CFN-dyncast: Load Balancing the Edges via the Network

Bing Liu, Jianwei Mao, Ling Xu, Ruizhao Hu, Xia Chen
Huawei Technologies Co., Ltd.
China

***Abstract*—Multi-access Edge Computing (MEC) is a promising business paradigm in the 5G network. The principle "the nearest is the best" may not apply in some cases. How to break the boundaries among individual MEC sites and leverage the computing resources as an integrity, is the key to improve the user experience and improve the usage efficiency of computing and network resources. This paper proposes *CFN-dyncast*, a distributed technique that dispatches clients' demands to an optimal site according to the load of each computing site and the network status. This paper also introduces the related design considerations, the implementation and the evaluation comparing to other load balancing techniques.**

Keywords*—Compute first networking, Multi-access Edge Computing, Load Balancing, Carrier Network, Computing-aware*

I. INTRODUCTION

Edge Computing is a paradigm that seeks the optimal trade-off between computing, storage, transmission and latency. Its deployment forms varies in different scenarios. Multi-access edge computing (MEC) is a promising form for Internet service providers (ISP). Base stations, central offices and other facilities can be used as candidate edges (MEC sites) to host third party services.

However, the deployment of a MEC site often has many constraints. Firstly, the location of a MEC site must be chosen wisely considering the geographical environment, the population nearby, the power supply and the network access. Secondly, a MEC site may be limited in its physical size. Excluding the space for network devices, power system and cooling system, there is little space left for computing devices, e.g., servers, NPUs and GPUs. Thus the computing resource of each individual MEC site is limited and the ceiling is much easier to be touched compared to large data centers.

When dispatching clients' demand to a certain service, the intuitive idea is to send the demand to the nearest MEC site. In this case, the individual sites are considered as silos and each site only covers its own neighborhood area. Some sites can be saturated easily at peak hours, therefore longer response time or even demand dropping can be experienced by the users. Meanwhile some other sites may be running idly, which is a waste of compute resources and investment. Scale-out of the saturated sites may not be possible because the physical size is limited, and scale-up needs more investment.

Thus it is necessary to consider the compute resources of the MEC sites as a pool and load balancing among them. A traditional way to load balance between data centers is DNS-based. Normally, the DNS server returns an IP address based on the geographic location of the client IP or based on pre-provisioned weights. This technique is static and may cause the unbalanced work load on different sites. Some L4 load balancer like LVS supports the statistics of connection number to service instances based on L4 header inspection, and dispatches the client's demand to the instance that has the least connections. Since the L4 load balancer is normally deployed inside the data center, where the network condition is thought to be equivalent to different instances, the real-time network condition is not considered during the dispatch process.

The MEC sites at different locations are interconnected via the carrier network. The logical distance between MEC sites differs, and the network status varies. When load balancing among the MEC sites, it is necessary to consider both the compute dynamics and the network dynamics. This idea argues the job completion time as the summation of the computing delay and the network delay. As long as this summation is under the tolerance of the client's request, any application instance that meets this condition, no matter which MEC site it is deployed on, can be the one that serves this request. In this way, the capacity of a MEC site could even be set to lower than the requirement at peak hours, at the same time the gap can be filled by other MEC sites. This solution can save investment under the prerequisite of ensuring the user experience.

CFN-dyncast, as proposed in this paper, is a technique that follows the resource pooling idea. In CFN-dyncast, a dedicated IP address is assigned to each application/service, and this address can be used to reach any of the service endpoints deploying on the edge. It sounds like traditional "anycast in the network", but it is not the same thing. In normal anycast, the packets are always sent to the nearest location. In CFN-dyncast, the network is aware of the load of each computing site (edge and cloud), thus the network is in charge of dispatching the client's requests to a proper site. Since the load of each site and the network status are dynamic, this kind of anycast is called "dyncast", which is short for "dynamic anycast".

The paper makes the following contributions:

(1) Design of the CFN-dyncast control plane: collect the raw compute load status of the service instances from cloud-native platform; advertise load status from server to network device; and from device to device, via network protocols.

(2) Design of the CFN-dyncast data plane: choose the optimal MEC site when a client's request arrives, and record the decision into a session table to achieve session affinity.

(3) Implementation of a prototype using servers and physical network devices.

