SIoTFog: Byzantine-resilient IoT fog networking

Frontiers of Information Technology & Electronic Engineering
www.jzus.zju.edu.cn; engineering.cae.cn; www.springerlink.com
ISSN 2095-9184 (print); ISSN 2095-9230 (online)
E-mail: jzus@zju.edu.cn

SIoTFog: Byzantine Resilient IoT Fog Networking*

Jian-wen XU[1], Kaoru OTA[1], Mian-xiong DONG‡[1], An-feng LIU[2], Qiang LI[3]

[1] *Department of Information and Electronic Engineering, Muroran Institute of Technology, Muroran 0508585, Japan*

[2] *School of Information Science and Engineering, Central South University, Changsha 410083, China*

[3] *Key Laboratory of Symbol Computation and Knowledge Engineering (Jilin University), Ministry of Education, Changchun 130012, China*

†E-mail: {17096011, ota, mxdong}@mmm.muroran-it.ac.jp; afengliu@mail.csu.edu.cn; li_qiang@jlu.edu.cn

Received Aug. 31, 2018; Revision accepted Nov. 19, 2018; Crosschecked mmm. dd, 2018

Abstract: The current boom in IoT is changing daily life in many ways, from wearable devices to connected vehicles and smart cities. We used to regard fog computing as an extension of cloud computing, but it is now becoming an ideal solution for transmitting and processing large-scale geo-distributed big data. In this paper, we propose a Byzantine fault tolerant networking method and two resource allocation strategies for IoT fog computing. Our aim is to build a secure fog network called SIoTFog to resist Byzantine faults and improve the efficiency of transmitting and processing IoT big data. We consider two cases: a case with a single Byzantine fault and a case with multiple faults to compare their performances when facing different degrees of risk. We chose latency, forwarding hops in the transmission and device use rate as the metrics for analysis of the simulation results. The simulation results show that our strategies can help achieve an efficient and reliable fog network.

Key words: Byzantine Fault Tolerance; Fog Computing; Resource Allocation; Internet of Things
https://doi.org/10.1631/FITEE.1000000 **CLC number:** TP

1 Introduction

Recent years have witnessed the boom in the Internet of Things and the hypergrowth of cloud computing which again overturned our perception of information technology. By 2020 there will be more than 20 billion IoT devices manufactured and put into use after increases of 15 percent occurring year-after-year (IHS Markit report, 2017). Originally, as an extension of cloud computing, fog computing relied on collaborative end-user clients or near-user edge devices to provide a substantial amount of storage capacity and communication solutions. Now, the fog has already become a research hotspot which not only broadens our perspective in distributed computation but also provides brand new ideas to exploit the potential of "Things" besides the "Internet."

Byzantine fault tolerance (BFT) describes the dependability of fault-tolerant computing systems, especially distributed ones. The Byzantine Generals' Problem or the BFT Problem was first raised by Leslie Lamport, Robert Shostak and Marshall Pease early in 1982 (Lamport et al., 1982). In BFT, a group of generals are trying to reach an agreement to decide whether to attack or retreat according to their votes in the majority. Considering the appearances of messengers or the presence of traitors who want to disrupt the whole group, the final agreement may run in a direction opposite of the original intentions of all loyal generals. A Byzantine fault stands for the inconsistency whereby generals receive different messages from a single general, and Byzantine failure is the system malfunction caused by Byzantine fault.

The occurrence of Byzantine faults can be very common in distributed systems such as fog networks. Sometimes fog nodes may fail and there is imperfect

‡ Corresponding author

* This work is partially supported by JSPS KAKENHI (Grant No. JP16K00117), and the KDDI Foundation.

ORCID: Mian-xiong DONG,
http://orcid.org/0000-0002-2788-3451

information about whether a particular node has failed. The only way to completely solve this problem is to find the failed node. However, we cannot ask a running distributed system to stop and troubleshoot all nodes. Instead, a relative compromise is a solution that designs a fault tolerance mechanism. That is, what we prefer to do is to cope with BFT while introducing as little impact as possible to the network computing performance.

In this paper, we focus on the issue of Byzantine fault tolerance in resource allocation of fog computing for IoT applications. The property of fault tolerance enables a system to continue working properly when some of its components go down. Therefore, good fault-tolerance performance can greatly avoid the interruption of retransmissions in network communications and reduce extra energy consumption and time costs.

In fog computing, the role of large central servers is carried out by a massive number of geo-distributed small- and medium-sized fog devices at the edge of the network structure. Thus, rather than setting dedicated standby replicas for all fog devices, we can simply let the fog devices help each other for state machine replication. Thus, a fog device can serve as the replica of its neighbor to reduce the influence of a possible Byzantine fault. Taking into account the mobility of IoT devices, the relationship between replicas and primary devices can also change while the entire network is running. As a result, we need a dynamic resource allocation strategy to solve BFT in fog computing. The main contributions of our work are as follows.

• A 3-tiered heterogeneous fog network model is designed in which the routers as fog devices are providing services to IoT users such as sensors, smart devices and vehicles.

• A Byzantine-resilient fog networking method and two resource allocation strategies are proposed to help reduce the influence of Byzantine faults.

• The case of a single Byzantine fault and the case of multiple faults are considered to test the performance of the strategies when facing different degrees of risk.

• Total latency, forwarding hops in the transmission and the device use rate are chosen as the metrics for analysis of the simulation results.

This paper is divided into six sections to elaborate on our work on Byzantine fault tolerant resource allocation for an IoT fog network. Section 2 introduces related work in the fields of fog computing and the Byzantine fault tolerance problem. Section 3 formulates the mathematical model of a 3-tiered heterogeneous IoT fog network structure and describes the problems to solve. Section 4 proposes the resource allocation methods. Section 5 describes the simulation experiments carried out and analyzes the performance of the proposed methods. Section 6 summarizes the work.

2 Related work

In this section, we present the related work on fog computing and the Byzantine fault tolerance problem.

