

rtgwg
Internet-Draft
Intended status: Informational
Expires: May 4, 2021

Y. Li
L. Iannone
Huawei Technologies
J. He
City University of Hong Kong
L. Geng
P. Liu
China Mobile
Y. Cui
Tsinghua University
October 31, 2020

Architecture of Dynamic-Anycast in Compute First Networking (CFN-
Dyncast)
draft-li-rtgwg-cfn-dyncast-architecture-00

Abstract

Compute First Networking (CFN) Dynamic Anycast refers to in-network edge computing, where a single service offered by a provider has multiple instances attached to multiple edge sites. In this scenario, flows are assigned and consistently forwarded to a specific instance through an anycast approach based on the network status as well as the status of the different instance.

This document describes an architecture for the Dynamic Anycast (Dyncast) in Compute First Networking (CFN). It provides an overview, a description of the various components, and a workflow example showing how to provide a balanced multi-edge based service in terms of both computing and networking resources through dynamic anycast in real time.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of [BCP 78](#) and [BCP 79](#).

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

Internet-Draft

CFN-dyncast Architecture

October 2020

This Internet-Draft will expire on May 4, 2021.

Copyright Notice

Copyright (c) 2020 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to [BCP 78](#) and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1.	Introduction	2
2.	Definition of Terms	3
3.	CFN-Dyncast Architecture Overview	4
4.	Architectural Components and Interactions	5
4.1.	Service Identity and Bindings	5
4.2.	Service Notification between Instances and CFN node . . .	7
4.3.	CFN Dyncast Control Plane	9
4.4.	Service Demand Dispatching	9
4.5.	CFN Dispatcher	10
5.	Summary of the key elements of CFN Dyncast Architecture . . .	12
6.	Conclusion (and call for contributions)	13
7.	Security Considerations	13
8.	IANA Considerations	13
9.	Informative References	14
	Acknowledgements	14
	Authors' Addresses	14

[1.](#) Introduction

Dynamic anycast in Compute First Networking (CFN-Dyncast) use cases and problem statements document

[[I-D.geng-rtgwg-cfn-dyncast-ps-usecase](#)] shows the usage scenarios that require an edge to be dynamically selected from multiple edge sites to serve an edge computing service demand based on computing

resource available at the site and network status in real time. Multiple edges provide service equivalency and service dynamism in CFN. The current network architecture in edge computing provides relatively static service dispatching, for example, to the closest edge, or to the server with the most computing resources without

considering the network status. Dynamic Anycast takes the dynamic nature of computing load as well as the network status as metrics for deciding flow's service dispatch and at the same time maintains the flow affinity in a service life cycle.

CFN-Dyncast architecture presents an anycast based service and access models. The aim is to solve the problematic aspects of existing network layer edge computing service deployment, including the unawareness of computing resource information of service, static edge selection, isolated network and computing metrics and/or slow refresh of status.

CFN-Dyncast assumes there are multiple equivalent edge instances implementing the same single service (think about the same service function instantiated on several edge nodes). A single edge node has limited computing resources attached, and different edge nodes may have different resources available such as CPU or GPU. Because multiple edge nodes are interconnected and can collaborate with each other, it is possible to balance the service load and network load in CFN. Computing resource available to serve a request is usually considered the main metric to assign a service demand to an instance of the service. However, the status of the network, in particular paths toward the instances, varies over time and may get congested, hence, becoming another key attribute to be considered. CFN-Dyncast aims at providing a layer 3 protocol framework able to dispatch the service demand to the "best" edge node in terms of both computing resources and network status, in real time and no application and/or service specific dependencies.

This document describes the a general architecture for the service notification, status update and service dispatch in CFN edge computing.

[2.](#) Definition of Terms

CFN: Compute First Networking

SID: Service ID, an anycast IP address representing a service and the clients use it to access that service. SID is independent of which service instance serves the service demand. Usually multiple service instances serve a single service.

BID: Binding ID, an address to reach a service instance for a given SID. It is usually a unicast IP. A service can be provided by multiple service instances with different BID.