(4) Evaluation of the technique and comparison with normal DNS and compute-aware DNS, a technique dispatches the demands in a centralized way.

The rest of this paper is organized as follows. Section II gives a brief glance of the related works. In Section III, we present the key issues that we've considered in our design. We describe the overall design in section IV. In Section V, we

present the prototype that we've implemented in the laboratory environment. In section VI, the implementation is evaluated and compared to other load balancing techniques. We conclude with some discussion and plans for future work in Section VII.

II. Related Works

In 2016, ETSI has published the standard on MEC framework and architecture [1], which defines the functionality of different modules, such as Multi-access edge orchestrator (MEO), Multi-access edge platform manager (MEPM), MEC platform (MEP), Data plane and MEC app, along with the interfaces between these modules. The project EdgeGallery is a reference implementation of ETSI MEC.

Although the standards for MEC architecture have been published for several years, the discussion on the design of MEC network just begins recently. Reference [2] analyzes the key requirements for the edge computing network infrastructure, then divides the infrastructure into three parts: ECA (the access network), ECN (the network inside a MEC site) and ECI (interconnect network), and suggests key technologies for each part. Reference [3] thinks a step further and investigates enhanced functionality of the carrier network by introducing computing-aware features. Within IETF, the CFN related works are emerging. Reference [4] describes the use cases and problem statement of CFN and Reference [5] gives a primary architectural design of CFN-dyncast. Some extensions to network protocols, like BGP and OSPF, are proposed to advertise compute metrics inside the network, and the related works can be found in Reference [6] and [7]. Reference [8] explores the design of CFN over a ICN network layer, realizing functionalities like request forwarding, caching, and load management.

III. Problem statement

A. Deployment Environment

The MEC related infrastructure can be divided into three parts. Firstly, the access network, which is the network infrastructure that the flow goes through from the user system to the MEC site, e.g., the campus network, the cellular network, PON, 5G UPF and BNG, etc.. Secondly, the compute and network devices inside the MEC site, including several servers hosting accelerating hardware (GPU, NPU), one or two gateways, several switches if needed. Thirdly, the network interconnecting the MEC sites, in which tunnels can be built among the sites. The tunneling technologies can be categorized into overlay (represented by VXLAN and GRE) and underlay (represented by SRv6).

According to ETSI standards, a MEC site runs a MEC platform (MEP) to host the service instances. The service instances can be deployed inside VMs or containers. The MEC operator can choose to use OpenStack (or equivalent) and Kubernetes (or equivalent) to orchestrate them.

B. Considerations

When design the CFN-dyncast technique, we mainly consider the following aspects:

- Compute status acquisition with application granularity. Since different applications can be collocated on the same MEC site, they share the same compute resources. The status of a server can not represent the status of an individual application. Therefore it is necessary to acquire the compute status at the application level.
- Compute status advertisement. The compute status is acquired on the MEC platform, and the status needs to be advertised to the network, including all the network devices who need to make the dispatching decision. The status at different devices should be synchronized so that each device can have the correct overall perspective.
- Backward compatibility. The design should be as less invasive as possible, so that most of the infrastructure will be left untouched.
- Session affinity. Once the a client's demand is decided to be dispatched to a certain MEC site, the following packets within this session must be sent to the same MEC site, so that the session is not interrupted. After the current session is accomplished, the next demand of the same client, in a new session, can be dispatched to another MEC site.

IV. Solution Design

A. Overview

This paper uses the following terminology to facilitate the description.

- CFN router: A network device that is capable to be aware of compute metrics and dispatch client's demand accordingly. In MEC scenario, the CFN router feature could be implemented on the MEC gateway.
- Local CFN router: From the perspective of a MEC site, it is the CFN router that is responsible to accept the compute metric advertisement directly from the cloud platform of this particular site.
- Remote CFN router: From the perspective of a MEC site, a CFN router that accepts the compute metrics advertisement from the local CFN router of this particular site.
- Ingress CFN router: The first CFN router that a client's demand goes through. Normally, it is the local CFN router of the nearest MEC site to the client. It encapsulates, if necessary, the original demand with outer headers and forwards it to the egress CFN router.
- Egress CFN router: The CFN router that decapsulates the outer header, if exists, and forwards the original demand to the destination MEC site.

