electronics

Article

On-Demand Computation Offloading Architecture in Fog Networks

Yeonjin Jin and HyungJune Lee *,†

Department of Computer Science and Engineering, Ewha Womans University, Seoul 03760, Korea; yeonjin13@ewhain.net

* Correspondence: hyungjune.lee@ewha.ac.kr; Tel.: +82-2-3277-6644

† Current Address: 52 Ewhayeodae-gil, Asan Eng. 334, Seodaemun-gu, Seoul 03760, Korea.

Received: 20 August 2019; Accepted: 18 September 2019; Published: 23 September 2019

Abstract: With the advent of the Internet-of-Things (IoT), end-devices have been served as sensors, gateways, or local storage equipment. Due to their scarce resource capability, cloud-based computing is currently a necessary companion. However, raw data collected at devices should be uploaded to a cloud server, taking a significantly large amount of network bandwidth. In this paper, we propose an on-demand computation offloading architecture in fog networks, by soliciting available resources from nearby edge devices and distributing a suitable amount of computation tasks to them. The proposed architecture aims to finish a necessary computation job within a distinct deadline with a reduced network overhead. Our work consists of three elements: (1) resource provider network formation by classifying nodes into stem or leaf depending on network stability, (2) task allocation based on each node's resource availability and soliciting status, and (3) task redistribution in preparation for possible network and computation losses. Simulation-driven validation in the iFogSim simulator demonstrates that our work achieves a high task completion rate within a designated deadline, while drastically reducing unnecessary network overhead, by selecting only some effective edge devices as computation delegates via locally networked computation.

Keywords: computation offloading; in-network resource allocation; fog networks; edge computing

1. Introduction

With a recent advance in information technology, sensors and consumer devices have become connected with each other via Internet-of-Things (IoT) to provide intelligent applications based on data driven decisions. Various IoT applications such as smart homes and connected cars are actively being developed by collecting various data in local sites and performing intensive pattern analysis. In general, however, IoT devices have too limited resources in terms of computation, storage, and networking capabilities to satisfy the requirement of those applications that need massive data processing.

To bridge the gap between utility and constraint on IoT applications, cloud computing [1] is a way to flexibly provide resources in need through powerful virtual servers connected in the network. Although cloud computing has been an easy-to-use solution with IoT, the innate centralized approach has critical limitations of high network bandwidth usage and large delay for continuously uploading a large bulk of raw data collected at sensors.

The paradigm of fog networking [2–6] brings computation and storage services to the network edge devices. A fog network is formed with locally connected edge devices that can perform a certain task computation in a collaborative manner. However, due to the volatile wireless connectivity and mobility nature, flexible yet reliable resource provisioning and management is a key challenge for its successful evolution.

Previous endeavours focus on establishing a stable fog network, or configuring and mapping resources to participating nodes [7]. In particular, the problem of constructing on-demand ad hoc networks has been studied in the ad hoc network community by selecting some selected high performance nodes as spine nodes that lie in crucial transmission paths to effectively connect other parts [8], or by constructing a network cluster considering dynamic node environments [9,10]. However, a reliable network construction is not tightly coupled with resource provisioning. A more resource-specific network formation topic is not well studied in the context of computation offloading.

Regarding computation offloading, several computation offloading models with memory replication [11], a game-theoretic approach [12], or a convex optimization formulation [13] are proposed [14]. However, these approaches do not explicitly consider a critical aspect of how to form an on-demand resource provider network in an ad hoc manner.

In this paper, we propose a computation offloading architecture that embeds two essential components of resource provider network construction and flexible resource provisioning coupled in a framework. A host node that needs additional computation resources beyond itself finds available nodes in a progressive fashion and forms an on-demand local resource provider network. At the same time, as long as a node joins the network, its parent node decides to allocate a suitable amount of jobs to compute into the node based on its processing capability relative to its sibling nodes and link quality with the parent node. In case of losing either a parent node or a child node, task redistribution and parent re-selection are followed in a reconfiguration process. In this way, the framework accomplishes on-demand computation job processing using nearby edge devices within a distinct deadline, with high fidelity.

The contributions of this paper are three-fold: (1) This work designs both network construction and resource provisioning for distributed computation offloading over edge devices; (2) To cope with unexpected network and resource dynamics, a flexible distributed task redistribution is built; (3) In order to guarantee a certain level of quality of service with a certain time constraint, our framework progressively accommodates a suitable number of edge devices to compensate for the scarcity of computation resources on the device side by forming a connected resource pool.

2. Related Work

Edge devices that can support the ad hoc resource sharing in the fog network architecture tend to have relatively limited capability compared to cloud systems [15]. Since those devices use a wireless medium for communication, network connection is volatile, and data delivery is not guaranteed.

To deal with the volatile network challenge, various approaches are proposed by assorting connection-wise stable nodes. The spine routing structure [8] selects a stable *spine* node as a virtual backbone for ad hoc networks so that network routes are discovered and maintained via a way of connecting two adjacent spine-based sub-networks. Particularly for dynamic mobile networks, some distributed weighted cluster-based algorithms [10,16,17] choose some crucial cluster heads by calculating a weight measure based on a node's state information and its link status with the surrounding neighbor nodes. However, these approaches focus only on network construction, lacking the computation resource aspect.

To take into account resource sharing, as well as network routing in the distributed network context, the resource sharing in storage has widely been investigated in the peer-to-peer (P2P) networks. By constructing a virtual abstract overlay layer of complex network systems, resources (e.g., file, content streaming) can be shared in a decentralized manner through overlay routing [18]. Structured P2P approaches of Chord [19], CAN [20], Pastry [21], and Tapestry [22] have a logarithmic hashing-based lookup and routing performance in a relatively stable topology. Unstructured P2P overlays such as Freenet [23] and Gnutella [24] have been proposed for a more dynamic topology. However, these approaches suffer from high communication overhead in resource discovery based on flooding. Thus, in wireless mobile networks where dynamicity is even more severe due to node movement and wireless medium itself, these P2P-based approaches mostly for storage resource sharing can not directly be migrated to computation offloading in practice.

In the field of mobile edge computing (MEC), a task offloading algorithm using a wireless network has been proposed to overcome the resource limitation at the device itself. Chen et al. [25] has proposed an efficient task offloading algorithm to limited access points (APs) for multiple mobile users. Some algorithms [26,27] attempt to efficiently offload tasks to an MEC server considering the efficiency of transmission power and computing intensity. However, the previous studies constrain themselves on some selected computing devices such as MEC servers and APs still in a somewhat centralized manner where resource at the devices is not limited.

The multi-tier communication network is another task offloading approach designed to use resources efficiently in IoT devices [28]. Previous studies have used a heuristic structure by adjusting the cell size depending on different transmission power to efficiently distribute tasks by placing them into two computing layers at local and remote MEC servers [27]. However, these studies do not take into account time constraints and network dynamics due to node mobility.

This work provides a computation offload architecture that supports on-demand lightweight local network formation for resource discovery and dynamic task allocation under a continuously changing topology.

3. System Model

We address the problem of computation offloading in a distributed computing environment. In case that a device temporarily suffers from computation resource scarcity with a certain time period in the near future, we consider decomposing one complete task into a sequence of discrete sub-tasks that can be computed by nearby available edge devices. The processed results after being computed by the edge devices need to be delivered to the original requested device within a given distinct deadline.

We assume that a task can be decomposed into several unit tasks that are mutually exclusive. Each edge device is designed to communicate using the same wireless radio interface (e.g., 802.11, 802.15.4, or Bluetooth). Edge devices are willing to join the resource sharing for the computation offloading framework without considering incentives. Designing a suitable incentive mechanism is orthogonal to this work and is not considered in the paper. We consider a practical scenario that some devices retain mobility to move around the network, lacking stable connectivity to neighbor devices.

To tackle the problem, we design a framework to form a *resource provider network*, and distribute and compute unit tasks on behalf of the original requested device. In preparation for various procedure failures that can occur from erratic network or computation resource situations, the framework embeds the following reconfiguration as a make-up stage. Insufficient computation resources are additionally recruited from already participating devices with additional resources or newly found ones.

Our framework aims to complete the computation of a task of which several unit tasks are assigned to nearby edge devices within a designated time limit with a low task loss rate and marginal network overhead in a dynamic network environment. Our work consists of three key components: (1) network formation, (2) task distribution, and (3) reconfiguration in a fog network as in Figure 1.

(1) Network Formation: We form an on-demand resource provider network by nearby devices that can voluntarily lend some amount of computation resource in a certain period of time. A resource requestor node that needs computation services sends a network join request to all neighboring nodes that can be connected within a few hops and configures a network consisting of the nodes that have responded. This process continues until sufficient resources are gathered.

