Efficient Data Placement and Retrieval Services in Edge Computing

Junjie Xie*, Chen Qian†, Deke Guo*, Xin Li†, Shouqian Shi†, Honghui Chen*
*Science and Technology on Information Systems Engineering Laboratory
National University of Defense Technology, Changsha Hunan 410073, China
†Department of Computer Engineering, University of California Santa Cruz, CA 95064, USA

***Abstract*—Edge computing is a new paradigm in which the computing and storage resources are placed at the edge of the Internet. Data placement and retrieval are fundamental services of edge computing when a network of edge servers collaboratively provide data storage. These services require short-latency and low-overhead implementation in network devices and load balance on edge servers. However existing methods such as distributed hash tables (DHTs) are not able to achieve efficient data placement and retrieval services in the edge computing environment. This paper presents GRED, an efficient data placement and retrieval service for edge computing, which is efficient in not only the load balance but also routing path lengths and forwarding table sizes. GRED utilizes the software defined networking paradigm to support a virtual-space based DHT with only one overlay hop. We implement GRED in a P4 prototype. Experimental results show that GRED uses <30% routing path lengths and achieves better load balance among edge servers compared to using Chord, a well-known DHT solution.**

I. Introduction

A recent trend is to offload computing and storage to the network edges so as to enable computation-intensive and latency-critical applications. This technology, called *Edge Computing*, has been proposed to shift computing and storage capacities from the remote Cloud to the network edge in close proximity to mobile devices, sensors, and end users [1]. Edge computing promises the dramatic reduction in network latency and traffic volume, tackling the key challenges for materializing 5G vision. Meanwhile, the edge of the Internet is also an optimal site for aggregating, analyzing and distilling bandwidth-hungry sensor data from devices such as video cameras and the appropriate platform for critical IoT services and applications [2]. Terms such as 'cloudlets', 'micro data centers', and 'fog computing' have been used in the literature to refer to similar edge-located services [3] [4].

Edge nodes can perform computing offloading, data storage, caching and processing for edge users, where edge nodes consist of one or multiple edge servers. Unlike cloud data-centers, edge nodes are usually geographically distributed and have heterogeneous computation and storage capacities [1]. Always offloading the data and computation of a user at the closest edge node may not be a valid solution because 1) the user may be mobile and 2) one edge node may have limited resource. Hence we consider a large number of edge nodes in an interconnected edge network that collaboratively serve the resource pool of storage and computation offloading for users.

A core operation in edge computing is to support the efficient *data placement and retrieval* when multiple edge nodes work together. In this work, we define "data placement" as the process of delivering a given data item to an edge node for storage and "data retrieval" as the process of finding the storage node of a given data item and requesting the node to deliver the data to a user. Hence they are essentially *overlay services with network-layer implementation*.

However, the data placement and retrieval services in edge computing face at least two challenges. First, these services should have short-latency and low-overhead implementation on the user side and network routers/switches. For example, it is impractical to maintain a complete index of all data-to-location mappings at an edge device or inside a router. Second, achieving load balance among edge nodes is very important, which requires that no server should be overloaded when there is the available resource on other servers. The limited and heterogeneous computation and storage capacities of different edge nodes further complicate the problem.

To solve these problems, we propose short-latency and low-overhead data placement and retrieval services for edge computing, called Greedy Routing for Edge Data (GRED). GRED include two innovative ideas. First, GRED supports a DHT of edge nodes with only one overlay hop. Second, GRED utilizes the Software Defined Networking (SDN) paradigm [5] to implement efficient routing support of the one-hop DHT on programmable switches[1]. In particular, the SDN controller maintains a virtual space. Switches and data items are mapped to different positions in the space according to their IDs. The data will be stored in an edge server connected to the switch whose position is nearest to the data position in the virtual space.

GRED is efficient in terms of both routing path lengths and forwarding table sizes. Each data placement/retrieval request in GRED only needs one overlay hop. In detail, in the control plane of GRED, we design a virtual space construction algorithm to assign the switches to the points in the virtual space, such that the Euclidean distance between two switches is proportional to their network distance. It has been shown that under this circumstance, the routing stretch of the network can be optimized [6]. Furthermore, to achieve the load balance

[1]Hereafter we use "switches" to denote network forwarding devices for the compatibility to the SDN conetxt, although they can be routers.

among edge nodes, we further optimize the switches' positions considering that the data is stored in the network based on their postions in the virtual space.

Meanwhile, to minimize the forwarding table size, the data plane of GRED does not need a new flow entry for every placement/retrieval request. Instead, the data plane performs greedy forwarding based on the next-hop switch's position determined by greedy forwarding, which is implemented in P4 [7], a programmable data plane development tool. Hence the forwarding table size is independent of the network size and the number of flows in the network. We conducted extensive experiments, using both P4 implementation and simulations, to evaluate the performance of GRED. Theoretical analysis shows the correctness and efficiency of GRED. Experimental results show that GRED uses $<30\%$ routing cost and achieves better load balance among edge nodes compared to using Chord [8], a well-known DHT.

The rest of this paper is organized as follows. In section II, we introduce the background and motivation of this paper. Section III presents the system overview. In Section IV, we describe the virtual position construction in the control plane, which is the base of GRED. Section V details the mechanism of GRED placement and retrieval. We discuss the network dynamic and the data copies in the network in Section VI. In Section VII we evaluate the performance of GRED. We introduce the related work in Section VIII and conclude this paper in Section IX.

II. Background and Motivation

A. Motivation

This work focuses on the core function in edge computing: data placement and retrieval, which provide efficient support for a large number of upper-layer applications. First, these services should have the efficient implementation on the user side and network routers/switches. The efficiency discussed in this work aims at both network efficiency such as short routing path lengths that imply short latency and low bandwidth cost, and forwarding efficiency such as small table size that achieves low infrastructure cost. The second challenge is the load balance when a large amount of data are stored in those edge servers. Load balance requires that no edge server should be overloaded when there is the available resource on other servers. The limited and heterogeneous computation and storage capacities of different edge nodes further complicate the problem.

To enable the data placement and retrieval services, we recall that there has been some related work in peer-to-peer (P2P) networks. However, existing approaches in P2P can not meet the low-latency routing requirement in edge computing. In those systems, a data object is associated with a key and each node in the system is responsible for storing a certain range of keys. For example, Chord [8] is a widely used DHT solution for the data storage and lookup in P2P networks. As shown in Fig. 1, an edge network consists of 12 edge nodes where each edge node has a unique identifier. The data with the key 12 is stored in node 15 based on the storage principle in

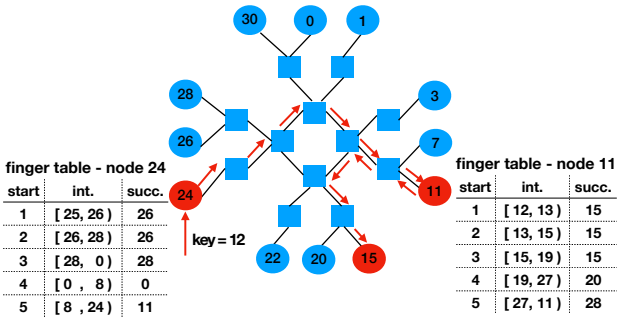


Fig. 1. Finger tables and key locations in DHT-based storage system.