CFN-Dyncast: as defined in [[I-D.geng-rtgwg-cfn-dyncast-ps-usecase](#)].

[3.](#) CFN-Dyncast Architecture Overview

Service instances can be hosted on servers, virtual machines, access routers or gateway in edge data center. The CFN node is the glue allowing CFN-Dyncast network to provide the capability to exchange the information about the computing resource information of service instances attached to it, but also to forward flows consistently toward such instances.

Figure 1 shows the architecture of CFN-Dyncast. CFN nodes are usually deployed at the edges of the operator infrastructure, where clients are connected. As such, we can consider that clients are logically connected to CFN nodes. A CFN node has the purpose to constantly direct flows coming from clients to an instance of the service the flow is supposed to go through. Service instances are initiated at different edge sites, where a CFN node is also running. A single service can have a huge number of instances running on different CFN nodes. A "Service ID" (SID) is used to uniquely identify a service, at the same time identifying the whole set of instances of that specific service, no matter where those instances are running. There can be several instances of the service running on the the same CFN node (e.g., one instance per CPU core), there can also be on several different CFN nodes (e.g., one instance per PGW-U in a 5G network). Each instance is associated to a "Binding ID" indicating where the instance is running. Hence, there is a dynamic binding between an SID (the service) and a set of BIDs (the instances of the service) and such bindings are enriched with information concerning the network state and the available resources so that at each new service request (a new flow) CFN nodes can decide which

instance is the most appropriate to handle the request. This highlights the anycast part of CFN-Dyncast, since flow are routed toward one service end-point among a set of equivalent , i.e., one-to-one-out-of-many.

When a clients sends a service demand, it will be delivered to the most appropriate instance of the service attached to a CFN node. A service demand is normally the first packet of a data flow, not necessarily an explicit out of band service request. Once the CFN node has decided which instance has to serve the flow, flow affinity must be guaranteed, meaning that all packets belonging to the same flow have to go through the same service instance.

