RTGWG
INTERNET-DRAFT
Intended Status: Informational

S. Gu
G. Zhuang
Huawei Technologies
H. Yao
X. Li
China Mobile

Expires: June 4, 2020

December 2, 2019

# A Report on Compute First Networking (CFN) Field Trial
# draft-gu-rtgwg-cfn-field-trial-01

Abstract

Compute First Networking (CFN) enables the routing of the service request to an optimal edge site to improve the overall system load balancing and efficiency. Especially when an edge site is overloaded, other edges with service equivalency can dynamically serve the request. This document describes a CFN field trial to show the effect that CFN can achieve. Edge to edge interaction to get the available computing resources information for services and the network status to each other is introduced. Data plane to support late binding based dynamic anycast is illustrated too. The field trial shows that CFN can greatly improve the overall query per second served for a service hosted on multiple edges in a more balanced way.

Table of Contents

## 1. Introduction

Compute First Networking (CFN) Scenarios and Requirements [CFN-req] shows the usage scenarios and requirements to dynamically dispatch the service request to multiple edge sites in order to overcome the computing resource overloading problem in edge computing. Compute First Networking (CFN) framework document [CFN-fmwk] presents the basic system framework to dynamically route a service request to a selected edge in real time based on the computing load status and network conditions. This approach improves the load balancing between multiple edges with service equivalency in a distributed manner. This document introduces a more concrete CFN field trial and its performance.

### 1.1 Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

## 2 Testbed overview

We deployed CFN node on three edge sites in Hangzhou. The sites are approximately 30 kilometers apart. Figure 1 shows the topology and configuration we used for this CFN testbed.

