Dynamic-Anycast Architecture
draft-li-dyncast-architecture-03

Abstract

This document describes a proposal for an architecture for the Dynamic-Anycast (Dyncast). It includes an architecture overview, main components that shall exist, and the workflow. An example of workflow is provided, focusing on the load-balance multi-edge based service use-case, where load is distributed in terms of both computing and networking resources through the dynamic anycast architecture.

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."

This Internet-Draft will expire on 8 September 2022.

Copyright Notice

Copyright (c) 2022 IETF Trust and the persons identified as the document authors. All rights reserved.
1. Introduction

Edge computing is expanding from a single edge nodes to multiple networked collaborating edge nodes to solve the issues like response time, resource optimization, and network efficiency.

The current network architecture in edge computing provides relatively static service dispatching, for example, to the closest edge from an IGP perspective, or to the server with the most computing resources without considering the network status, and even sometimes just based on static configuration.

Networking taking into account computing resource metrics seems to be an interesting paradigm that fits numbers of use-cases that would benefit from such capability [I-D.liu-dyncast-ps-usecases]. Yet, more investigation is still needed in key areas for this paradigm and, to this end, this document aims at providing an architectural framework, which will enable service notification, status update, and service dispatch in edge computing.
The Dyncast architecture presents an anycast based service and access model addressing 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.

Dyncast assumes that there are multiple equivalent service instances running on different edge nodes, globally providing (from a logical point of view) one single service. A single edge may have limited computing resources available, and different edges likely have different resources available, such as CPU or GPU. The main principle of Dyncast is that multiple edge nodes are interconnected and collaborate with each other to achieve a holistic objective, namely to dispatch service demands taking into account both service instances status as well as network state (e.g., paths length and their congestion). For this, computing resources available to serve a request is one of the top metrics to be considered. At the same time, the quality of the network path to an edge node may vary over time and may hence be another key attribute to be considered for said dispatching of service demands.

2. Definition of Terms

Dyncast: As defined in [I-D.liu-dyncast-ps-usecases], Dynamic Anycast, taking the dynamic nature of computing resource metrics into account to steer an anycast routing decision.

Service: As defined in [I-D.liu-dyncast-ps-usecases], a monolithic functionality that is provided by an endpoint according to the specification for said service. A composite service can be built by orchestrating monolithic services.

Service instance: As defined in [I-D.liu-dyncast-ps-usecases], running environment (e.g., a node) that makes the functionality of a service available. One service can have several instances running at different network locations.

D-Router: A node supporting Dyncast functionalities as described in this document. Namely it is able to understand both network-related and service-instances-related metrics, take forwarding decision based upon and maintain instance affinity, i.e., forwards packets belonging to the same service demand to the same instance.

D-MA: Dyncast Metric Agent (D-MA): A dyncast specific agent able to gather and send metric updates (from both network and instance perspective) but not performing forwarding decisions. May run on a D-Router, but it can be also implemented as a separate module (e.g., a software library) collocated with a service instance.
D-SID: Dyncast Service ID, an identifier representing a service, which the clients use to access said service. Such identifier identifies all of the instances of the same service, no matter on where they are actually running. D-SID is independent of which service instance serves the service demand. Usually multiple instances provide a (logically) single service, and service demands are dispatched to the different instance through an anycast model, i.e., choosing one instance among all available instances.

D-BID: Dyncast Binding ID, an address to reach a service instance for a given D-SID. It is usually a unicast IP where service instances are attached. Different service instances provide the same service identified through D-SID but with different Dyncast Binding IDs.

Service demand: The demand for a specific service and addressed to a specific D-SID.

Service request: The request for a specific service and addressed to a specific service instance identified with D-BID.

3. Architecture Main Concepts

Dyncast assumes that there are multiple equivalent service instances running on different edge sites, globally providing one single service which is represented by D-SID. The network will take forwarding decision for the service demand from the client according to both service instances status as well as network state.

The architecture of Dyncast has two typical modes, distributed or centralized.

