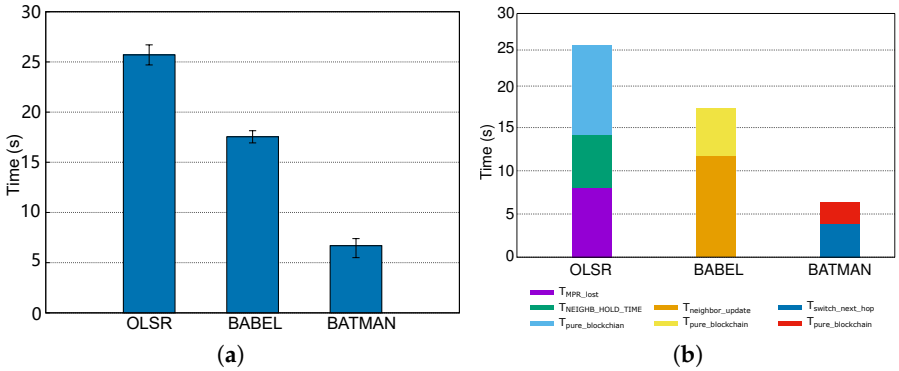


Figure 5. Recovery time of real IoT-blockchain in mobile scenario. (**a**) Recovery time; (**b**) Average recovery time with components.

5.2. Emulated IoT-Blockchain

5.2.1. Static Scenario

The previous evaluation provides insights into the simple IoT-blockchain system with the protocols. This section investigates a bigger system, as shown in Figure 6, where the IoT nodes form a 3 × 3 grid topology. We chose the grid topology because it is a classic topology for investigating ad hoc routing protocols. In the system, each IoT device can have at least two neighbors and the ad hoc routing protocols have multiple choices of routing paths. Our topology has also been investigated in [34], where the authors built wireless sensor networks with a nine-node grid in an indoor environment. We consider placing the devices at different distances from each other because, in most cases, the network nodes are at different, or even very random, distances. In Figure 6, each IoT device is 50 m away from its neighbors in the horizontal direction and 30 m away from its neighbors in the vertical order. In Figure 6a, the black dotted line represents the ad hoc connection between each IoT device. Figure 6b shows the design of the blockchain network; the red dotted lines represent the blockchain connection. All IoT devices join the blockchain and are fully connected. The environment and software for constructing the system are listed in Table 6. We use Mininet-WiFi on an Ubuntu 20.04 machine, Ethereum, and the routing protocols are similar to the ones in the real IoT-blockchain system.

Similar to the previous section, we first measure the system's network performance; the results are shown in Figure 7. Figure 7a,b shows the throughput and RTT between different nodes in the IoT-blockchain system, respectively. We investigated not only the performance of the two neighboring nodes' links, but also the link between the sender and the receiver. From the results, we can see that the performance of the emulated IoT-blockchain system is the same when the links have similar roles. The maximum throughput between a node and its one-hop neighbor node is about 7 Mbps, while the minimum RTT is about 0.5 ms. When the node has more hops (e.g., four hops between the sender and the receiver), the throughput value is lowest and the RTT value is highest, i.e., the average throughput is 2.695 Mbps, and the average RTT is 2.75 ms.

Table 6. Emulated environment.

OS	Ubuntu 20.04.3 LTS
CPU	Intel Core i7-8565U CPU@1.80 GHz × 8
Mininet-WiFi	version 2.6
Ethereum	Geth 1.10.9-stable-eae3b194
OLSR	olsrd 0.9.9
BATMAN	batman IV
BABEL	babeld 1.12.1