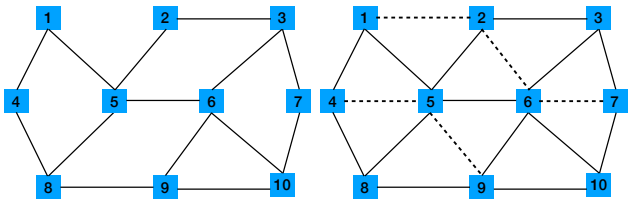
Chord [8]. When a user accessing node 24 needs to retrieve the data with the key 12 located in the interval $[8, 24)$, the lookup request is first sent to node 11 based on the finger table of node 24. Note that the finger table indicates the successor node to find a data. Then, node 11 will continue to forward the lookup request to node 15 based on its finger table. In this case, the path length of the lookup request is 11, which is significantly longer than the shortest path between node 24 and node 15 with only 5 physical hops.

In the DHT-based storage systems, the overlay routing takes up to $O(\log n)$ overlay hops for n nodes and each overlay hop may include multiple network-layer hops. The main reason is the mismatch between the overlay network and the physical network. That is, the path length for locating a data item is heavily longer than the shortest path. Furthermore, such mismatch causes a high routing stretch, which results in the long response delay. In addition, the load balance in Chord [8] is not perfect. Although Chord can achieve a better load balance by adding more virtual nodes to each real node, it also increases the routing table space usage and makes the system more complicated. Therefore, in this paper, we look for a better design with low routing stretch and better load balance for data placement and retrieval in edge computing.

B. Guaranteed delivery on a DT Graph

In our design, each switch does a greedy forwarding. To achieve the guaranteed delivery, a virtual Delaunay Triangulation (DT) graph is maintained in the control plane of the netowrk. Note that greedy routing on an arbitrary graph is prone to the risk of being trapped at a local optimum, i.e., routing stops at a non-destination node that is closer to the destination than any of its neighbors. However, on a DT, it is guaranteed that greedy routing always succeeds to find the node closest to destination p. For a given set P of discrete points (called nodes) in a plane is a triangulation $DT(P)$ such that no point in P is inside the circumcircle of any triangle in $DT(P)$. If two nodes share a DT edge, they are called DT neighbors. One important property of DT is that greedy routing to a destination location p on a DT graph always stops at a node that is closest to p among all nodes [9].

Note the main difficulty of maintaining a DT graph in a network of edge nodes is that two DT neighbors may not be connected by a physical link. Hence they cannot directly



(a) A physical connectivity graph. (b) A multi-hop DT graph.

Fig. 2. An illustration of a physical network and the multi-hop DT

forward messages between them. For an arbitrary layer-2 network, the MDT [10] protocol was designed for nodes to construct a distributed multi-hop DT graph. As shown in Fig. 2(b), there is a DT graph of 10 nodes in a 2D Euclidean space, and the physical connectivity of those 10 nodes is shown in Fig. 2(a). In Fig. 2(b), Nodes 5 and 1 are both physical neighbors and DT neighbors. However, DT neighbors, nodes 1 and 2, are not connected directly. Hence in a multi-hop DT graph, node 1 transfers packets to node 2 by the multi-hop path $\{1, 5, 2\}$ in Fig. 2(a). Therefore, node 2 is called the multi-hop DT neighbor of node 1. For a set of nodes that maintain a correct multi-hop DT, given a destination p, it is proved that MDT-greedy forwarding always succeeds to find a node that is closest to p, for nodes located in a Euclidean space (2D, 3D, or a higher dimension) [10].

III. System Overview

The GRED protocol specifies how to place a data item and to retrieve it from the edge servers given a data identifier. We design the GRED protocol while utilizing the advantage of software-defined networking [5], which centralizes the network intelligence in the network controller. The switches in the data plane only forward packets according to the installed rules derived from the controller. When we apply the principle of SDN to the edge computing, the network is called a Software-Defined Edge Network (SDEN). As shown in Fig. 3, we define the general hierarchical architecture of an SDEN, which consists of the control plane, the switch plane, the edge plane and the user plane. The user plane includes the mobile users and various edge devices, such as autonomous vehicles and IoT devices. In SDEN, the users access the network by wireless Access Points (APs). Those APs and edge servers are connected to network switches and constitute the edge plane. The switches provide data communication services among edge servers based on the forwarding entries derived from the controller in the control plane.

The GRED protocol mainly consists of the functions in the control plane and the switch plane.

Control plane associates each switch to a point in the virtual space and computes a DT graph of all points. It then inserts related forwarding entries into switches based on their DT neighbors in the virtual space. It is worth noting that the control plane proactively distributes forwarding rules to switches to perform greedy forwarding based on the destination position rather than per-flow information. The mechanism

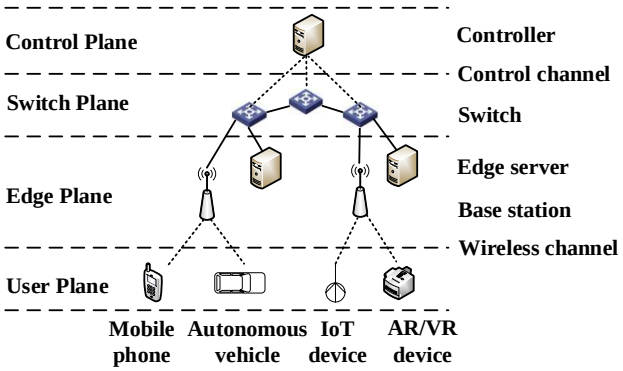


Fig. 3. General architecture of software-defined edge network.

can efficiently reduce the load of the control plane and the size of forwarding tables, because the switches can forward data requests based on the pre-installed rules without the interaction of the control plane.

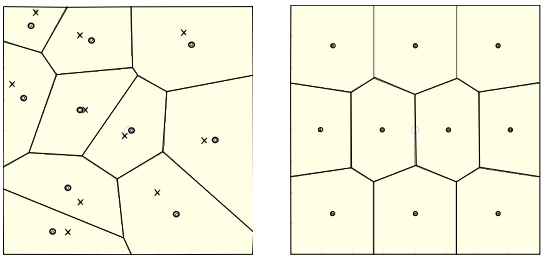
Switch plane consists of switches and transfer links. The switch greedily forwards a data request to the correct server based on the installed rules. More precisely, the switch first achieves the data position in the virtual space by hashing the data identifier. Then the switch finds a DT neighbor that is closest to the data position and forwards the packet to it, by either a direct link or a multi-hop path.