* Distributed mode: The resources and status of the different service instances are propagated from the D-Routers connecting the edge sites where the service is deployed to the D-Routers with clients. In addition D-Routers have the network topology and status. The ingress D-Router which receives the service demand from the client decides independently which service instance to access according to the service instances status and network state and maintains instance affinity.

* Centralized mode: The resources and status of the different service instances are reported to the network controller from the D-Routers connecting the edge sites where the service is deployed. At the same time the controller collects the network topology and status. The controller makes routing decisions for each ingress D-Router according to the service instances status and network state and downloads the decisions to all the ingress D-Routers.
When the ingress D-Router receives the service demand from the client, it selects which service instance to access according to the decision made by the controller, and maintains the instance affinity subsequently.

This document mainly introduces the detailed process of the distributed mode, and the centralized mode will be introduced in detail in the future.

Edge sites (edges for short) are normally the sites where edge computing is performed. Service instances are initiated at different edge sites. Thus, a single service can actually have a significant number of instances running on different edges. A Dyncast Service ID (D-SID) is used to uniquely identify a service (e.g., a matrix computation for face recognition, or a game server). Service instances can be hosted on servers, virtual machines, access routers or gateway in edge data center.

Close to (one or more) Service instances is the Dyncast Metric Agent (D-MA). This element has the task to gather information about resources and status of the different instances as well as network-related information. Such element may also run in a dyncast-enable router (named D-Router), while other deployment scenarios may lead to this element running separately on edge nodes.

A D-Router is actually the main element in a Dyncast network, providing the capability to exchange the information about the computing resources information of service instances which have been gathered through D-MAs. A D-Router can also be a service access point for clients. When a service demand arrives, it will be delivered to the most appropriate service instance. A service demand may be the first packet of a data flow rather than an explicit out of band service request. This architectural document does not make any specific assumption on this matter. This documents only assumes that:

* D-Routers are able to identify new service demands. The Dyncast architecture presented in this document allows then to deliver such a packet to the most appropriate service instance according to information received from D-MAs and other D-Routers.

* D-Router are able to identify packets belonging to an existing service demand. The Dyncast architecture presented in this document allows to deliver these packets always to the same service instance selected at the initial service demand. We term this capability as ‘instance affinity’.
Note: As described above, D-Router can make routing decision based on per-service-instance computing-aware information. Actually, the D-Router can make the decision based on per-site computing-aware information. In this case, the egress D-Router can send the packet to the specific instance based on local policy, Load balancing, etc. This will be described in the future.

The element introduced above are depicted in Figure 1, which shows the proposed Dyncast architecture. In Figure 1, the "infrastructure" indicates the general IP infrastructure that does not necessarily need to support Dyncats, i.e., not all routers of the infrastructure need to be D-Routers.

![Figure 1: Dyncast Architecture.](image)

Figure 2 shows an example of Dyncast deployment, with 2 service instantiated twice (2 instances) on two different edges, namely edge site 2 and 3. Those service instances utilize different D-BIDs to serve service demands. D-Router 1 doesn’t connect the edge site
directly and needn’t collect the metric updates by D-MA. But it has
client to access and need to take forwarding decision for the client.
D-Router 2 gets metric updates by D-MA which runs on it. Edge site 2
has client present, so D-Router 2 need to take forwarding decision.
D-Router 3 gets metric updates from D-MA which is a separate software
module on edge computing platform in edge site 3. No client is
present at edge site 3, so D-Router 3 doesn’t need take forwarding
decision.

D-SID: Dyncast Service ID
D-BID: Dyncast Binding ID

Service/Metrics Information
(D-SID 1, D-BID 21, <metrics>)
(D-SID 2, D-BID 22, <metrics>)

----------

---------+----------
| Clients | D-SID 1 |
---------+----------

---------
| D-Router 2| D-MA |
---------+----------

---------
| D-Router 3| D-MA |
---------+----------

----------

---------
| Client | D-Router 1 |
---------+----------

----------

---------
| Edge 2 |
---------+----------

----------

---------
| Edge 3 |
---------+----------

----------

<D-SID 1, D-BID 31, <metrics>)
(D-SID 2, D-BID 32, <metrics>)

Service/Metrics Information
In Figure 2, the Dyncast Service ID (D-SID) follows an anycast semantic, such as provided through an IP anycast address. It is used to access a specific service no matter which service instance eventually handles the service demand of the client. Clients or other entities which want to access a service need to know about its D-SID in advance. It can be achieved in different ways, for example, using a special range of addresses associated to a certain service or coding of anycast IP address as D-SID, or using DNS.