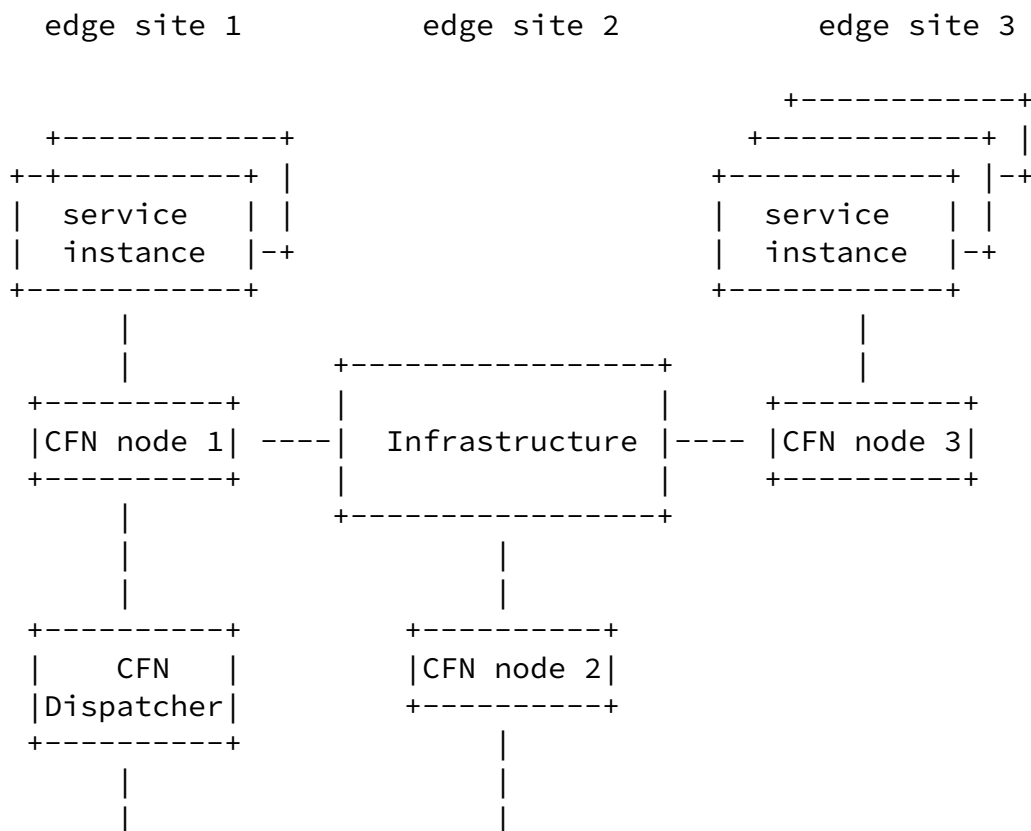




Figure 1: CFN-Dyncast Architecture

4. Architectural Components and Interactions

Figure 1 also shows that the local components of the architecture are service instance, CFN node, CFN dispatcher and client. The following subsections provide an overview of how some of these architectural components interact. The figures accompanying the examples do not show the interconnecting infrastructure to avoid making them too cluttered.

4.1. Service Identity and Bindings

As previously stated, the CFN-Dyncast architecture uses Service ID (SID) and Binding ID (BID) in order to identify services and their instances.

Service ID (SID) is an anycast service identifier (which may or may not be a routable IP address). It is used to access a specific service no matter which service instance eventually handles the

client's flow. CFN nodes must be able to know SIDs (and their bindings) in advance and must be able to identify which flow needs which service. This can be achieved in different ways, for example, use a special range or coding of anycast IP address as SID, or use DNS.

Binding ID (BID) is a unicast IP address. It is usually the interface IP address of a service instance. Mapping and binding from a SID to a BID is dynamic and depends on the computing resources and network state at the time the service demand is made. The CFN node must be able to guarantee flow affinity, i.e., steering the flow always toward the same instance.

Figure 2 shows an abstract example of the use of SIDs and BIDs. There are three services, namely SID1, SID2, and SID3. In

particular, SID2 has two instances on different CFN nodes (CFN node 2 and CFN node 3). In this case the complete list of bindings (only in term of SID and BID, no network or resource state) are:

- o SID1:BID21
- o SID2:BID22,BID32
- o SID3:BID33

SID: Service ID

BID: Binding ID

```

SID1
+-----+ service
+---| BID21 | instance1
```

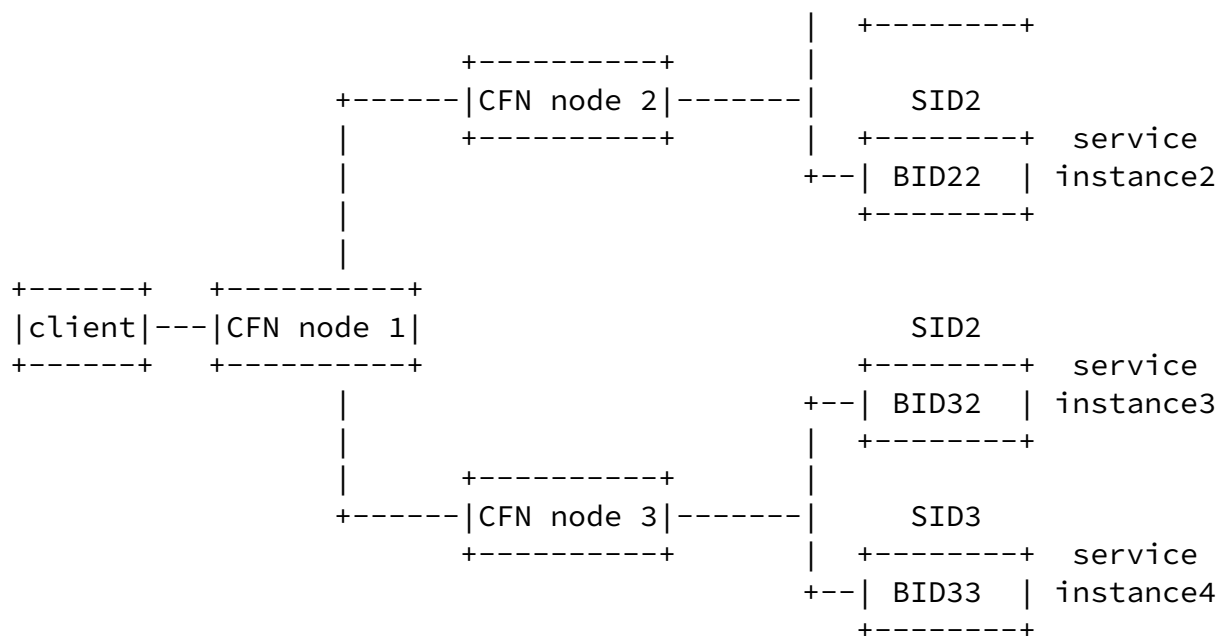


Figure 2: CFN-Dyncast Architectural Concept Example

4.2. Service Notification between Instances and CFN node

CFN-Dyncast service side is responsible to notify its attaching CFN node about the mapping information of SID and BID when a new service is instantiated, terminated, or its metrics (e.g., load) change, as shown in Figure 3.