2.1 From the cloud to the fog

First put forward by Cisco Systems Inc. (Bonomi et al., 2012), fog computing serves as an extension of cloud computing as a way to share responsibility for data storage and processing at the edge of the network structure. Vaquero et al. from the Hewlett-Packard Company (HP) offered a comprehensive view of fog computing, and correlated it with existing technologies such as the cloud, sensor networks, peer-to-peer networks and the network virtualization function (NFV) to reach a definition of "the fog" (Vaquero and Rodero-Merino, 2014). The group led by Satyanarayanan et al. conducted research for years into mobility-enhanced small-scale instances of cloud datacenters, the cloudlet, to mobile edge computing (MEC) in IoT (Satyanarayanan et al., 2009; Satyanarayanan, 2017). Liu et al. focused on streaming media in heterogeneous edge networks and proposed a device-to-device relay-assisted scheme to help solve video frame recovery for picocell edge users (Liu et al., 2016). Tao et al. integrated fog and cloud computing to build a hybrid network model for Vehicle-to-Grid (V2G) and 5G services (Tao et al., 2017a). Stojmenovic et al. analyzed the real-world application scenarios of the fog such as in smart grids, smart traffic and software defined networks (SDN). In these scenarios, the man-in-the-middle attack is regarded as a typical security issue to represent new features in the fog (Stojmenovic and Wen, 2014a). Yi et al. focused on the new security and privacy challenges

besides those inherited from the cloud and proposed ideas for solutions (Yi et al., 2015). Alrawais et al. considered the fog and IoT as a whole and put forward a mechanism to improve the distribution of certificate revocation information for security enhancement among IoT devices in the fog (Alrawais et al., 2017). Li et al. propose the idea of introducing deep learning to solve problems in edge computing (Li et al., 2018). Hu et al. addressed face identification and resolution technology, and implemented a prototype system to evaluate their proposed security and privacy preservation method (Hu et al., 2017).

Compared with cloud computing, fog computing was originally intended to share the high load of a central architecture and help save on the extra cost that occurs between cloud servers and IoT devices at the edge of a network. Jalali et al. believed that fog computing could just help to reduce the energy consumption in cloud computing (Jalali et al., 2016). Tao et al. investigated the problem of energy efficiency in mobile-edge computing and applied a request offloading scheme to improve the performance of energy consumption and bandwidth capacity (Tao et al., 2017b). Perera et al. surveyed the existed research and the problems to solve in fog computing for sustainable smart cities (Perera et al., 2017). Castillo-Cara et al. put forward a fog node design to solve the energy consumption problem and network resilience provisioning in wireless sensor networks (WSNs) (Castillo-Cara et al., 2018). Zeng et al. study how to explore energy generation diversity in a cyber physical fog system (CPFS) considering source rate control, service replica deployment and load balancing (Zeng et al., 2018). Wu et al. combine information-centric networks (ICNs) in designing content awareness filtering to help increase the safety factor of fog computing (Wu et al., 2018b).

2.2 Research on the Byzantine fault tolerance problem

Fault tolerance refers to the property where no global errors or interruptions occur in a system due to local faults. As a result, fault-tolerant design is very common and important in research fields related to an overall system structure (Khosravi and Seifi Kavian, 2016; Gao et al., 2017; Zhang et al., 2018). Since the idea was first raised by Lamport et al. (1982), research into Byzantine fault tolerance has undergone decades of development. Castro et al. first explored in depth the practice of BFT and implemented a generic program library and the first BFT network file system (NFS). Their experiment results showed that a NFS with BFT, i.e., a BFS, performs better than the NFS protocol without replicas (Castro and Liskov, 2002). Driscoll et al. redefined the concepts in Byzantine problems including the widely known existence of Byzantine faults and their possibility of leading to Byzantine failures. Their work points out some misunderstandings about the conditions needed for the appearance of a Byzantine attack as well as preventing those mechanism (Driscoll et al., 2003, 2004). Kotla et al. propose a speculative BFT protocol, the Zyzzyva, to simplify the design of BFT state machine replication and ensure that responses to the correct clients become stable. They compare this with existing BFT protocols including (Castro and Liskov, 2002) in cost, throughput and latency, and proved that Zyzzyva can maintain properties of safety and liveness (Kotla et al., 2010).

Today BFT is now widely accepted as a basic security necessity especially for distributed systems with system-level consensus requirements and mutual clock synchronization (Driscoll et al., 2004). Aublin et al. designed a Redundant-BFT (RBFT) approach to closely monitor the performance of instances from the primary to replicas on different machines (Aublin et al., 2013). Bessani et al. improved the previous BFT protocols by applying an open Java-based library source to make the state machine replication robust (Bessani et al., 2014). Li et al. designed a secure software-defined network (SDN) structure to resist Byzantine attacks on the communication links between SDN controllers and switches (Li et al., 2014). Wu et al. present optimization algorithms to achieve secure cluster management in SDNs (Wu et al., 2018a). Zhang et al. focused on the case of a cognitive radio network (CRN) and introduced the Byzantine attack and defense in cooperative spectrum sensing which is one of the key security issues in a CRN (Zhang et al., 2015). Miller et al. argued that the former synchronous BFT protocols relied critically on network time assumptions and came up with the idea of an asynchronous one to extend the adaptability to asynchronous systems such as blockchain technology (Miller et al., 2016).

3 Problem formulation

In this section, we design the system model and formulate the problem of BFT in fog computing.

In contrast to a traditional centralized network design, fog computing prioritizes local distributed devices at the edge of the network to provide low-latency resource-constrained processing and storage services. In the fog, there can exist more complex relations between users and the fog devices as service providers. That is, each user may not stay in contact with the same service provider all the time. Rather than a dedicated wide bandwidth, fog users prefer flexible dynamic resource allocation which may save extra time and energy consumption in multi-hop forwarding.

To resist the influence of Byzantine faults, we need to set replicas for network nodes as backups to restore and recover data when necessary. In the case of the fog, we may not need to prepare dedicated devices for Byzantine fault tolerance and may just be able to assign neighbor fog devices to serve as replicas.

$$n \geq 3f + 1 \tag{1}$$

As shown in Equation (1), existing BFT protocols including PBFT (Castro and Liskov, 2002), Zyzzyva (Kotla et al., 2010) and Honey Badger BFT (Miller et al., 2016) elaborate on how we need at least 3f extra devices as replicas to tolerate f Byzantine faults while all communications are synchronous or in bounded delays. For example, if the number of Byzantine faults reaches 3, we may need at least 10 fog nodes to avoid Byzantine failure.