When placing a data item to an edge server, the hash value $H(d)$ of the data identifier d is firstly calculated. In this paper, we adopt the hash function, *SHA-256* [11], which outputs a 32-byte binary value. Furthermore, the hash value $H(d)$ is reduced to the scope of the 2D virtual space, which is constructed by the control plane. We only use the last 8 bytes of $H(d)$ and convert them to two 4-byte binary numbers, x and y. We limit that the coordinate value ranges from 0 to 1 in each dimension. Then, the position of a data in 2D is $(\frac{x}{2^{32}-1}, \frac{y}{2^{32}-1})$. The position can be stored in decimal format, using 4 bytes per dimension. Hereafter, for any data identifier, d, we use $H(d)$ to represent its position. Last, the data is greedily forwarded to the switch whose position is the nearest to the data position in the virtual space, and further, the switch determines a unique edge server to store the data.

The GRED protocol greedily forwards the data request based on the data position and the switches' positions in the virtual space. Determining the positions of switches is the key to achieve the advantages of GRED. It is because bad virtual positions will result in long routing path and bad load balance among edge servers. Therefore, we first detail the procedure of the virtual position construction in the next section.

IV. Virtual Position Construction

The control plane of GRED first determines the positions of all switches in a virtual 2D Euclidean space, then constructs a multi-hop DT based on those virtual positions. After that, the control plane inserts the forwarding entries into the switches. Then, switches performs greedy forwarding based on those forwarding entries.



(a) A generic voronoi diagram. (b) A centroidal voronoi diagram.

Fig. 4. The voronoi tessellation of 10 points.

A. Calculating the positions of switches

To ensure the low routing stretch of greedy routing, it is required that the Euclidean distance of two switches in the virtual space is proportional to their network distance, which is called greedy network embedding [6]. Note that the network topology and state can be obtained in the control plane by collecting switch, port, link, and host information [5][12]. Then, the control plane can compute the shortest path matrix between switches. However, the key challenge is how to achieve the coordinate matrix of n points where the shortest path lengths between n switches can be indirectly reflected by the distances between points in the virtual space. In other words, we need to solve the problem of finding a point configuration that represents a given scalar-product matrix [13].

To achieve this goal, the M-position algorithm is designed to calculate the switches' positions in the virtual space. The shortest path matrix L=$[l_{ij}]$, where l_{ij} is the length of the shortest path between the i^{th} and j^{th} switches, is first calculated. The M-position algorithm utilizes the fact that the coordinate matrix can be derived by eigenvalue decomposition from B=QQ' where the matrix B can be computed from the distance matrix L. Then, the matrix Q can be uniquely determined by the matrix B. Therefore, the M-position algorithm first constructs the scalar product matrix B by multiplying the squared ditance matrix $L^{(2)}$ with the matrix J=$I-\frac{1}{n}A$. That is B=$-\frac{1}{2}JL^{(2)}J$, where n is the number of switches, and A is the squared matrix with all elements are 1. This procedure is called double centering [14]. Then, the m largest eigenvalues $\lambda_1, \lambda_2, ..., \lambda_m$ and corresponding eigenvectors $e_1, e_2, ..., e_m$ of the matrix B is determined, where m is the number of dimensions. Last, the coordinates of the switches Q=$E_m\Lambda_m^{1/2}$ are achieved, where E_m is the matrix of m eigenvectors and Λ_m is the diagonal matrix of m eigenvalues of the matrix B, respectively. After that, each switch will be assigned a point in the virtual space from the coordinate matrix Q.

B. Refining the positions of switches

One potential problem is that the M-position algorithm determines the positions of switches without considering the load balance among edge nodes. Fig. 4(a) shows a Voronoi Diagram [15] of 10 crosses where each cross is associated with a Voronoi cell. In each cell, the distance from a point to the corresponding cross in the same region is not greater than its distance to the other crosses in the diagram. Recall that a data item is stored at an edge server connected to the switch whose position is nearest to the data position in the virtual space. Assume that the switches are located in those crosses in Fig. 4(a). Then, when data items are mapped into the whole space evenly, it is obvious that there would be load imbalance among those switches because these Voronoi cells have different sizes. To achieve the load balance among those switches, it is necessary to further refine the coordinates of switches so that each Voronoi cell has the equal size.

Algorithm 1 Refine the coordinates of switches in the virtual space while achieving the load balance.

Require: The coordinates of the switches Q achieved in Section IV-A.

Ensure: The updated coordinates of the switches Q^*.

1: Set Q^*←Q; set j_i=1 for i=$1, \ldots, n$ where q_i∈Q;
2: Obtain a random sample W of 1000 points from the virtual space Ω that is constructed by the control plane with uniform probability;
3: For each point w∈W, find the q_i that is closest to w; denote the index of that q_i by i^*;
4: Set q_{i^*}←$\frac{j_{i^*}q_{i^*}+w}{j_{i^*}+1}$ and j_{i^*}←j_{i^*}+1; this new q_{i^*}, along with the unchanged q_i, i≠i^*, form the new set of points Q^*. Note that j_i-1 equals the number of times that the point q_i has been updated.
5: If this new set of points meets some convergence criterion, terminate; otherwise, go back to step 2.

In Fig. 4(a) the crosses are the Voronoi sites and the circles are the centroids of the corresponding Voronoi regions. Note that the sites and the centroids do not coincide in Fig. 4(a). However, Fig. 4(b) shows a 10-point Centroidal Voronoi Tesselation (CVT) [16], which can be viewed as an optimal partition corresponding to an optimal distribution of sites. That is, the crosses are the sites for the Voronoi tessellation and the centroids of the Voronoi regions. Further, we hope that the coordinates of switches are also the centroids of the corresponding Voronoi regions. After that, we can achieve the proper load balance when those data items are mapped into the virtual space evenly.

Inspired by the CVT, we design the C-regulation method, as shown in Algorithm 1, to further refine the positions of switches obtained by the M-position algorithm in Section IV-A. Given the number of sites n, a CVT is a minimizer (or a local minimizer) of the CVT *energy*, defined to be the square of the distance between each point in the region and its nearest site. Let Ω be a metric space with distance function ϕ. Assume that there are n sites, and $(q_k)_{1\leq k\leq n}$ be a site in the space Ω. If $\phi(r, P)$=$inf\{\phi(r, p)|p$∈$P\}$ denotes the distance between the point r and the subset P, then we define a region R_k associated with the site q_k as follows.

$$R_k=\{r\in\Omega|\phi(r, q_k)\leq\phi(r, q_j), j = 1, \ldots, n, j\neq k\} \quad (1)$$

That is, the region R_k is the set of all points in Ω whose distance to q_k is not greater than their distance to the other sites q_j, where j is any index different from k. Accordingly, these

regions are called Voronoi cells, and the diagram is a general Voronoi diagram. Furthermore, given a density function $\rho(\cdot)$ defined on Ω, the formulation of the CVT energy is as follows:

$$F((q_i, R_i), i{=}1, \ldots, n){=}\sum_{i=1}^{n}\int_{r\in R_i}\rho(r)|r-q_i|^2dr \quad (2)$$

The C-regulation algorithm is a sampling technique, which supplies a discrete estimate of this CVT energy. Each time this C-regualtion iteration is carried out, an attempt is made to modify the coordinates of switches in such a way that they are closer and closer to being the centroids of the Voronoi cells they generate. The iteration will terminate when the CVT energy is lower than a given threshold. We set that the number of sample points is 1000 in each iteration, and that can be more. Note that the C-regulation method could require fewer iterations when more points are sampled in each iteration. However, more sample points would incur more computing time in each iteration. In addition, the number of iterations can also be set as the termination condition. During a iteration of the C-regulation method, it should generally be the case that the CVT energy decreases from step to step. Furthermore, the impact of the number of iterations on the load balance is evaluated in Section VII-E3. When the C-regulation method terminates, we can achieve the updated coordinates of switches, which are indicated by the set of points Q^* in Algorithm 1.

