High-Throughput Anycast Routing and Congestion-Free Reconfiguration for SDNs

IEEE INFOCOM 2015 Subsmission, Paper # 1570007389

***Abstract*—Software Defined Network (SDN) separates the network control plane from the data plane, thus providing more flexible functionalities. A number of research challenges have been recently addressed for this new networking paradigm. In this work, we focus on designing high-throughput anycast mechanisms in SDN, as anycast routing serves as a fundamental building block in many applications, e.g., campus network. Specifically, we investigate two main challenges for efficient implementation of SDN-based anycast system: high throughput anycast routing (HTAR) and congestion-free anycast reconfiguration (CFAR). For the first challenge, we design an approximate algorithm, SAR, based on the fully polynomial time approximation scheme and the single-source unsplittable flow routing method. We prove that the proposed algorithm produces a routing that achieves a throughput that is at least $\frac{1}{2(1+\xi)}$ of the optimum for the HTAR problem, where $\xi > 0$ is an arbitrarily small constant. We then propose an efficient algorithm, called ARC, for congestion-free anycast reconfiguration. This algorithm tries to find a routing reconfiguration procedure consisting of at most $t(> 1)$ transitive configurations while meeting all the flow demands. When the traffic demands can not be satisfied, ARC can maximize the minimum throughput using one transitive configuration. We show, by mininet simulations, the performance gains that are achievable using these algorithms are much better than the SSPF method.**

Index Terms*—*Software Defined Networks, Anycast, High Throughput, Anycast Routing Reconfiguration, Congestion-free.

I. Introduction

The software Defined Network [**?**] [**?**] [**?**] is a new networking paradigm that separates the control and data (or forwarding) planes on the independent devices. In general, an SDN usually comprises of two main components: SDN-Controller (SDN-C) and SDN Forwarding Element (SDN-FE). The SDN-C is a logically centralized function unit [**?**] [**?**], and constitutes the control plane of a SDN. One or several controllers will determine the forwarding path of each traffic flow in the network with a centralized control manner. A set of SDN-FEs constitutes the data plane of a SDN, and responses for data forwarding of each flow. The logic for forwarding the packets is determined by the SDN controller, and is implemented in the forwarding table at the SDN-FEs.

The implementation of control and forwarding plane functions on separate hardware platforms has various operational benefits. First, since the SDN-C can dynamically adjust the forwarding paths for all flows with a global view, SDNs are more flexible compared with traditional (or non-SDN) networks [**?**], [**?**]. Second, since the SDN-C can provide smarter scheduling for all flows, SDN helps to alleviate network congestion, and explore a higher utilization of transmission links [**?**]. With these significant advantages, SDN can be used in a wide range of fields, such as campus networks [**?**], datacenter [**?**], and wide-area networks [**?**].

Previous studies have pointed out the importance of anycast routing in different applications, such as resource acquirement [**?**] and DNS [**?**], etc. We consider a typical scenario of resource acquirement application in a campus network. There usually deploy several mirror servers, in which each server contains different kinds of resources, such as academia resource, office resource and multimedia resource, etc. When a user (or host) requests for resource access, it enters the network via a SDN-FE and obtains the resources or services from any server. Under this application scenario, each SDN-FE should connect with one of the servers to cope with the requests, which is called as *anycast routing*. Compared with the unicast routing [**?**], the main difference is that the destination in the anycast scheme is one of the given address set. Since anycast routing permits all the SDN-FEs to obtain the required resource from proper servers, it improves the network throughput and reduces the congestion. However, to fully explore the advantage of anycast, an efficient route control is necessary. Thus, it is of great significance to implement a high-quality anycast routing with the help of SDN.

In an anycast system, when a lot of users request for resource access, an improper anycast routing method may result in a serious congestion on some links, thus decreasing the network throughput. Due to the special features of both anycast and SDN, we should deal with two main tasks for anycast routing in software defined networks.

Anycast Routing With Both Server and Path Selection: With development of information technology, resources or services cost much more bandwidth during forwarding in the network. When one user u wants to acquire some resource, it should connect with one of these servers. The traditional way may let each device (e.g., the router) connect with the "closest" server in a network using the static routing protocol, such as OSPF [**?**]. Without efficient route control, congestion may occur on some links or devices under this specific routing strategy, which will be illustrated in the next section. If some links are over-loaded, the throughput and reliability will be reduced significantly. Moreover, it may result in performance oscillation with the distributed route control mechanism. In this work, by taking advantage of the fact that SDN can provide the centralized route control, given traffic demands on all the SDN-FEs, we will design a high throughput anycast routing (HTAR) mechanism with capacity constraints on both links and servers so as to provide high QoS for users.

Dynamical Anycast Routing Reconfiguration: Different

anycast routing configurations are necessary response to various user requests and changed network topologies. Thus, the route table on each SDN-FE should be reconfigured from an "old" routing configuration to a "new" one. However, due to asynchronous routing reconfiguration on all the SDN-FEs, immediate reconfiguration may result in transient congestion due to the capacity constraints on links and servers. For unicast routing, by assuming arbitrarily *splittable* flows, Hong et. al. [**?**] proposed a routing reconfiguration (or updating) method using linear program. Unfortunately, this method cannot be applied for anycast route-table reconfiguration here as in anycast scheme, we often do not allow a flow to be splitted into many fractional flows with different destinations. Otherwise, if one SDN-FE is permitted to connect with multiple servers simultaneously for resource acquirement, it needs additional coordination among these servers, and thus makes the system management more complex. Observe that due to the linear programming nature, method in [**?**] cannot ensure such a property. Notice that, we do allow a flow to to split into fractional flows with the same destination. Therefore, we need a congestion-free anycast routing reconfiguration (CFAR) mechanism to avoid the transient congestion while ensuring that any flow is only splitted into sub-flows with the same destination.

In this paper, we mainly study the design of a SDN-based anycast system, which aims to provide the efficient anycast routing and reconfiguration. We first analyze two main challenges of an anycast system in a SDN, high throughput anycast routing and congestion-free anycast reconfiguration, respectively, and formulate two challenges into mixed integer programs. For the first challenge, we propose a SAR algorithm based on the fully polynomial time approximation scheme (FPTAS) [**?**] and the single-source unsplittable flow routing (UFP) method [**?**]. Our theoretical analysis shows that the SAR algorithm has an approximation ratio of $2(1+\xi)$ for maximizing throughput, where $\xi > 0$ is an arbitrarily small constant. Then, we design an efficient algorithm (ARC) for congestion-free anycast reconfiguration. This algorithm, given a parameter $t > 1$, tries to find a routing table reconfiguration procedure consisting of at most t phases and in each phase the throughput demand of each flow is satisfied. When the throughput cannot be satisfied, it will find one-phase anycast reconfiguration that will maximize the minimum throughput, *i.e.*, achieving max-min fairness, using a linear program method. We show high-efficiency of the proposed algorithms by both analysis and mininet simulations. For example, the SAR algorithm can obtain 35% improvements on network load ratio compared with the SSPF method.

The rest of this paper is organized as follows. Section II analyzes two main challenges for a SDN-based anycast system, and defines these challenges. In Section III, we propose two algorithms to deal with two challenges. The simulation results are presented in Section **??**. Section **??** discusses the related works on anycast routing under different networking paradigms. We conclude the paper in Section **??**.

II. Anycast System Overview and Challenges

This section first describes the overview of an anycast system in software defined networks. Then, we introduce the network model and flow model in anycast applications. Moreover, we emphasize two main challenges for a SDN-based anycast system, and give the formal definitions.