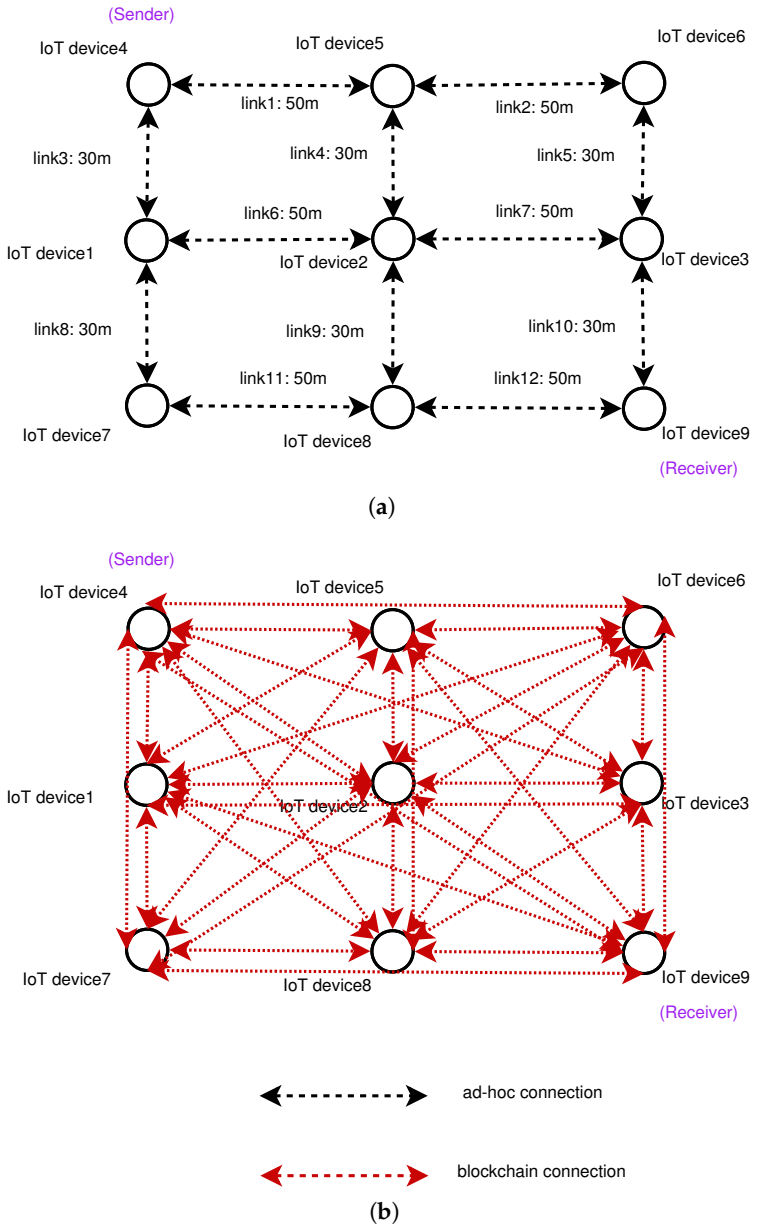
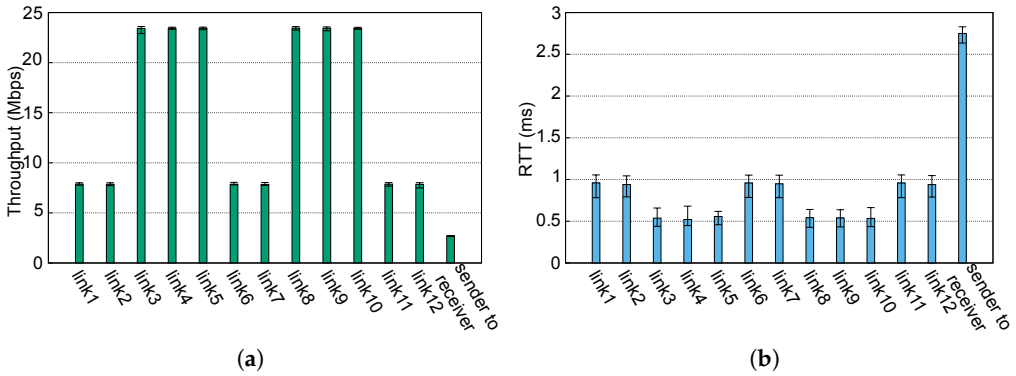


Figure 6. Emulated IoT-blockchain system. (**a**) Ad-hoc network connection; (**b**) Blockchain connection.