The Dyncast Binding ID (D-BID) is a unicast IP address. It is usually the interface IP address through to reach a specific service instance. Mapping and binding a D-SID to a D-BID is dynamic and depends on the computing and network status at the time the service demand first arrives (see Section 4.1 for the reporting of such status). To ensure instance affinity, D-Routers are requested to remember the instance that has been selected (e.g., by storing the mapping) for delivering all packets to the same instance (see Section 4.2 for discussing this aspect).

4. Dyncast Architecture Workflow

The following subsections provide an overview of how the architectural elements introduced in the previous section do work together.

4.1. Service Notification/Metrics Update

When a service instance is instantiated/terminated the service information consisting in the mapping between the D-SID and the D-BID has to be updated/deleted as well. An update can also be triggered by a change in relevant metrics (e.g., an instance becomes overloaded). Computing resource information of service instance is key information in Dyncast. Some of them may be relatively static like CPU/GPU capacity, and some may be very dynamic, for example, CPU/GPU utilization, number of sessions associated, number of queuing requests. Changes in service-related relevant information has to be collected by D-MA associated to each service instance. Various ways can be used, for example, via routing protocols like EBGP or via an API of a management system. Conceptually a D-Router collects information coming from D-MA and keeps track of the IDs and computing metrics of all service instances.
Figure 2 shows an example of information shared by the Dyncast elements. The D-MA which is deployed with D-Router2 shares binding information concerning the two instances of the two services running on edge 2 (upper right hand side of the figure). These information is:

* (D-SID 1, D-BID 21, metrics)
* (D-SID 2, D-BID 22, metrics)

The D-MA which is deployed as a separate module on edge 3 (lower right hand side of the figure) shares binding information concerning the two instances of the two services running on edge 3. These information is:

* (D-SID 1, D-BID 31, metrics)
* (D-SID 2, D-BID 32, metrics)

Dyncast nodes share among themselves the service information including the associated computing metrics for the service instances attached to them. As a network node, a D-Router can also monitor the network cost or metrics (e.g., congestion) to reach other D-Routers. This is the focus of Dyncast control plane. Different mechanisms can be used to share such information, for instance BGP ([RFC4760]), an IGP, or a controller based mechanism. The specific mechanism is beyond the scope of this document. The architecture assumes that the Dyncast elements are able to share relevant information.

If, for instance, the client on the left hand side of Figure 2 sends a service demand for D-SID1, D-Router1 has the knowledge of the status of the service instance on both edge 2 and edge 3 and can make a decision toward which D-BID to forward the demand.

There are 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 used for simplicity reasons. However, it may not accurately reflect the computing resources of interest. Multi-dimensional values give finer information. This architectural document does not make any specific assumption about metrics and how to encode or even use them. As stated in Section 3, the only assumption is that a D-Router is able to use such metrics so to take a decision when a service demand arrives in order to map the demand onto a suitable service request.

As explained in the problem statement document [I-D.liu-dyncast-ps-usecases], computing metrics may change very frequently, when and how frequent such information should be
exchanged among Dyncats elements should be determined also in
accordance with the distribution protocol used for such purpose. A
spectrum of approaches can be employed, such as interval based
updates, threshold triggered updates, policy based updates, etc.

4.2. Service Demand Dispatch and Instance Affinity

This is the focus of the Dyncast data plane. When a new flow
(representing a service demand) arrives at a Dyncast ingress, such
ingress node selects the most appropriate egress according to the
network status and the computing resource of the attached service
instances.

Instance affinity is one of the key features that Dyncast should
support. It means that packets from the same 'flow' for a service
should always be sent to the same egress to be processed by the same
service instance. The affinity is determined at the time of newly
formulated service demand.