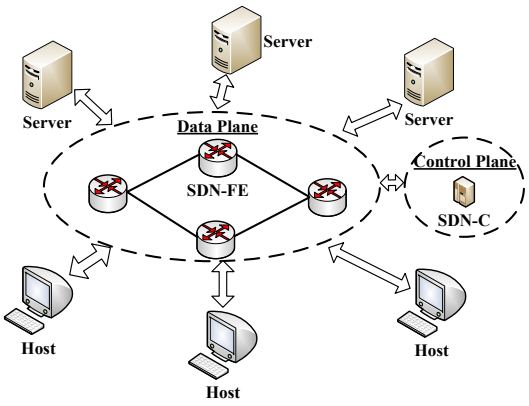


Fig. 1: Architecture of a SDN-based Anycast System

A. SDN-based Anycast System Overview

A typical architecture of a SDN-based anycast system is shown in Fig. 1. There usually deploys a set of mirror servers, in which each one can provide the uniform services for all the users. A set of SDN-FEs, shown in the left dotted circle, constitutes the forwarding backbone in the anycast system. These SDN-FEs connect the hosts (or users) and servers, and response for data forwarding by matching the packets to items in the routing tables. In addition, SDN-FEs should take some traffic measurement which they report to the SDN-C, shown in the right dotted circle. To provide the flexible control, one or several SDN controllers, peer with the network and gather link-state information so as to compute the routes for all the flows. To adapt to different traffic demands, the SDN-C should dynamically reconfigure the routing tables at SDN-FEs.

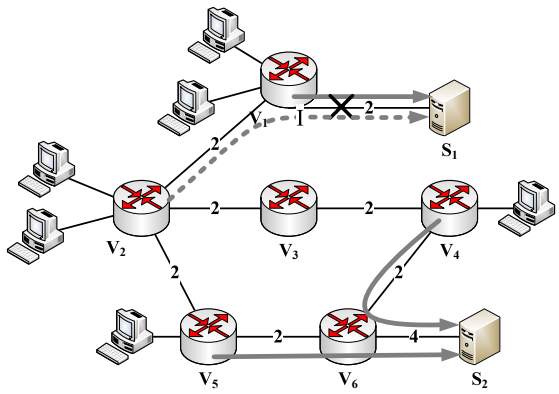
B. Network and Flow Models

Before describing two anycast challenges in a SDN, we first introduce the network model and flow model, respectively.

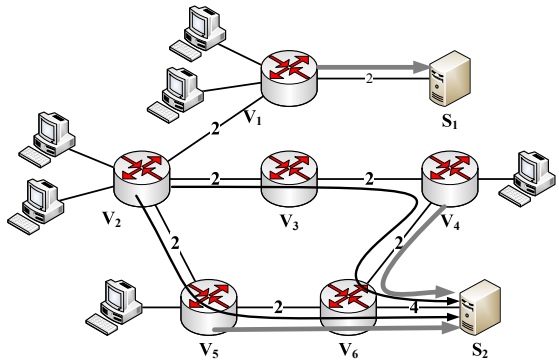
Network Model: An anycast system mainly consists of three device sets: a server set, $S = \{s_1, ..., s_m\}$, with $m = |S|$; a SDN-FE set, $V = \{v_1, ..., v_n\}$, with $n = |V|$; and a SDN-C device, C. In the SDN, since the SDN-C mainly responses for route selection of all the flows, it will not participate in the packet forwarding. Thus, the network topology can be modeled by $G = (S \cup V, E)$, where E is the link set in the graph G. Notice that, two servers are not directly connected in the network. For ease expression, let $c(e)$ and $c(s)$ denote the capacities of link $e \in E$ and server $s \in S$, respectively. Moreover, the statistical information of traffic loads on all links are available to the controller by the Openflow interface.

Flow Model: Each SDN-FE v will aggregate the traffic demands from all the connected hosts, denoted by $R(v)$. Under

the anycast scheme, each SDN-FE v should be connected with one of the servers, denoted by s. The traffic can be splitted and transmitted from SDN-FE v to the selected server s through a set of paths connecting v and s. Given a routing configuration, the traffic loads on each link $e \in E$ and each server $s \in S$ are the sum of all the traffics through this link or server, respectively. To avoid the congestion, the traffic loads on each link and server should not exceed their capacities.



(a) Illustration of the SSPF Protocol. SDN-FE v_1 connects with server s_1, and its anycast path is $v_1 - s_1$. SDN-FE v_5 connects with server s_2, and its anycast path is $v_5 - v_6 - s_2$. When SDN-FE v_2 tries to build an anycast route, its anycast path is $v_2 - v_1 - s_1$, which results in congestion.



(b) Illustration of High-Throughput Anycast Routing. SDN-FE v_2 will connect with server s_2. Two paths, $v_2 - v_3 - v_4 - v_6 - s_2$ and $v_2 - v_5 - v_6 - s_2$, are used for anycast routing, and each path is allocated one unit traffic. Then, network congestion is avoided.

Fig. 2: Illustration of Different Anycast Routing

C. High Throughput Anycast Routing (HTAR): Challenge and Formulation

In this section, we will define the high throughput anycast routing problem for a software defined network. As described above, each user or host will obtain the resource from a server via a certain SDN-FE. Since all the servers are assumed to have the uniform resource set, each SDN-FE should build a connection with one of the servers, which is called as anycast routing [**?**]. Notice that here we require that at any moment, the request from a SDN-FE v is only served by exactly one server, say s, while the flow between v and s may be carried by several paths connecting v and s. Without efficient anycast mechanism, congestion may occur on some links or servers, which reduces the network throughput accordingly.

Next, we give an example of anycast routing, as shown in Fig. 2. The value attached with each link denotes its capacity. Each host will generate one unit traffic demand for resource acquirement, and each SDN-FE aggregates the traffic loads from all the connected hosts. For example, the traffic demands of SDN-FEs v_1 and v_5 are 2 and 1, respectively. The previous methods, such as SSPF [**?**], usually select the "closest" server as the destination. Followed by this rule, SDN-FE v_1 will connect with server s_1, and its anycast path is $v_1 - s_1$. SDN-FE v_5 connects with server s_2, and its anycast path is $v_5 - v_6 - s_2$. When another SDN-FE v_2 requests for resource access, it should associate with server s_1 by the SSPF rule, and its anycast path is $v_2 - v_1 - s_1$. Then, the traffic load on link $e_{v_1 s_1}$ is 4, which exceeds its capacity. Congestion occurs on this link, as shown in Fig. 2(a). We consider a high throughput anycast routing, as shown in Fig. 2(b). SDN-FE v_2 is connected with server s_2. Moreover, the total traffic will be transmitted through two paths, $v_2 - v_3 - v_4 - v_6 - s_2$ and $v_2 - v_5 - v_6 - s_2$, respectively, and each path is allocated one-unit traffic. As a result, though we do not select the shortest path as anycast route, it helps to avoid the congestion, thus improve the network throughput. Next, we give the definition of the HTAR problem as follows.

Let P_{vs} denote the path set from each SDN-FE $v \in V$ to each server $s \in S$. The HTAR problem should select a server, denoted by s, for each SDN-FE v as the destination of anycast routing. Then, each path $p \in P_{vs}$ will be allocated a bandwidth, denoted by $x(p)$. We assume that the achievable throughput of each SDN-FE v is denoted by $\lambda \cdot R(v)$, where λ is throughput factor. That is, $\sum_{p \in P_{vs}} x(p) \geq \lambda \cdot R(v)$. Due to capacity limit, there are two constraints for anycast routing. One is that the total traffic load on each link $e \in E$ should not exceed its capacity $c(e)$. The other is that the total traffic load on each server $s \in S$ should not exceed its processing capacity $c(s)$. Otherwise, congestion will occur on some links or servers. Our objective is to maximize the network throughput fairness, that is, $\max\ \lambda$.