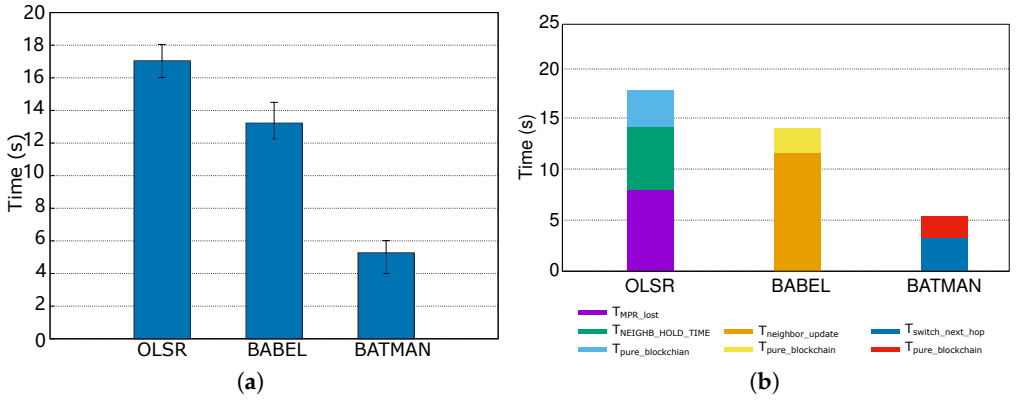
In our system, to interact with the smart contract, one of the IoT devices sends a transaction containing the fixed value. In this evaluation, IoT device4 acts as the sender sending the transaction file and IoT device9 acts as the receiver. When evaluating recovery performance, we must understand the routing path between the sender and receiver when using each routing protocol. We found that with OLSR, IoT device4 always chose the same route to propagate packets to IoT device9 (i.e., IoT device4-IoT device1-IoT device2-IoT device3-IoT device9). In the case of BABEL, IoT device4 also selects a fixed path to route to an IoT device9 like OLSR. However, BATMAN is different from the other two protocols. The routing path from IoT device4 to IoT device9 changes, but we find that IoT device4 frequently selects a route to IoT device9 (i.e., IoT device4-IoT device1-IoT device2-IoT device3-IoT device9). We shut down one of the relay nodes between the sender and the receiver to make the path fail. With OLSR and BATMAN, we shut down IoT device2 while the BABEL IoT device6's interface is down. We then evaluate and collect the communication recovery time of the IoT-blockchain system. Figure 8a shows the average, maximum, and minimum recovery time for the ten experiments in the static scenario with the IoT-blockchain system and three routing protocols. The results show that BATMAN

achieves the shortest recovery time. It has an average recovery time of 5.2656 s, which is 69% faster than OLSR and 60% quicker than BABEL. When the sender transmits the transactions to the receiver, the failure of one of the two-hop relay nodes, device2, prevents the sender from continuing to receive OGM packets from it. However, the sender keeps receiving OGM packets from the other two-hop relay nodes (e.g., IoT device6 and IoT device7). Once the number of OGM packets received from the two nodes is more significant than that obtained from IoT device2, BATMAN can select a new relay node. A closer examnination of the different components of the average recovery time is provided in Figure 8b. In OLSR, T_{MPR_lost} and $T_{NEIGHB\_HOLD\_TIME}$ consumed 82.2% of the total recovery time in this evaluation. On the other hand, the value of $T_{pure\_blockchain}$ is about 17.8%. For BABEL, $T_{neighbor\_update}$ occupied 90.8% of the total process's time and $T_{pure\_blockchain}$ occupied 9.2%. For BATMAN, $T_{switch\_next\_hop}$ was about 57% of the total recovery time and $T_{pure\_blockchain}$ was about 43%.



(**a**) (**b**)

Figure 7. Throughput and RTT of emulated IoT-blockchain system. (**a**) Throughput; (**b**) RTT.



(**a**) (**b**)

Figure 8. Recovery time of emulated IoT-blockchain in static scenario. (**a**) Recovery time; (**b**) Average recovery time with components.

5.2.2. Mobile Scenario

In this section, we investigate blockchain recovery when the receiver moves. The receiver gradually starts from a position close to the sender and has a speed of 1 m per second. The yellow arrow indicates the direction of movement in Figure 9, where Figure 9a shows the starting point and Figure 9d shows the final destination. When the receiver is within the signal range of the sender, they can communicate directly. However, when the receiver moves outside the sender's signal range, the first communication failure occurs, and the routing protocols need to recalculate and find a new one-hop relay node. Moreover, the second communication failure occurs when the receiver continues to move outside the communication range of the first-hop relay node. At this point, the routing protocol looks for a new second-hop relay node from the sender to the receiver again. Finally, the third communication failure occurs when the receiver moves outside the communication