It is worth noting that different services may have different notions
of what constitutes a 'flow' and may thus identify a flow
differently. Typically a flow is identified by the 5-tuple value.
However, 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 by the same service instance. Therefore a 3-tuple based flow
identifier is more suitable for this case. Hence, it is desired to
provide certain level of flexibility in identifying flows, or from a
more general perspective, in identifying the set of packets for which
to apply instance affinity. More importantly, the means for
identifying a flow for the purpose of ensuring instance affinity must
be application-independent to avoid the need for service-specific
instance affinity methods.

Specifically, Instance affinity information should be configurable on
a per-service basis. For each service, the information can include
the flow/packets identification type and means, affinity timeout
value, and etc. For instance, the affinity configuration can
indicate what are the values, e.g., 5-tuple or 3-tuple, to be used as
the flow identifier.
When the most appropriate egress and service instance is determined when a new flow for a service demand arrives, a binding table should save this association between new service demand and service instance selection. The information in such binding table may include flow/packets identification, affinity timeout value, etc. The subsequent packets matching the entry are forwarded based on the table. Figure 3 shows a possible example of flow binding table at the ingress D-Router.

<table>
<thead>
<tr>
<th>Flow/Packets Identifier</th>
<th>D-BID egress</th>
<th>timeout</th>
</tr>
</thead>
<tbody>
<tr>
<td>src_IP</td>
<td>dst_IP</td>
<td>src_port</td>
</tr>
<tr>
<td>X</td>
<td>D-SID 2</td>
<td>-</td>
</tr>
<tr>
<td>Y</td>
<td>D-SID 2</td>
<td>-</td>
</tr>
</tbody>
</table>

Figure 3: Example of what a binding table can look like.

5. Dyncast Control-plane vs Data-plane operations

In summary, Dyncast consists of the following Control-plane and Data-plane operations:

* Dyncast Control Plane:
  
  - Dyncast Service ID Notification: the D-SID, an anycast IP address, should be available and known. This can be achieved in different ways. For example, use a special range or coding of anycast IP address as service IDs or using the DNS.
  
  - Dyncast Binding ID Notification: the mapping of (D-SID, D-BID), i.e., service ID and the binding address, should be notified to the D-Routers when the service instance starts (or stops). Various ways can be used, for example, EBGP or management system notification.
  
  - Metrics Notification: D-MA have to be able to share the metrics for a service and its binding ID so that D-Routers can select the "best" instance for each new service demand.

* Dyncast Data Plane:
  
  - New service demand: an ingress D-Router selects the most appropriate egress in terms of the network status and the computing resources of the instances of the requested service.
- Instance Affinity: Make sure the subsequent packets of an existing service demand are always delivered to the same service instance so that they can be served by the same service instance.

6. Summary

This draft introduces a Dyncast architecture that enables the service demand to be sent to an optimal service instance. It can dynamically adapt to the computing resources consumption and network status change. 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.

More discussion and input on control plane and data plane approach are welcome.

7. Security Considerations

The computing resource information changes over time very frequent with the creation and termination of service instance. When such information is carried in routing protocol, too many updates can make the network fluctuate. Control plane approach should take it into considerations.

More thorough security analysis to be provided in future revisions.

8. IANA Considerations

This document does not make any request to IANA.

9. Contributors

Huijuan Yao
yaohuijuan@chinamobile.com
China Mobile

Xia Chen
jescia.chenxia@huawei.com
Huawei
10. Informative References


Acknowledgements

TBD

Authors’ Addresses

Yizhou Li
Huawei Technologies
Email: liyizhou@huawei.com

Luigi Iannone
Huawei Technologies
Email: Luigi.iannone@huawei.com

Dirk Trossen
Huawei Technologies
Email: dirk.trossen@huawei.com

Peng Liu
China Mobile
Email: liupengyj@chinamobile.com

Cheng Li
Huawei Technologies
Email: c.l@huawei.com
Abstract

Many service providers have been exploring distributed computing techniques to achieve better service response time and optimized energy consumption. Such techniques rely upon the distribution of computing services and capabilities over many locations in the network, such as its edge, the metro region, virtualized central office, and other locations. In such a distributed computing environment, providing services by utilizing computing resources hosted in various computing facilities (e.g., edges) is being considered, e.g., for computationally intensive and delay sensitive services. Ideally, services should be computationally balanced using service-specific metrics instead of simply dispatching the service requests in a static way or optimizing solely connectivity metrics. For example, systematically directing end user-originated service requests to the geographically closest edge or some small computing units may lead to an unbalanced usage of computing resources, which may then degrade both the user experience and the overall service performance.