We give the formulation for the HTAR problem below. In the formulation, the traffic demand of SDN-FE v is denoted by $R(v)$. y_{vs} is a 0-1 variable, and denotes whether SDN-FE v is associated with server s or not, and $x(p)$ denotes the allocated traffic load through path p. The HTAR problem solves the following optimization problem, expressed in equation (1).

$$\text{Max-Throughput Anycast:} \quad \max\ \lambda$$
$$S.t. \quad \begin{cases} \sum_{s \in S} y_{vs} = 1, & \forall v \in V \\ \sum_{s \in S} \sum_{p \in P_{vs}} x(p) y_{vs} \geq \lambda \cdot R(v), & \forall v \in V \\ \sum_{e \in p} x(p) \leq c(e), & \forall e \in E \\ \sum_{v \in V} \sum_{p \in P_{vs}} x(p) \leq c(s), & \forall s \in S \\ x(p) \geq 0, & \forall p \\ y_{vs} \in \{0,1\}, & \forall v \in V, \forall s \in S \end{cases} \tag{1}$$

The first set of equations means that each SDN-FE should

be connected with only one server. The second set of equations denotes that the achievable throughput of each SDN-FE v should be no less than $\lambda \cdot R(v)$, where λ is throughput factor. The third and fourth sets of inequalities ensure that the total traffic loads on each link and each server are no more than their capacities. The above two sets of equations guarantee congestion-free for anycast routing. The fifth set of inequalities denotes that the flow on any path is non-negative. The objective is to maximize the network throughput, λ.

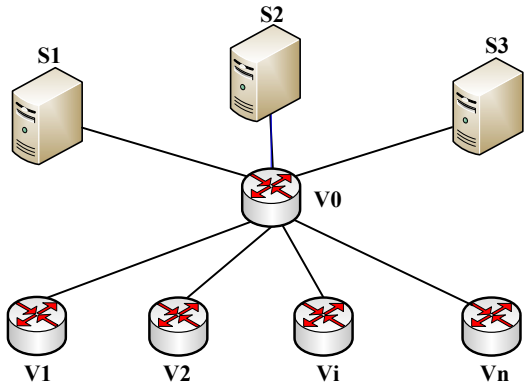


Fig. 3: A Special Example of the HTAR Problem

Theorem 1 (Intractable): The maximum throughput anycast routing problem (defined in Eq.(1)) is an NP-hard problem.

Proof: We consider a special example of the HTAR problem, shown in Fig. 3. Assume that the server capacity is infinite, so that we can ignore the constraint on server capacity. Moreover, the capacities of links are as follows: $c(e_{v_1v_0})$=... = $c(e_{v_nv_0})$=500Mbps, and $c(e_{v_0s_1})$=$c(e_{v_0s_2})$ =$c(e_{v_0s_3})$=100Mbps. When a lot of SDN-FEs request for resource access, each traffic flow will be scheduled to a server through one of three links $\{e_{v_1s_1}, e_{v_1s_2}, e_{v_1s_3}\}$. To maximize the network throughput by definition in Eq.(1), we should reduce the maximum traffic loads of links $e_{v_0s_1}$, $e_{v_0s_2}$ and $e_{v_0s_3}$. As a result, this is just an unrelated processor scheduling problem [**?**], which is NP-Hard. As the unrelated processor scheduling problem is a special case of HTAR, HTAR is an NP-Hard problem too. This finishes the proof. ■

D. Congestion-free Anycast Reconfiguration (CFAR): Challenge and Formulation

To achieve high-efficiency of an anycast system, the anycast routing configuration should be dynamically computed in response to changes of traffic demands and network topologies. We give the detailed explanation by an example. As shown in Fig. 4, the value attached with each link denotes its capacity, the value in a box denotes the traffic demand of a SDN-FE, and the value in a bracket denotes the traffic splitted on a path. Initially, two SDN-FEs a and d request traffic demands of 6 and 4, respectively, and the current routing configuration is illustrated in Fig. 4(a). When the traffic demand of SDN-FE b increases to 5, a new anycast configuration is illustrated in Fig. 4(b). However, improper route reconfiguration will result

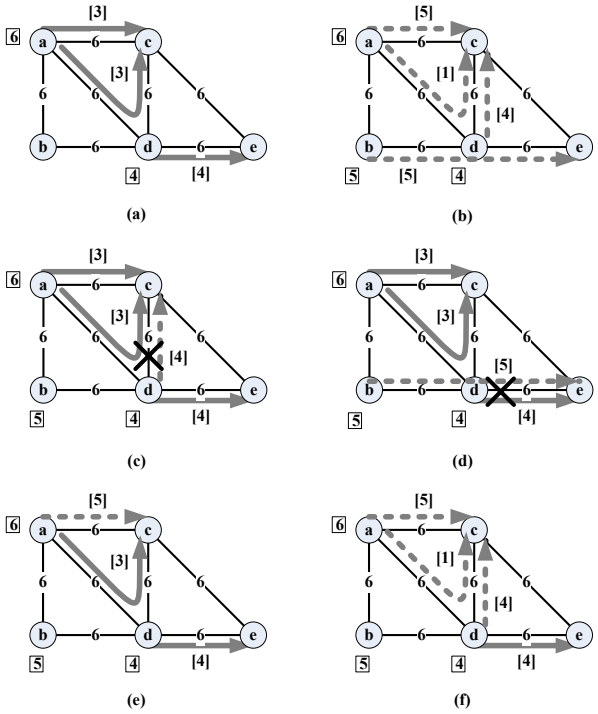


Fig. 4: Illustration of Anycast Routing Reconfiguration. Improper anycast reconfiguration will result in congestion in (c) and (d). The congestion-free reconfiguration should be (a)-(e)-(f)-(b).

in congestion. For example, when SDN-FE d first reconfigures its anycast routing, the total traffic load on link e_{cd} may reach to 7, which exceeds its capacity, in Fig. 4(c). When SDN-FE b first reconfigures its anycast routing, the total traffic load on link e_{de} may reach to 9, which also exceeds its capacity, in Fig. 4(c). To find a congestion-free route reconfiguration, we should reconfigure the anycast route of SDN-FEs a, d, b one by one, shown in Fig. 4(e,f,b). From this example, it is important to explore a congestion-free anycast reconfiguration. To reduce the coordination overhead between the servers, and also make the system management simpler, in this work, we require that each SDN-FE should be connected with only one server in any time instant. For routing-table updating, we try to find an updating procedure that will minimize the number of updating phases. In fact, the CFAR problem with minimum number of phases, where in each phase the anycast throughput of any flow is preserved, can be formalized into the integer program below.

$$\text{Congestion-Free Reconfiguration:} \quad \min \quad k$$

$$S.t. \quad \begin{cases} x^0(p) = x^s(p), & \forall p \\ x^{k+1}(p) = x^f(p), & \forall p \\ \sum_s b^i_{vs} = 1, & \forall i, v \in V \\ \sum_{p \in P_{vs}} x^i(p) \geq R(v) b^i_{vs}, & \forall i, v \in V \\ \sum_{p:p \ni e} \max[x^{i-1}(p), x^i(p)] \leq c(e), & \forall i, e \in E \\ \sum_{p:p \ni s} \max[x^{i-1}(p), x^i(p)] \leq c(e), & \forall i, s \in E \\ x^i(p) \geq 0, & \forall i, p \\ b^i_{vs} \in \{0, 1\}, & \forall i, v \in V, s \in S \end{cases} \tag{2}$$

In equation (2), the boolean variable b_{vs}^i indicates whether SDN-FE v is connected with server s in the i^{th} phase or not. $x^i(p)$ denotes the traffic load of path p in the i^{th} anycast configuration. More specifically, $x^s(p)$ and $x^f(p)$ denotes the traffic loads of each path p in the start and final anycast routing configurations, respectively. The former two sets of equations express that there are k anycast configuration at most from the start configuration to the final one. The third set of equations denotes that each SDN-FE v should be connected with only one server s in each configuration. The fourth set of equations means that the throughput of each SDN-FE v will be preserved no less than $R(v)$ in each configuration, so that all traffic demands from the connected hosts can be satisfied. The fifth and sixth inequalities ensure congestion-free on both links and servers during anycast reconfiguration. That is, the traffic loads on links and servers should not exceed their capacities. The seventh set of inequalities ensures that the flow on any path is non-negative.