3.1 System outline

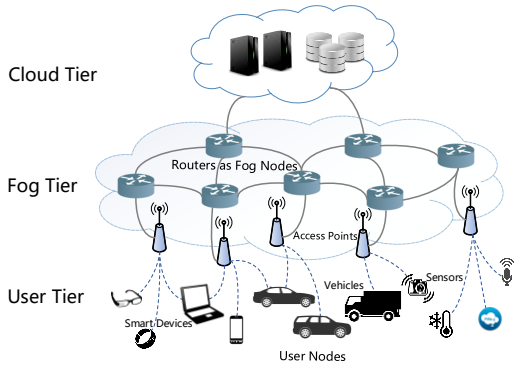


Fig. 1 A 3-tiered heterogeneous IoT fog network structure.

Shown in Fig. 1, we formulate the mathematical model of a 3-tiered heterogeneous IoT fog network structure (Stojmenovic and Wen, 2014a) (Reznik et al., 2017). Our aim is to reduce the impact of Byzantine faults in resource allocation for fog computing. As a result, we consider that this three-tiered model can more intuitively show the relationship between the fog nodes as service providers and the users as service receivers than models with more tiers.

User nodes (u_1, u_2, ..., u_n) in the User Tier send requests upwards to ask the routers as fog nodes (f_1, f_2, ..., f_n) in the Fog Tier for computational resources through access points. The Cloud Tier serves as reliable data centers providing stable network connections. The solid and dotted lines, respectively, stand for Ethernet and wireless connections. That is, communications between the User Tier and Fog Tier are wireless broadcasting, those inside the Fog Tier are wired broadcasting, and those between the Fog Tier and the Cloud Tier are wired point-to-point.

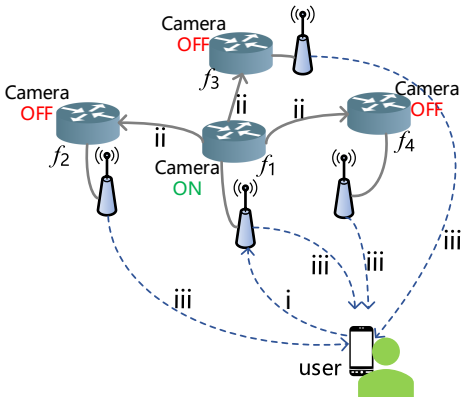


Fig. 2 BFT threat model in the fog service.

Figure 2 shows the threat model in our work. When users choose some fog nodes as service providers, they also need to accept some permissions for authority. The situation is similar to a pop-up window that appears before installation or the first time one opens an app on a smart device; for example, when the user in the figure chooses f_1 to finish a task on a mobile phone. For account certification, a user allows f_1 to use the camera when a service is provided. In normal cases, f_1 will send a message to let the user turn off the camera after certification. However, when f_1 is controlled by someone who wants to obtain additional personal privacy information, the message may be modified to remain open. To guarantee the operation of the fog network, we cannot interrupt service extensively to troubleshoot some individual

malicious nodes. It is better to draw support from a suitable fault tolerant strategy to avoid possible system failures.

To implement Byzantine fault tolerance in this 3-tiered fog network, we need geo-distributed routers to work as fog nodes to help each other when facing Byzantine faults. After choosing f_1 as a service provider, we also set replicas (f_2, f_3 and f_4) to ensure state machine replication when necessary. Here, we consider the case of a single Byzantine fault in which three replicas are required by one primary fog node. The entire procedure of Byzantine-resilient communication in the design of our fog network is as follows.

• i. A mobile user in the User Tier requests computational resources from the Fog Tier.

• ii. A fog node in the Fog Tier within a suitable distance to the user accepts the request and then forwards it to the other 3f fog nodes as replicas.

• iii. Both the primary and the replicas execute the task and send back responses to the original user.

As a result, the original user can then check the responses that even if f_1 wants to keep the camera on, he/she is still able to avoid a Byzantine failure from the single fault by checking responses from f_2 to f_4.

3.2 Performance metrics

To compare the actual performances of our proposed BFT resource allocation strategy, we choose total latency and forwarding hops in the transmission as the two main metrics to evaluate the performance of our work.

To achieve Byzantine fault tolerance and avoid Byzantine failures, our strategies operate at the expense of reducing some computing performance in the fog network. That is, in the process of multiple fog nodes working together to complete a user request, an additional information exchange is implemented to eliminate the possible impact of the failed nodes. Latency is a basic metric widely used in performance evaluation in engineering. Here we use it to prove that our strategies can achieve BFT with as little time cost as possible.

$$\begin{aligned} L_{all} &= L_{trans} + L_{prop} + L_{proc} \\ &= n_{hop} \sum_{i}^{n_{pkt}} s_{pkt}(i) / r_{bit} + l_{e2e} / v_{prop} \\ &\quad + \sum_{i}^{n_{pkt}} s_{pkt}(i) / r_{MTR} \end{aligned} \tag{2}$$

As shown in Equation (2), the total end-to-end latency L_{all} mainly includes three parts, the latencies of transmission, propagation and processing procedures. Transmission latency L_{trans} represents the amount of time for pushing all bits of packets into the transmission medium like the wires or air. L_{trans} has nothing to do with the distance between any two nodes and only relates to the total size of the packets. n_{pkt} and s_{pkt} are the number and size of the packets. rbit is the bandwidth or bit rate of the transmission link. In contrast, propagation latency L_{prop} depends on the travel distance between the sender and receiver and the property of the transmission medium. For wireless communication, v_{prop} is equal to the speed of light c, and for wired communication, it ranges from $0.59c$ to $0.77c$. Thus, l_{e2e} may stand for the end-to-end length added up by all distances between any two nodes taking part in the current communication. Lastly, to calculate the processing latency, we need the maximum transfer rate r_{MTR} of the fog devices and the total packet size.

$$L_{user}^{all} = L_{\text{i.}}^{trans} + L_{\text{i.}}^{prop} + L_{\text{ii.+iii.}} \tag{3}$$

Then, to obtain the latency in practice, in the entire procedure described in Fig. 2, as shown in Equation (3), we may need to calculate each part of the three steps. For step i., we just sum up the transmission and propagation latencies since there is only one connection between the user and primary fog node f_1.

$$L_{\text{ii.+iii.}} = max\{L_{\text{ii.+iii.}}^{f_i} \mid i = 1, 2, 3, 4\} \tag{4}$$

However, for steps ii and iii, since the replicas may differ from each other in their positions from the user and primary fog node and processing capacity, etc., we need to figure out the practical latency values of the primary and each replica, and pick the maximum one as shown in Equation (4).

$$L_{\text{ii.+iii.}}^{f_2} = L_{f_1, f_2}^{trans+prop} + L_{f_2, user}^{trans+prop} + L_{f_2}^{proc} \tag{5}$$

Lastly, Equation (5) gives the expression of summation. We take f_2 in Fig. 2 as the example, with latencies in steps ii and iii. includes the two steps for L_{trans}, L_{prop}, and L_{proc}.