range of the sender's second-hop relay node. The IoT-blockchain system performs a final communication recovery, finding a third-hop relay node from the sender to the receiver. The recovery time results for the different occasions in the emulated mobile IoT-blockchain system are shown in Figure 10a. From the results, we observe that, in our nine-node IoT-blockchain system, the receiver's movement causes three occasions of communication recovery. The result of each communication recovery shows that BATMAN is the least time-consuming. On the first occasion, BATMAN takes 6.22 s to recover, which is 72.3% and 72.1% faster than OLSR and BABEL, respectively. For the second occasion, the recovery time of BATMAN is 5.92 s, which is 71.35% quicker than OLSR and 70.4% faster than BABEL. On the last occasion, BATMAN is 5.95 s, 73.3% and 72.36% faster than OLSR and BABEL, respectively. Similar to the previous evaluation, we plot the average values with different components of the recovery process with the protocols in Figure 10b. We can see that OLSR's T_{MPR_lost} and $T_{NEIGHB\_HOLD\_TIME}$ is about 65.2% of the total recovery time and $T_{pure\_blockchain}$ is 34.8%. With BABEL, $T_{neighbor\_update}$, in this case, is about 57.3% of the recovery time and $T_{pure\_blockchain}$ consumes the remaining 42.7%. With BATMAN, $T_{switch\_next\_hop}$, $T_{pure\_blockchain}$ are 49.8% and 50.2% of the recovery time, respectively.

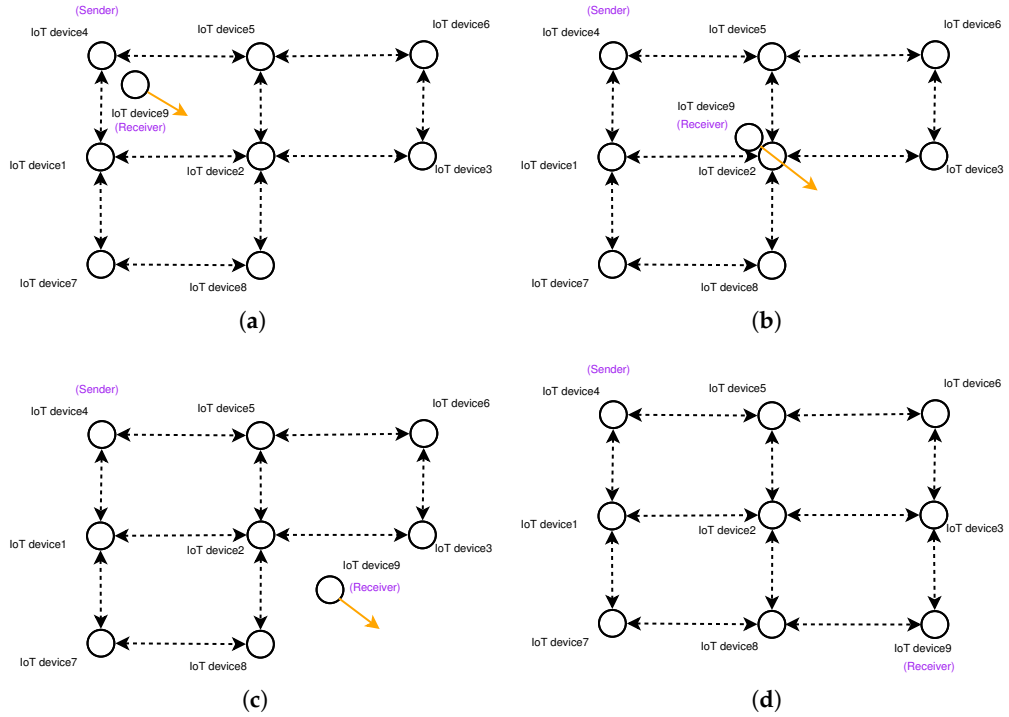


Figure 9. Emulated IoT-blockchain system with a mobile node. (**a**) Receiver's initial position; (**b**) During the movement (1); (**c**) During the movement (2); (**d**) Final position.