III. Our Routing and Reconfiguration Algorithms

This section mainly presents two algorithms to deal with the challenges of high throughput anycast routing and congestion-free anycast reconfiguration, respectively. Each SDN-FE in the network aggregates traffic demands from all the connected hosts at the granularity of OpenFlow rules, and reports to the SDN controller periodically, e.g., every 5 minutes [**?**]. The controller computes a high throughput anycast routing under given traffic demands *in Section III-A*. When the controller detects that the total traffic loads on some links or servers exceed their capacities, it will allocate the feasible bandwidth with fairness for SDN-FEs, which can be implemented through distributed rate limiting mechanism [**?**]. In the running process, new anycast routing should be computed in response to changes of network states. To avoid the transient congestion, we should determine a certain number of transitive configurations for congestion-free anycast reconfiguration *in Section* **??**. Moreover, each transitive configuration will last for a short period, such as 10 seconds. In the following, we will describe the algorithms for the above two operations in detail.

A. Algorithm for High Throughput Anycast Routing

This sub-section mainly describes an approximate algorithm, called SAR, for high throughput anycast routing in a SDN. The proposed algorithm consists of three main steps. As HTAR is an NP-hard problem, we will relax the mixed integer program formulation to a linear program problem. In the first step, we implement a fractional anycast flow scheduling using the fully polynomial time approximation scheme (FPTAS) [**?**], and obtain a routing solution with approximatable throughput. Notice that the solution of the linear program may associate a SDN-FE with multiple servers. Thus, the second step of our method will determine a unique server for each SDN-FE, such that the achievable throughput by all flows is still within a constant factor of the optimum. After associating each SDN-FE with a server, in the final step, we compute a multi-commodity flow with given traffic demands (between each SDN-FE and its associated server), for sake of load balance in the network. Next, we describe three steps of the algorithm in detail.

1) Factional Anycast Scheduling (FAS): The HTAR problem cares for capacity constraints of both links and servers, which brings much difficulties in solving this problem. To transform a server capacity constraint into a link capacity constraint, this step first builds a virtual graph based on network deployment. We add a virtual node, denoted by s', to the original network. For each server $s \in S$, we add a virtual link between this server and the virtual node, and its capacity is set as capacity of server s. That is, $c(e_{ss'}) = c(s)$. We have built a new graph, denoted by $G' = (V \cup S \cup \{s'\}, E')$, where $E' = E \cup \bigcup_{s \in S} \{e_{ss'}\}$. Then, the algorithm constructs a linear program as a relaxation of the HTAR problem. More specifically, HTAR assumes that each SDN-FE should connect with only one server for resource access. By relaxing this assumption, that is, each SDN-FE is permitted to connect with multiple servers, we formulate the following linear program LP_1 inequation (3), which is solvable in polynomial time.

$$\begin{aligned} &\max \quad \lambda \\ S.t. \quad &\begin{cases} \sum_{p:p \in P_{vs'}} x(p) \geq \lambda \cdot R(v), & \forall v \in V \\ \sum_{p:p \ni e} x(p) \leq c(e), & \forall e \in E' \\ x(p) \geq 0, & \forall p \end{cases} \end{aligned} \tag{3}$$

Note that, $P_{vs'}$ denotes the path set from each SDN-FE $v \in V$ to virtual node s'. $x(p)$ denotes the allocated load for each path p from each SDN-FE to the virtual node.

However, since equation (3) may contain exponential number of variables in the worst case, solving such a linear program is rather costly in practice. Therefore, we devise an approximate solution instead using a fully polynomial time approximation scheme (FPTAS) [**?**]. Notice that, an FPTAS provides the following performance guarantees: for any $\varepsilon > 0$, the solution can reach $(1+\varepsilon)$-factor of the optimum, and the running time is at most a polynomial function of the network size and $\frac{1}{\varepsilon}$.

$$\begin{aligned} &\max \quad \sum_{e \in E'} l(e)c(e) \\ S.t. \quad &\begin{cases} \sum_{v \in V} R(v)z(v) \geq 1 & \\ \sum_{e \in p} l(e) \geq z(v) & \forall p \in P_{vs'}, v \in V \\ z(v) \geq 0, & \forall v \in V \\ l(e) \geq 0, & \forall e \in E' \end{cases} \end{aligned} \tag{4}$$

The FPTAS in our case is a primal dual method. The FAS problem is set to be the primal prolem, and In the dual problem, the weight of each link $e \in E'$ is denoted by $l(e)$, which is initialized to $l(e) = \frac{\delta}{c(e)}$. Then, the path weight could be the total weight of all the links on this path. The weight of the lightest path from SDN-FE v to s' is denoted by $z(v)$. Equation 4 gives the formulation of the dual problem.

The FPTAS in our case is a primal dual method, which works as follows: this method first computes a variable δ that is a function of the desired accuracy level ε, and the number

of links ϕ. In the dual problem, the weight of each link $e \in E'$ is denoted by $l(e)$, which is initialized to $l(e) = \frac{\delta}{c(e)}$. Then, the path weight could be the total weight of all the links on this path. The primal dual method operates in phases from line **??** *to line* **??**, *where in each primal phase, flow is routed from all SDN-FEs to the virtual node s' along the lightest weight path. We augment the traffic flow for each SDN-FE along the lightest weight path from line* **??** *to line* **??**. *Once each SDN-FE v has shipped a flow of $R(v)$ to s', the weight of all the links on which the flow has been sent will be increased. The process will be repeated until the problem is dual feasible. The algorithm is formally described in Alg.* **??**.

Algorithm 1 FAS: Fractional Anycast Scheduling Algorithm

Input: $\phi = |E'|$,$\delta := \frac{1}{(1+\xi)^{\frac{1-\xi}{\xi}}}(\frac{1-\xi}{\phi})^{\frac{1}{\xi}}$,$l(e) = \frac{\delta}{c(e)}$,$x_e(v) = 0, \forall e \in E, v \in V$,$D(l) := \phi\delta$;
Output: traffic x, γ;

while $(D(l) < 1)$ **do**
  $R'(v) = R(v), \forall v \in V$;
  while $D(l) < 1$ and $R'(v) > 0$ for some v **do**
    p_v : shortest path from v to s' using l;
    $c'(e) = c(e), \forall e \in U_{v:v\in V} P_{vs'}$;
    for each $v \in V$ **do**
      $c := min_{e\in P_{vs'}} c'(e)$;
      if $R'(v) \leq c$ **then**
        $x_e(v) = x_e(v) + R'(v) \quad \forall e \in P_{vs'}$;
        $c'(e) = c'(e) - R'(v) \quad \forall e \in P_{vs'}$;
        $f_v = R'(v)$;
        $R'(v) = 0$;
      else
        $x_e(v) = x_e(v) + c \quad \forall e \in P_v$;
        $f_v = c$;
        $R'(v) = R'(v) - c$;
      end if
    end for
    $l(e) := l(e)(1 + \xi \cdot \frac{\sum_{v:e\in P_{vs'}} f_v}{c(e)})$, $\forall e \in \cup_{v:v\in V} P_{vs'}$
    $D(l) = \sum_e c(e)l(e)$
  end while
end while
$x_e(v) := x_e(v)/\log_{1+\xi}\frac{1+\xi}{\delta}$, $\forall e \in E, v \in V$;
$\gamma := min_v \frac{\sum_{p:p\in P_{vs'}} x(p)}{R(v)}$.