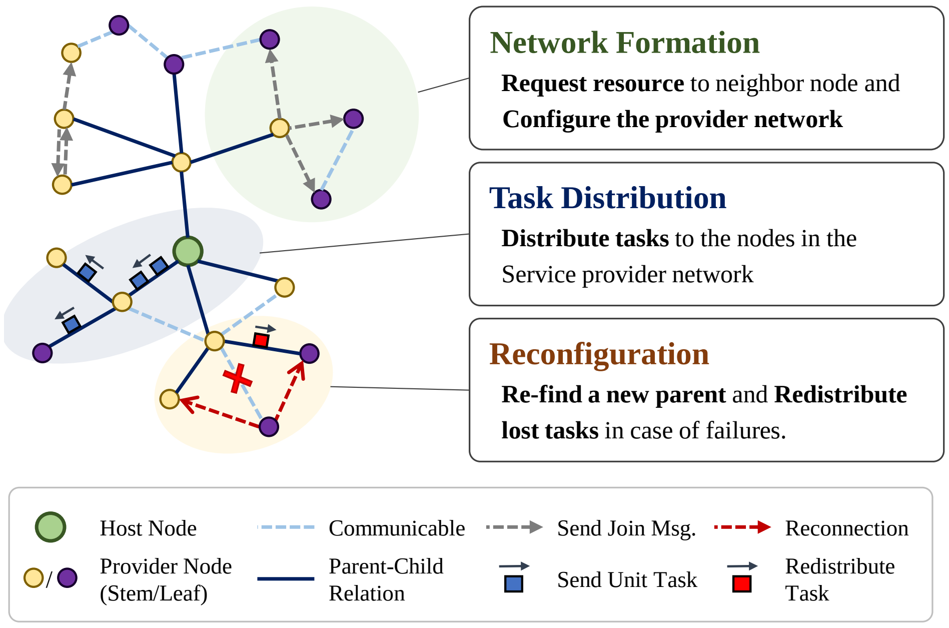

Network Formation
Request resource to neighbor node and
Configure the provider network
Task Distribution
Distribute tasks to the nodes in the
Service provider network
Reconfiguration
Re-find a new parent and **Redistribute**
lost tasks in case of failures.
Host Node
Provider Node (Stem/Leaf)
Communicable
Parent-Child Relation
Send Join Msg.
Send Unit Task
Reconnection
Redistribute Task

Figure 1. Computation offloading system framework consisting of network formation, task distribution, and reconfiguration.

We classify all the participating nodes in the resource provider network into two types: host nodes and provider nodes. The *host node* (HN) is the principal object of managing task allocation, which requests its required computation resource to nearby edge devices. Upon receiving the request from HN, the edge devices that agree to provide parts of their own computation resource and decide to participate in the resource provider network are called *provider nodes* (PNs). The PN node is further divided into stable PN node as *stem PN* and volatile PN node as *leaf PN* depending on connectivity volatility relative to its surrounding network devices. We aim to construct a reliable yet agile network tree structure to support the resource discovery, task distribution, and computation result delivery with modest network overhead.

(2) Task Distribution: Once a node joins the resource provider network as a child node to offer its own computation resource for the HN node, its parent node (either HN or stem PN) distributes some parts of unit tasks. The parent node calculates the suitable number of jobs (unit tasks) to allocate to the node considering the maximum workload that the node can contribute in terms of computation power and timely result delivery.

(3) Reconfiguration: In case that the already allocated tasks cannot be performed, or a child PN node is lost in connection, a reconfiguration process is performed to prevent its following service abruption. We design a reconfiguration stage to refresh the resource provider network by finding a new parent or a new child, or by redistributing the interrupted unit tasks to other PNs.

These three stages can concurrently make progress until the complete computation results arrive at the HN node, or the given time limit is reached.

4. Network Formation

We consider a scenario in which a computation resource can be acquired by other adjacent devices to handle a time-limited task in an infrastructure-less fog network environment. The goal is to construct a stable resource provider network that can gather sufficient computation resources from its participating nodes considering volatile network connectivity in an on-demand basis.

To classify stable nodes into stem PN nodes and volatile nodes into leaf PN nodes, we observe the dynamics of link quality from a node to its neighboring nodes. We adopt a bi-directional link quality estimator based on packet reception rate (PRR) and apply the exponential weighted moving average (EWMA) [29] to capture both short-term and long-term link variations as follows:

$$Q_t(i,j) = PRR_{i\to j} \times PRR_{j\to i}, \tag{1a}$$

$$LQ_t(i,j) = \alpha \cdot Q_t(i,j) + (1-\alpha) \cdot LQ_{t-1}(i,j), \tag{1b}$$

where $Q_t(i,j)$ is a bi-directional end-to-end delivery rate, and the expected number of end-to-end transmissions (ETX) is defined as $1/Q_t(i,j)$. We primarily use $LQ_t(i,j)$, which is the net link quality measure between node i and node j at time t considering its past link quality as well as the current end-to-end link quality. The direct influence of temporary link quality change in $Q_t(i,j)$ can be controlled by smoothing with its past link quality $LQ_{t-1}(i,j)$ via a smoothing factor α, as in Equation (1b).

Once we search resource provider nodes around the HN node in terms of requested resource, individual processing power, and time limit, we identify stem PN nodes that can have child nodes among resource provider nodes by using the above link quality-related criteria. In case the required amount of resource is not fulfilled by the direct neighboring child nodes of the HN node, the stem PN nodes among them recursively follow the resource provider search process.

4.1. Selecting Stem Provider Nodes

The stem provider node (PN) is defined as a core node of the resource provider network. A parent node (either HN node or stem PN node) chooses a neighboring node that can maintain connectivity-wise stable link connection to their surrounding neighboring nodes as a stem PN node.

We define a connection quality (CQ) measure as the average link variation over the certain number of check trials with the check interval of T_W at the center of node j as follows:

$$LQ\_dev_t(j) = \frac{\sum_{k=1}^{N_K}(LQ_t(j,j_k) - LQ_{t-1}(j,j_k))}{N_K}, \tag{2a}$$

$$CQ(j) = \frac{\sum_{t=t_1}^{t_{N_T}} LQ\_dev_t(j)}{N_T}, \tag{2b}$$

where node j_k is a neighboring node of node j, N_K is the total number of neighboring nodes of node j, N_T is the number of stability check trials, and $T_W = t_{m+1} - t_m$, where $m = 1, \ldots, N_T - 1$.

We adopt both criteria of link quality and connection quality to evaluate network stability toward node j from its parent node i as follows:

$$LQ_t(i,j) \geq T_{LQ}, \tag{3a}$$

$$CQ_t(j) \geq T_{CQ}, \tag{3b}$$

where T_{LQ} is the threshold for LQ measure, and T_{CQ} is the threshold for CQ measure. If both $LQ_t(i,j)$ and $CQ_t(j)$ satisfy their respective thresholds at the decision time t, then a child node j is promoted to a stem PN node of the parent node i.

As in Figure 2, the procedure of stem provider node selection starts at an HN node or a stem PN node. Once the link quality between itself and its neighboring node satisfies Equation (3a), then it asks the connection quality information to the neighboring node.

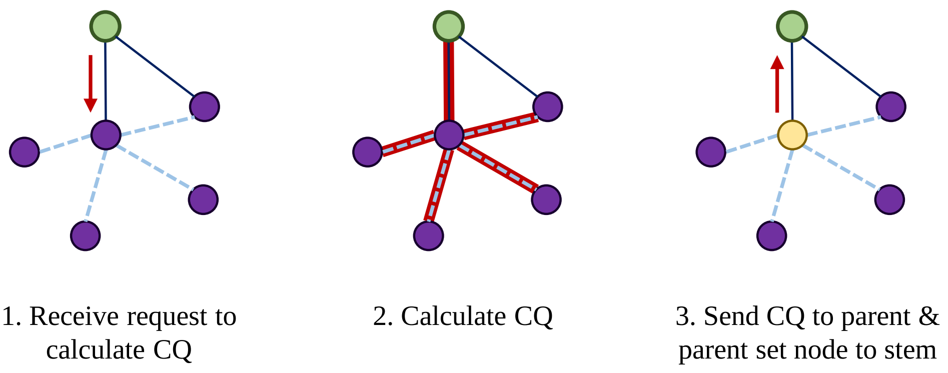

1. Receive request to calculate CQ
2. Calculate CQ
3. Send CQ to parent & parent set node to stem

Figure 2. Sequence of connection quality calculation.

Once the neighboring node calculates the CQ measure after communicating with its neighbors, it reports the measure back to its parent node. If the value satisfies Equation (3b), the neighboring node is selected as a stem PN node.

4.2. Selecting Resource Provider Nodes

An HN node or a stem PN node send a network join request to its 1-hop neighboring nodes with the given time limit information. There are two ways for an HN node or a stem PN node to find resource provider nodes among its neighboring nodes depending on the type of its neighboring node: (1) by directly asking a neighboring node with resource to join the source provider network as a leaf PN node, and (2) by searching resource provider nodes as a stem PN node from its sub-tree nodes.

4.2.1. Network Join as Leaf PN

To invite a new node as a leaf PN node in the resource provider network, it is necessary to receive the information about the number of unit tasks that can be processed and supported for the current request from a leaf PN node candidate as in Figure 3. Since each device has different processing capability that results from a distinct processor, memory, storage, and other workloads, the requesting node sends a reference unit task so that a leaf PN candidate can measure the time elapsed for the reference task. Based on the measured processing time, it calculates the maximum number of unit tasks that the node can support and sends a response message to the sender with its affordable capability:

$$2 \cdot t_d + Task_j \cdot t_{cl} \leq TaskDeadline, \tag{4a}$$

$$1 \leq Task_j \leq MaxStorage, \tag{4b}$$

where nodes i and j are the sender and the receiver of a network join request, t_d is the network delay between two nodes, and t_{cl} is the unit task computation time at node j.