Moreover, we also choose the number of forwarding hops in the transmission which can reflect the quantity of work in our fog network and show the practical efficiency.

$$n_{hop}^{all} = 2 \cdot n_{user,pri}^{hop} + \sum_{i=2}^{4} \left(n_{f_1,f_i}^{hop} + n_{f_i,user}^{hop} \right) \quad (6)$$

As shown in Equation (6), in contrast to the total latency, to calculate total number of forwarding hops in the transmission we need to consider all the connections in the three steps of Fig. 2.

Besides total latency forwarding hops, to gain a thorough understanding of the network structure, we add the fog nodes' use rates and the percentages of workload capacity occupied by the primary/replicas as two auxiliary metrics to provide more details into the analysis of the simulation results.

The use rate stands for the overall resource occupancy of all fog nodes. Here, we use it to study the actual working conditions of the entire IoT fog network, and the possible changes brought about by resource allocation strategies. In the section on simulation, we consider the cases of both workload capacities occupied as primary service providers and as replicas. Moreover, we treat the two cases separately to calculate the percentages.

Table 1 summarizes and lists the main symbols used in this paper.

Table 1 Notations in the design of the Byzantine-resilient fog network

Symbol	Meaning
U, u_i	Set of user nodes and one in it
F, f_i	Set of fog nodes and one in it
$L_{trans/prop/proc}$	Latencies of total, transmission, propagation and processing
n_{hop}	Number of forwarding hops in the transmission
n_{pkt}, S_{pkt}	Number and size of packets
r_{bit}	Bit rate of the transmission link
l_{e2e}	End-to-end length of the network connection
v_{prop}	Wave propagation speed of the transmission medium
r_{MTR}	Maximum transfer rate of the device
$C_{i,j}$	Distance or number of forwarding hops between f_i and f_j
$\boldsymbol{P}_i$	Position coordinates of f_i
$path(\boldsymbol{P}_i, \boldsymbol{P}_j)$	Summation of all connections between f_i and f_j
w_{this}	Needed workload capacity of the current request from the user
C_{pri}, C_{rep}	Workload capacities of the fog nodes occupied as primary and replicas

4 Byzantine fault tolerant resource sllocation strategy

In this section, we propose resource allocation strategies for a fog network aimed at resisting the influence of Byzantine faults.

4.1 BFT fog networking

Before choosing the primary nodes and replicas for users in need of fog service, we first build up a BFT fog network which considers all neighbor relationships among the routers as fog nodes. Here our target is to fulfill the requirements of the BFT protocol called Zyzzyva in (Kotla et al., 2010) ($n = 3f + 1$).

Algorithm 1 Breadth-First BFT Fog Networking

Input: $F = \{f_1, f_2, ..., f_n\}$ // all n fog nodes in the network structure
$\boldsymbol{P}_i$ // position coordinates of all the fog nodes
$f_i.adj$ // the list of all adjacent nodes to f_i
Q_{fog}, Q_{save} // FIFO queues to keep fog nodes
layer, leaves (layer) // layers and leaves in tree map

Output: $\{c_{i,j} \mid i, j \in \{1,2,...,n\}, i \neq j\}$ //connections between any f_i and f_j

```
1  for i ← 2 to n do
2    for j ← 1 to i − 1 do
3      Q_fog ← ∅ , Q_save ← ∅
4      if find(f_i.adj = j) then
5        c_i,j.n_hop ← 1, c_i,j.l_e2e ← path(P_i, P_j)
6      end if
7      push all f_i.adj into Q_fog and Q_save
8      layer ← 1, leaves (layer) ← sizeof (f_i.adj)
9      while Q_fog ≠ ∅ do
10       if leaves(layer) = 0 then
11         layer ← layer + 1
12       end if
13       this ← Q_fog.pop()
14       leaves(layer) ← leaves(layer) − 1
15       if find(f_this.adj = j) then
16         c_i,j.n_hop ← layer + 1, c_i,j.l_e2e ← path(P_i, P_j)
17         break
18       end if
19       drop any f_this.adj already in Q_save and push into Q_fog and Q_save one by one
20       leaves(layer) ← leaves(layer) + sizeof(f'_this.adj)
```

```
21        end while
22      end for
23    end for
```

Algorithm 1 is based on a non-recursive breadth-first search (BFS) method to implement BFT fast networking. To obtain the connection situations $c_{i,j}$ between any two fog nodes { $f_1, f_2, \ldots, f_n$} including geographical distances and forwarding hops in routing, we need the two-dimension positions ($\boldsymbol{P}_i$) and neighbor lists recording all adjacent nodes ($f_i.adj$). The two first-in first-out (FIFO) queues Q_{fog}, Q_{save} and the variables layer, leaves are used to build the tree maps formulated by the BFS method. Some key points are as follows.

• Q_{fog} is used as the main data structure to take the whole situation into account. The cyclic condition in line 9 could not be broken unless no existing path is found between f_i and f_j after traversing all other nodes.

• Q_{save} is an instrumental variable to save all non-repetitive nodes which means no path will be tried twice. In line 19 we drop the neighbors of f_{this} which are already covered by Q_{save} before pushing the rest into two queues.

$$path(\boldsymbol{P}_i, \boldsymbol{P}_j) = \sum_{n=1}^{n_{i,j}-1} length(\boldsymbol{P}_n, \boldsymbol{P}_{n+1}) \qquad (7)$$

• The path ($\boldsymbol{P}_i$,$\boldsymbol{P}_j$) function in lines 5 and 16 stands for the summation of all connections between any two of the nodes in the full path from f_i to f_j. Equation (7) gives the calculation of *path()* in which $n_{i,j}$ represents the number of nodes in the path between f_i and f_j.