2) SDN-FE Server Association (SSA): In the first step, each SDN-FE is permitted to connect with multiple servers by relaxing the assumption. Thus, this subsection mainly designs the SSA approach to determine the SDN-FE server association. That is, each SDN-FE should be associated with only one destination server due to the specific feature of anycast routing. Our SSA approach can guarantee that the traffic flow of each SDN-FE will be forwarded only to one server, and the achievable throughput of each SDN-FE would be not less than $\frac{1}{2}$ of the optimum. For simplicity, we use x_e, $x_e(v)$, $x(p)$ to denote the traffic load on link e, the traffic load of node v on link e, and the traffic load on path p, respectively.

After the first step, we get a flow graph G^f, which is constructed by attaching a traffic load x_e on each link e in graph G'. The flow graph G^f obeys two rules below:

1) *capacity constraint: $0 \leq x_e \leq c(e), \ \forall e \in E'$*
2) *flow conservation: Denote the in-edges of node u as $E_{in}(u)$, the out-edges of node u as $E_{out}(u)$. If $u \in V$, then $\sum_{e\in E_{in}(u)} x_e + R(u) == \sum_{e\in E_{out}(u)} x_e$, note that for some node $u \in V \cup S$, $R(u) = 0$; if $u = s'$, then $\sum_{e\in E_{out}(u)} x_e = 0$.*

Being inspired by the single-source unsplittable flow (UFP) solution [?], we introduce our approach to implement the SDN-FE server association. UFP is designed for a unidirectional graph. However, there may have bi-directional links in the derived flow graph G^f. So, before we apply the UFP solution [?], we should set a single direction for each of these bi-directional links, which can be achieved by the following steps:

1) *Check all the nodes in the flow graph. If node u has a link connecting with node v, denoted by e_1, and node v has a link connecting with node u, denoted by e_2, then we find a bi-directional link.*
2) *if $x_{e_1} = x_{e_2}$, then we delete both links. Otherwise, without loss of generality, suppose that $x_{e_1} < x_{e_2}$. We delete the link with less traffic load (i.e., e_1), and change the value of another link (i.e., e_2) to $x_{e_2} - x_{e_1}$.*

Then, we will apply the UFP method on the flow graph G^f to implement the SDN-FE server association. Basically, the UFP method tries to move flows from the sink nodes to a source node, which, in our case, is to move the traffics from the SDN-FEs to the virtual node. The traffic generated by a SDN-FE $v \in V$ is denoted by $R(v)$. Initially, Traffic $R(v)$ is located on node v. If there exists a link $e_{vu} \in E'$, with $x(e) \geq R(v)$, then we say traffic $R(v)$ could be moved to node u along link e, and after the move its location was updated to u. If traffic $R(v)$ is moved along link e, then x_e is updated to $x(e) := x(e) - R(v)$.

After all the traffics of SDN-FEs are moved to the virtual node, the UFP method terminates, and the SDN-FE server associations are built at the same time. To achieve this, the UFP method was designed to execute the move and the modification process repeatedly. The formal description of the SSA algorithm is given in Alg. **??**. *For each expression, UFP introduces two definitions [?]:*

Definition 1: A link $e_{u_1u_2}$ is singular *if u_2 and all the vertices reachable from u_2 have out-degree at most 1.*

Definition 2: Traffic $R(v)$ located on node u_1 is irregular *if for each link $e \in E_{out}(u_1)$, $x_e < R(v)$; Otherwise, it is called* regular.

In the moving process, each regular traffic will be moved immediately. After moving, all the traffics become irregular. Then, it needs a modification process to find new regular traffics. The modification process randomly picks a node, and follows one of its out links as far as possible, which forms a forward path. According to the flow conservation constraints, this forwarding process repeats until it finds a node with traffic

located on. Beginning from this node, the UFP method follows one of its incoming links, and continue this operation as far as possible, which forms a backward path. The steps of building the forward and backward paths are executed repeatedly, until some of these paths form into a cycle. Then, UFP can compute a value, by which the flows of the forward and backward paths are augmented and weakened, respectively. This modification will generate at least one regular traffic, thus some traffics becomes movable again.

Algorithm 2 SSA: SDN-FE Server Association Algorithm

Input: The original demand of each switch v sent on each link e: $x_e(v)$, here x is used to denote the original traffic flow

Output: A new traffic flow y, in which the total demand of every switch v is sent to only one of the servers. Note that the demand of different switches could be sent to different servers.

$y_e(v) := 0$, $\forall e \in E$, $\forall v \in V$
for each $e = (u, v)$, $v \in V$ **do**
  if $x_e \geq R(v)$ **then**
    Move vertex v to u;
    $x_e = x_e - R(v)$;
    $y_e(v) = y_e(v) + R(v)$;
  end if
end for
while some $v \in V$ is not at s' **do**
  Find an alternating path $L = F \cup B$, $f \in F$ is a forward path, with a flow of x^f ,and $b \in B$ is a backward path,with a flow of x^b;
  $\Delta = \min\{\min_{f\in F}\{\min_{e\in f} x_e\}, \min_{b\in B}\{\min_{v\in b}(R(v)- x_e)\}\}$;
  $x^f = x^f - \Delta$;
  $x^b = x^b + \Delta$;
  for each $e = (u, v)$, $v \in V$ **do**
    if (e is a singular edge and $x_e = R(v)$) or (e is not a singular edge and $x_e \geqq R(v)$) **then**
      Move v to u;
      $x_e = x_e - R(v)$;
      $y_e(v) = y_e(v) + R(v)$;
      if $x_e = 0$ **then**
        Vanish e from graph G'
      end if
    end if
  end for
end while
output $y_e(v)$, $\forall e \in E, v \in V_{in}$.

3) Load-Balanced Bandwidth Allocation: After determination of SDN-FE server association, we will allocate the bandwidth among all the paths between each SDN-FE and the associated server so as to reach the load balance or minimize the congestion, which can be formulated into a linear program LP_2, as follows.

$$\min \quad \gamma$$
$$S.t. \begin{cases} \sum_{p:p\in P_{vs}} x(p) \geq R(v), & \forall v \in V \\ \sum_{p:p\ni e} x(p) \leq \gamma c(e), & \forall e \in E' \\ x(p) \geq 0, & \forall p \end{cases} \quad (5)$$

*As equation (**??**) is a linear program, we can solve it in polynomial time with a linear program solver. As equation (**??**) is equivalent to the following equation.*

$$\max \quad \gamma'$$
$$S.t. \begin{cases} \sum_{p:p\in P_{vs}} x(p) \geq \gamma' R(v), & \forall v \in V \\ \sum_{p:p\ni e} x(p) \leq c(e), & \forall e \in E' \\ x(p) \geq 0, & \forall p \end{cases} \quad (6)$$

*Therefore, the FPTAS method [**?**] can also be applied to deal with this linear program with lower time complexity. As this is similar to the method introduced in previous subsection, we omit the details here.*

4) Performance Analysis: For ease of expression, the optimal throughput factor of the HTAR problem is denoted by λ_{opt}. In the fist step, the FAS approach solves the linear program LP_1, which is the relaxation of the HTAR problem. The result of LP_1, denoted by λ_{upper}, is an upper bound for the HTAR problem. That is, $\lambda_{upper} \geq \lambda_{opt}$. As LP_1 may contain an exponential number of variables in equation (3)*, we apply the FPTAS method to solve this problem fast, and obtain the throughput factor as λ_f. Thus,*

Lemma 2: $\lambda_f \geq \frac{1}{(1+\xi)}\lambda_{opt}$.