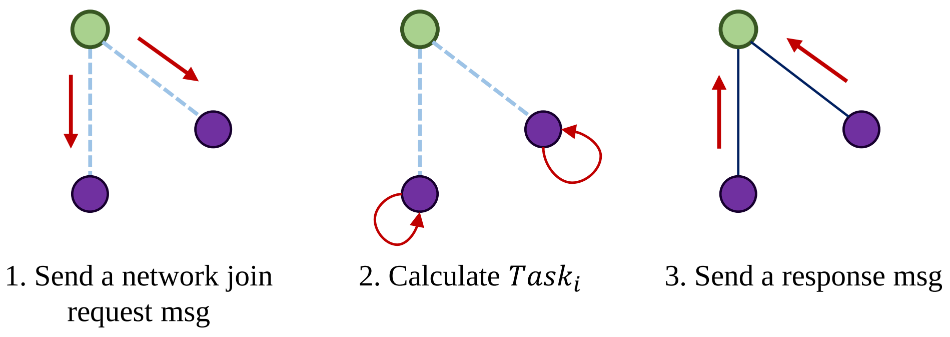


Figure 3. Sequence of network join request.

The receiver node picks up the maximum number of unit tasks to support by satisfying the above constraints and replies with it. Once the sender receives a response message with the computation resource to support from the receiver node, it calculates the net resource affordability by reflecting the link quality between two nodes:

$$TaskCapacity_j = LQ_t(i, j) \cdot Task_j \tag{5}$$

for child node j from the perspective of parent node i.

The HN node or the stem PN node calculates total resource affordability collected from the current resource provider network. If the total resource estimate is beyond the requirement amount, it stops finding other available provider nodes.

If a node receives several duplicate network join requests from other neighboring nodes, which are its parent node candidates, it responds to the first request message and chooses the first sender node as its parent node.

4.2.2. Resource Search Request at Stem PN

If the plan of gathering computation resource is not fulfilled by the 1-hop neighboring nodes of the HN node, it requests its stem PN nodes to extend the resource provider network as a proxy with a resource search request message.

If a stem PN node receives the request message, it sends a network join request message to its neighboring nodes that are not a part of the resource provider network. The stem PN node waits until all of the network join response messages arrive, or the maximum time limit is reached. It collects the net resource affordability information from itself and its child nodes and reports it to its parent.

The procedure of the resource search request is illustrated in Figure 4.

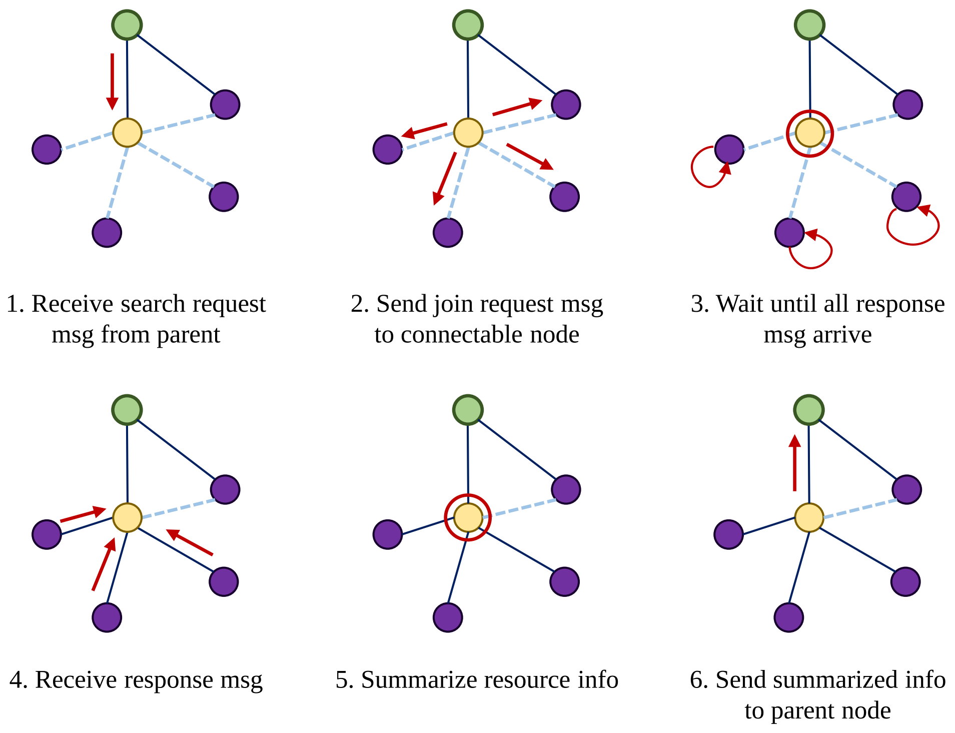

1. Receive search request msg from parent
2. Send join request msg to connectable node
3. Wait until all response msg arrive
4. Receive response msg
5. Summarize resource info
6. Send summarized info to parent node

Figure 4. Sequence of resource search request.

5. Task Distribution

Once a resource provider network is formed at an HN node or a stem PN node, it becomes aware of all the connectivity-related and computation-related information (e.g., $LQ_t(i,j)$, $CQ_t(j)$, $TaskCapacity_j$) of its child nodes. Based on the resource information gathered from its own network, it performs task distribution considering the computation capability status of itself and its child nodes.

5.1. Task Allocation Based on Resource Capability

Depending on whether a task is distributed to either a leaf PN node or a stem PN node, we classify the distribution type into inner-cluster distribution to a leaf PN node and outer-cluster distribution to a stem PN node, as illustrated in Figure 5. An HN node or a stem PN node check the task capacity and distribute a suitable amount of tasks only to each sub-network cluster node. Thus, the complexity of the task distribution algorithm leads to the complexity of $O(n)$ per job distribution for n sub-network cluster nodes.

We describe the task allocation procedure in Figure 6. In order for an HN node or a stem PN node to distribute a number of unit tasks, $Task\_R$ to its child nodes, it allocates them among the remaining number of tasks, $Task\_Tr(j)$, excluding the already-assigned tasks to a child node j, i.e., $Task\_A(j)$.

$$TaskAlloc_j = \left\lfloor \frac{TaskCapacity_j}{\sum_k TaskCapacity_k} \cdot J_t(i) \right\rfloor, \tag{6}$$

where $J_t(i)$ is the number of unit tasks that node i needs to allocate to its child node j at time t.

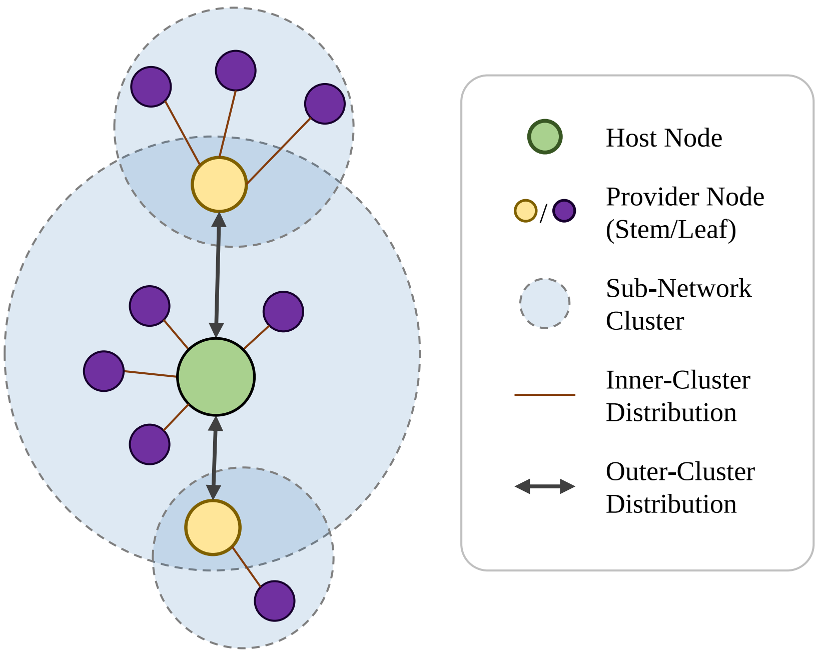
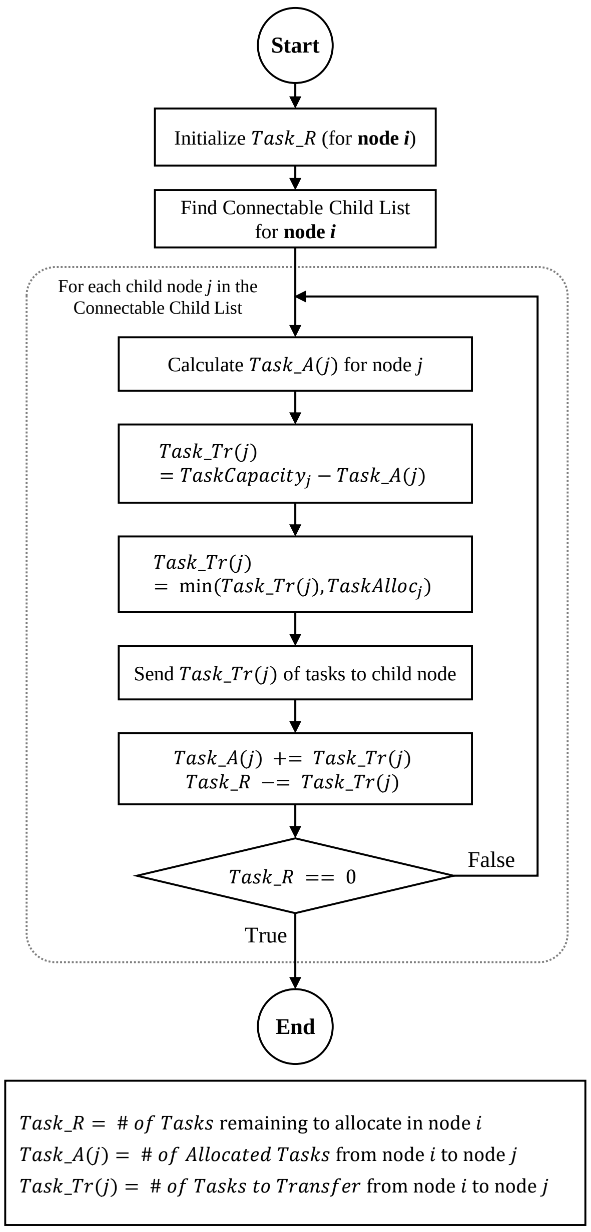

Host Node
Provider Node (Stem/Leaf)
Sub-Network Cluster
Inner-Cluster Distribution
Outer-Cluster Distribution

Figure 5. Resource provider network structure for task distribution.