BID: Binding ID

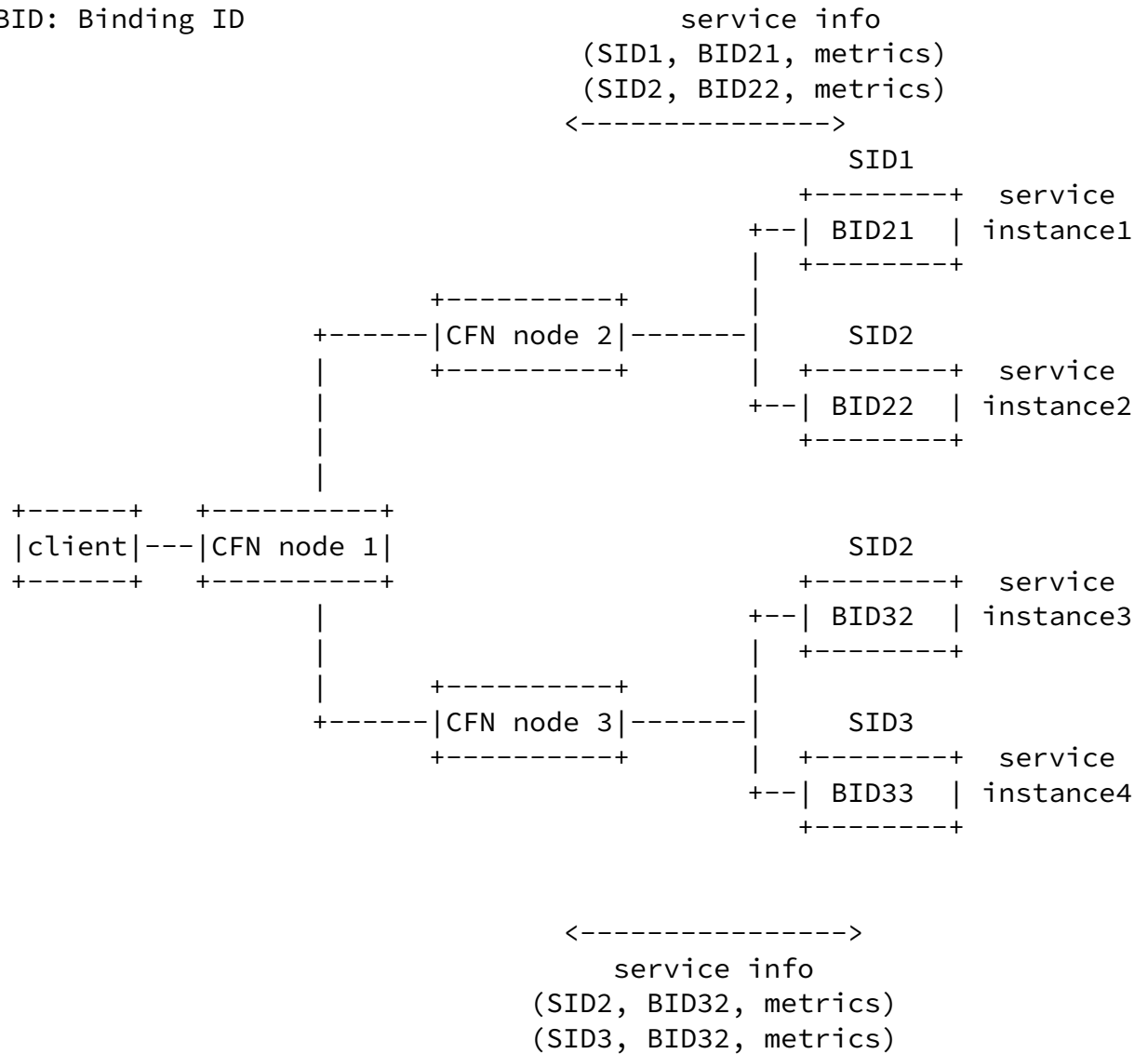


Figure 3: CFN-Dyncast Service Notification

Computing resource information of service instances is key information in CFN-Dyncast. Some of them are relatively static like CPU/GPU capacity, and some are very dynamic, for example, CPU/GPU utilization, number of sessions associated, number of queuing requests. The service side has to notify and refresh this information to its attaching CFN node. Various ways can be used, for instance via protocol or via an API of the management system. Conceptually, a CFN node keeps track of the SIDs and computing metrics of all service instances attached to it in real-time.

[4.3.](#) CFN Dyncast Control Plane

CFN Dyncast needs a control plane allowing to share information about resources and costs. Through the control plane, CFN nodes share and update among themselves the service information and the associated computing metrics for the service instances attached to it. As a network node, CFN node also monitors the network state to other CFN nodes. In this way, each CFN node is able to aggregate the information and create a complete vision of the resources available and the cost to reach them. For instance, for the scenario in Figure 3, the different CFN nodes will learn that there exists two instances of SID2, each of which has a certain computational capacity expressed in the metrics. Different mechanisms can be used in updating the status, for instance, BGP [[RFC4760](#)], IGP or controller based mechanism.

An important question CFN Dyncast raises is on the different ways to represent the computing metrics. A single digitalized value calculated from weighted attributes like CPU/GPU consumption and/or number of sessions associated may be the easiest. However, it may not accurately reflect the computing resources of interest. Multi-dimensional variables may give finer information, however the structure and the algorithmic processing should be sufficiently general to accommodate different type of services (i.e., metrics).

A second important issue is related to the system stability and signaling overhead. As computing metrics may change very frequently, when and how frequent such information should be exchanged among CFN nodes should be determined. A spectrum of approaches can be employed, interval based update, threshold update, policy based update, etc.

[4.4.](#) Service Demand Dispatching

Assuming that the set of metric are well defined and that the update rate is tailored so to have a stable system, the CFN Dyncast data plane has the task to dispatch flows to the "best" service instance. When a new flow comes to a CFN ingress, CFN ingress node selects the most appropriate CFN egress in terms of the network status and the computing resources of the attached service instances and guarantees flow affinity for the flow from now on.

Flow affinity is one of the critical features that CFN-Dyncast should support. The flow affinity means the packets from the same flow for a service should always be sent to the same CFN egress to be processed by the same service instance.

At the time that the most appropriate CFN egress and service instance is determined when a new flow comes, a flow binding table should save this flow binding information which may include flow identifier, selected CFN node, affinity timeout value, etc. The subsequent packets of the flow are forwarded based on the table. Figure 4 shows an example of what a flow binding table at CFN ingress node can look like.

Flow Identifier					CFN egress	timeout
src_IP	dst_IP	src_port	dst_port	proto		
X	SID2	-	8888	tcp	CFN node 2	xxx
Y	SID2	-	8888	tcp	CFN node 3	xxx

Figure 4: Example of flow binding table

A flow entry in the flow binding table can be identified using the classic 5-tuple value. However, it is worth noting that different services may have different granularity of flow identification. For instance, an RTP video streaming may use different port numbers for video and audio, and it may be identified as two flows if 5-tuple flow identifier is used. However they certainly should be treated as the same flow. Therefore 3-tuple based flow identifier is more suitable for this case. Hence, it is desired to provide certain level of flexibility in identifying flows in order to apply flow affinity.