Proof: By features of the FPTAS method, we know that,

$$\lambda_f \geq \frac{1}{(1+\xi)}\lambda_{upper} \quad (7)$$

Since $\lambda_{upper} \geq \lambda_{opt}$, it follows that $\lambda_f \geq \frac{1}{(1+\xi)}\lambda_{opt}$. The lemma is proved. ■

In the second step, our algorithm adopts the UFP method to establish the SDN-FE server association. Under the optimal scheme, UFP can guarantee that the throughput factor of each SDN-FE should not be less than $\frac{1}{2}$. Suppose that the throughput factor is denoted by λ_s at the end of the second step. As the FPTAS method can reach $(1+\xi)$-approximation for fractional anycast scheduling, it follows that:

$$\lambda_s \geq \frac{1}{2}\lambda_f \geq \frac{1}{2(1+\xi)}\lambda_{opt}. \quad (8)$$

In the third step, the algorithm enhances the load balance through a linear program method, compared with that in the second step. We use λ_{sar} to denote the final throughput factor by the SAR algorithm. It follows that $\lambda_{sar} \geq \lambda_s$. As a result, it follows that $\lambda_{sar} \geq \frac{1}{2(1+\xi)}\lambda_{opt}$. Now, we give the approximate performance of the proposed algorithm.

Theorem 3: The SAR algorithm can obtain the network throughput more than $\frac{1}{2(1+\xi)}$ times of the optimal for HTAR.

*5) Complexity Analysis: In this subsection, we analyze the time complexity of the proposed SAR algorithm. In the first step, the FAS approach solves the linear program LP_1 using the FPTAS method. By [**?**], we know that the time complexity of this step is $\mathrm{O}(\xi^{-2}(\phi^2 + \varphi^2\phi)\log^{\mathrm{O}(1)}\phi)$, where $\xi > 0$ is an arbitrarily small number, φ is the total number of SDN-FEs and servers in a network, i.e., $\varphi = m+n$, and ϕ is the amount of links in the graph G'. Usually, $\varphi = \mathrm{O}(\phi)$. In the second*

step, the algorithm mainly adopts the UFP method to solve the SDN-FE server association problem. The time complexity is $O(\varphi\phi+\varphi^2\log\varphi)$ [?]. In the third step, the algorithm will solve the bandwidth allocation for sake of load balance. The time complexity of the third step is $\mathrm{O}(\xi^{-2}(\phi^2+\varphi^2\phi)\log^{\mathrm{O}(1)}\phi)$ [?]. Thus, we obtain the following lemma.

Theorem 4: The total time complexity of the SAR algorithm is $\mathrm{O}(\xi^{-2}(\phi^2+\varphi^2\phi)\log^{\mathrm{O}(1)}\phi)$.

B. Algorithm for Congestion-free Anycast Reconfiguration

Since traffic demands and network topologies may change dynamically, the SDN-C should compute a new anycast routing configuration for SDN-FEs. Given a start configuration C^s and a final configuration C^f, immediate reconfiguration may result in congestion on some links and servers. Thus, we propose an anycast reconfiguration algorithm (ARC) for congestion-free routing reconfiguration. First, this algorithm tries to find a sequence of $t(\geq 1)$ transitive configurations at most, where t is a predefined constant, to fulfill the congestion-free routing reconfiguration while satisfying traffic demands of all the SDN-FE, i.e., throughput-guarantee. Notice that, a subset of SDN-FEs will finish anycast routing reconfiguration in each transitive configuration. If the heuristical step fails, the second step shall reduce the throughput with max-min fairness for congestion-free reconfiguration.

First, the algorithm tries to find a sequence of at most t transitive configurations for anycast reconfiguration, denoted by $\{C_0(=C^s)...C_i...C_{t+1}=(C^f)\}$. In each configuration C_i, we search for a set of SDN-FEs $\widetilde{V}$, which can simultaneously reconfigure the anycast routing with respect of capacity constraints on links and servers. The capacity constraints are expressed by the following conditions.

1) $\sum_{v\in V}\max\{x_e^i(v), x_e^{i-1}(v)\}\leq c(e),\ \forall e\in E$

2) $\sum_{v\in V}\max\{b_{vs}^i R^i(v), b_{vs}^{i-1}R^{i-1}(v)\}\leq c(s),\ \forall s\in S$

Then, we use a greedy method to construct set $\widetilde{V}$. Initially, set $\widetilde{V}$ is null. We pick a SDN-FE v, whose routing update will still satisfy the above two conditions, and put this SDN-FE in the set $\widetilde{V}$. This procedure is repeated until there is no SDN-FE satisfying the above two constraints. Anycast routing of all the SDN-FEs in set $\widetilde{V}$ will be updated in the transitive configuration C_i. If all the SDN-FEs can reconfigure their anycast routing without congestion within t configurations, we get a feasible solution for the CFAR challenge, and the algorithm terminates. Otherwise, the algorithm will continue to find a new transitive configuration C_{i+1}. However, while the number of transitive configurations exceeds a given parameter t, the first step terminates.

Due to capacity constraints on links and servers, we may not find a serial of transitive configurations for congestion-free routing reconfiguration with throughput-guarantee by a heuristical method. Thus, the second step will design one transitive configuration for anycast routing reconfiguration, which aims to congestion-free with max-min throughput fairness for all SDN-FEs. To implement this, we try to find a configuration, in which each SDN-FE will set its destination server as that in the final configuration. To compute the max-min throughput, we construct a linear program LP_3 as follows.

Algorithm 3 ARC: Anycast Routing Reconfiguration Algorithm

Input: The original routing plan denoted by traffic flow x^0, and a vector of b^0 where $b_{vs}^0\in b^0$ representing that the demand of switch v is sent to server s. The target plan denoted by traffic flow x^k and b_{vs}^k, where k is the given amount of phases the route should be updated.

Output: A list of routing plan $\{x^i, b^i\}$, $i=0,1,2...k-1$

```
Step 1: Greedy Routing Update
i = 1;
c(s) := c(s) − Σ_{v∈V} b^i_{vs} R^i(v);
C_i = ∅;
while i ≤ k do
    for each v ∈ V do
        if Ṽ = v ∪ C_i satisfies conditions(1)(2): then
            c(e) = c(e) − x^i_e(v), ∀e ∈ p, p ∈ P^i_v;
            c^i(s) = c^i(s) − R(v), ∀s ∈ {s|b^i_vs = 1};
            V = V − {v};
            C_i = C_i ∪ {v};
        end if
    end for
    for each v ∈ C_i do
        c(e) = c(e) + x_e(v), ∀e ∈ p, p ∈ P^i_v;
        c(s) = c(s) − R(v), ∀s ∈ {s|b^i_vs = 1};
    end for
    if (|V| = 0) then
        Construct a routing update phase
    else
        i = i + 1;
        C_i = ∅;
    end if
end while
Step 2: LP-based Routing Update
c(s) := c(s) − Σ_{v∈V} b^0_vs R(v);
Solve the equation (??)
Anycast Routing Update during one phase.
```

$$\max\ \ \gamma$$
$$S.t.\quad \begin{cases} \sum_{p\in P_v^2} x^1(p)\geq \gamma R^f(v), & \forall v\in V \\ \sum_{p\ni e} max\left(x^s(p), x^1(p)\right)\leq c(e), & \forall e\in E \\ \sum_{v\in V} max\{b_{vs}^0 R^s(v), b_{vs}^2\sum_{p\in P_v} x^1(p)\}\leq c(s), & \forall s\in S \\ x^1(p)\geq 0, & \forall p \end{cases} \tag{9}$$

Note that, b_{vs}^i represents whether SDN-FE v is connected with server s in the configuration C_i. More specifically, b_{vs}^0 and b_{vs}^2 mean the association results in the start and final anycast configurations, respectively. The first set of equations denotes to preserve the max-min fairness of all the SDN-FEs. The second and third sets of inequations ensure congestion-free on each link and each server. The forth set of inequalities ensures that the flow on any path is non-negative. As this is a linear program, we can solve this in polynomial time and

obtain an anycast routing configuration. The ARC algorithm is described in Alg. **??**.

IV. Performance Evaluation

In this section, we evaluate our anycast routing and updating algorithms by comparing with the previous methods. The simulations are executed on the Mininet platform [?], which is a widely-used simulator for software defined networks.