Note that the ingress and the egress can be the same device if the local (nearest) MEC is the one being selected for this demand.

As shown in Fig. 1., The operational mechanism of the control plane and data plane can be divided into the following steps.

- Control Plane
  1) *Compute status acquisition*: a compute-aware module deployed as software, called station daemon, is in charge of collecting the compute status like CPU usage, memory usage, number of connections from the instances of an application, and then calculates a compute metric accordingly. The compute metric is a

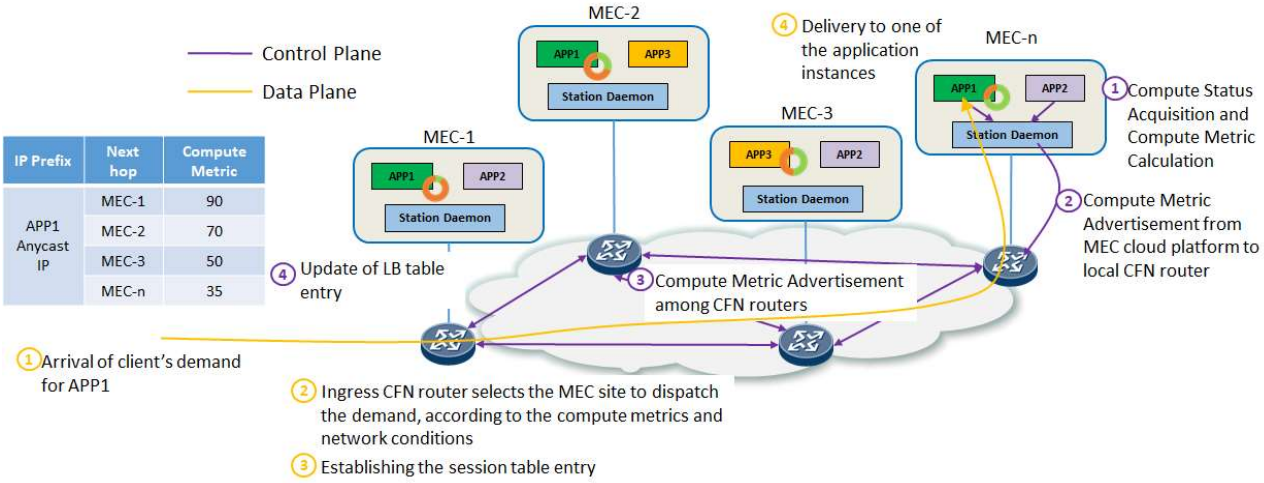

Fig. 1. Overview of the CFN-dyncast control plane and data plane

scalar value, which represents the workload of a certain application deployed on this MEC site.

2) *Advertisement from the MEC platform to the local CFN router:* the station daemon advertises the calculated metric to the local CFN router via a BGP speaker.

3) *Advertisement among CFN routers:* the local CFN router advertises the compute metric to the remote CFN routers via network protocol. Each CFN router advertises the metric of the site behind it to the others, so that every CFN router can have an overall perspective of all the related MEC sites.

4) *Update of load balancing table entry:* based on the overall perspective, each CFN router updates its local load balancing table.

- Data Plane

1) *Arrival of the client's demand:* The first packet from the client side arrives at the ingress CFN router. The ingress identifies this packet by its destination IP address (DIP). If the DIP is one of the pre-provisioned addresses, then go to the following steps. If not, the packet is forwarded in the ordinary way.

2) *Selection of the MEC site:* The ingress CFN router finds the optimal MEC site for this demand, according to the compute metric of the sites and the related network conditions. If the selected MEC site is the local one, the egress and the ingress are the same router, and the demand can be forwarded directly to the local site. If not, the ingress CFN router encapsulates the original packet into a tunnel destinating the egress CFN router.

3) *Setting up the session table:* The selection result must be recorded to a session table, so that the subsequent packets in the same session from this client to this application can be forwarded to the same MEC site. The related table entry should not be outdated until the current session is accomplished.

4) *Delivery to one of the application instances:* The egress CFN router decapsulates (if needed) the tunnel and forward the original packet to one of the application instances.

B. *Compute status acquisition*

Generally, the compute status is not limited to CPU usage, memory usage, GPU usage, number of connections, compute latency, etc. The status related to different applications can also be different, depending on the working principle of the application.