This document provides an overview of scenarios and problems associated with realizing such scenarios, identifying key engineering investigation areas which require adequate architectures and protocols to achieve balanced computing and networking resource utilization among facilities providing the services.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.
1. Introduction

Edge computing aims to provide better response times and transfer rates compared to Cloud Computing, by moving the computing towards the edge of a network. Edge computing can be built on embedded systems, gateways, and others, all being located close to end users' premises. There is an emerging requirement that multiple edge sites (called "edges", for for short) are deployed at different locations to provide a service. There are millions of home gateways, thousands of base stations, and hundreds of central offices in a city that can serve as candidate edges for behaving as service nodes. Depending on the location of an edge and its capacity, different computing resources can be contributed by each edge to deliver a service. At peak hours, computing resources attached to a client’s closest edge may not be sufficient to handle all the incoming service requests. Longer response times or even dropping of requests can be experienced by users. Increasing the computing resources hosted on each edge to the potential maximum capacity is neither feasible nor economically viable in many cases.

Some user devices are battery-dependent. Offloading computation intensive processing to the edge can save battery power. Moreover, the edge may use a data set (for the computation) that may not exist on the user device because of the size of data pool or due to data governance reasons.

At the same time, with new technologies such as serverless computing and container based virtual functions, the service node at an edge can be easily created and terminated in a sub-second scale, which in turn changes the availability of a computing resources for a service dramatically over time, therefore impacting the possibly "best" decision on where to send a service request from a client.

Traditional techniques to manage the overall load balancing process of clients issuing requests include choose-the-closest or round-robin. Those solutions are relatively static, which may cause an unbalanced distribution in terms of network load and computational load among available sources. For example, DNS-based load balancing usually configures a domain in the Domain Name System (DNS) such that client requests to that domain name are distributed across a group of servers. It usually provides several IP addresses for a domain name.

There are some dynamic solutions to distribute the requests to the server that best fits a service-specific metric, such as the best available resources and minimal load. They usually require Layer 4 - Layer 7 handling of the packet processing, such as through DNS-based or indirection servers. Such an approach is inefficient for large number of short connections. At the same time, such approaches can
Internet-Draft Dynamic-Anycast (Dyncast) Use Cases & Pr    March 2022

often not retrieve the desired metric, such as the network status, in real time. Therefore, the choice of the service node is almost entirely determined by the computing status, rather than the comprehensive considerations of both computing and network metrics or makes rather long-term decisions due to the (upper layer) overhead in the decision making itself.

Distributing service requests to a specific service having multiple instances attached to multiple edges, while taking into account computing as well as service-specific metrics in the distribution decision, is seen as a dynamic anycast (or "dyncast", for short) problem of sending service requests, without prescribing the use of a routing solution.

As a problem statement, this document describes sample usage scenarios as well as key areas in which current solutions lead to problems that ultimately affect the deployment (including the performance) of edge services. Those key areas target the identification of candidate solution components.

2. Definition of Terms

This document makes use of the following terms:

Service: A monolithic functionality that is provided by an endpoint according to the specification for said service. A composite service can be built by orchestrating monolithic services.

Service instance: Running environment (e.g., a node) that makes the functionality of a service available. One service can have several instances running at different network locations.

Service identifier: Used to uniquely identify a service, at the same time identifying the whole set of service instances that each represent the same service behavior, no matter where those service instances are running.

Anycast: An addressing and packet forwarding approach that assigns an "anycast" identifier for one or more service instances to which requests to an "anycast" identifier could be routed/forwarded, following the definition in[RFC4786] as anycast being "the practice of making a particular Service Address available in multiple, discrete, autonomous locations, such that datagrams sent are routed to one of several available locations".