• A tree map is obtained through the BFS method and we use the layer and leaves(layer) to record the current layer and how many nodes are within this layer.

To set the primary fog node and replicas for user in a request, we need to ensure the protocol communications between any two different nodes. That is to say, although our fog network is not a real full connected network, we still can make sure that f_i can exchange messages with f_j at any time after limited forwarding hops. The time complexity of Algorithm 1 is $O(n^2(1 + n)) = O(n^3 + n^2) = O(n^3)$.

4.2 BFT resource allocation strategy

To resist f Byzantine faults in our fog network, we set the nearby fog nodes as replicas to help achieve state machine replication. Thus, each replica needs to repeat what the primary fog node is doing and send back the processing result to the user in a request.

Algorithm 2 OPMD: One Phase Minimum Distance

Input: w_{this} // workload need by user
C_{pri},C_{rep} // capacities of fog nodes used as primary and replicas
$c_{this,j}$ // connections between user and fog node f_j
f_{need} // 3f + 1 fog nodes as primary and replicas
C_{left} // resource capacity of fog nodes
Output: resource allocation results for all users

```
1   for i ← 2 to 3f + 1 do
2     find f_j with minimum c_{this,j}.l_{e2e} and set as f_need(i)
3     if !find(f_need(1 to i - 1) = f_need(i)) && f_need ( i) .C_left ≥ w_this then
4       if i = 1 then
5         f_need.C_pri ← f_need.C_pri + w_this
6       else
7         f_need.C_rep ← f_need.C_rep + w_this
8       end if
9       f_need.C_left ← f_need.C_left - w_this
10    else
11      continue
12    end if
13  end for
```

OPMD gives an entire procedure for setting one primary fog node and 3f replicas for workload w_{this} requested by the current user. C_{pri} and C_{rep}, respectively, stand for the resource allocation result where a part of the workload capacity is set as the primary or replica. We sort all the fog nodes by their distances away from the position of the current user and judge if the remaining workload capacity C_{left} is full as well as if no fog node is being requested twice. Some key points are as follows.

• We use the fneed as a set of temporary choices of fog nodes during the 3f + 1 cycles and regard the first choice as the primary fog node.

• $!find(f_{need}(1\ to\ i\ -\ 1) = f_{need}(i))$ in line 3 is a function to make sure that the current chosen $f_{need}(i)$ is not included in the former f_{need}.

OPMD focuses on shortening the communication distances between users and fog nodes, which may extensively cut down on the propagation latencies L_{prop} in Equation (2). The algorithm itself makes full use of the advantages of fast networking in the BFS method and is able to find all 3f + 1 required fog

nodes in a simple and straightforward way. The time complexity of Algorithm 2 is $O((3f + 1)(n + 1)) = O(3fn + 3f + n + 1) = O(fn)$.

However, OPMD may also place an extra burden on communications among the primary fog nodes and replicas providing service for the same users to some extent. As a result, we put forward a two-phase algorithm to optimize this issue between the primary fog nodes and replicas.

Algorithm 3 TPSP: Two-Phase Shortest Path

Input: w_{this} // workload need by user
C_{pri},C_{rep} // capacities of fog nodes used as primary and replicas
f_{pri}, f_{rep} // fog nodes set as primary and replicas
$c_{this,j}$ // connections between user and fog node f_j
f_{need} // 3f + 1 fog nodes as primary and replicas
C_{left} // resource capacity of fog nodes
Output: resource allocation results for all users

1 find f_j with minimum $c_{this,j}.l_{e2e}$ && $f_j.C_{left} \geqslant w_{this}$ and set as $f_{pri}(i)$
2 $f_{pri}.C_{pri} \leftarrow f_{pri}.C_{pri} + w_{this}$
3 $f_{pri}.C_{left} \leftarrow f_{pri}.C_{left} - w_{this}$
4 **for** $i \leftarrow 1$ *to* $3f$ **do**
5 find f_j with least $c_{pri,j}.n_{hop}$ or minimum $c_{pri,j}.l_{e2e}$ and set as $f_{rep}(i)$
6 **if** $f_{rep} \neq f_{pri}$ && *!find*($f_{rep}(1$ *to* $i - 1) = f_{rep}(i)$) && $f_{rep}(i).C_{left} \geqslant w_{this}$ **then**
7 $f_{rep}.C_{rep} \leftarrow f_{rep}.C_{rep} + w_{this}$
8 $f_{rep}.C_{left} \leftarrow f_{rep}.C_{left} - w_{this}$
9 **else**
10 **continue**
11 **end if**
12 **end for**

Compared to OPMD, TPSP adopts a two-phase design which first chooses the optimal fog node as primary and lets the primary look for its 3f replicas. Thus, after choosing one of the fog nodes as f_j, subsequent sorting and other work will be carried out around it instead of the current user who requests w_{this}. Some key points are as follows.

• The sum of f_{pri} and f_{rep} is equal to f_{need} in TPSP.

• Line 5 shows two selections in choosing suitable neighbor fog nodes as replicas which may show different performances such as the majority in total latency as shown in Equation (2) whereby L_{trans} pays more attention to the number of forwarding hops and L_{prop} relies on the transmission distance.

The time complexity of Algorithm 3 is $O(1+3f(n+ 1)) = O(3fn + 3f + 1) = O(fn)$.

5 Simulation and analysis

In this section, we carry out experimental simulations to evaluate the performance of the resource allocation strategies designed for a BFT fog network in two cases: a case of a single Byzantine fault and a case of multiple Byzantine faults. The simulation scenario is a 10 km^2 square open area in which we set up 100 routers with access points as fog nodes. There are 50 to 500 mobile IoT users requesting fog service from nearby fog nodes.

Table 2 Experimental setups

Bit rate of transmission	
Wireless (802.11ad)	6.8 Gbit/s
Ethernet	10 Gbit/s
Maximum transfer unit (802.11)	2304 Bytes
Wave propagation speed of transmission	
Wireless (air)	c (speed of light)
Ethernet (thick coax)	0.77 c
Device settings of the fog nodes	
Maximum transfer rate (SATA3)	750 MB/s
Workload capacity of fog node	32~512 MB

As shown in Table 2, we consider the conditions of both wireless and Ethernet connections with their respective transmission bit rate and wave propagation speed. The workload capacity of a single fog node would be in one of {32, 64, 128, 256, 512} MB according to the case of a single Byzantine fault or of multiple faults. We take multiple time-slots for sending and answering requests, processing and storage. We repeat each set of experiments 10 times with different numbers of IoT users.