On the basis of the experimental results, we can see that BATMAN is faster than OLSR and BABEL in resuming blockchain communication. This is because, when using BATMAN, the sender only transmits OGM packets to probe its neighbor nodes. In contrast to OLSR, BATMAN does not require the sender to know the topology of the entire network. It only needs to know its best next hop to reach the receiver. The means of knowing the best next-hop is determined by the number of OGM packets the sender receives from different one-hop nodes transmitted or forwarded. These OGM packets are deposited in a sliding window between the sender and its one-hop neighbor nodes. When the number of OGM packets received from a one-hop node within the sliding window is more significant than those received from other one-hop neighbor nodes, the sender identifies this node as its best next-hop to the receiver. So once the sender's current best next-hop fails, it does not send any more OGM packets to the sender but keeps the number of OGM packets before the failure. However, the other next-hops of the sender are still normal, so they will continue to

send OGM packets to the sender. Even if the number of OGM packets received from other next-hops is one larger than the number received from the failed best next-hop, BATMAN will immediately switch the route and choose the one with better link quality.

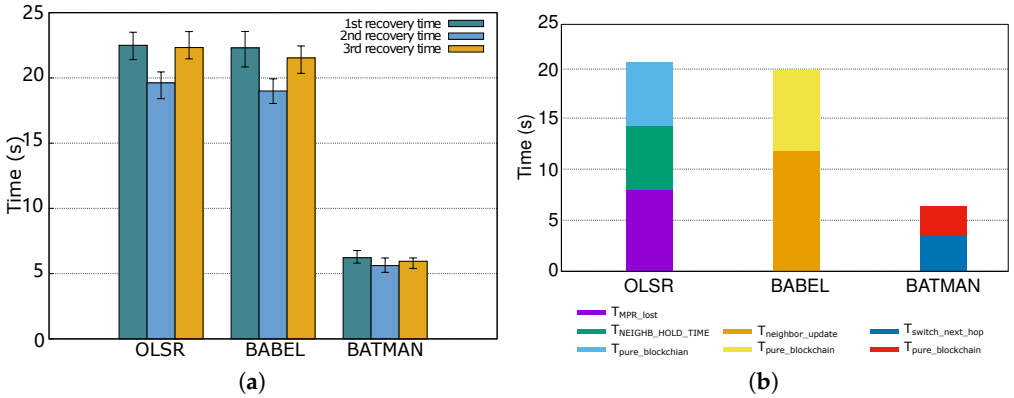


Figure 10. Recovery time of emulated IoT-blockchain in mobile scenario. (**a**) Recovery time; (**b**) Average recovery time with components.

For BABEL, the recovery time is the second smallest among the three. This is, firstly, because it does not send only one type of packet to probe its neighbors to establish routes like BATMAN. Instead, it sends two packages, HELLO and IHU, and it takes time to enable the nodes and neighboring nodes to recognize each other's existence and establish connections. BABEL has a faster recovery time than OLSR because, when the sender realizes a failure with the selected link, it requests new routing information instead of waiting to receive new topology information about the network, as in OLSR.

6. Conclusions

This study investigates the recovery time of IoT-blockchain systems with infrastructureless underlying networks that can avoid SPoF. We first create emulated and real IoT-blockchain networks with the private Ethereum blockchain and three ad hoc routing protocols, OLSR, BATMAN, and BABEL. We then extensively evaluate the recovery time of the systems in static and mobile scenarios. The experimental results show that BATMAN has the best ability to recover the blockchain communication in a real IoT-blockchain system. More specifically, it recovers 74.9% faster than OLSR and 59.8% faster than BABEL in the static scenario and 73.9% faster than OLSR and 61.8% faster than BABEL in the mobile one. In the emulated IoT-blockchain system with more nodes, BATMAN still maintains its outstanding features. In the static scenario, BATMAN achieves 69% and 60% faster recovery than OLSR and BABEL, respectively. BABEL, in turn, restores faster than OLSR, which has the second-lowest recovery time. In a bigger network with node mobility, BATMAN outperforms the others and can recover the blockchain in 6.22 s, 5.92 s, and 5.95 s, following the different stages, respectively

Author Contributions: Conceptualization, Y.S. and K.N.; methodology, Y.S. and K.N.; software, Y.S.; validation, Y.S. and K.N.; formal analysis, Y.S.; investigation, Y.S. and K.N.; resources, Y.S.; writing—original draft preparation, Y.S. and K.N.; writing—review and editing, K.N. and H.S.; supervision, K.N. and H.S.; project administration, K.N.; funding acquisition, K.N. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported in part by the Japan Society for the Promotion of Science (JSPS) under Grant 20H0417; and in part by the Japan Science and Technology Agency (JST) through the establishment of university fellowships towards the creation of science technology innovation, Grant Number JPMJFS2107.