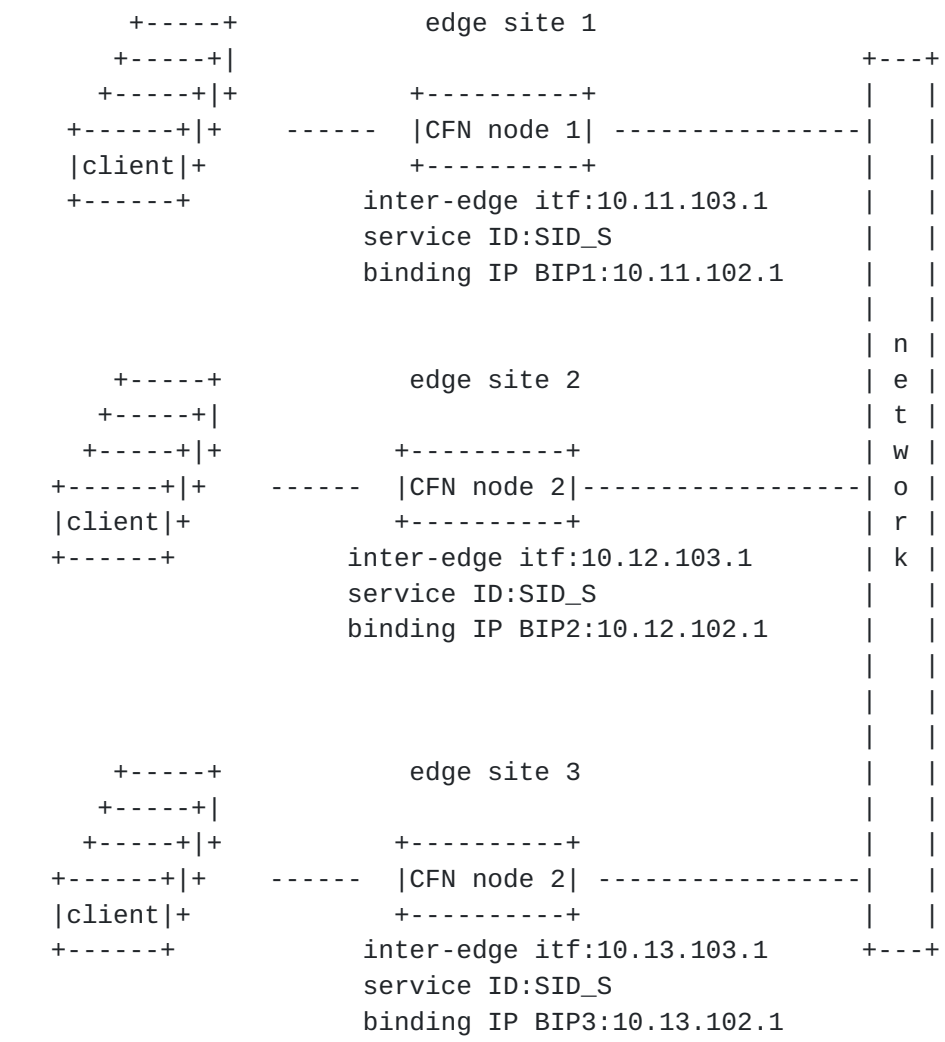



Figure 1. CFN testbed overview

A matrix multiplication service S is provided by all three edge sites (or edges for simplicity in this document). The CFN nodes use a unique service ID SID_S to announce the its reachability to service S. In our test, we use 200.200.200.201 for SID_S. Consider SID_S here as a anycast IP address. Though this service is reachable by a single SID_S in network, 3 edges indeed serve SID_S using 3 different binding IP (BIP) addresses , BIP1/2/3 with address 10.11/12/13.102.1 via CFN node 1/2/3 respectively. Service node hosted on or attached to a CFN node only knows that it uses its BIP to serve service S and has no knowledge about SID_S.

Each CFN node has an inter-edge interface IP address for communicating the computing load information among CFN nodes. About 200 simulated clients connect to each CFN node in the test.

## 3. Procedures

The procedures are introduced in [CFN-fmwk]. For easy reference, control plane and data plane timeline diagrams are shown here too.

### 3.1 Control Plane

When a service node is initiated for service S, the edge platform manager will send the registration information about service ID SID_S and binding IP (BIP) to access SID_S to the CFN node that the service node attaches to.

Each CFN node regularly gets the computing load information about the service node attached to it for SID_S. The computing load information can be CPU consumption for SID_S, number of current connections, query per second processed, total capacity, or other performance metrics. In our test, we give each type of metrics a weight. CFN nodes distribute those information to each other by BGP extensions. Figure 2 shows the CFN control plane procedures.