A. Simulation Setting

We run three groups of experiments to check the effectiveness of our algorithms. The first group of simulations mainly observes the impact of traffic demands on the performance of anycast routing under different topologies. The second group of simulations studies the performance of our routing updating algorithms under different networks. The third simulation observes the time cost while running these proposed algorithms in this work. In the simulation, we mainly adopt two practical topologies with different network sizes from [?]. The first topology, denoted by (a), contains 4 servers, 10 SDN-FEs, 16 hosts and 48 links. The second topology, denoted by (b), contains 10 servers, 100 SDN-FEs, 190 hosts and 397 links. For the first topology, all the link capacities were assumed to be equal. For the second topologies, the link capacities are random.

For performance evaluation, we adopt different metrics for different simulations. For anycast routing, this work uses two important metrics. The first one is the network load ratio. Given traffic demands of all the SDN-FEs, the simulator can return the traffic load $f(e)$ of each link $e \in E$, and compute the server load $f(s)$ of each server $s \in S$. Then, the network load ratio (NLR) is defined as:

$$NLR = \max\{\max\{\frac{f(e)}{c(e)},\ e \in E\}, \max\{\frac{f(s)}{c(s)},\ s \in S\}\} \tag{10}$$

Note that, the above equation reflects the load ratio on both links and servers. The second one is the packet loss ratio, which reflects the transmission efficiency of a SDN and quality of experience. For anycast routing reconfiguration, two metrics are adopted for performance comparison in the second group of simulation.

B. Numerical Results for Anycast routing

To evaluate how well our SAR algorithm performs, we compare it with two reference methods. One is the SSPF algorithm [?], in which each SDN-FE will connect with the closest server for resource access using shortest path first method. That is, the SSPF method selects the shortest shortest path for anycast routing. The other is RR [?], in which each SDN-FE will connect with one closest server, and the traffic demand of each SDN-FE is split equally on all the shortest paths. This set of simulations mainly observes how traffic demand of each SDN-FE affects the network performance, such as network load ratio and packet loss ratio, etc.

The first simulation will apply our SAR algorithm on the topology (a), which is small. From Fig. **??**, *the network load ratio is increasing with traffic demands for three anycast protocols. That's because, all the links and servers will burden much more traffics on average when all the SDN-FEs request more and more traffics. Moreover, our SAR algorithm can significantly decrease the network load ratios 27.5% and 73.6% compared with the RR and SSPF methods, respectively. This also indicates that SAR can reach a higher network throughput compared with two other anycast methods. From Fig.* **??**, *we find that packet loss ratios of three algorithms are becoming higher with the increasing of traffic demands. Under the light traffic demands, the packet loss ratios of three algorithms also low and almost similar. However, under the heavy traffic demands, the SAR and RR method can perform better than the SSPF method, which shows the advantage of multi-path anycast transmission in the network. Moreover, SAR can reach a lower packet loss than the RR method, which indicates the advantage of the SDN technology.*

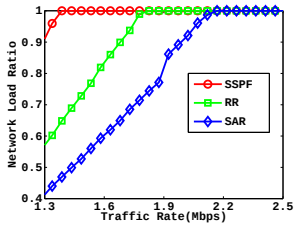


Fig. 5: Network Load Ratio vs. Traffic Rate for Topology (a)

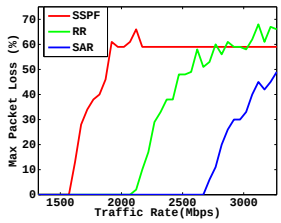


Fig. 6: Maximum Packet Loss vs. Traffic Rate for Topology (a)

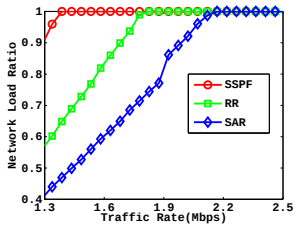


Fig. 7: Network Load Ratio vs. Traffic Rate for Topology (b)

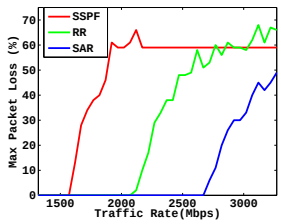


Fig. 8: Maximum Packet Loss vs. Traffic Rate for Topology (b)

The second simulation will apply our SAR algorithm on the topology (b), which is much larger than topology (a). From Fig. **??**, *the network load ratio is increasing with traffic demands for three anycast protocols. Moreover, our SAR algorithm can significantly decrease the network load ratios 26.5% and 70.2% compared with the RR and SSPF methods, respectively. This also indicates that SAR can reach a higher network throughput compared with two other anycast methods. Fig.* **??** *shows that packet loss ratios of three algorithms are becoming higher with the increasing of traffic demands. Under the light traffic demands, the packet loss ratios of three algorithms also low and almost similar. However, under the heavy traffic demands, the SAR and RR method can perform better than the SSPF method, which shows the advantage of multi-path anycast transmission in the network. Moreover, SAR can reach a lower packet loss than the RR method, which indicates the advantage of the SDN technology.*

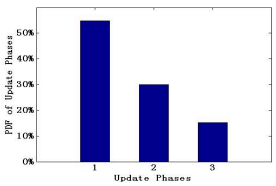


Fig. 9: Probability Distribution Frequency (PDF) of Update Phases

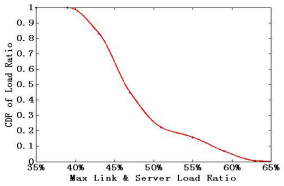


Fig. 10: CDF of Load Ratio vs Network Load Ratio

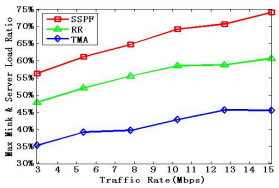


Fig. 11: Network Load Ratio vs. Traffic Rate

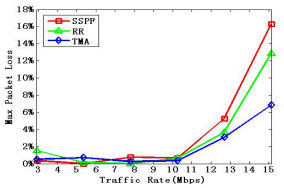


Fig. 12: Maximum Packet Loss vs. Traffic Rate

C. Numerical Results for Anycast routing Update

We then study the performance of our proposed anycast updating algorithm. First, we observe the number of required phases for anycast updating under different topologies. In the simulations, we random generate 180 sets of traffic demand distributions, and run our routing updating algorithm to compute the average number of updating phases. The simulation results show that the anycast routing update needs only 1-3 phases in the most cases. We give the probability distribution frequency of phase number in Fig. **??***. From this figure, the probability that the route update can be finished in one phase is more than 50%. Almost all the anycast routing updates will be finished in three phases, which shows the efficiency of our ARC algorithm. Fig.* **??** *shows the CDF of network load ratio during the anycast routing updating. This figure shows that the algorithm can control the congestion during the anycast update. Two figures show that our ARC algorithm can deal with the congestion-free anycast routing update in a small number of phases, which indicates a shorter time for anycast updating. Moreover, the network load ratio will not be very high during routing updating, which also helps to provide high network performance.*

D. Time Consuming of The Proposed Algorithms

As traffic demands may dynamically change, it is necessary to compute the anycast routing and updating in timely. Thus, we observe the consumed time of the proposed algorithm under three topologies with different network sizes. As shown in Fig. **??***, two algorithms can finish the computation in 2 seconds, though the network size is very large, such as 200 SDN-FEs and 400 hosts, etc, which shows the algorithm feasibility in the practical scenarios.*

V. Related Works

Since anycast has been widely used in different applications, such as resource acquirement and DNS, etc, many studies have devoted to anycast routing. Partridge et. al. [?] first introduced the anycast routing problem, in which data packets would be transmitted to one member of the anycast destination group, and discussed the potential applications of anycast routing. Xuan et. al. [?] proposed the shortest shortest path first (SSPF) algorithm for anycast routing selection. This algorithm was a simple extension to the traditional shortest path first method for unicast routing. The authors [?] divided the anycast routing into two stages, routing exploration and packet forwarding. They proposed four anycast routing protocols for purpose of loop prevention. These methods differed from each other on information used to maintain orders, the impact on QoS, and compatibility to the existing routing protocols. The work [?] presented a practical anycast approach with near-optimum delays while taking into account the processing loads at routers and processing elements of a computer network. The anycast routing was generalized into the problem of minimum-delay load-balanced routing to account for processing delays on network devices. They proposed a distributed algorithm for load balanced anycast in computer networks. A new mechanism, called "Anycast Address Mapper" [?], enabled to find out the corresponding unicast address from the anycast address using Internet Control Message Protocol (ICMP). They implemented and evaluated the mapper mechanism on FreeBSD-2.x/3.x. Hussein et. al. [?] evaluated anycast by proposing a load-aware IP anycast content distribution network (CDN) architecture. This architecture made use of route control mechanisms to take server and network load into account to realize load-aware anycast. They formulated the anycast routing as a Generalized Assignment Problem and presented practical algorithms that addressed the numbers of concurrency while at the same time limiting connection disruptions. Most of these methods are designed on the distributed route control mechanism, which can not provide flexible route control, and may result in performance oscillation.