C. Multi-hop DT construction

To achieve the guaranteed delivery, the control plane constructs a multi-hop DT in the virtual space. As shown in Fig. 2(b), that is a multi-hop DT graph of 10 points. Furthermore, greedy routing in a multi-hop DT provides the property of guaranteed delivery [10], which is based on a rigorous theoretical foundation. For a given set of nodes in a 2D space, a triangulation is to construct edges between pairs of nodes such that the edges form a non-overlapping set of triangles that cover the convex hull of the nodes. A DT in a 2D space is usually defined as a triangulation such that the circumcircle of each triangle does not include any node other than the vertexes of the triangle.

After obtaining the switches' positions in a set of points Q^*, a randomized incremental algorithm is designed to construct the DT $DT(Q^*)$ in the 2D virtual space [17]. We first add an appropriate triangle boundingbox to contain P. The points in P are inserted in random order, and a DT corresponding to the current point set is maintained and updated throughout the whole process. Last, we remove the boundingbox and relative triangles which cotains any vertex of the boudingbox triangle. Meanwhile, it is necessary to ensure that the union of all simplices in the triangulation is the convex hull of those points. Furthermore, greedy routing on a DT graph can achieve the guaranteed delivery [9]. That is, given a destination location p, the data packets always stop at a node that is closest to p among all nodes.

Considering the case of inserting v_i, $DT(v_1, v_2, \ldots, v_{i-1})$ formed by inserting all previous points $v_1, v_2, \ldots, v_{i-1}$ is already a DT. The change caused by inserting v_i is adjusted and $DT(v_1, v_2, \ldots, v_{i-1}) \cup v_i$ is made a new $DT(v_1, v_2, \ldots, v_i)$. The adjustment process is as follows. First, we determine which triangle (or edge) v_i falls on, and then connect v_i with the three vertices of the triangle to form three triangles (or connect the vertices of two triangles of the common edge to form four triangles). Since the newly generated edges and the original edges may not be Delaunay edges, a flipping [18] is conducted to make them all Delaunay edges to get $DT(v_1, v_2, \ldots, v_i)$. Take $DT(A, B, C, D)$ for example, we change the common edge $<B, D>$ to the common edge $<A, C>$ to produce two triangles that do meet the Delaunay condition when two original triangles do not meet the Delaunay condition [18]. This operation is called a flipping.

However, a key challenge is to ensure that each switch can transfer data packets to its DT neighbors note that a DT neighbor of a switch may not be the physical neighbor of the switch. Therefore, to achieve the guaranteed delivery, each switch maintains two kinds of flow entries in the GRED protocol, one makes it can forward requests to its physical neighbors, and another makes it forward requests to its multi-hop DT neighbors. Note that the switches that are not directly connected to some edge servers will not participate in the construction of the DT. Those switches are just used as the intermediate nodes to transfer data to the multi-hop DT neighbors. For a node u, each entry in its forwarding table F_u is a 4-tuple as follows.

$$<sour, pred, succ, dest>,$$

which is a sequence of nodes with $sour$ and $dest$ being the source and destination nodes of a path, and $pred$ and $succ$ being node u's predecessor and successor nodes in the path. F_u is used to forward packets to multi-hop DT neighbors. For a specific tuple t, we use $t.sour$, $t.pred$, $t.succ$, and $t.dest$ to denote the corresponding nodes in the tuple t. Although greedy routing does not always find a shortest route, the quality of the greedy route is often very good. The length of an optimal route between a pair of nodes on a DT is within a constant time of the direct distance [19].

V. Data Placement and Retrieval using GRED

In this section, we detail how the GRED is designed to support data placement and retrieval services.

A. Placing data in the edge network

In GRED, the switches are associated with their coordinates in the virtual space, which is maintained by the control plane. A switch knows its own coordinates, its physical neighbors' coordinates, and the coordinates of its DT neighbors. The Euclidean distance between any two switches can be calculated from their coordinates where the network-wide distance has been embedded in Section IV-A. The key idea of GRED forwarding at a switch, say u, is conceptually simple: For a data with ID d. The place to store the data is position $H(d)$, which will be converted to the coordinate in the virtual space, as shown in Section III. u forwards the packet to the DT-neighbor switch closest to $H(d)$. If the neighbor is a physical

Algorithm 2 GRED(u, d) forwarding at switch u.

1: For each physical neighbor v, $R_v \leftarrow ED(v,d)$, Euclidean distance between v and d;
2: For each DT neighbor $\tilde{v}$, $R_{\tilde{v}} \leftarrow ED(\tilde{v},d)$;
3: Let v^* be the neighbor where $R_{v^*} = \min\{R_v, R_{\tilde{v}}\}$;
4: **if** $R_{v^*} < ED(u,d)$ **then**
5: Send the packet to v^* directly or by the multi-hop path;
6: **else**
7: Switch u is closest to d, and determines a unique edge server to place the data;
8: **end if**

neighbor, the packet is directly forwarded; else, the packet is forwarded via a virtual link, to a DT neighbor closest to $H(d)$. If there is no neighbor of u closer to $H(d)$ than u itself, it is proved that u is the switch closest to $H(d)$ [19]. When the data arrives at the switch closest to $H(d)$, the switch determines a unique edge server to place the data. The detailed algorithm is presented in Algorithm 2.

Transfer in a virtual link. Consider a switch u that has received a data d to forward. Switch u stores it with the format: $d{=}{<}d.dest, d.sour, d.relay, d.data{>}$ in a local data structure. When $d.relay{\neq}null$, data d is traversing a virtual link. Note that $d.dest$ is the end switch of the virtual link, $d.sour$ is the source switch, $d.relay$ is the relay switch, and $d.data$ is the payload of the data. When switch u receives a packet that is being forwarded in a virtual link, the packet is processed as follows. When $u{=}d.dest$, switch u is the endpoint of the virtual link, and continues to forward the data based on Algorithm 2. When $u{=}d.relay$, switch u first finds tuple t from the forwarding table F_u with $t.dest{=}d.dest$ where F_u is defined in Section IV-C. Then, switch u revises $d.relay{=}t.succ$ based on the matched tuple t. The last step in switch u is to transmit the data to $d.relay$. Based on this setting, messages can be forwarded to a DT neighbor of a switch. However, it is worth noting that a global minimum may not be unique, for those data mapped to a Voronoi edge in Fig. 4(b). The tie can be broken by ranking the x coordinate, then y coordinate.