Once the amount of task allocation is determined, the actual corresponding unit tasks are distributed so that each child node can execute the allocated tasks, and the result is reported back to its parent node.


Start
Initialize $Task\_R$ (for **node *i***)
Find Connectable Child List for **node *i***
For each child node *j* in the Connectable Child List
Calculate $Task\_A(j)$ for node *j*
$Task\_Tr(j) = TaskCapacity_j - Task\_A(j)$
$Task\_Tr(j) = \min(Task\_Tr(j), TaskAlloc_j)$
Send $Task\_Tr(j)$ of tasks to child node
$Task\_A(j) \mathrel{+}= Task\_Tr(j)$
$Task\_R \mathrel{-}= Task\_Tr(j)$
$Task\_R == 0$
False
True
End

$Task\_R = \# \, of \, Tasks$ remaining to allocate in node i
$Task\_A(j) = \# \, of \, Allocated \, Tasks$ from node i to node j
$Task\_Tr(j) = \# \, of \, Tasks \, to \, Transfer$ from node i to node j

Figure 6. Flowchart of task distribution procedure.

5.2. Reconfiguration

In the case that the established link between a stem PN node and its child node becomes disconnected, previously allocated tasks cannot be kept track of, and may be lost. To tackle the uncertainty in the resource provider network, our scheme is accompanied with a following reconfiguration process.

We use the net link quality measure $LQ_i(i, j)$ at a parent node side and its child node side, respectively.

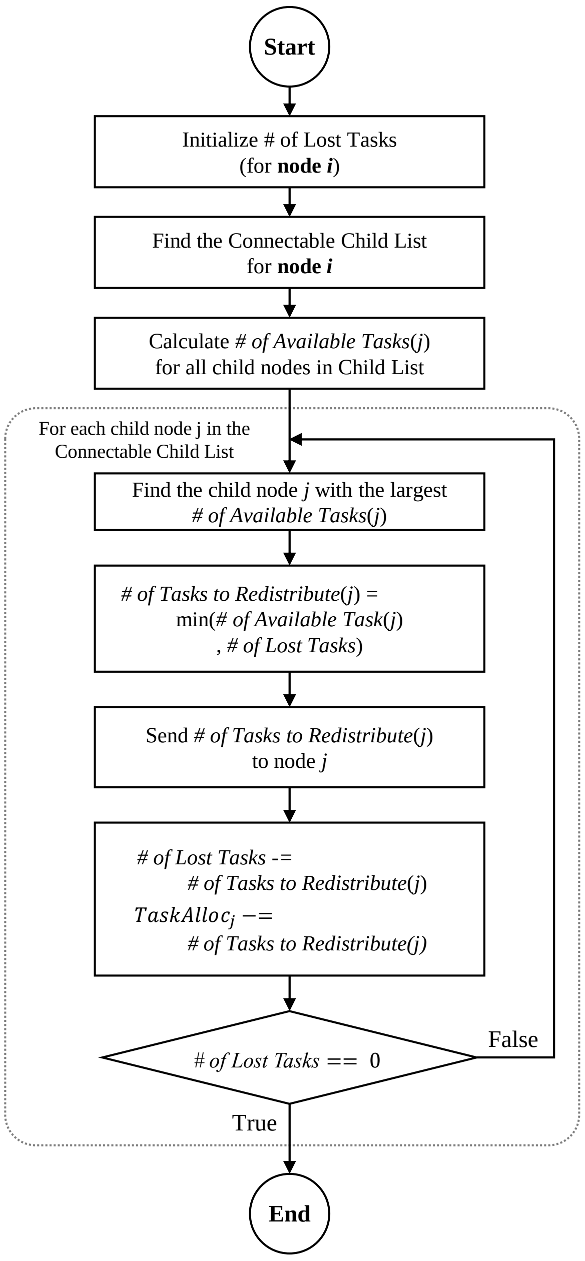
First, if a parent node loses a connection with a child node or evaluates its link quality to not be stable, it performs task redistribution. It re-finds other provider nodes to replace the lost child node so that the previously allocated tasks to the lost node are allocated to newly discovered nodes.

Second, if a child node detects a connection loss with its parent node, it runs the new parent search and re-connection procedure.

Since the parent node or its child node proceeds with its own reconfiguration process for child nodes within its 1-hope neighborhood, each algorithm proceeds with $O(n)$ complexity for its respective n child nodes.

5.2.1. Task Redistribution

If a parent node evaluates the link quality to its child node that does not satisfy the condition in Equation (3a) (e.g., in case of highly mobile child nodes), it performs task redistribution as described in Figure 7. The parent node finds a substitute child node that has the largest available tasks, ($\# \, of \;\, Available \; Tasks(j)$) to process beyond its previous allocated tasks, in the inner cluster. It chooses a child node with the highest value of ($TaskCapacity_j - TaskAlloc_j$) and allocates the tasks that were previously allocated to the lost child.



Start
Initialize # of Lost Tasks (for **node *i***)
Find the Connectable Child List for **node *i***
Calculate *# of Available Tasks*(*j*) for all child nodes in Child List
For each child node j in the Connectable Child List
Find the child node *j* with the largest *# of Available Tasks*(*j*)
# of Tasks to Redistribute(*j*) = min(*# of Available Task*(*j*) , *# of Lost Tasks*)
Send *# of Tasks to Redistribute*(*j*) to node *j*
# of Lost Tasks -= *# of Tasks to Redistribute*(*j*)
$TaskAlloc_j$ −= *# of Tasks to Redistribute*(*j*)
# of Lost Tasks == 0
False
True
End

Figure 7. Flowchart of a task redistribution procedure.

5.2.2. New Parent Node Selection

Second, if a child node detects a connection loss with its parent node by checking that the corresponding link quality does not hold as in Equation (3a), it unpairs with its current parent node and runs the new parent search and re-connection procedure.

As illustrated in Figure 8, the child node sends a network join request as a child to its neighboring nodes. If all of its response messages arrive, it sorts out stem PN nodes. Among them, it re-establishes its new parent connection with the one with the highest link quality. If none of the nodes are stem PN nodes, a leaf PN node that has the highest link quality among the ones that have responded is selected as a temporary leaf-to-leaf connection. The parent leaf PN node operates only some limited functionality of delivering task results from the child leaf PN node toward the HN node.

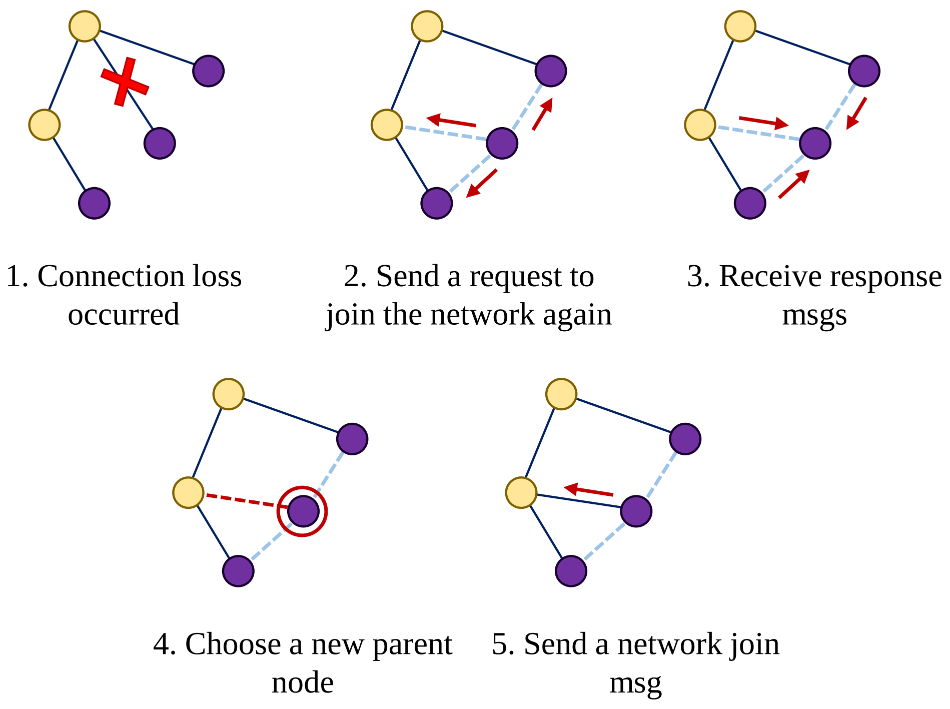

1. Connection loss occurred
2. Send a request to join the network again
3. Receive response msgs
4. Choose a new parent node
5. Send a network join msg

Figure 8. Sequence of reconfiguration with an updated resource provider network by finding a new parent node.