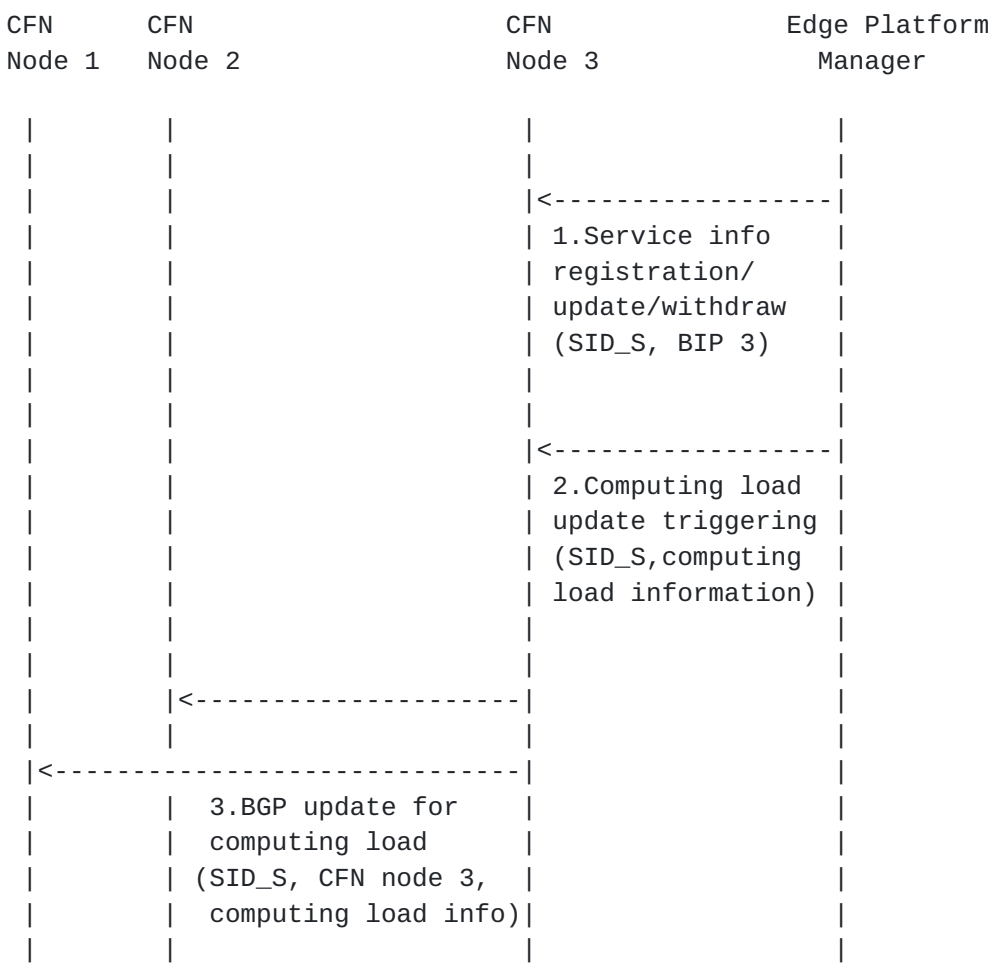



Figure 2. CFN control plane

## 3.2 Data Plane

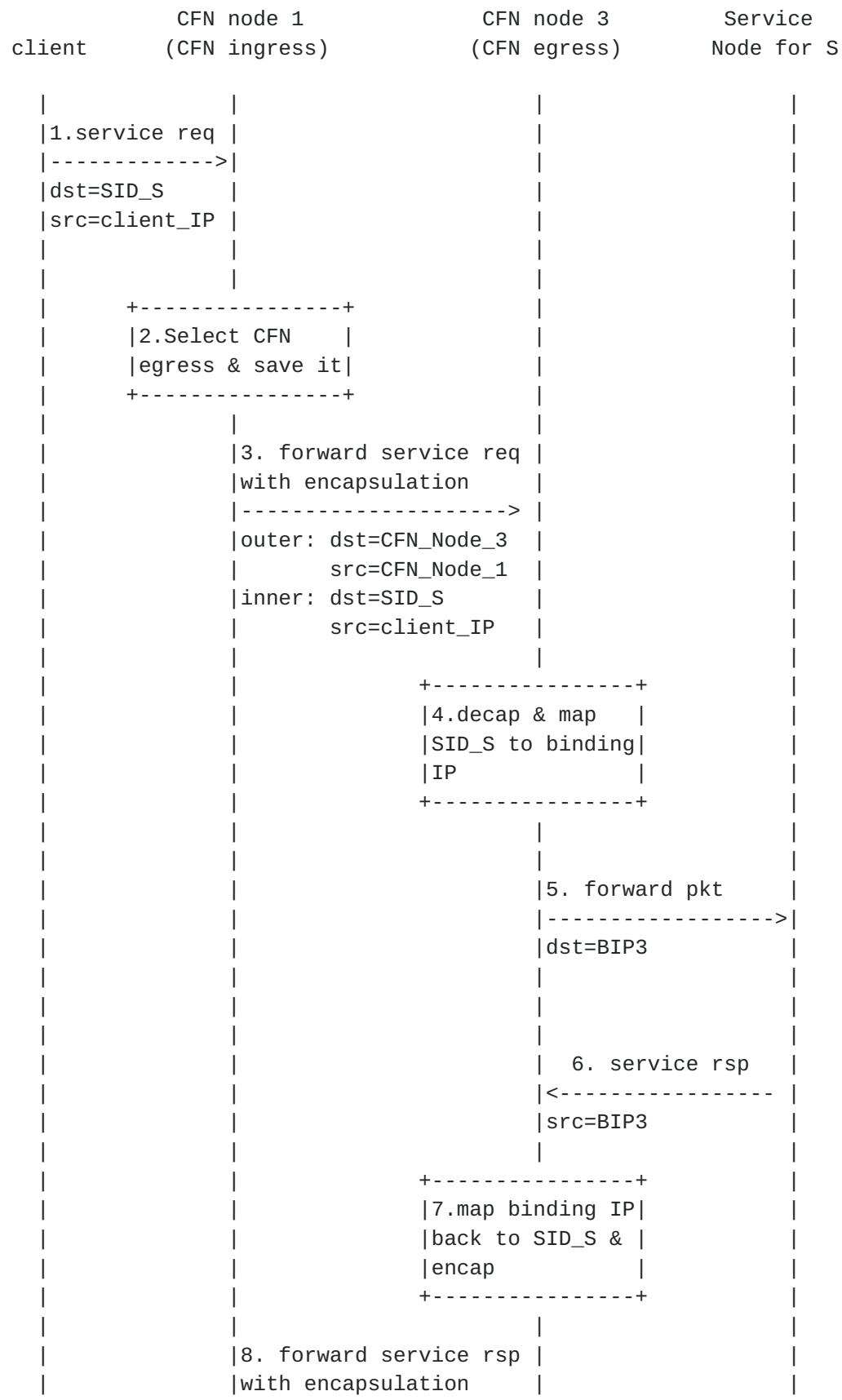
When a client sends a service request for service S, it uses SID_S as destination IP. In the test, SID_S is an anycast address. There are various ways that a client can get the SID_S for a service, such as by DNS or static configuration.