Dyncast: Dynamic Anycast, taking the dynamic nature of computing resource metrics into account to steer an anycast-like decision in sending an incoming service request.
3. Sample Use Cases

This section presents a non-exhaustive list of scenarios which require multiple edge sites to interconnect and to coordinate at the network layer to meet the service requirements and ensure better user experience.

Before outlining the use cases, however, let us describe a basic model that we assume through which those use cases are being realized. This model justifies the choice of the terminology introduced in Section 2.

We assume that clients access one or more services with an objective to meet a desired user experience. Each participating service may be realized at one or more places in the network (called, service instances). Such service instances are instantiated and deployed as part of the overall service deployment process, e.g., using existing orchestration frameworks, within so-called edge sites, which in turn are reachable through a network infrastructure via an egress router.

When a client issues a service request to a required service, the request is being steered by its ingress router to one of the available service instances that realize the requested service. Each service instance may act as a client towards another service, thereby seeing its own outbound traffic steered to a suitable service instance of the request service and so on, achieving service composition and chaining as a result.

The aforementioned selection of one of candidate service instances is done using traffic steering methods, where the steering decision may take into account pre-planned policies (assignment of certain clients to certain service instances), realize shortest-path to the 'closest' service instance, or utilize more complex and possibly dynamic metric information, such as load of service instances, latencies experienced or similar, for a more dynamic selection of a suitable service instance.

It is important to note that clients may move throughout the execution of a service, which may, as a result, position other service instance 'better' in terms of latency, load, or other metrics. This creates a (physical) dynamicity that will need to be catered for.
Apart from the input into the traffic steering decision, under the aforementioned constraint of possible client mobility, its realization may differ in terms of the layer of the protocol stack at which the needed operations for the decision are implemented. Possible layers are application, transport, or network layers. Section 4 discusses some choice realization issues.

As a summary, Figure 1 outlines the main aspects of the assumed system model for realizing the use cases that follow next.

Figure 1: Dyncast Use Case Model

3.1. Cloud Virtual Reality (VR) or Augmented Reality (AR)

Cloud VR/AR services are used in some exhibitions, scenic spots, and celebration ceremonies. In the future, they might be used in more applications, such as industrial internet, medical industry, and metaverse.

Cloud VR/AR introduces the concept of cloud computing to the rendering of audiovisual assets in such applications. Here, the edge cloud helps encode/decode and render content. The end device usually
only uploads posture or control information to the edge and then VR/
AR contents are rendered in the edge cloud. The video and audio
outputs generated from the edge cloud are encoded, compressed, and
transmitted back to the end device or further transmitted to central
data center via high bandwidth networks.

Edge sites may use CPU or GPU for encode/decode. GPU usually has
better performance but CPU is simpler and more straightforward to use
as well as possibly more widespread in deployment. Available
remaining resources determines if a service instance can be started.
The instance’s CPU, GPU and memory utilization has a high impact on
the processing delay on encoding, decoding and rendering. At the
same time, the network path quality to the edge site is a key for
user experience of quality of audio/ video and input command response
times.

A Cloud VR service, such as a mobile gaming service, brings
challenging requirements to both network and computing so that the
edge node to serve a service request has to be carefully selected to
make sure it has sufficient computing resource and good network path.
For example, for an entry-level Cloud VR (panoramic 8K 2D video) with
110-degree Field of View (FOV) transmission, the typical network
requirements are bandwidth 40Mbps, 20ms for motion-to-photon latency,
packet loss rate is 2.4E-5; the typical computing requirements are 8K
H.265 real-time decoding, 2K H.264 real-time encoding. We can
further divide the 20ms latency budget into (i) sensor sampling
delay, (ii) image/frame rendering delay, (iii) display refresh delay,
and (iv) network delay. With upcoming high display refresh rate
(e.g., 144Hz) and GPU resources being used for frame rendering, we
can expect an upper bound of roughly 5ms for the round-trip latency
in these scenarios, which is close to the frame rendering computing
delay.
Furthermore, specific techniques may be employed to divide the overall rendering into base assets that are common across a number of clients participating in the service, while the client-specific input data is being utilized to render additional assets. When being delivered to the client, those two assets are being combined into the overall content being consumed by the client. The requirements for sending the client input data as well as the requests for the base assets may be different in terms of which service instances may serve the request, where base assets may be served from any nearby service instance (since those base assets may be served without requiring cross-request state being maintained), while the client-specific input data is being processed by a stateful service instance that changes, if at all, only slowly over time due to the stickiness of the service that is being created by the client-specific data. Other splits of rendering and input tasks can be found in [TR22.874] for further reading.