6. Evaluation

We validate our proposed architecture in a simulated network consisting of one host node and 20 edge devices, which are distributed over 300 m $\times$ 300 m as illustrated in Figure 9. To show the effect of dynamic network environments, we run experiments under three different topology environments. Before running our algorithm on these topologies, we show how different the node distribution in simulation environments is between stationary and mobile nodes among 20 edge nodes with respect to hop distance in Figure 10. As the topology goes from 1 to 3, a network becomes more dynamic with a larger portion of mobile nodes. In Topology 1, all of the edge devices used in experiments are stationary. As for Topology 2 and Topology 3, we increase the portion of mobile nodes, while keeping the same total number of edge devices in experiments.

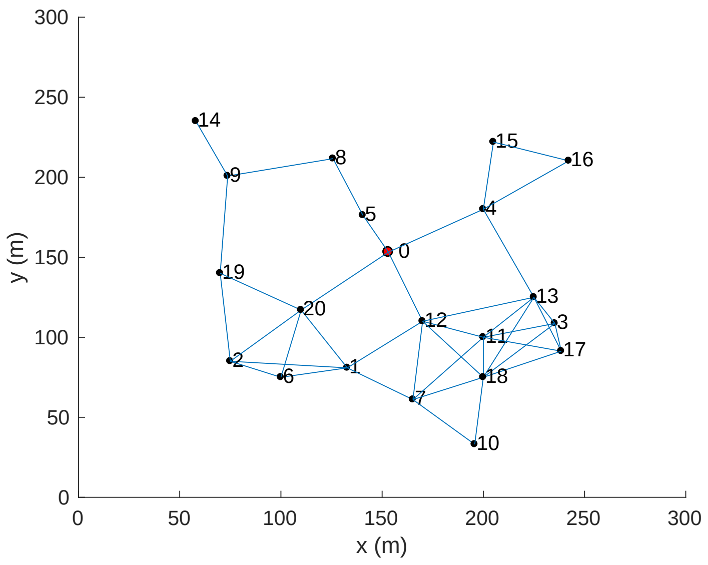


Figure 9. Network topology of one host node and 20 edge devices over 300 m $\times$ 300 m, located in a relative 2D Cartesian coordinate for simulation experiments, showing wireless links with packet reception rate $(PRR) \geq 0.7$.

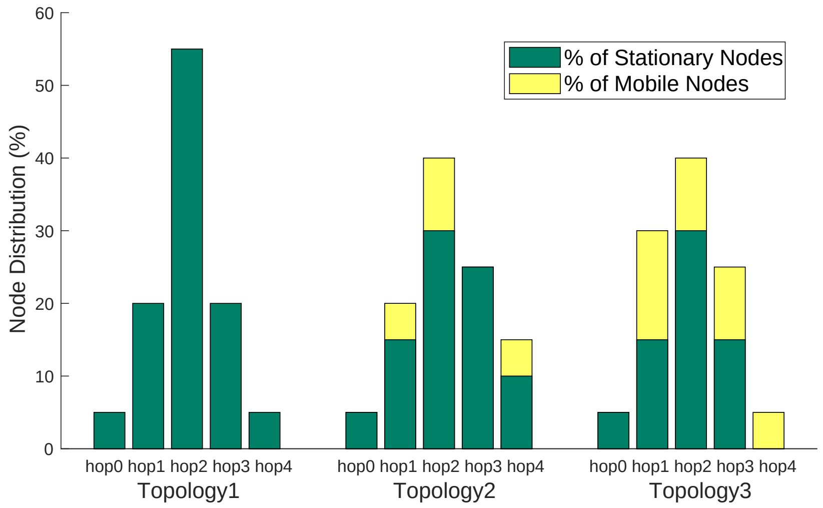


Figure 10. Three node topologies with different portions of mobile nodes with respect to hop distance to a host node.

We assume that edge devices use a wireless radio interface of the short range radio like 802.11 or 802.15.4 under the fixed transmission power setting. To simulate wireless links among them, a path-loss shadowing model is used with a path-loss exponent of 3.0, a reference loss of 46.6777 dB, and an additive Gaussian noise of $N(0, 3^2)$ in dB, which is one of the popular urban area parameter settings [30]. We let each node establish its connection with a neighboring node through a reliable link where its packet reception rate (PRR) is 0.7 or higher with the communication range of 60 m. The speed of mobile nodes ranges from 0 to 2.2 m/s, which belongs to the human walking speed. All other simulation environments and parameters are listed in Table 1.

Table 1. Simulation environment and parameters.

Environment	Value
Simulation Area	300 m $\times$ 300 m
# of Host Node	1
# of Edge Nodes	20
Communication Range	60 m
Speed of Mobile Nodes	0–2.2 m/s
N_T	0–7
T_W	1 s
T_{LQ}	0.6
T_{CQ}	-0.01
$TaskDeadline$	60 s

We consider a data intensive application such that the collected data amount is huge, incurring higher network bandwidth and latency in case of uploading to a remote server, while the computation workload is close to around the level of an image compressor or electroencephalogram (EEG) game [31]. It is also assumed that each unit task has no inter-dependency. We simulate all of networking and computation procedures in a fog computing and IoT network simulator, iFogSim simulator [32] under a certain ordinary task environment as in Table 2 and a task deadline of 60 s, unless otherwise noted. We conduct 20 simulation runs per experiment and report the average and the standard deviation values. It is assumed that the computation power on the cloud side is 10 times more powerful compared to edge devices.

Table 2. Task environment.

Environment	Value
Input Task Size	1.5 MB
Output Task Size	25 KB
Network Packet Size	56 B
Task Computation Latency (Edge)	1 s
Task Computation Latency (Cloud)	0.1 s

We investigate network performance in terms of task completion rate and network overhead. We quantify the task completion rate by counting how many tasks are successfully completed within a distinct deadline. As for network overhead, we use the *Network Usage* metric provided by iFogSim, which is the measure of message size by network delay in the (byte $\cdot$ second) unit basis.

We compare our algorithm under three different network topologies against a simple tree-based resource provider network [33] and a cloud-based computation. The tree-based resource provider network is formed purely based on the parent-to-child link establishment, without using a concept of stem or leaf node. Regarding the cloud-based service, a Wi-Fi gateway and ISP gateway are configured to be connected to a host node. Some empirical network latency setting [32] is used in experiments according to random selection per experiment within those ranges as in Table 3.

Table 3. Network latency.

Source	Destination	Latency
Edge Node	Edge Node	2–4 ms
Edge Node	Cloud Datacenter	106–108 ms

To achieve the feasibility of experimental results, we design experiments such that network performance of our algorithm is validated under various environmental factors: node mobility and interference level by varying task intensity and task deadline. While comparing against traditional approaches, we show how effectively our algorithm reacts to resource scarcity and instability due to mobility by dynamically accommodating necessary resources from the connected fog device pool.

6.1. Effects of Parameter and Topology

To find out a suitable operation environment, we investigate how the connection quality measure should consider dynamic link variations. As Figure 11 shows, we vary the number of CQ calculation trials, N_T from 0 to 7 where their consecutive interval, T_W is 1 second. As for all topology cases, increasing N_T up to 3 leads to the highest task completion rate, whereas the rate beyond 3 starts slightly decreasing. This means that, across a stable stationary network and even a dynamic network with mobile nodes, checking connection quality with a certain window of 4 seconds (i.e., $N_T + 1$) is an essential step to choose suitable stem PN nodes out of neighboring nodes. Beyond three trials, the connection checking procedure may be considered as a critical burden that affects a given task completion deadline. Thus, this rather degrades the task completion rate to some degree. In preceding experiments, $N_T = 3$ is used.

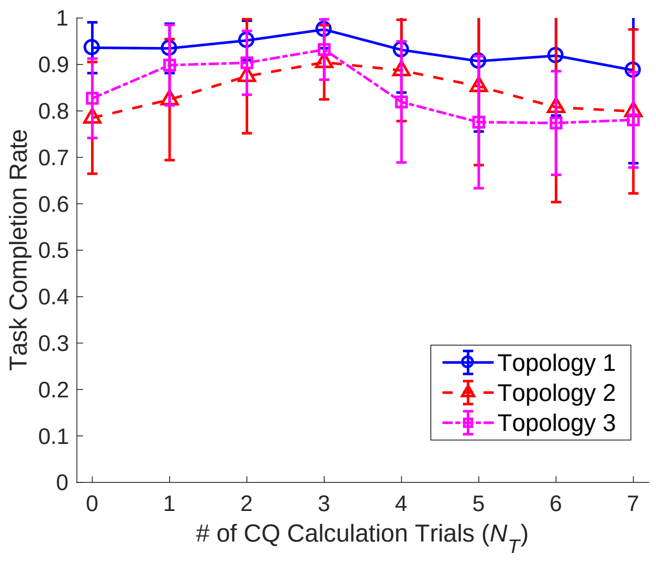
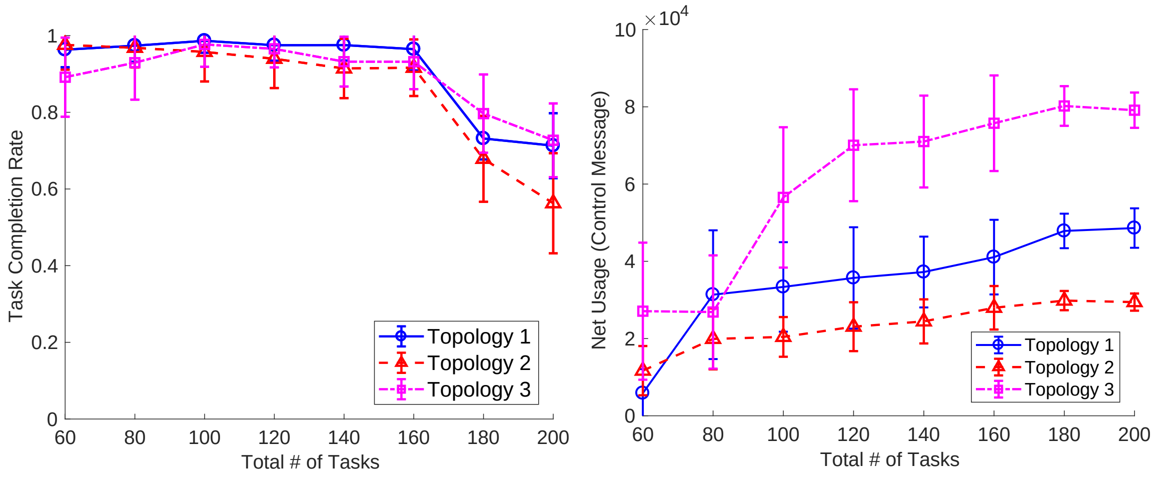


Figure 11. Task completion rate with respect to connection quality (CQ) calculation trials where the number of tasks is 140.