When the CFN ingress which is CFN node 1 in figure 3 receives the request, it dynamically selects the most appropriate CFN egress based on computing load information received. As figure 4 shows, CFN node 3 is selected as CFN egress in this case. CFN ingress further tunnels the data packet to CFN egress.

When CFN egress receives the packet, it decapsulates the packet and maps the destination address from SID_S to binding IP BIP3. The service node for service S gets the packet and processes it. The

service response is returned back to CFN node 3. CFN node 3 is conceptually the gateway of attached service nodes for CFN services. It maps BIP3 to SID_S as source IP and then tunnels it to CFN node 1. CFN node 1 further decapsulates the packet and sends it to the client.

For the subsequent service request packets sent to CFN node 1 from the same flow, CFN node always uses CFN node 3 as the egress to ensure the flow affinity.


CFN node 1
(CFN ingress)
CFN node 3
(CFN egress)
Service
Node for S
client
1.service req
dst=SID_S
src=client_IP
2.Select CFN
egress & save it
3. forward service req
with encapsulation
outer: dst=CFN_Node_3
src=CFN_Node_1
inner: dst=SID_S
src=client_IP
4.decap & map
SID_S to binding
IP
5. forward pkt
dst=BIP3
6. service rsp
src=BIP3
7.map binding IP
back to SID_S &
encap
8. forward service rsp
with encapsulation