Flow affinity attributes information can be configured per service in advance. For each service, the information can include the flow identifier type, affinity timeout value, etc. The flow identifier type can indicate what are the values, for instance, 5-tuple, 3-tuple or anything else that can be used as the flow identifier. Because we deal with single services the matching rules have to be disjoint, meaning that two different services need not have non-overlapping matching flow set.

[4.5.](#) CFN Dispatcher

When a CFN node maintains the flow binding table, the memory consumed is determined by the number of flows that CFN ingress node handles. The ingress node can be an edge data center gateway, hence it may cover hundreds of thousands of users and each user may have tens of flows. The memory space consumption on binding table at the CFN

ingress node can be a concern. To alleviate it, a functional entity called CFN Dispatcher can help.

CFN Dispatcher is deployed closer to the clients and it normally handles the flows for a limited number of clients. In this case, the memory space required by the binding table will be much smaller. CFN dispatcher is a client side located entity which directs traffic to an CFN egress node. It is not a CFN node itself, that is to say, it does not participate in the status update about network and computing metrics among CFN nodes. CFN dispatcher does not determine the best CFN egress to forward packets for a new flow by itself. It has to learn such information from a CFN node and maintains it to ensure the flow affinity for the subsequent packets. In this way, the CFN node simply selects the most appropriate egress for the new flows and informs CFN dispatcher in explicit or implicit way. It is relieved from flow binding table maintenance.

Figure 5 shows the interaction between an CFN Dispatcher and a CFN node. After CFN node makes the service demand dispatch, it informs the CFN dispatcher about the selected CFN egress node for the flow. Then CFN dispatcher maintains the flow binding table to ensure the flow affinity. Message exchange between the CFN dispatcher and its corresponding CFN node needs to be defined. The CFN dispatcher can simply forward the first packet of a flow to the CFN node, who takes the decision of which instance to use and pushes this information in the flow binding table of the CFN dispatcher. However, in case of failures, e.g., CFN egress not reachable anymore, further interaction is needed between the CFN dispatcher and the CFN node.