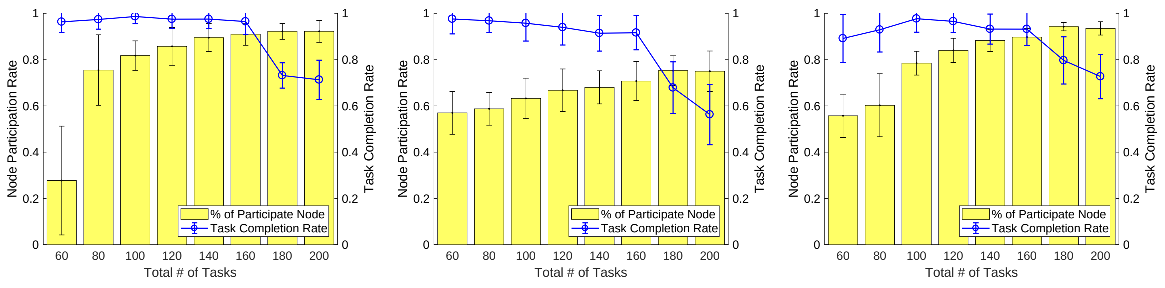
To examine how task intensity affects network performance, we measure task completion rate and network usage with respect to total number of tasks. Under all three topology environments, our algorithm achieves the high task completion rate above 90% in the range of 60–160 tasks as in Figure 12a. However, beyond that task range, such as 180 and 200 tasks, the task completion rate starts being reduced down under 80% due to lack of available computation resources in the network topology. As for network overhead, the control overhead increases up to some point and saturates as the total number of tasks does. The network overhead under Topology 3 is higher than other topologies since more dynamic link variation and its resulting reconfiguration may have occurred more frequently.



(**a**) Task completion rate with respect to task intensity (**b**) Control network overhead with respect to task intensity

Figure 12. Performance in three different node topologies with respect to task intensity.

To identify the degree of fog node involvement with respect to computational needs for a host node, we quantify the number of participating fog nodes to complete a given task by varying the task intensity as in Figure 13. As the total number of tasks increases, our algorithm dynamically and progressively accommodates more fog devices to meet the offloading computation requirement in all three environments. This implies that our work reacts to the scarcity of computing resource at the device side by making fog nodes participate in the network side.



(**a**) Topology 1 (**b**) Topology 2 (**c**) Topology 3

Figure 13. Fog node participation rate with respect to task intensity in three different node topologies.

While keeping the same task intensity, we investigate the effect of task completion deadline on task completion rate. As we vary the deadline from 10 s to 60 s in Figure 14, our algorithm successfully puts most of the given tasks from 20 s and above by letting more edge devices be engaged with the offloading. As the deadline gets relaxed at 40 s, our algorithm effectively utilizes the time margin, while reducing the number of participating nodes for offloading. This means that our offloading scheme finds a good balance between time and resources, while achieving the high completion rate of 90% and above.

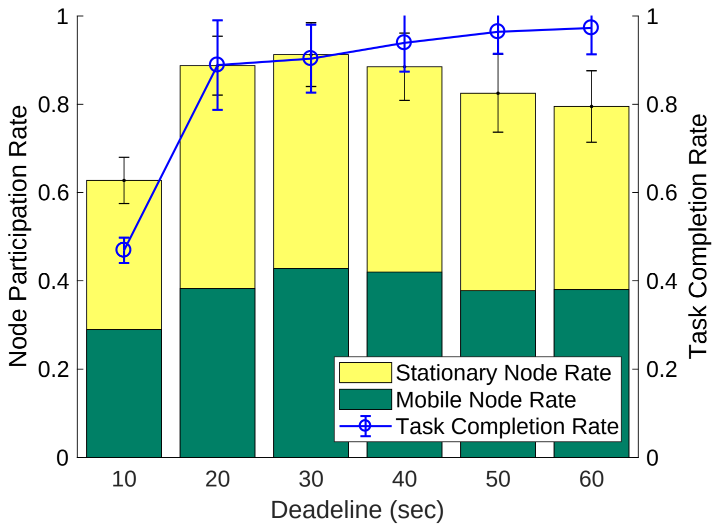
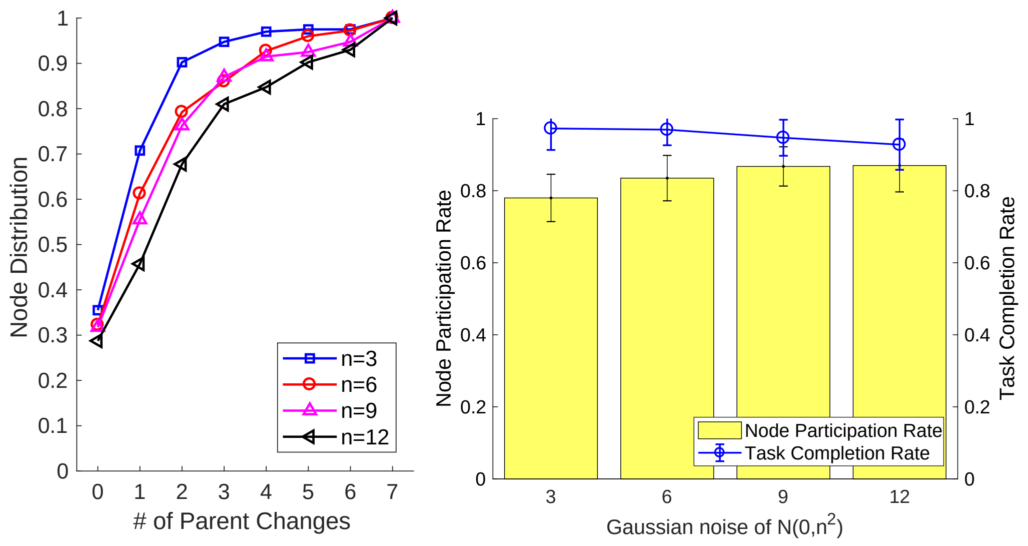


Figure 14. Task completion rate and node participation rate with respect to task deadline where the number of tasks is 120 in Topology 3.

We also investigate how the exterior wireless interference affects resource network formation and task completion performance. By varying the noise deviation n as in the additive White Gaussian noise $N(0, n^2)$ in the path-loss shadowing model, we count the number of parent changes in the resource network, and measure task completion rate and node participation rate in Figure 15. As Figure 15a shows, more frequent parent changes in the resource provider network are observed. Its resulting task completion performance, on the other hand, does not degrade much: by extending the range of fog device pool, our algorithm successfully manages network instability for achieving a high level of task completion performance as in Figure 15b.



(**a**) Cumulative node distribution with respect to parent changes

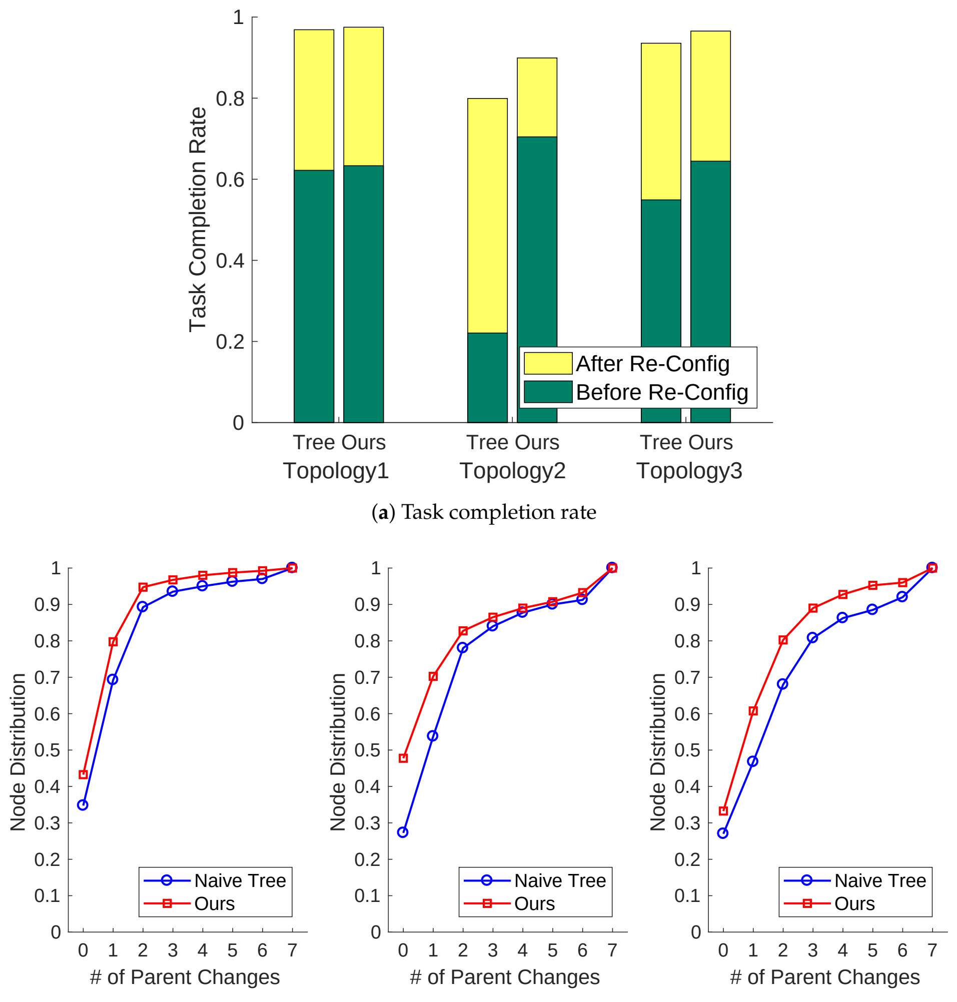
(**b**) Node participation rate and task completion

Figure 15. Performance under different interference levels with the Gaussian noise of $N(0, n^2)$ in dB where the number of tasks is 120 in Topology 3.