Data Availability Statement: Not applicable, the study does not report any data.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Yli-Huumo, J.; Ko, D.; Choi, S.; Park, S.; Smolander, K. Where is current research on blockchain technology?—A systematic review. *PLoS ONE* **2016**, *11*, e0163477. [CrossRef] [PubMed]
2. Vranken, H. Sustainability of bitcoin and blockchains. *Curr. Opin. Environ. Sustain.* **2017**, *28*, 1–9. [CrossRef]
3. Tahir, M.; Habaebi, M.H.; Dabbagh, M.; Mughees, A.; Ahad, A.; Ahmed, K.I. A review on application of blockchain in 5G and beyond networks: Taxonomy, field-trials, challenges and opportunities. *IEEE Access* **2020**, *8*, 115876–115904. [CrossRef]
4. Tijan, E.; Aksentijević, S.; Ivanić, K.; Jardas, M. Blockchain technology implementation in logistics. *Sustainability* **2019**, *11*, 1185. [CrossRef]
5. Gatteschi, V.; Lamberti, F.; Demartini, C.; Pranteda, C.; Santamaría, V. Blockchain and smart contracts for insurance: Is the technology mature enough? *Future Internet* **2018**, *10*, 20. [CrossRef]
6. Reyna, A.; Martín, C.; Chen, J.; Soler, E.; Díaz, M. On blockchain and its integration with IoT. Challenges and opportunities. *Future Gener. Comput. Syst.* **2018**, *88*, 173–190. [CrossRef]
7. Wood, G. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum Proj. Yellow Pap.* **2014**, *151*, 1–32.
8. Baliga, A.; Subhod, I.; Kamat, P.; Chatterjee, S. Performance evaluation of the quorum blockchain platform. *arXiv* **2018**, arXiv:1809.03421.
9. Nakada, R.; Li, Z.; Pei, T.; Nguyen, K.; Sekiya, H. An iota-based micropayment system for air quality monitoring application. In Proceedings of the 2021 IEEE 94th Vehicular Technology Conference (VTC2021-Fall), Norman, OK, USA, 27–30 September 2021; pp. 1–6.
10. Kamath, R. Food traceability on blockchain: Walmart's pork and mango pilots with IBM. *J. Br. Blockchain Assoc.* **2018**, *1*, 3712. [CrossRef]
11. Chen, X.; Tian, S.; Nguyen, K.; Sekiya, H. Decentralizing private blockchain-iot network with olsr. *Future Internet* **2021**, *13*, 168. [CrossRef]
12. Su, Y.; Nguyen, K.; Sekiya, H. Recovery Time Evaluation of Ad-hoc Routing Protocols in IoT-Blockchain. In Proceedings of the IEEE 4th Global Conference on Life Sciences and Technologies (LifeTech), Osaka, Japan, 7–9 March 2022; pp. 265–269.
13. Chroboczek, J. The Babel Routing Protocol. 2011. Technical Report. Available online: https://www.rfc-editor.org/rfc/rfc8966.html (accessed on 9 October 2022).
14. Abolhasan, M.; Hagelstein, B.; Wang, J.P. Real-world performance of current proactive multi-hop mesh protocols. In Proceedings of the 2009 15th Asia-Pacific Conference on Communications, Shanghai, China, 8–10 October 2009; pp. 44–47.
15. Murray, D.; Dixon, M.; Koziniec, T. An experimental comparison of routing protocols in multi hop ad hoc networks. In Proceedings of the 2010 Australasian Telecommunication Networks and Applications Conference, Auckland, New Zealand, 3–31 November 2010; pp. 159–164. [CrossRef]
16. Fontes, R.R.; Afzal, S.; Brito, S.H.; Santos, M.A.; Rothenberg, C.E. Mininet-WiFi: Emulating software-defined wireless networks. In Proceedings of the International Conference on Network and Service Management (CNSM), Barcelona, Spain, 9–13 November 2015; pp. 384–389.
17. Walker, M.A.; Dubey, A.; Laszka, A.; Schmidt, D.C. Platibart: A platform for transactive iot blockchain applications with repeatable testing. In Proceedings of the 4th Workshop on Middleware and Applications for the Internet of Things, Las Vegas, NV, USA, 11 December 2017; pp. 17–22.
18. Rejeb, A.; Keogh, J.G.; Treiblmaier, H. Leveraging the internet of things and blockchain technology in supply chain management. *Future Internet* **2019**, *11*, 161. [CrossRef]
19. Li, Y.; Susilo, W.; Yang, G.; Yu, Y.; Liu, D.; Du, X.; Guizani, M. A blockchain-based self-tallying voting protocol in decentralized IoT. *IEEE Trans. Dependable Secur. Comput.* **2020**, *19*, 119–130. Available online: https://ieeexplore.ieee.org/document/9031381 (accessed on 9 October 2022). [CrossRef]
20. Ren, Q.; Man, K.L.; Li, M.; Gao, B. Using blockchain to enhance and optimize IoT-based intelligent traffic system. In Proceedings of the 2019 International Conference on Platform Technology and Service (PlatCon), Jeju, Korea, 28–30 January 2019; pp. 1–4.
21. Dwivedi, A.D.; Srivastava, G.; Dhar, S.; Singh, R. A decentralized privacy-preserving healthcare blockchain for IoT. *Sensors* **2019**, *19*, 326. [CrossRef]
22. Abolhasan, M.; Wysocki, T.; Dutkiewicz, E. A review of routing protocols for mobile ad hoc networks. *Ad Hoc Netw.* **2004**, *2*, 1–22. [CrossRef]
23. Boukerche, A. Performance evaluation of routing protocols for ad hoc wireless networks. *Mob. Netw. Appl.* **2004**, *9*, 333–342. [CrossRef]
24. Barolli, L.; Ikeda, M.; De Marco, G.; Durresi, A.; Xhafa, F. Performance analysis of OLSR and BATMAN protocols considering link quality parameter. In Proceedings of the International Conference on Advanced Information Networking and Applications, Bradford, UK, 26–29 May 2009; pp. 307–314.
25. Bujari, A.; Gaggi, O.; Palazzi, C.E.; Ronzani, D. Would current ad-hoc routing protocols be adequate for the internet of vehicles? a comparative study. *IEEE Internet Things J.* **2018**, *5*, 3683–3691. [CrossRef]
26. Sanchez-Iborra, R.; Cano, M.D.; Garcia-Haro, J. Performance evaluation of BATMAN routing protocol for VoIP services: a QoE perspective. *IEEE Trans. Wirel. Commun.* **2014**, *13*, 4947–4958. [CrossRef]
27. Kulla, E.; Hiyama, M.; Ikeda, M.; Barolli, L. Performance comparison of OLSR and BATMAN routing protocols by a MANET testbed in stairs environment. *Comput. Math. Appl.* **2012**, *63*, 339–349. [CrossRef]