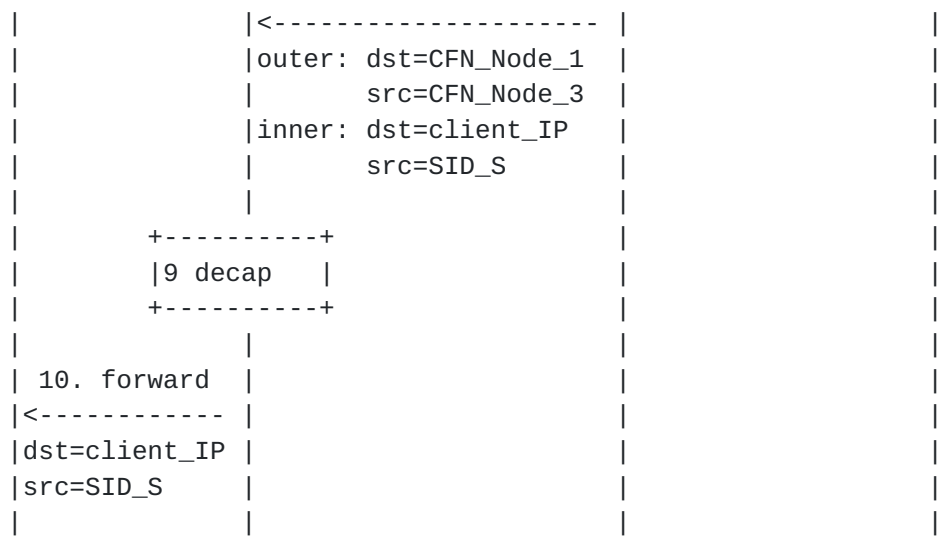



Figure 3. CFN data plane for the first request of a flow

## 4. Preliminary Tests

### 4.1 Requests rush to an edge (no system background load)

In this test, we assume the service nodes capacities attached to all three edges are the same and there is no background computing tasks running. The overall computing task handling capacity from service nodes can handle about 670 queries per second (qps).

The clients attached to edge 1 generating service request to it at about 40 qps. The number of clients simultaneously send requests varies. When 10 clients send requests, the computing power consumed by the system can reach approximately 60% of its overall maximum. The requests are all short-processing tasks and based on observation each request roughly take 4ms to be completed at the server side.

CFN leverages the computing load reported by different edges and together with network status to spread the service request. On the other hand, a pure random selection from the edges to handle the request is used for comparison.

We tested for 5, 10 and 15 clients attached to one edge which result in the consumption of medium low, medium high and high computing resources of the whole system respectively. Note it exceeds a single edge capacity in any case. For 15 clients case, it almost reaches the maximum system capacity. Figure 4 shows the average delay between a request being sent and the response being received by a client and

system qps.

| number of clients | system | average delay (ms) | qps |
|---|---|---|---|
| 5 (medium low) | CFN | 3.954 | 208.5 |
| | random | 5.316 | 197.7 |
| 10 (medium high) | CFN | 4.700 | 402.3 |
| | random | 5.595 | 302.1 |
| 15 (high) | CFN | 5.506 | 559.3 |
| | random | 5.718 | 546.0 |

Figure 4. Test results when service requests rush to a single edge when no system background load

The CFN achieves better results compared with random selection based application layer service dispatch. Average delay decreased by 25.62% and 16.00% and total qps increased by 5.5% and 33.17% in medium low and medium high computing load respectively. The unbalanced incoming traffic is spread to all edges. Unlike random selection, CFN will dispatch more requests to the local edge since its network cost is the lowest. CFN balances between higher computing resources available at the remote sites and lower network cost at the local site to make a choice. Hence it outperforms the random selection. In high number of clients case, as the maximum system capacity is almost reached, the performance are similar for CFN and random case.

## 4.2 Requests rush to an edge (system background load exists)

In this test, different edge has different background computing tasks to handle. We randomly select an edge to make it suffer from a computing intensive burst which consumes almost 90% of its capacity for about 4 seconds. Then computing load returns to zero for 2 seconds. It creates the busy edge and idle edges scenario. The other settings are same as shown in section 4.1.

Figure 5 shows the average delay between a request being sent and the

response being received by a client and system qps for this case.

| number of clients | system | average delay (ms) | qps |
|---|---|---|---|
| 5 (medium low) | CFN | 6.291 | 185.6 |
| | random | 9.630 | 165.3 |
| 10 (medium high) | CFN | 6.854 | 360.9 |
| | random | 10.592 | 316.3 |
| 15 (high) | CFN | 7.987 | 512.4 |
| | random | 12.156 | 441.7 |

Figure 5. Test results when service requests rush to a single edge when system background load exists

The results show that CFN has average delay decreased by 34.67%, 35.29% and 34.30% in medium low, medium high and high computing load respectively. And total qps is increased by 12.28%, 14.10% and 16.01% in medium low, medium high and high computing load respectively.

The performance gain of CFN shown in this test case is much higher than that in section 4.1 The reason is that the random service dispatching has more than 20% chance to send the request to an edge with service node with very high background computing load while CFN can greatly reduce such possibility.