6.2. Performance Comparison

We compare the task computation performance with a counterpart algorithm that uses a naive tree structure, but keeps its following same reconfiguration to examine how our network formation criteria affect the performance. The naive yet effective tree algorithm like broadcast tree [33] forms a parent-to-child relationship based on a parent's broadcast connection timing without considering link stability and mobility variations of child nodes. We quantify task completion rate separately for before and after reconfiguration.

As Figure 16a shows, under relatively dynamic network environments such as Topologies 2 and 3, our network formation outperforms the naive tree-based algorithm in terms of net task completion rate. It is interesting to see the result that the performance based on the naive tree before reconfiguration is mediocre or low in dynamic environments where more frequent parent re-selection is needed (in Figure 16c,d), but it has significantly been improved thanks to our reconfiguration process.

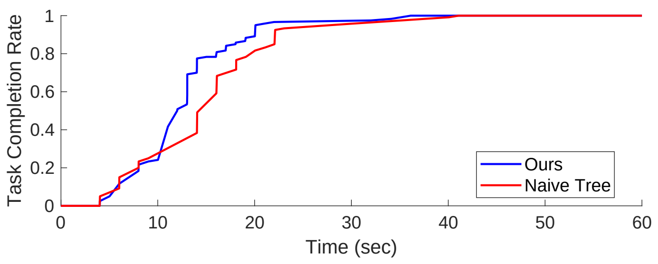


(**a**) Task completion rate

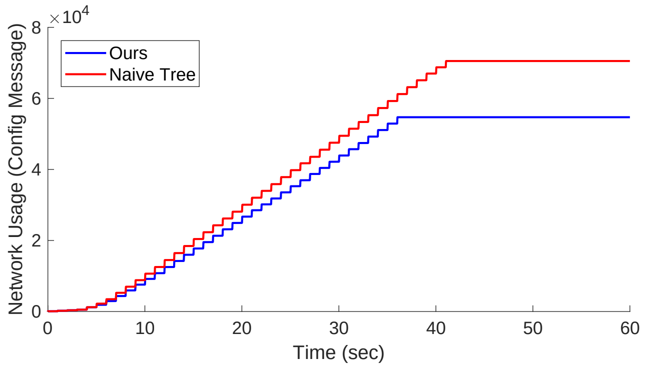
(**b**) Topology 1 (**c**) Topology 2 (**d**) Topology 3

Figure 16. Effect of reconfiguration on our algorithm and naive tree-based algorithm and cumulative node distribution with respect to parent changes at each topology.

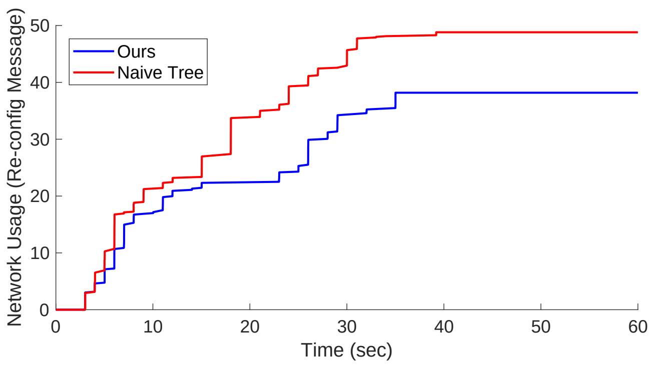
We show dynamic computation performance of our algorithm and the naive tree-based algorithm over time in Figure 17. As the running time passes, both algorithms try to manage their computation by distributing the given tasks to edge devices. Our algorithm accomplishes the goal of task completion within deadlines more quickly based on the more effective stem-leaf PN node-based tree structure in Figure 17a. As for network overhead in terms of control and data, both network overheads with our algorithm are lower than the naive tree-based algorithm as in Figure 17b–d. It should be noted that the network overhead for task redistribution is very low compared to the normal control signalling overhead for constructing a resource provider network and managing task distribution. This implies that, by investing some minimal network cost for reconfiguration, an even higher task completion rate can effectively be achieved (as shown in Figure 16a).



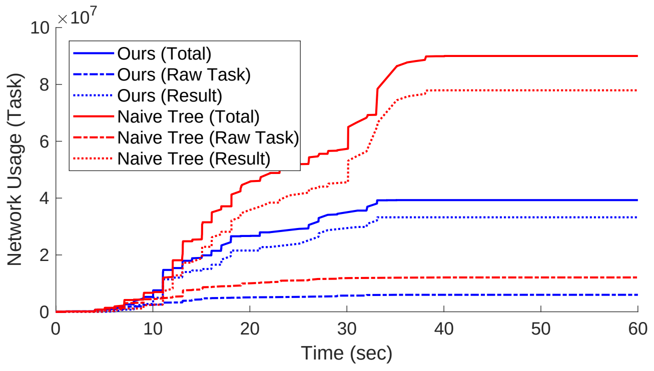
(**a**) Task completion rate over time



(**b**) Network overhead for control except reconfiguration over time



(**c**) Network overhead for control of reconfiguration over time



(**d**) Network overhead for task data over time

Figure 17. Dynamic computation performance for our algorithm and a naive tree-based algorithm over time until the deadline where the number of tasks is 120 in Topology 3.

Lastly, we compare our algorithm against a cloud-based computation where a required task is delivered from a host node towards a cloud server through a Wi-Fi gateway and ISP gateway, and its computed result at the cloud is delivered to the host. If we are allowed to use stationary edge devices with the highest stability in networks, the overall task completion time of both approaches is within the deadline, whereas our algorithm significantly reduces its required network overhead with a factor of up to 4.6, compared to the cloud-based computation as in Figure 18. As the stability of nodes degrades, the task completion time and the network overhead increase. The performance between ours in Topology 3 and Cloud is almost similar: even under an unstable network environment consisting of mobile nodes of 40%, our algorithm effectively turns nearby edge devices into available common computation resources, making computation performance almost indistinguishable in a distributed manner.

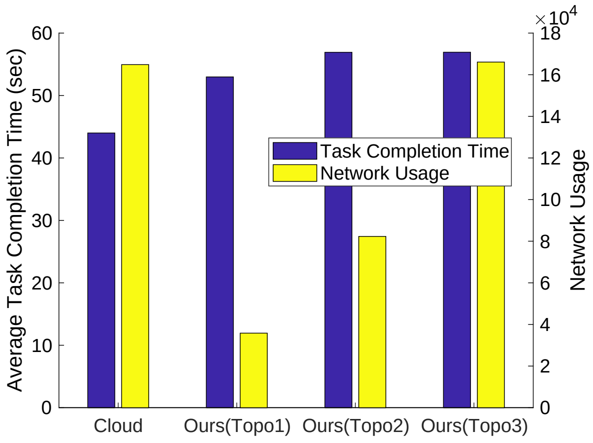


Figure 18. Distributed computation offloading vs. centralized cloud-based computation under 160 tasks and a 60 s deadline.

7. Conclusions

We have presented an on-demand distributed computation offloading framework in a dynamic fog network environment. Our approach designs an efficient yet reliable resource provider network by soliciting from nearby edge devices and a task allocation algorithm on top of it in a progressive manner. To make the architecture resilient to computation losses due to link volatility from wireless and node mobility, our work has embedded a following task redistribution phase so that some additional resources can be solicited flexibly and quickly. The proposed computation offloading architecture achieves high task completion rate within a designated computation deadline, while reducing a huge amount of network overhead.

We have validated our algorithm based on simulation experiments in the iFogSim environment under various network settings, compared to counterpart algorithms including traditional cloud-based computation.

For future work, it would be interesting to implement our proposed architecture on real-world edge devices so that they learn and execute deep neural networks in a distributed collaborative fashion. In addition, designing an energy-aware offloading algorithm considering limited energy availability and node loss due to the energy shortage would make the framework more feasible in real-world scenarios. Once the energy aspect is considered, we may extend to mid range and wide range wireless interfaces such as 5G considering various aspects of transmission power and data rate control beyond a relatively short range wireless under a fixed transmission power considered in this paper.

The current work focuses on a single end-user that wants to offload computation tasks with somewhat not-too-high computing intensity. It would be possible to take a hybrid approach by mixing local edge computing resources with cloud computing resources depending on the degree of task intensity and the type of multi-user application.