5.1 Single Byzantine fault

First, we consider the condition of a single Byzantine fault (f = 1) which means our aim here is to resist the influence of a single fault in the procedure of answering a request from an IoT user. As a result, we need to choose four fog nodes in total for one user in each time-slot as the primary device and replicas. The workload capacity range set for the fog nodes is {32, 64, 128, 256}.

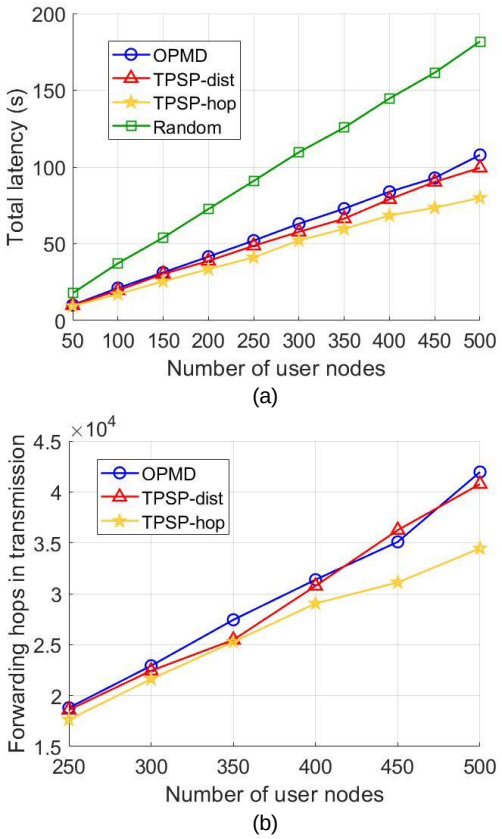


Fig. 3 Simulation results of a single Byzantine fault: (a) total latency and (b) total forwarding hops in the transmission.

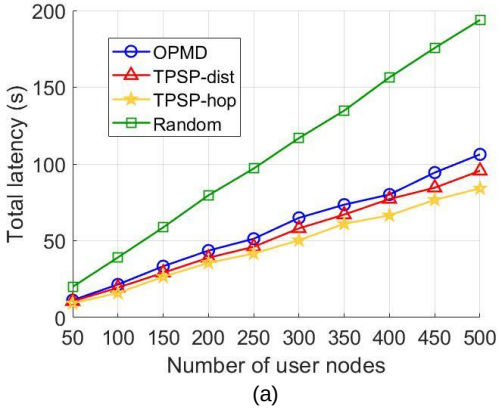
As shown in Fig. 3, we calculate the total latencies and forwarding hops in the transmission of different numbers of users. The blue, red, yellow and green broken lines, respectively, stand for OPMD, two cases of TPSP and a random method as a contrast. In Fig. 3a, all four methods show a linear increase in the trend from 50 to 500 IoT users requesting fog services from 100 routers. Although the differences among the four methods are not large when there are only a few users, the gap between the random one and the other three appears with the growth in number of users. The performance of OPMD is relatively poor in the three methods which matches our expectation. Compared with TPSP which applies two phases in fog node selections, the same treatment for the primary fog device and replicas does generate some impact on the total latencies. That is, the position of the primary fog node is more of an issue in not only receiving the request but also distributing it to all replicas. Therefore, the overlong distance or redundant forwarding hops between the primary fog node and replicas may cost extra time in data transmission.

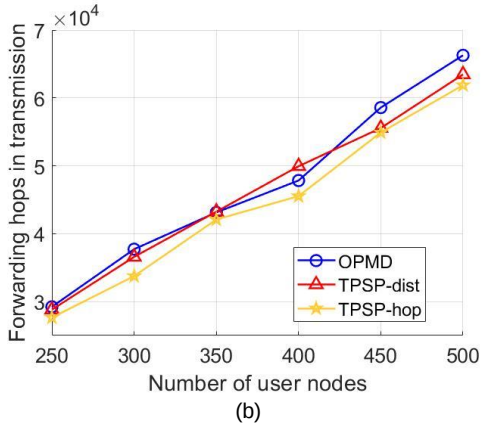
For TPSP, the red and yellow lines represent the simulation results of different standards in choosing suitable neighbor fog nodes shown in line 5 of Algorithm 3. The yellow line in consideration of the forwarding hops exceeds the red one in distance in total latencies, which may illustrate that the time cost for transmission latency L_{trans} takes up a larger proportion than that of propagation latency L_{prop}.

The total forwarding hops is the second metric we chose to compare and analyze for the performance of the simulation results for BFT resource allocation in the 3-tiered heterogeneous IoT fog network. From Fig. 3b we can see that the yellow broken line of the hop standard in TPSP still holds the lead in practical efficiency whereby more transmission hops mean extra energy consumption in the transmissions between IoT users and the fog nodes. In particular, when there is a large number of users, the TPSP-hop may behave better in solving situations where demand exceeds supply. Thus, service capacities could be insufficient relative to the user's needs, and sometimes the user may have to choose a service node with a relatively high cost in time and energy consumption.

5.2 Multiple Byzantine faults

Because we cannot be sure that only one Byzantine fault would occur in the BFT communication procedure shown in Fig. 2, the case of multiple Byzantine faults also should be taken into account. In this part of the simulation, f relates to the size of the requested workload capacity which means the possibility of multiple Byzantine faults is proportional to how many resources are being allocated to users. To fulfill the larger need of available resources in total, we also adjusted the workload capacity range set of the fog nodes to {64, 128, 256, 512}.





(b)

Fig. 4 Simulation results of multiple Byzantine faults: (a) total latency and (b) total forwarding hops in the transmission.

5.3 Device use rate and percentage for the primary and replicas

Lastly, to figure out the composition and the actual working conditions of the entire IoT fog network, we add the results of the fog nodes' use rates as well as the percentage of workload capacity occupied by the primary or replicas as auxiliary metrics to provide more detail.