28. Lee, Y.; Rathore, S.; Park, J.H.; Park, J.H. A blockchain-based smart home gateway architecture for preventing data forgery. *Hum.-Centric Comput. Inf. Sci.* **2020**, *10*, 1–14. [CrossRef]
29. Tan, Y.; Liu, J.; Kato, N. Blockchain-based key management for heterogeneous flying ad hoc network. *IEEE Trans. Ind. Inform.* **2020**, *17*, 7629–7638. [CrossRef]
30. Li, X.; Lu, R.; Liang, X.; Shen, X.; Chen, J.; Lin, X. Smart community: An internet of things application. *IEEE Commun. Mag.* **2011**, *49*, 68–75. [CrossRef]
31. Clausen, T.; Jacquet, P. Optimized Link State Routing Protocol (OLSR). Technical Report. 2003. Available online: https://www.rfc-editor.org/rfc/rfc3626.html (accessed on 9 October 2022).
32. Neumann, A.; Aichele, C.; Lindner, M.; Wunderlich, S. Better Approach to Mobile Ad-Hoc Networking (B.A.T.M.A.N.). Internet-Draft Draft-Wunderlich-Openmesh-Manet-Routing-00, Internet Engineering Task Force, 2008. Work in Progress. Available online: https://datatracker.ietf.org/doc/draft-openmesh-b-a-t-m-a-n/ (accessed on 9 October 2022).
33. Chen, X.; Nguyen, K.; Sekiya, H. On the Latency Performance in Private Blockchain Networks. *IEEE Internet Things J.* **2022**, *9*, 19246–19259. [CrossRef]
34. Machado, K.; Rosário, D.; Cerqueira, E.; Loureiro, A.A.; Neto, A.; De Souza, J.N. A routing protocol based on energy and link quality for internet of things applications. *Sensors* **2013**, *13*, 1942–1964. [CrossRef] [PubMed]