Recently, SDN [?] [?] [?] is introduced to improve the network performance with centralized route control, which makes the network more flexible and high-efficiency. Thus, it has become an emerging technology for the future network. The Google cooperation [?] presented the design, implementation and evaluation of B4, which connected their data centers across the planet. B4 took a software defined networking architecture using OpenFlow to control SDN-FEs. The evaluation on the hardware platform showed that centralized traffic engineering could drive links to near 100% utilization through flow splitting among multiple paths. The authors [?] presented the SWAN system that boosted the utilization of inter-datacenter networks by centrally controlling when and how much traffic each service sent. They also developed a novel technique based on a linear program method to apply updates in a congestion-free manner. The work [?] studied the effective use of SDNs for traffic engineering especially when SDNs were incrementally deployed into an existing network. They showed that these improvements could be obtained even in cases where there was only a partial deployment of SDN capability in a network. All these works are studying the SDN-

based routing. However, no work has been devoted to SDN-based anycast routing.

In all, the previous works [?] [?] [?] [?] on anycast routing are mostly based on the distributed route control, which may result in performance oscillation. Thus, they can not fully explore the advantages of the software defined network. Some works [?] [?] have devoted to routing in a software defined network. However, as the specific features of anycast routing, these works [?] [?] may not achieve the high-efficiency for anycast routing. Moreover, the anycast routing updating is not found in the previous studies.

VI. Conclusion

In this paper, we analyze two main challenges for a SDN-based anycast system: high throughput anycast routing and congestion-free anycast reconfiguration. We propose efficient algorithms for both anycast routing and routing-table reconfiguration. We prove that our routing method achieves a throughput that is at least a constant fraction of the optimum. We then show by simulations that our algorithms perform significantly better than the SSPF anycast method. As QoS is another important measurement for anycast routing in many applications, a future work is to study QoS-aware anycast routing and scheduling in a software defined network, including minimum delay or delay/reliability guarantee, etc.

References

[1] *M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker, "Ethane: Taking control of the enterprise,"* ACM SIGCOMM Computer Communication Review, *vol. 37, no. 4, pp. 1–12, 2007.*

[2] *N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: enabling innovation in campus networks,"* ACM SIGCOMM Computer Communication Review, *vol. 38, no. 2, pp. 69–74, 2008.*

[3] *"The openflow switch," openflowswitch.org.*

[4] *N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker, "Nox: towards an operating system for networks,"* ACM SIGCOMM Computer Communication Review, *vol. 38, no. 3, pp. 105–110, 2008.*

[5] *T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama,* et al., *"Onix: A distributed control platform for large-scale production networks." in* OSDI, *vol. 10, 2010, pp. 1–6.*

[6] *M. Caesar, D. Caldwell, N. Feamster, J. Rexford, A. Shaikh, and J. van der Merwe, "Design and implementation of a routing control platform," in* Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation, *2005, pp. 15–28.*

[7] *N. Feamster, H. Balakrishnan, J. Rexford, A. Shaikh, and J. Van Der Merwe, "The case for separating routing from routers," in* Proceedings of the ACM SIGCOMM workshop on Future directions in network architecture, *2004, pp. 5–12.*

[8] *C.-Y. Hong, S. Kandula, R. Mahajan, M. Zhang, V. Gill, M. Nanduri, and R. Wattenhofer, "Achieving high utilization with software-driven wan," in* Proceedings of the ACM SIGCOMM 2013, *2013, pp. 15–26.*

[9] *"Software defined flexible and efficient passive optical networks for intra-datacenter communications,"* Optical Switching and Networking.

[10] *H. Egilmez, S. Dane, K. Bagci, and A. Tekalp, "Openqos: An openflow controller design for multimedia delivery with end-to-end quality of service over software-defined networks," in* Signal Information Processing Association Annual Summit and Conference (APSIPA ASC), 2012 Asia-Pacific, *Dec 2012, pp. 1–8.*

[11] *C. Partridge, T. Mendez, and W. Milliken, "Host anycasting service," RFC 1546, November, Tech. Rep., 1993.*

[12] *Y. Dinitz, N. Garg, and M. X. Goemans, "On the single-source unsplittable flow problem,"* Combinatorica, *vol. 19, no. 1, pp. 17–41, 1999.*

[13] *S. Agarwal, M. Kodialam, and T. Lakshman, "Traffic engineering in software defined networks," in* INFOCOM, 2013 Proceedings IEEE, *2013, pp. 2211–2219.*

[14] *G. Karakostas, "Faster approximation schemes for fractional multicommodity flow problems," in* Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms. *Society for Industrial and Applied Mathematics, 2002, pp. 166–173.*

[15] *C. Partridge, T. Mendez, and W. Milliken, "Host anycasting service," RFC 1546, November, Tech. Rep., 1993.*

[16] *D. Xuan, W. Jia, and W. Zhao, "Routing algorithms for anycast messages," in* Parallel Processing, IEEE International Conference on, *1998, pp. 122–130.*

[17] *V. V. Vazirani,* Approximation algorithms. *springer, 2001.*

[18] *H. Ballani, P. Costa, T. Karagiannis, and A. Rowstron, "Towards predictable datacenter networks," in* ACM SIGCOMM Computer Communication Review, *vol. 41, no. 4. ACM, 2011, pp. 242–253.*

[19] *"The mininet platform," http://mininet.org/.*

[20] *"The network topology from the monash university," http://www.ecse.monash.edu.au/twiki/bin/view/InFocus/LargePacket-switchingNetworkTopologies.*

[21] *M. J. Freedman, K. Lakshminarayanan, and D. Mazières, "Oasis: Anycast for any service." in* NSDI, *vol. 6, 2006, pp. 10–10.*

[22] *D. Xuan, W. Jia, W. Zhao, and H. Zhu, "A routing protocol for anycast messages,"* Parallel and Distributed Systems, IEEE Transactions on, *vol. 11, no. 6, pp. 571–588, 2000.*

[23] *W. T. Zaumen, S. Vutukury, and J. Garcia-Luna-Aceves, "Load-balanced anycast routing in computer networks," in* IEEE Symposium on Computers and Communications, ISCC 2000. *IEEE, 2000, pp. 566–574.*

[24] *M. Oe and S. Yamaguchi, "Implementation and evaluation of ipv6 anycast," in* Proceedings of 10th Annual Internet Society Conference, Yokoham, Japan, *2000.*

[25] *H. A. Alzoubi, S. Lee, M. Rabinovich, O. Spatscheck, and J. Van Der Merwe, "A practical architecture for an anycast cdn,"* ACM Transactions on the Web (TWEB), *vol. 5, no. 4, p. 17, 2011.*

[26] *S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu,* et al., *"B4: Experience with a globally-deployed software defined wan," in* Proceedings of the ACM SIGCOMM 2013. *ACM, 2013, pp. 3–14.*