The six subfigures in Figs. 5 and 6 show the use rates and primary and replica percentages of three resource allocation methods in cases of single and multiple faults. The green broken lines stand for the actual occupancy rates calculated from the average of 10 time slots. The blue and red bars are the average values of percentages of workload capacity occupied by the replicas and primary. First, in the comparisons between two cases of the same method, the occupied workload capacity proportions of the replicas all increase when there are more replicas needed as well as the times a single fog node is set as replicas in multiple requests. Second, in Fig. 5 the use rates of TPSP-hop are always lower than the other two methods for the range between 5% to 10% which can also be an asset for efficiency. That is, TPSP-hop may be able to complete the same amount of work using fewer computational resources. Third, compared to the second point above, in Fig. 6 the gap between TPSP-hop and the other two methods in the device use rate is narrowed when more than one Byzantine fault occurs in a single BFT communication procedure.

In summary, from the simulation results in the two cases of a single Byzantine fault and multiple faults, TPSP with the selection standard of fewer transmission hops shows better performance in total latency, number of forwarding hops and device use rate. As a result, our BFT resource allocation strategy does help build a reliable fog network structure to resist the influence of a single Byzantine fault or multiple faults.

6 Conclusion

In this paper, our SIoTFog focuses on how to resist the influence of Byzantine faults and improve the transmission and processing efficiency in fog computing for IoT. First, we designed a 3-tiered heterogeneous IoT fog network model which was mainly made up of routers as fog nodes to provide fog service to IoT users. To solve the problem of Byzantine fault tolerance in fog services, we proposed a fog networking method based on breath-first searching and two BFT resource allocation strategies to distribute workload capacities of the fog nodes to users upon request. We consider the cases of both a single Byzantine fault and multiple faults in experimental settings. The simulation results show that our proposed strategies can help build an efficient and reliable fog network when faced with Byzantine faults.

In the future, we will focus on further improving our approach to deal with the various situations that may occur in actual network operations. Two performance bounds in our proposed strategies that are a priority to be solved are the following. First, to ensure BFT in fog computing, we rely on the mutual assistance of the geographically distributed fog nodes themselves, which means there may be significant differences in performance for different node distributions. Second, in a distributed network composed of large-scale fog nodes, the fact that BFT does increase the relationships among the nodes may lead to new issues when the network topology changes.

References

Alrawais A, Alhothaily A, Hu C, et al., 2017. Fog computing for the internet of things: Security and privacy issues. IEEE Internet Computing, 21(2):34-42. https://doi.org/10.1109/MIC.2017.37

Aublin P, Mokhtar SB, Quema V, 2013. Rbft: Redundant byzantine fault tolerance. 2013 IEEE 33rd International Conference on Distributed Computing Systems,

p.297-306.

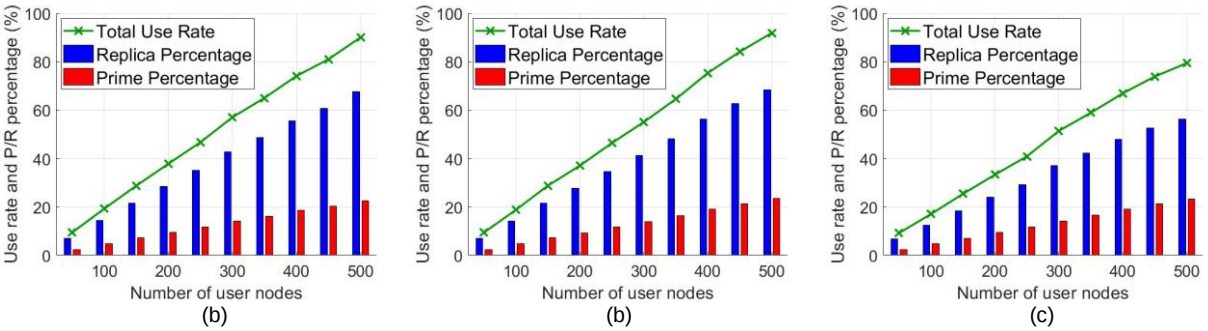


Fig. 5 Device use rate and percentage of the primary and replicas (single Byzantine fault): (a) OPMD, (b) TPSP-dist, and (c) TPSP-hop.

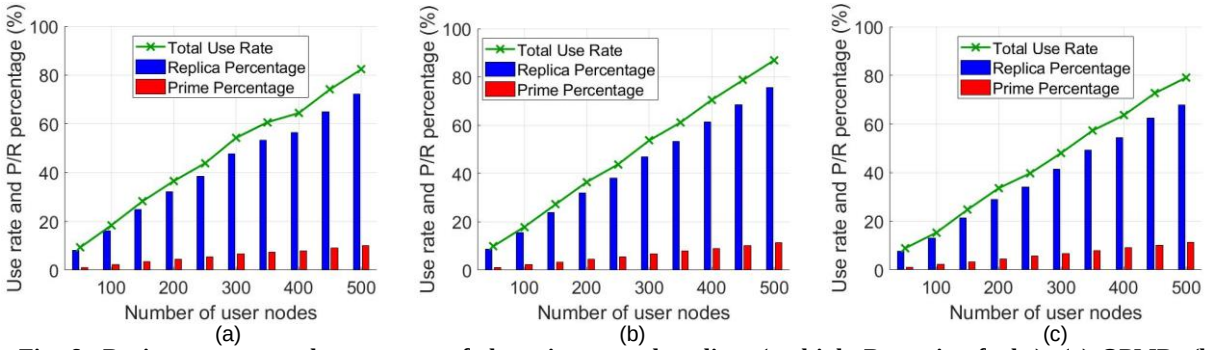


Fig. 6 Device use rate and percentage of the primary and replicas (multiple Byzantine faults): (a) OPMD, (b) TPSP-dist, and (c) TPSP-hop.

https://doi.org/10.1109/ICDCS.2013.53

Bessani A, Sousa J, Alchieri EEP, 2014. State machine replication for the masses with bft-smart. 2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, p.355-362. https://doi.org/10.1109/DSN.2014.43

Bonomi F, Milito R, Zhu J, et al., 2012. Fog computing and its role in the internet of things. Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing, New York, NY, USA, p.13-16. https://doi.org/10.1145/2342509.2342513

A Castillo-Cara M, Huaranga-Junco E, Quispe-Montesinos M, et al., 2018. Frog: A robust and green wireless sensor node for fog computing platforms. Journal of Sensors, 2018. https://doi.org/10.1155/2018/3406858

Castro M, Liskov B, 2002. Practical byzantine fault tolerance and proactive recovery. ACM Trans Comput Syst, 20(4):398-461. https://doi.org/10.1145/571637.571640

Driscoll K, Hall B, Paulitsch M, et al., 2004. The real byzantine generals. The 23rd Digital Avionics Systems Conference (IEEE Cat No04CH37576), 2:6.D.4-61. https://doi.org/10.1109/DASC.2004.1390734

Driscoll K, Hall B, Sivencrona H, et al., 2003. Byzantine fault tolerance, from theory to reality. Computer Safety, Reliability, and Security, Berlin, Heidelberg, p.235-248.