In addition, compare with the results in section 4.1, delay increases 59.10%, 45.83% and 45.06% in different computing load level in CFN and 81.15%, 89.31%, 112.60% in random selection. It shows CFN can much better adapt to dynamic computing load change especially when system background load is high.

### 4.3 Mixed requests rush to an edge (no system background load)

We changed the characteristics of service requests to reflect the co-existence nature of long-processing tasks and short-processing tasks. Short-processing task takes roughly 4ms to complete and long-processing task takes roughly 400ms to complete. And the ratio of

long and short tasks is approximately 1:100.

Figure 6 shows the average delay between a request being sent and the response being received by a client and system qps for this case.

| number of clients | system | average delay (ms) | qps |
|---|---|---|---|
| 5 (medium low) | CFN | 5.205 | 193.5 |
| | random | 5.398 | 193.5 |
| 10 (medium high) | CFN | 5.201 | 393.4 |
| | random | 5.985 | 385 |
| 15 (high) | CFN | 6.147 | 559.4 |
| | random | 8.499 | 559.4 |

Figure 6. Test results when mixed service requests rush to a single edge when no system background load

The results show that CFN has average delay decreased by 3.58%, 13.10% and 27.76% in medium low, medium high and high computing load respectively. The qps has no much difference for different levels of computing load especially for the medium low and high case.

## 4.4 Impact from update frequency

The computing load information is updated and distributed when its metric changes exceed some threshold compared to the last distributed information. In the test, we used the 10% of maximum number of connections allowed and 5% CPU consumption as threshold. Frequency of update affects the system performance. We tested for different update interval to see their impact. The clients keep sending requests to make the computing resource consumption on each edge maintained at medium low which is about 5 connections. Update internal has been set to 10s, 5s, 1s, 100ms, 10ms, 1ms. Figure 7 shows the average delay between a request being sent and the response being received by a client under different update intervals and the improvement of delay when comparing to the case of 10 second interval.

The results shows that the higher frequency of updates distributed the better performance.

| # of clients | Interval | 10s | 5s | 1s | 100ms | 10ms | 1ms |
|---|---|---|---|---|---|---|---|
| 5 (medium low) | Delay(us) | 6445 | 6255 | 5741 | 5312 | 4883 | 4058 |
| | Improvement(%) | 0 | 3.5 | 12.3 | 21.3 | 32.3 | 58.8 |

Figure 7. Test results under different update intervals

## 5. Summary

This draft presents a field trial for CFN system with three edge sites in different locations. CFN enables a network-based fast-react system to serve multi-edge based computing service in a more balanced way. Computing load information are exchanged regularly between CFN nodes. CFN egress bound to serve a particular service is determined in real time and maintained to ensure flow affinity.

The tests show that the overall clients' request delay is greatly decreased and the system qps has some improvement too. CFN is a feasible and efficient way in edge computing to provide multi-edge service balancing.

## 6. Security Considerations

The security risks mentioned in [CFN-fmwk] apply in the tests. As a preliminary tests, no extra security risks control is implemented currently. Mechanisms such as authentication of edge node and fluctuation avoidance should be considered in deployment.

## 7. IANA Considerations

No IANA action is required.

## 8. Acknowledgements

The authors would like to thank Xunwen Li's team members for their help in setting up the testbed in Hangzhou.

## 9. References

### 9.1 Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

## 9.2 Informative References

[CFN-req] Geng, L., et al, "Compute First Networking (CFN) Scenarios and Requirements", draft-geng-cfn-req-00, November 2019.

[CFN-fmwk] Li, Y., et al, "Framework of Compute First Networking (CFN)", draft-li-cfn-framework-00, November 2019.

Authors' Addresses

Shuheng Gu
Huawei Technologies

EMail: gushuheng@huawei.com

Guanhua Zhuang
Huawei Technologies

EMail: zhuangguanhua@huawei.com

Huijuan Yao
China Mobile

EMail: yaohuijuan@chinamobile.com

Xunwen Li
China Mobile

EMail: lixunwen@zj.chinamobile.com