B. Determining the placement server

Based on the above analysis, GRED can ensure that a data item can be forwarded to a unique switch, whose position is closest to the position of the data. Furthermore, the switch determines a unique server to place the data. Assume that switch u is closest to the data position in the virtual space, and switch u is directly interconnected with s edge servers. In GRED protocol, switch u maintains a serial number for each edge server from 0 to $s{-}1$. Then, switch u transmits the data with the identifier d to the server whose serial number is $[H(d) \bmod s]$ where we still use a uniform hash function [11]. Furthermore, the method to determine the server can efficient balance the load among those edge servers since the hash function can map the expected inputs as evenly as possible over its output range.

TABLE I
THE FLOW ENTRY IN SWITCH 1 BEFORE UPDATING.

	Match	Action
1	$d.dest{=}h3.address$	Output: port $p3$

TABLE II
THE FLOW ENTRY IN SWITCH 1 AFTER UPDATING.

	Match	Action
1	$d.dest{=}h3.address$	Set: $d.dest{=}h6.address$; Output: port $p5$

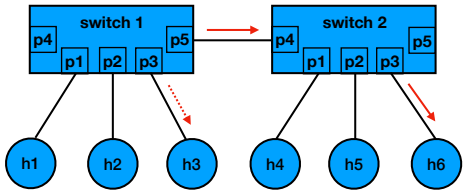


Fig. 5. Data item that should be placed in server $h3$ is placed in server $h6$ when server $h3$ would be overloaded.

The range extension. Consider that edge servers could be heterogeneous. Some edge servers with low storage capacity would be overloaded when switches connect to different numbers of edge servers with heterogeneous capacity. To solve this problem, We further extend the management range of the switches. The management range of a switch is determined by the edge servers that the switch can place data. In prior discussion, the management range is one-hop. That is, the data whose position is closest to a switch position would be placed in the edge server directly connected to the corresponding switch. Furthermore, GRED allows that a switch can manage servers with more than one hop. When the upper layer application finds that an edge server would be overloaded, the corresponding switch sends an extending request to the control plane, which can be achieved in the context of SDN [5]. Accordingly, the control plane assigns the edge server with the most remaining capacity from the physical neighbor switches to take over the corresponding storage load. To enable this, the control plane needs to update the corresponding forwarding entries into the related switches.

As shown in Fig. 5, when the server $h3$ that connected to switch 1 would be overloaded, the switch 1 sends an extending request to the control plane. Then, the control plane assigns server $h6$ to take over the load of server $h3$ where the edge server $h6$ is connected to switch 2. Before that, for switch 1, the data that should be placed in server $h3$ would be forwarded to port $p3$ based on the flow entry in Table I. However, the data is forwarded to port $p5$ after that the control plane replaces the forwarding entry in Table I with the flow entry in Table II. Table II shows that switch 1 first sets the destination address of the data as the address of server $h6$, and then forwards the data to port $p5$, when the destination address of the data is the address of server $h3$. Meanwhile, switch 2 also receives the corresponding forwarding entry, which indicates to forward the related data to its edge server $h6$. Furthermore, when

some edge servers in switch 2's range would be overloaded, switch 2 will also send an extending request to the control plane. Therefore, the range extension can efficiently avoid the overload of edge servers and share the resources of multiple edge servers.

In addition, consider that the data placement in edge servers is not everlasting. That is, the overloaded edge server could become underloaded again since some data could be invalid or migrated to the Cloud. In this case, the edge server will first retrieve the data, which should be placed in the edge server, but is placed in other edge servers. When all the corresponding data has been retrieved, the corresponding extended forwarding entries will also be deleted from the related switch.

C. Data retrieval using GRED

So far, we have introduced the procedure of data placement. The data retrieval using GRED is similar to the data placement. The retrieval is also to use the data identifier, and each switch greedily forwards the retrieval request to the switch whose position is closest to the data position in the virtual space. Furthermore, the switch uses the same method shown in Section V-B to determine the edge server for responding to the retrieval request. However, the key challenge is how to determine the edge server that has stored a data when the correponding switch has extended its management range.

As shown in Fig. 5, the data that should be placed in server $h3$ that is connected to switch 1 is forwarded to server $h6$ connected to switch 2 when switch 1 extends its management range. In this case, when we retrieve a data that is directed to the edge server $h3$ based on the value of $[H(d) \bmod s]$, we could not determine that the data has been placed in server $h3$ connected to switch 1 or server $h6$ connected to switch 2. Therefore, to efficiently retrieve a data, the retrieval request is forwarded to the two edge servers at the same time, and the edge server that has stored the data will respond to the retrieval request. Note that a tag is used in the packet header to indicate a placement/retrieval request. After that, we can ensure to efficiently locate a data that has been placed in the edge network when a data retrieval request is received.

VI. Discussion

The network dynamic. Consider that some edge nodes could be added into the edge network. Meanwhile, some failures of switches or edge nodes could result in that some edge nodes leave from the edge network. Therefore, the GRED is required to accommodate the network dynamic. Recall that we utilize an incremental method to construct the DT graph in the control plane in Section IV-C. When an edge node is added in the edge network, some edges will be added in the DT graph to connect the new edge node and its neighbors, which have existed in the DT graph. It is worth noting that the new edge node has no effect on the other edge nodes. It only affects its neighbors. First, the control plane will add the corresponding forwarding entries into the new edge nodes and its neighbors. Then, those data in the neighboring edge nodes of the new edge node will be calculated again. If those data is closest to the new edge node, they will be forwarded to the new edge node. Furthermore, when an edge node leaves from the edge network, the related edges between it and its neighbors will be deleted, and then some new edges will be added between those neighbors to form a new DT graph. After that, those related data will be stored in those neighbors based on their positions in the virtual space, which has been described in Section V-A.

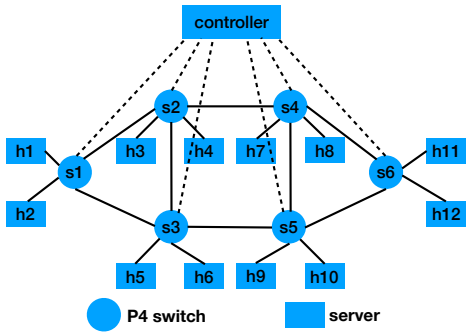


Fig. 6. Prototype with 1 controller, 6 P4 switches and 12 servers.