Author Contributions: Conceptualization, H.L.; Funding acquisition, H.L.; Investigation, Y.J. and H.L.; Methodology, Y.J. and H.L.; Project administration, H.L.; Resources, H.L.; Software, Y.J.; Supervision, H.L.; Validation, Y.J.; Writing—original draft, Y.J.; Writing—review and editing, H.L.

Funding: This work was supported by Samsung Research Funding Center of Samsung Electronics under Project Number SRFC-IT1803-00.

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

References

1. Botta, A.; De Donato, W.; Persico, V.; Pescapé, A. Integration of cloud computing and internet of things: A survey. *Future Gener. Comput. Syst.* **2016**, *56*, 684–700. [CrossRef]
2. Baccarelli, E.; Naranjo, P.G.V.; Scarpiniti, M.; Shojafar, M.; Abawajy, J.H. Fog of everything: Energy-efficient networked computing architectures, research challenges, and a case study. *IEEE Access* **2017**, *5*, 9882–9910. [CrossRef]
3. Bonomi, F.; Milito, R.A.; Zhu, J.; Addepalli, S. Fog computing and its role in the internet of things. In Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing, MCC@SIGCOMM, Helsinki, Finland, 17 August 2012.
4. Firdhous, M.F.M.; Ghazali, S. Fog Computing: Will it be the Future of Cloud Computing? In Proceedings of the Third International Conference on Informatics & Applications (ICIA), Kuala Terengganu, Malaysia, 8–10 October 2014.
5. Ravandi, B.; Papapanagiotou, I. A Self-Learning Scheduling in Cloud Software Defined Block Storage. In Proceedings of the 2017 IEEE 10th International Conference on Cloud Computing (CLOUD), Honolulu, CA, USA, 25–30 June 2017 pp. 415–422.
6. Skala, K.; Davidovic, D.; Afgan, E.; Sovic, I.; Sojat, Z. Scalable Distributed Computing Hierarchy: Cloud, Fog and Dew Computing. *OJCC* **2015**, *2*, 16–24.
7. Nishio, T.; Shinkuma, R.; Takahashi, T.S.; Mandayam, N.B. Service-oriented heterogeneous resource sharing for optimizing service latency in mobile cloud. In Proceedings of the First, International Workshop on Mobile Cloud Computing & Networking MobileCloud '13, Bangalore, India, 29 July 2013.
8. Sivakumar, R.; Das, B.; Bharghavan, V. Spine routing in ad hoc networks. *Clust. Comput.* **1998**, *1*, 237–248. [CrossRef]
9. Adabi, S.; Jabbehdari, S.; Rahmani, A.M.; Adabi, S. A Novel Distributed Clustering Algorithm for Mobile Ad-hoc Networks. *J. Comput. Sci.* **2008**, *4*, 161–166. [CrossRef]
10. Chauhan, N.; Awasthi, L.K.; Chand, N.; Katiyar, V.; Chugh, A. A Distributed Weighted Cluster Based Routing Protocol for MANETs. *Wirel. Sens. Netw.* **2011**, *3*, 54–60. [CrossRef]
11. Abdelwahab, S.; Hamdaoui, B.; Guizani, M.; Znati, T. Replisom: Disciplined tiny memory replication for massive iot devices in lte edge cloud. *IEEE Internet Things J.* **2015**, *3*, 327–338. [CrossRef]
12. Chen, X.; Jiao, L.; Li, W.; Fu, X. Efficient multi-user computation offloading for mobile-edge cloud computing. *IEEE/ACM Trans. Netw.* **2015**, *24*, 2795–2808. [CrossRef]
13. Chen, M.; Hao, Y. Task offloading for mobile edge computing in software defined ultra-dense network. *IEEE J. Sel. Areas Commun.* **2018**, *36*, 587–597. [CrossRef]
14. Taleb, T.; Samdanis, K.; Mada, B.; Flinck, H.; Dutta, S.; Sabella, D. On multi-access edge computing: A survey of the emerging 5G network edge cloud architecture and orchestration. *IEEE Commun. Surv. Tutor.* **2017**, *19*, 1657–1681. [CrossRef]
15. Yousefpour, A.; Fung, C.; Nguyen, T.; Kadiyala, K.; Jalali, F.; Niakanlahiji, A.; Kong, J.; Jue, J.P. All one needs to know about fog computing and related edge computing paradigms: A complete survey. *J. Syst. Archit.* **2019**, *98*, 289–330. [CrossRef]
16. Chatterjee, M.; Das, S.K.; Turgut, D. WCA: A weighted clustering algorithm for mobile ad hoc networks. *Clust. Comput.* **2002**, *5*, 193–204. [CrossRef]
17. Zhang, D.; Ge, H.; Zhang, T.; Cui, Y.Y.; Liu, X.; Mao, G. New multi-hop clustering algorithm for vehicular ad hoc networks. *IEEE Trans. Intell. Transp. Syst.* **2018**, *20*, 1517–1530. [CrossRef]
18. Malatras, A. State-of-the-art survey on P2P overlay networks in pervasive computing environments. *J. Netw. Comput. Appl.* **2015**, *55*, 1–23. [CrossRef]
19. Stoica, I.; Morris, R.; Liben-Nowell, D.; Karger, D.R.; Kaashoek, M.F.; Dabek, F.; Balakrishnan, H. Chord: A scalable peer-to-peer lookup protocol for internet applications. *IEEE/ACM Trans. Netw. (TON)* **2003**, *11*, 17–32. [CrossRef]

20. Ratnasamy, S.; Francis, P.; Handley, M.; Karp, R.; Shenker, S. A Scalable Content-addressable Network. In Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, SIGCOMM '01, San Diego, CA, USA, 27–31 August 2001; pp. 161–172.
21. Rowstron, A.; Druschel, P. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms and Open Distributed Processing, Heidelberg, Germany, 12–16 November 2001*; Springer: Berlin/Heidelberg, Germany, 2001; pp. 329–350.
22. Zhao, B.Y.; Huang, L.; Stribling, J.; Rhea, S.C.; Joseph, A.D.; Kubiatowicz, J.D. Tapestry: A resilient global-scale overlay for service deployment. *IEEE J. Sel. Areas Commun.* **2004**, *22*, 41–53. [CrossRef]
23. Clarke, I.; Sandberg, O.; Wiley, B.; Hong, T.W. Freenet: A distributed anonymous information storage and retrieval system. In *Designing Privacy Enhancing Technologies*; Springer: Berlin/Heidelberg, Germany, 2001; pp. 46–66.
24. Gnutella. *Gnutella Protocol Specification*, v0.6; Gnutella Developer Forum, 2002. Available online: http://rfc-gnutella.sourceforge.net/src/rfc-0_6-draft.html (accessed on 23 September 2019)
25. Chen, W.; Wang, D.; Li, K. Multi-user Multi-task Computation Offloading in Green Mobile Edge Cloud Computing. *IEEE Trans. Serv. Comput.* **2018**, 1. [CrossRef]
26. Apostolopoulos, P.A.; Tsiropoulou, E.E.; Papavassiliou, S. Game-theoretic Learning-based QoS Satisfaction in Autonomous Mobile Edge Computing. In Proceedings of the 2018 Global Information Infrastructure and Networking Symposium (GIIS), Thessaloniki, Greece, 23–25 October 2018; pp. 1–5. [CrossRef]
27. Mao, Y.; Zhang, J.; Letaief, K.B. Dynamic Computation Offloading for Mobile-Edge Computing With Energy Harvesting Devices. *IEEE J. Sel. Areas Commun.* **2016**, *34*, 3590–3605. [CrossRef]
28. Tsiropoulou, E.E.; Vamvakas, P.; Katsinis, G.K.; Papavassiliou, S. Combined Power and Rate Allocation in Self-optimized Multi-service Two-tier Femtocell Networks. *Comput. Commun.* **2015**, *72*, 38–48. [CrossRef]
29. Lucas, J.M.; Saccucci, M.S.; Baxley, R.V.; Woodall, W.H.; Maragh, H.D.; Faltin, F.W.; Hahn, G.J.; Tucker, W.T.; Hunter, J.S.; MacGregor, J.F.; et al. Exponentially weighted moving average control schemes: Properties and enhancements. *Technometrics* **1990**, *32*, 1–12. [CrossRef]
30. Masini, B.M.; Bazzi, A.; Zanella, A. A Survey on the Roadmap to Mandate on Board Connectivity and Enable V2V-Based Vehicular Sensor Networks. *Sensors* **2018**, *18*, 2207. [CrossRef] [PubMed]
31. Zao, J.K.; Gan, T.T.; You, C.K.; Méndez, S.J.R.; Chung, C.E.; Wang, Y.T.; Mullen, T.; Jung, T.P. Augmented Brain Computer Interaction Based on Fog Computing and Linked Data. In Proceedings of the 2014 International Conference on Intelligent Environments, Shanghai, China, 30 June–4 July 2014; pp. 374–377. [CrossRef]
32. Gupta, H.; Dastjerdi, A.V.; Ghosh, S.K.; Buyya, R. iFogSim: A Toolkit for Modeling and Simulation of Resource Management Techniques in Internet of Things, Edge and Fog Computing Environments. *Softw. Pract. Exp.* **2017**, *47*, 1275–1296. [CrossRef]
33. Gandhi, R.; Mishra, A.; Parthasarathy, S. Minimizing Broadcast Latency and Redundancy in Ad Hoc Networks. *IEEE/ACM Trans. Netw.* **2008**, *16*, 840–851. [CrossRef]