Gao D, Wang Q, Lei Y, 2017. Distributed fault-tolerant strategy for electric swing system of hybrid excavators under communication errors. Frontiers of Information Technology & Electronic Engineering, 18(7):941-954. https://doi.org/10.1631/FITEE.1601021

Hu P, Ning H, Qiu T, et al., 2017. Security and privacy preservation scheme of face identification and resolution framework using fog computing in internet of things. IEEE Internet of Things Journal, 4(5):1143-1155. https://doi.org/10.1109/JIOT.2017.2659783

Jalali F, Hinton K, Ayre R, et al., 2016. Fog computing may help to save energy in cloud computing. IEEE Journal on Selected Areas in Communications, 34(5):1728-1739. https://doi.org/10.1109/JSAC.2016.2545559

Khosravi A, Seifi Kavian Y, 2016. Autonomous faultdiagnosis and decision-making algorithm for determining faulty nodes in distributed wireless networks. Frontiers of Information Technology & Electronic Engineering, 17(9):885-896. https://doi.org/10.1631/FITEE.1500176

Kotla R, Alvisi L, Dahlin M, et al., 2010. Zyzzyva: Speculative byzantine fault tolerance. ACM Trans Comput Syst, 27(4):7:1-7:39. https://doi.org/10.1145/1658357.1658358

Lamport L, Shostak R, Pease M, 1982. The byzantine generals problem. ACM Trans Program Lang Syst, 4(3):382-401. https://doi.org/10.1145/357172.357176

Li H, Li P, Guo S, et al., 2014. Byzantine-resilient secure

software-defined networks with multiple controllers in cloud. IEEE Transactions on Cloud Computing, 2(4):436-447. https://doi.org/10.1109/TCC.2014.2355227

Li H, Ota K, Dong M, 2018. Learning iot in edge: Deep learning for the internet of things with edge computing. IEEE Network, 32(1):96-101. https://doi.org/10.1109/MNET.2018.1700202

Liu Z, Dong M, Zhou H, et al., 2016. Device-to-device assisted video frame recovery for picocell edge users in heterogeneous networks, :1-6. https://doi.org/10.1109/ICC.2016.7511342

IHS Markit, 2017. IoT Trend Watch 2017 https://ihsmarkit.com/Info/0117/IoT-trend-watch-2017.html [Accessed on Aug. 29, 2018]

Miller A, Xia Y, Croman K, et al., 2016. The honey badger of bft protocols. Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, New York, NY, USA, p.31-42. https://doi.org/10.1145/2976749.2978399

Perera C, Qin Y, Estrella JC, et al., 2017. Fog computing for sustainable smart cities: A survey. ACM Comput Surv, 50(3):32:1-32:43. https://doi.org/10.1145/3057266

Reznik A, Arora R, Cannon M, et al., 2017. Developing software for multi-access edge computing. ETSI, White Paper, (20).

Satyanarayanan M, Bahl P, Caceres R, et al., 2009. The case for vm-based cloudlets in mobile computing. IEEE Pervasive Computing, 8(4):14-23. https://doi.org/10.1109/MPRV.2009.82

Satyanarayanan M, 2017. The emergence of edge computing. Computer, 50(1):30-39. https://doi.org/10.1109/MC.2017.9

Stojmenovic I, Wen S, 2014a. The fog computing paradigm: Scenarios and security issues. 2014 Federated Conference on Computer Science and Information Systems, p.1-8. https://doi.org/10.15439/2014F503

Tao M, Ota K, Dong M, 2017a. Foud: Integrating fog and cloud for 5g-enabled v2g networks. IEEE Network, 31(2):8-13. https://doi.org/10.1109/MNET.2017.1600213NM

Tao X, Ota K, Dong M, et al., 2017b. Performance guaranteed computation offloading for mobile-edge cloud computing. IEEE Wireless Communications Letters, 6(6):774-777. https://doi.org/10.1109/LWC.2017.2740927

Vaquero LM, Rodero-Merino L, 2014. Finding your way in the fog: Towards a comprehensive definition of fog computing. SIGCOMM Comput Commun Rev, 44(5):27-32. https://doi.org/10.1145/2677046.2677052

Wu J, Dong M, Ota K, et al., 2018a. Big data analysis-based secure cluster management for optimized control plane in software-defined networks. IEEE Transactions on Network and Service Management, 15(1):27-38. https://doi.org/10.1109/TNSM.2018.2799000

Wu J, Dong M, Ota K, et al., 2018b. Fcss: Fog computing based content-aware filtering for security services in information centric social networks. IEEE Transactions on Emerging Topics in Computing, :1-1. https://doi.org/10.1109/TETC.2017.2747158

Yi S, Li C, Li Q, 2015. A survey of fog computing: Concepts, applications and issues. Proceedings of the 2015 Workshop on Mobile Big Data, New York, NY, USA, p.37-42. https://doi.org/10.1145/2757384.2757397

Zeng D, Gu L, Yao H, 2018. Towards energy efficient service composition in green energy powered Cyber–Physical Fog Systems. Future Generation Computer Systems, :1-1 . https://doi.org/10.1016/j.future.2018.01.060

Zhang L, Ding G, Wu Q, et al., 2015. Byzantine attack and defense in cognitive radio networks: A survey. IEEE Communications Surveys Tutorials, 17(3):1342-1363. https://doi.org/10.1109/COMST.2015.2422735

Zhang W, Lu K, Wang X, 2018. Versionized process based on non-volatile random-access memory for fine-grained fault tolerance. Frontiers of Information Technology & Electronic Engineering, 19(2):192-205. https://doi.org/10.1631/FITEE.1601477