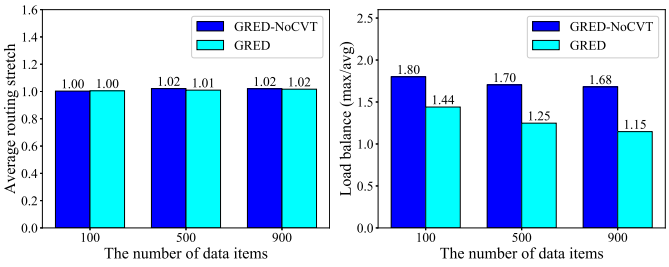
Data copies. The data copies are fundamental for the fault tolerance. Meanwhile, multiple data copies can also help to achieve better performance. Therefore, it is necessary for the GRED to support multiple data copies in the edge network. Recall that we store a data item in the edge network by hashing its ID. Furthermore, when there exists multiple data copies, it is required to add a serial number for each data copy. Then, the ID and the serial number are concatenated to form a new string. By hashing the new string, we can achieve the position of the corresponding data copy in the virtual space. After that, the data copy can be stored in the edge network based on the scheme in Section V-A. An advantage of the GRED is that it is easy to determine which copy is cloeset to the access point. Consider that we have embedded the network-wide distance between switches into the Euclidean distance between the related two points in the virtual space in Section IV-A. Therefore, we can know which copy is closest to the access point by calculating their distances to the access point in the virtual space after embedding the network distance in Section IV-A.

VII. Performance Evaluation

In this section, we first introduce the implementation and evaluation of the GRED on a small-size testbed. Then, we conduct large-scale simulations to evaluate the performance of the GRED.

A. Implementation using P4

We have implemented a prototype of GRED, including all switch data plane and control plane features described in Section III, where the switch data plane is written in P4 [7], and the function in the control plane is written in Java. The P4 compiler generates Thrift APIs for the controller to insert the forwarding entries into the switches. The P4 switch supports a programmable parser to allow new headers to be defined.



(a) Routing stretch. (b) Load balance.

Fig. 7. The performance of the GRED protocol under different settings.

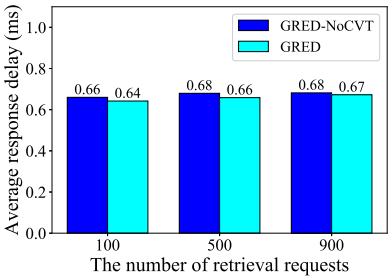


Fig. 8. The response delay under different number of retrieval requests.

Meanwhile, multiple match+action stages [7] are designed in series to achieve the neighboring switch whose position is closest to the position of the data. The P4 switch calculates the distance from a neighbor to the data in the virtual space in a match+action stage. The topology of our prototype is shown in Fig. 6. Our testbed consists of 1 controller and 6 P4 switches, where each switch connects to 2 servers. We use those servers to generate data requests including the data placement/retrieval requests. Furthermore, we evaluate the routing stretch and the load balance of the GRED protocol on our prototype. We implemented two variants of the GRED protocol including the GRED-NoCVT protocol and the GRED protocol on our testbed, where the GRED protocol sets the number of iterations is 50 for the C-regulation method shown in Section IV-B. GRED-NoCVT indicates the positions of switches are only generated by the M-position algorithm in Section IV-A, and not refined by the C-regulation method.

We first evaluate the performance of the GRED protocol based on our testbed. Fig 7(a) shows that the average routing stretches of GRED-NoCVT and GRED are close to 1, which is the optimal value of the routing stretch. However, Fig 7(b) shows that GRED achieves significantly better load balance than GRED-NoCVT due to the lower max/avg value, which is used to quantify the load balance of a networked storage system. The value of max is the number of data items received by the most loaded edge server, and the value of avg means the average load of all edge servers. The optimal value of max/avg is 1, which indicates perfect load balancing.

Furthermore, we test the average response delay of the GRED protocol where we have placed some data items in our testbed and then generated some data retrieval requests. Fig 8 shows that the average response delay of those retrieval requests. We can find that the average response delays of the two GRED variants are similar, and the average response delay has a modest change when we send the different number of retrieval requests. The routing stretch would affect the average response delay. Recall that the two GRED variants all have low routing stretches in Fig 7(a). Therefore, we can find that the response delay is low in Fig 8. That is, the GRED protocol can quickly respond to those retrieval requests in edge computing. However, it is worth noting that the network size is small since our testbed just consists of 6 P4 switches and 12 edge servers. So, we further conduct massive simulations to evaluate the performance of the GRED protocol including the routing stretch and the load balance in the next section.

B. The setting of large-scale simulations

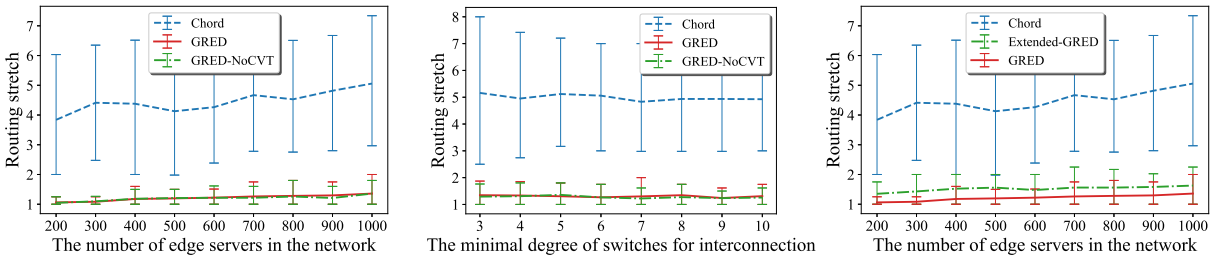
In simulations, unless otherwise specified, we use BRITE [20] with the Waxman model to generate synthetic topologies at the switch level where each switch connects to 10 edge servers. Switches could connect to different numbers of edge servers or servers with different capacity. Then, we compare the GRED protocol with the Chord [8] protocol, which can locate data in a peer-to-peer network. The GRED protocol includes two variants: GRED and GRED-NoCVT (without CVT). We use two performance metrics to evaluate the performance of GRED as follows.

- **Routing stretch.** The routing stretch value is defined to be the ratio of the hop count in the selected route to the hop count in the shortest route between a pair of source and destination nodes.
- **Load balance.** The *max/avg* metric quantifies the load balance, defined as the ratio of the number of data items received by the most loaded edge server (*max*) to the average load of all edge servers (*avg*).

We evaluate the routing stretch of GRED by varying the number of switches and the minimal degree of switches for interconnection. In each setting of the network, we randomly generate 100 data items to be placed in the network and randomly select an access point for each data. Each point in Fig. 9 is the average of 100 routing stretches where each error bar is constructed using a 90% confidence interval of the mean. Furthermore, we evaluate the load balance of GRED varying the number of switches and the amount of data. Meanwhile, we evaluate the impact of the number of iterations of the C-regulation method on the load balance of GRED.

C. Routing stretch

1) Varying network size: We first evaluate the impact of the network size on the routing stretch. Fig. 9(a) shows the routing stretches of Chord, GRED, and GRED-NoCVT. In Fig. 9(a), GRED and GRED-NoCVT achieve significantly lower routing stretches than Chord. It is because that the Chord takes $O(\log n)$ overlay hops to retrieve the data while the GRED costs only one overlay hop to get the data. The average routing stretch of Chord is higher than 3.5 under any network size in our experiments. However, the average routing stretches of



(a) The impact of the network size. (b) The impact of the minimal degree of switches. (c) The impact of the range extension.