We investigate how to acquire compute status of application instances deployed on the cloud-native platform such as Kubernetes, namely K8S. On K8S, an application server can be instantiated as a set of pods, and each pod can be assigned with an amount of compute resources. The pods with the same functionality can be abstracted as a service with a virtual IP address (cluster IP). The service could be exposed externally, so that it can serve the clients' demands coming from the access network.

Prometheus is one of the mainstream systems to monitor the K8S platform and the services deployed on it. Exporters can be implemented on demand, and Prometheus is capable to scrape the exposed information with a certain sampling rate, then stores it in the time series database. Prometheus has an HTTP API, which permits to extract data selectively from the database.

In this paper, we design and implement a compute status acquisition module, called *station daemon*. The station daemon extracts compute status in application granularity from Prometheus, then processes the raw compute status into compute metric, a scalar value, to represent the workload of an application on its hosting MEC site. The station daemon, as the name suggests, is deployed per MEC site.

C. *Compute metric advertisement*

The compute metric advertisement comprises 2 phases. Firstly, the metric is advertised from the cloud platform to the local CFN router. Secondly, the metric is advertised from local CFN router to remote CFN routers. We choose to extend BGP protocol to carry the metric, thus only the CFN routers such as MEC gateways need to be aware of this extension, and the network devices between the CFN routers just need to forward the BGP signaling as normal IP packets. Two segments of BGP sessions are required: the first one is between a software BGP speaker deployed on server to the CFN router, and the second one is between two CFN routers or between CFN router and the Route Reflector (RR).

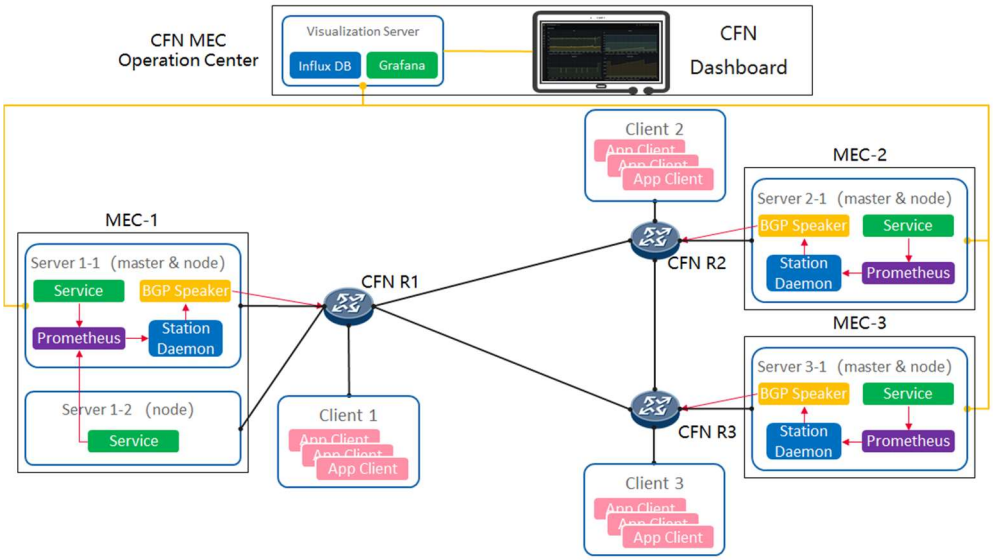

Fig. 2. Illustration of the testbed

Then each CFN router can have an overall perspective on compute metrics of all the MEC sites. Thus a routing table with compute metrics can be built. An example of the table is shown in TABLE I.

TABLE I. AN EXAMPLE OF THE ROUTING TABLE WITH COMPUTE METRICS

IP Prefix	Next Hop	Compute Metric
Anycast IP of APP1	MEC-1	90
	MEC-2	70
	MEC-n	30
Anycast IP of APP2	MEC-1	30
	MEC-3	40
	MEC-n	55

D. *Seission affinity*

Without session affinity, once the compute metrics of sites changes, a MEC site different from the current one may be selected as the optimal site, consequently the ingress CFN router will forward the subsequent packets of the ongoing session to this new optimal site. As a result, the current session between the client and the application instance is interrupted.

We use a session table on the ingress CFN router to record the selection result, so that the following packet of the session can be forwarded to the same MEC site. An example of the session table is given in TABLE II.