SID: Service ID

BID: Binding ID

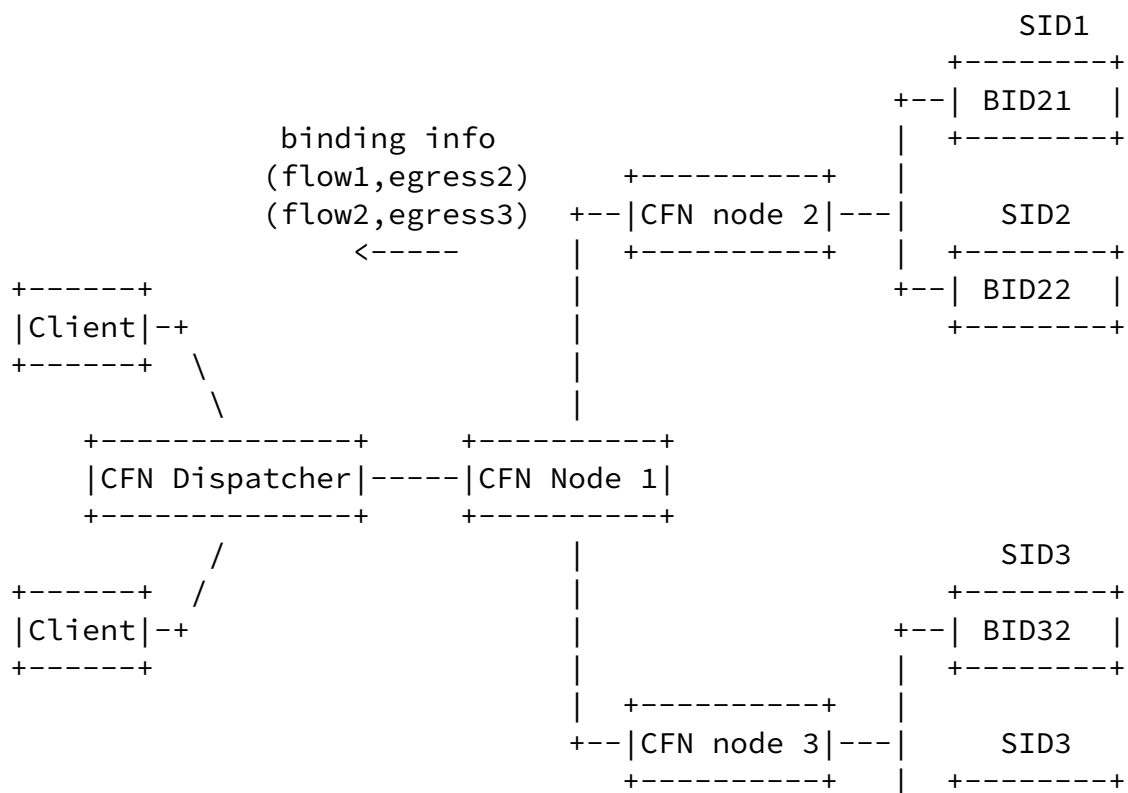


Figure 5: Service Demand Dispatch with CFN Dispatcher

5. Summary of the key elements of CFN Dyncast Architecture

o CFN Control Plane:

- * SID: CFN nodes have to made aware of existing services through the existence of the corresponding SID. It can be achieved in different ways. For example, use a special range or coding of anycast IP address as service IDs or use DNS.
- * BID bindings: SID are bound to a set of BID representing the different instances of the service. Associated to these BID there is as well a set of metrics describing the state of the instance. These bindings have to be shared among the CFN nodes so that they are aware of the different instances and their computing resource status.

- * Network state: CFN nodes have to be able to share network status so to have an idea on the impact of the dispatching decision in terms of link congestion.
- * Metric and network status updates need to be sufficiently sparse so to limit the signaling overhead and keep the system stable, but also sufficiently regular so to make the system reactive to sudden traffic fluctuations.

o CFN Data Plane:

- * In case of a new flow: CFN ingress node selects the most appropriate CFN egress in terms of the network status and the computing resource of the service instance attached to the egresses.
- * Flow affinity: CFN ingress nodes make sure the subsequent packets of an existing flow are always delivered to the same

CFN egress node so that they can be served by the same service instance.

6. Conclusion (and call for contributions)

This document introduces an architecture for CFN Dyncast, enabling the service demand request to be sent to an optimal edge to improve the overall system load balancing. It can dynamically adapt to the computing resources consumption and network status change and avoid overloading single edges. CFN-Dyncast is a network based architecture that supports a large number of edges and is independent of the applications or services hosted on the edge.

This present document is a strawman for defining CFN-Dyncast architecture.

More discussions on control plane and data plane approach are welcome.

7. Security Considerations

TBD

8. IANA Considerations

No IANA action is required so far.

9. Informative References

[RFC4760] Bates, T., Chandra, R., Katz, D., and Y. Rekhter, "Multiprotocol Extensions for BGP-4", [RFC 4760](#), DOI 10.17487/RFC4760, January 2007, <<https://www.rfc-editor.org/info/rfc4760>>.

[I-D.geng-rtgwg-cfn-dyncast-ps-usecase]
Geng, L., Liu, P., and P. Willis, "Dynamic-Anycast in Compute First Networking (CFN-Dyncast) Use Cases and Problem Statement", [draft-geng-rtgwg-cfn-dyncast-ps-](#)

Acknowledgements

TBD

Authors' Addresses

Yizhou Li
Huawei Technologies

Email: liyizhou@huawei.com

Luigi Iannone
Huawei Technologies

Email: Luigi.iannone@huawei.com

Jianfei He
City University of Hong Kong

Email: jianfeihe2-c@my.cityu.edu.hk

Liang Geng
China Mobile

Email: gengliang@chinamobile.com

Peng Liu
China Mobile

Email: liupengyjy@chinamobile.com

Yong Cui
Tsinghua University

Email: cuiyong@tsinghua.edu.cn