Fig. 9. Routing stretch comparison.

GRED and GRED-NoCVT are all lower than 1.5. It means that GRED uses $<30\%$ routing path lengths compared to using Chord. It is worth noting that shorter routing path indicates less bandwidth consumption and lower latency to place/retrieve data. Meanwhile, we can see that GRED has a little higher routing stretch than GRED-NoCVT in some cases. It is because the C-regulation method has influence on the distances between switches, which can be preserved as well as possible after using the M-position algorithm in Section IV-A.

2) Varying the minimum degree of switches: We evaluate the impact of the minimal degree of switches for interconnection on the routing stretch. The network employs 100 switches and 1000 edge servers, and the minimal degree of switches for interconnection varies from 3 to 10. Fig. 9(b) shows that GRED and GRED-NoCVT achieve obviously lower routing stretches than the Chord protocol. In Fig. 9(b), we can see that the degree of switches for interconnection has a modest impact on the routing stretch for the same protocol. Meanwhile, Fig. 9(b) shows that the routing stretch slightly decreases as the increase of the minimal degree of switches. When the switches provide more ports for interconnection, greedy routing has a higher possibility to find the shortest path.

3) Range extension: When an edge server will be overloaded, the corresponding switch needs to extend its management range. That is, the switch forwards data to the edge server connected to the neighboring switch. Range extension may increase the routing stretch. We compare the routing stretch achieved by GRED and the extended-GRED protocol where the number of iterations is 50 for the C-regulation method. The extended-GRED denotes the data would be placed in the edge server connected to the neighbor switch of the destination switch. We placed 100 data items to achieve the average routing stretch under each setting of the network size. Fig. 9(c) shows that the extended-GRED protocol achieves slightly higher routing stretch than GRED. However, the routing stretch of the extended-GRED is still significantly lower than Chord.

D. The number of forwarding table entries

In this section, we show the number of forwarding table entries per switch for the GRED protocol under different network sizes. In Figure 10, each point indicates the average

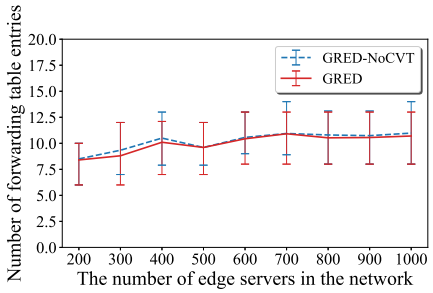


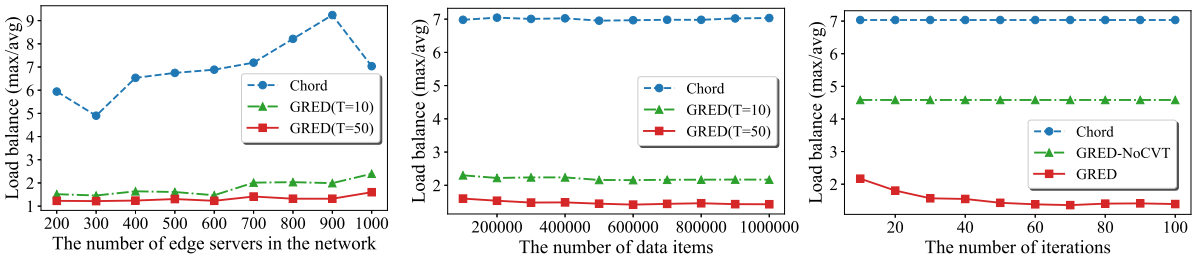
Fig. 10. The number of forwarding table entries.

number of forwarding table entries over all switches, where the error bar is constructed using a 90% confidence interval of the mean. We can see that the increase of the average number of forwarding entries is modest as the increase of the network size from Figure 10. That is, the GRED protocol only needs a few forwarding entries to achieve the data placement and retrieval services. GRED has the obvious advantage in scalability since the number of forwarding table entries is independent of the network size and the number of flows in the edge network.

E. Load balance

1) Varying the network Size: We first evaluate the impact of the network size on the load balance under different protocols where the number of edge servers varies from 200 to 1000. Fig. 11(a) shows that GRED (T=10) and GRED (T=50), T is the number of iterations, achieve significantly better load balance than Chord due to the lower value of *max/avg*. In Fig. 11(a), the value of *max/avg* goes up as the increase of the network size in Chord. However, we observe very little increase for GRED (T=10) and GRED (T=50) in Fig. 11(a). Fig. 11(a) shows that GRED (T=50) achieves better load balance than GRED (T=10), which means that the GRED protocol can achieve better load balance by increasing the number of iterations.

2) Varying the amount of data: We vary the amount of the placed data from 100,000 to 1,000,000 where 1000 edge servers are deployed in the network. Fig. 11(b) shows that GRED (T=50) achieves the best load balance among the three protocols. We can see that the Chord protocol has the worst load balance because the value of *max/avg* is higher that 6. Meanwhile, we can also see that the value of *max/avg* for



(a) The impact of the network size. (b) The impact of the number of data items. (c) The impact of the number of iterations.

Fig. 11. Comparison of load balance.

GRED (T=10) is lower than 2.5, and further the value of GRED (T=50) is lower than 2. Note that the value of *max/avg* is lower and better, and the optimal value for load balance is 1. Therefore, the GRED protocol can achieve the proper load balance among edge servers.

3) Varying the number of iterations: In this section, we test the impact of the number of iterations T on the load balance. Note that the number of iterations T for the C-regulation method will affect the positions of switches in the virtual space, and further affect the load balance of the GRED protocol. The setting of the network is the same as the setting in Section VII-E2, and we placed 100,000 data items in the network. Note that the Chord and the GRED-NoCVT are independent of T. Therefore, Fig. 11(c) shows that T has no influence on Chord and GRED-NoCVT. Furthermore, we can see that the value of *max/avg* decreases as the increase of T for the GRED protocol in Fig.11(c). That means that the GRED protocol can achieve better load balance when T increases. Meanwhile, Fig. 11(c) shows that GRED-NoCVT can also achieve better load balance than Chord even if GRED-NoCVT did not use the C-regulation method to refine the positions of switches. Furthermore, we can see that the value of *max/avg* is lower than 2 when T is more than 20 in Fig. 11(c). We also find that the value of *max/avg* stops to decrease when T is more than 70 in Fig. 11(c). It means that the C-regulation method has found the optimal positions of switches in the virtual space to achieve the proper load balance when T=70. After that, the increase of T has little improvement on the load balance of GRED.

VIII. Related Work

A. Edge computing

In edge computing, edge servers perform computing offloading, data storage, caching and processing, as well as distribute request and delivery service [1]. Mobility is an intrinsic trait of many mobile applications. In those applications, the edge servers could exploit the movement and trajectory of edge users to improve the efficiency of handling users computation requests. Some mobility models were proposed [21][22], which characterize the mobility by a sequence of networks that users can connect to and a two-dimensional location-time workflow, respectively.