TABLE II. AN EXAMPLE OF THE SESSION TABLE

Session Identifier					Egress	Timeout
SRC_IP	DST_IP	SRC_PORT	DST_PORT	PROTO		
Client1	APP1	aaaa	bbbb	TCP	CFN R1	xxx
Client2	APP2	cccc	dddd	TCP	CFN R3	yyy

V. PROTOTYPE IMPLEMENTATION

We set up a testbed to implement and evaluate our design. We use Huawei routers and Huawei 2288HV5 server to build the testbed. The server has 32 CPU cores@2.10GHz, and is installed Ubuntu 18.04 OS.

As shown in Fig. 2, the testbed includes three MEC sites and three CFN Routers (R1, R2, R3) connected to them. The site MEC-1 has two servers, while the other two sites have one server each. Each MEC site runs K8S as the cloud environment, and the application instances are deployed as pods on K8S. MEC-1 has 10 pods and the other two sites have 5 pods each, which is proportionate to the hardware capacity of the sites. Every pod is assigned with 6 cores. Each MEC site also hosts a Prometheus pod, a station daemon and a BGP speaker.

In the control plane, BGP sessions are set between the BGP speaker and the CFN router directly connected to the hosting MEC site, and between two CFN routers. The client programs which can run in parallel are deployed on three other servers. In the data plane, SRv6 tunnels are set between each two CFN routers. At the egress CFN router, the function End.DX is performed to decapsulate the outer IP header and forward the internal packet to the destination MEC site.

We also implement a compute-intensive application, in which the client side sends out requests with parameters and the server side does the calculation and returns the result. Thus the CPU usage of the pods can be chosen as the raw compute status of this application. The station daemon processes the raw status and normalizes it into the compute metric of MEC site.

DNS is one of the widely used load balancing techniques among sites across wide area network. We would like to compare CFN-dyncast to DNS-based dispatching schemes, in which the clients need to send DNS request to find a MEC site if its local cache expires. We implement a real DNS server based on CoreDNS which provides address records of the MEC sites. Besides the native DNS, we also implement a "compute-aware DNS": the etcd plugin of CoreDNS is enabled, and CFN Daemon publishes records to etcd based on the local compute metric. The number of records is inversely proportional to the compute metric, realizing a DNS server with dynamic weights related to compute metrics of the MEC sites.

As a summary, there are three dispatching schemes implemented on our testbed:

- Native DNS: The weights of the records are set statically based on the capacities of MEC sites. The weight is set to 2:1:1 on our testbed.
- Compute-aware DNS: The weights of the records are set dynamically, based on the compute metric.
- CFN-dyncast: All the instances of an application are hidden behind an anycast IP, the MEC sites' addresses are not visible by the clients. The CFN routers are responsible to dispatch the clients' demands.

In addition, to visualize the compute metrics of applications in multiple MEC sites, we persist all the acquired compute status and the compute metrics into Influx DB. Then we develop a Dashboard base on Grafana to read data from the database and provides visualization.

VI. Evaluation and Analysis

In the evaluation, we have 71 clients running in parallel: one of them is selected for observation, and the rest are used as background workload. Each client will send out a series of demands.

We use the job completion time (JCT) as the performance indicator. It is defined as the time duration that the client has to spend until it gets the response from the server. In CFN-dyncast, the anycast IP keeps the same, and this address can be acquired via one time DNS request or even be provisioned a priori at the client. For DNS-based dispatching schemes, the client has to send a DNS request to decide the destination MEC site to which to send the demand, every time the local DNS cache expires. The DNS cache timeout for native DNS is set to 60s, and for compute-aware DNS, it is set to 15s.

The testbed runs three times with all the same clients' demands, but separately dispatched by the three schemes described above. Each test lasts for more than 20 minutes, and we obtain more than 8000 data records in each test, i.e., more than 8000 JCT data points.

As shown in Fig.3, the average JCT for Native DNS scheme is 175.87ms, for Compute-aware DNS is 179.07ms, and for CFN-dyncast is 149.54ms. In summary, CFN-dyncast scheme can shorten the average JCT by about 15%.

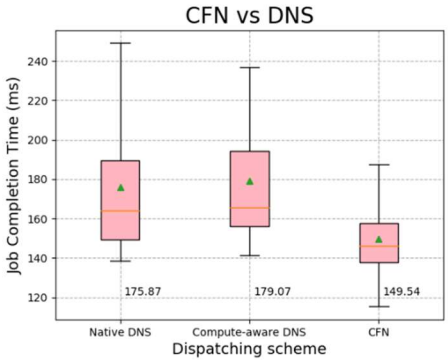

Fig. 3. Job completion time under different dispatching schemes

More details can be found in TABLE III. With CFN-dyncast, the span between the upper and lower bound of JCT is shorten by 34% compared to native DNS and 24% compared to compute-aware DNS. This result shows that the CFN-dyncast gains a more concentrated JCT distribution.

TABLE III. PERFROMENCE COMPARISON

	Native DNS	Compute-aware DNS	CFN-dyncast
Average JCT (ms)	175.87	179.07	149.54
Span between the upper and lower JCT bound (ms)	110.259	95.449	72.256
No. of completed job within 20min	7236	7278	7989

The number of completed jobs within the first 20 minutes is counted. It can be found out that the number of completed requests per unit time increased by about 10% when CFN-dyncast is used.

We analyzed the reason why CFN scheme performs better than DNS-based schemes. There are two main reasons: a) CFN just spends time on at most a single DNS request. And for compute-aware DNS, the client has to initiate DNS request once its local DNS cache expires, which add extra time to the JCT. b) The optimal MEC site in a client's local DNS cache can be outdated. Before the cache expires, the actual optimal MEC changes to another one, while the client keep sending requests to an outdated and suboptimal site, which leads to longer JCT.

In addition, we observe the curves of compute metric of the three sites dispatched by CFN-dyncast. As shown in Fig. 4. although the capability of the sites are greatly different, the compute metrics are converged to almost the same value after 4 minutes, which shows the computing payload of the application are balanced well among the sites.

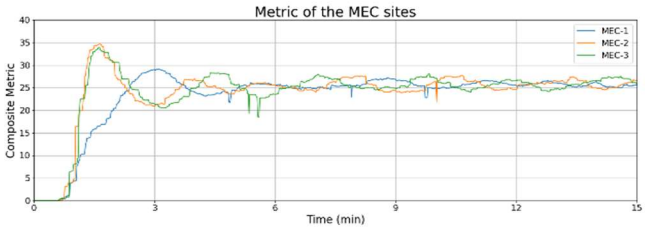

Fig. 4. Compute metric of MEC sites

VII. Conclusion

Integrated design of computing and network in the edge computing scenarios is emerging. In this paper, we design and implement CFN-dyncast, a technique that aims to load balance the MEC sites in consideration of the compute status in application granularity and the network conditions. Evaluation result shows that CFN-dyncast is capable to dynamically maintain the load of different MEC sites at the same level. Compared to centralized dispatching schemes, CFN-dyncast helps the clients get replied by the servers in a shorten period. Limited to the current lab environment, the difference of network conditions between each two CFN routers is not remarkable. For next steps, we will try to evaluate CFN-dyncast on wide area network, in which the network condition could be more decisive.

REFERENCES

[1] "Multi-access Edge Computing (MEC) Framework and Reference Architecture", ETSI ISG MEC, 2016.

[2] "Technical White Paper on Carrier Edge Computing Network", Edge Computing Consortium and Network 5.0 Industry and Technology Innovation Alliance, November 2019.

[3] "Technical White Paper on Computing-aware Networking", China Mobile and Huawei, November 2019.

[4] L. Geng, P. Liu and P. Willis, "Dynamic-Anycast in Compute First Networking (CFN-Dyncast) Use Cases and Problem Statement", IETF draft, October 2020.

[5] Y. Li, L. Iannone, et al. "Architecture of Dynamic-Anycast in Compute First Networking", IETF draft, October 2020.

[6] L. Dunbar, K. Majumdar and H. Wang, "BGP NLRI App Meta Data for 5G Edge Computing Service", IETF draft, November 2020.

[7] L. Dunbar and H. Chen, "OSPF extension for 5G Edge Computing Service", IETF draft, November 2020.

[8] Król, Michał, et al. "Compute first networking: Distributed computing meets icn." Proceedings of the 6th ACM Conference on Information-Centric Networking. 2019.