The burdens on an edge server can be lightened via peer-to-peer cooperative edge servers [23]. Resource sharing via the cooperation of edge servers can not only improve the resource utilization, but also provide more resources for edge users to enhance their user experience. The resource sharing framework was originally proposed in reference [24], which includes components such as resource allocation, revenue management and service provider cooperation. The framework was extended in [25], which considered both the local and remote resource sharing. Server cooperation can significantly improve the computation efficiency and resource utilization at edge servers.

B. Greedy routing with guaranteed delivery

Greedy geographic routing protocols have been designed for wireless sensor and ad hoc networks. GFG [26] and GPSR [27] use face routing to move packets out of local minimum. Furthermore, GHT [28] is designed for data-centric storage in a distributed sensing network. However, the GHT uses the GPSR [27] as the underlying routing system, which requires the network topology to be a planar graph in 2D to avoid routing failures. Lam and Qian proposed the MDT protocol [10], for any connectivity graph of nodes with arbitrary coordinates in a d-dimensional Euclidean space ($d \geq 2$). Virtual coordinate schemes have been studied for greedy routing when node location information is unavailable [6]. In this paper, we utilize the properties of MDT, which provides the guaranteed delivery.

IX. Conclusion

Edge computing needs to provide the data placement and retrieval services for many emerging applications such as IoT. However, it remains an open problem. A key challenge to enable this is to efficiently locate the data in the edge network. GRED solves this challenging problem by offering a powerful primitive: given a data identifier, it determines the edge server responsible for the data placement and retrieval, and does so efficiently. Attractive features of GRED include its routing simplicity, provable correctness, low routing stretch, and proper load balance. Our theoretical analysis, simulations, and experimental results confirm that the effectiveness and efficiency of GRED. We believe that GRED will be a valuable component for edge computing considering the user mobility and the cooperation among edge servers.

REFERENCES

[1] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637–646, 2016.

[2] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in *Proc. of the 1st MCC Workshop on Mobile Cloud Computing*, August 2012.

[3] M. Chiang and T. Zhang, "Fog and IoT: An Overview of Research Opportunities," *IEEE Internet of Things Journal*, 2016.

[4] Y. Elkhatib, B. Porter, H. B. Ribeiro, M. F. Zhani, J. Qadir, and E. Rivire, "On using micro-clouds to deliver the fog," *IEEE Internet Computing*, vol. 21, no. 2, pp. 8–15, 2017.

[5] D. Kreutz, F. M. V. Ramos, P. E. Verssimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, 2015.

[6] C. Qian and S. S. Lam, "Greedy routing by network distance embedding," *IEEE/ACM Transactions on Networking*, vol. 24, no. 4, pp. 2100–2113, 2016.

[7] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, and D. Walker, "P4: Programming protocol-independent packet processors," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 3, pp. 87–95, Jul. 2014.

[8] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," in *Proceedings of ACM SIGCOMM*, 2001, pp. 149–160.

[9] C. Qian and S. S. Lam, "ROME: Routing On Metropolitan-scale Ethernet," in *Proceedings of ICNP*, 2012.

[10] S. S. Lam and C. Qian, "Geographic routing in d-dimensional spaces with guaranteed delivery and low stretch," *IEEE/ACM Trans. Netw.*, vol. 21, no. 2, pp. 663–677, 2013.

[11] A. Biryukov, M. Lamberger, F. Mendel, and I. Nikolić, "Second-order differential collisions for reduced sha-256," in *Advances in Cryptology – ASIACRYPT*, 2011, pp. 270–287.

[12] P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz, B. O'Connor, P. Radoslavov, W. Snow, and G. Parulkar, "Onos: Towards an open, distributed sdn os," in *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*, ser. HotSDN, 2014, pp. 1–6.

[13] I. Borg and P. J. Groenen, *Modern multidimensional scaling: Theory and applications*. Springer Science & Business Media, 2005.

[14] F. Wickelmaier, "An introduction to mds," Sound Quality Research Unit, Aalborg University, Denmark, Tech. Rep., 2003.

[15] S. Fortune, "Voronoi diagrams and Delaunay triangulations," in *Handbook of Discrete and Computational Geometry*, 2nd ed., J. E. Goodman and J. O'Rourke, Eds. CRC Press, 2004.

[16] Q. Du, V. Faber, and M. Gunzburger, "Centroidal voronoi tessellations: Applications and algorithms," *SIAM Review*, vol. 41, no. 4, pp. 637–676, 1999.

[17] L. J. Guibas, D. E. Knuth, and M. Sharir, "Randomized incremental construction of delaunay and voronoi diagrams," *Algorithmica*, vol. 7, no. 1, pp. 381–413, 1992.

[18] J. A. De Loera, J. Rambau, and F. Santos, *Triangulations Structures for algorithms and applications*. Springer, 2010.

[19] D. Y. Lee and S. S. Lam, "Protocol design for dynamic delaunay triangulation," in *Proc. of 27th IEEE ICDCS*, June 2007.

[20] A. Medina, A. Lakhina, I. Matta, and J. Byers, "Brite: An approach to universal topology generation," in *Proc. 9th International Symposium on MASCOTS*, Cincinnati, OH, USA, August 2001.

[21] K. Lee and I. Shin, "User mobility model based computation offloading decision for mobile cloud," *Journal of Computing Science and Engineering*, vol. 9, no. 3, pp. 155–162, 2015.

[22] M. R. Rahimi, N. Venkatasubramanian, and A. V. Vasilakos, "Music: Mobility-aware optimal service allocation in mobile cloud computing," in *Proc. 6th International Conference on Cloud Computing*, June 2013.

[23] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A survey on mobile edge computing: The communication perspective," *IEEE Communications Surveys Tutorials*, vol. 19, no. 4, pp. 2322–2358, 2017.

[24] R. Kaewpuang, D. Niyato, P. Wang, and E. Hossain, "A framework for cooperative resource management in mobile cloud computing," *IEEE JSAC*, vol. 31, no. 12, pp. 2685–2700, 2013.

[25] R. Yu, J. Ding, S. Maharjan, S. Gjessing, Y. Zhang, and D. Tsang, "Decentralized and optimal resource cooperation in geo-distributed mobile cloud computing," *IEEE Transactions on Emerging Topics in Computing*, 2015.

[26] P. Bose, P. Morin, I. Stojmenović, and J. Urrutia, "Routing with guaranteed delivery in ad hoc wireless networks," *Wirel. Netw.*, vol. 7, no. 6, pp. 609–616, 2001.

[27] B. Karp and H. T. Kung, "Gpsr: Greedy perimeter stateless routing for wireless networks," in *Proc. 6th MobiCom*, August 2000.

[28] S. Ratnasamy, B. Karp, S. Shenker, D. Estrin, R. Govindan, L. Yin, and F. Yu, "Data-centric storage in sensornets with ght, a geographic hash table," *Mobile Networks and Applications*, vol. 8, no. 4, pp. 427–442, 2003.