When it comes to the service instances themselves, those may be instantiated on-demand, e.g., driven by network or client demand metrics, while resources may also be released, e.g., after an idle timeout, to free up resources for other services. Depending on the utilized node technologies, the lifetime of such "function as a service" may range from many minutes down to millisecond scale. Therefore computing resources across participating edges exhibit a distributed (in terms of locations) as well as dynamic (in terms of resource availability) nature. In order to achieve a satisfying service quality to end users, a service request will need to be sent to and served by an edge with sufficient computing resource and a good network path.

3.2. Intelligent Transportation

For the convenience of transportation, more video capture devices are required to be deployed as urban infrastructure, and the better video quality is also required to facilitate the content analysis. So, the transmission capacity of the network will need to be further increased, and the collected video data needs to be further processed, such as for pedestrian face recognition, vehicle moving track recognition, and prediction. This, in turn, also impacts the requirements for the video processing capacity of computing nodes.

In auxiliary driving scenarios, to help overcome the non-line-of-sight problem due to blind spot or obstacles, the edge node can collect comprehensive road and traffic information around the vehicle location and perform data processing, and then vehicles with high security risk can be warned accordingly, improving driving safety in complicated road conditions, like at intersections. This scenario is also called "Electronic Horizon", as explained in [HORITA]. For
instance, video image information captured by, e.g., an in-car, camera is transmitted to the nearest edge node for processing. The notion of sending the request to the "nearest" edge node is important for being able to collate the video information of "nearby" cars, using, for instance, relative location information. Furthermore, data privacy may lead to the requirement to process the data as close to the source as possible to limit data spread across too many network components in the network.

Nevertheless, load at specific "closest" nodes may greatly vary, leading to the possibility for the closest edge node becoming overloaded, leading to a higher response time and therefore a delay in responding to the auxiliary driving request with the possibility of traffic delays or even traffic accidents occurring as a result. Hence, in such cases, delay-insensitive services such as in-vehicle entertainment should be dispatched to other light loaded nodes instead of local edge nodes, so that the delay-sensitive service is preferentially processed locally to ensure the service availability and user experience.

In video recognition scenarios, when the number of waiting people and vehicles increases, more computing resources are needed to process the video content. For rush hour traffic congestion and weekend personnel flow from the edge of a city to the city center, efficient network and computing capacity scheduling is also required. Those would cause the overload of the nearest edge sites if there is no extra method used, and some of the service request flow might be steered to others edge site except the nearest one.

3.3. Digital Twin

A number of industry associations, such as the Industrial Digital Twin Association or the Digital Twin Consortium (https://www.digitaltwinconsortium.org/), have been founded to promote the concept of the Digital Twin (DT) for a number of use case areas, such as smart cities, transportation, industrial control, among others. The core concept of the DT is the "administrative shell" [Industry4.0], which serves as a digital representation of the information and technical functionality pertaining to the "assets" (such as an industrial machinery, a transportation vehicle, an object in a smart city or others) that is intended to be managed, controlled, and actuated.

As an example for industrial control, the programmable logic controller (PLC) may be virtualized and the functionality aggregated across a number of physical assets into a single administrative shell for the purpose of managing those assets. PLCs may be virtualized in order to move the PLC capabilities from the physical assets to the
edge cloud. Several PLC instances may exist to enable load balancing and fail-over capabilities, while also enabling physical mobility of the asset and the connection to a suitable "nearby" PLC instance. With this, traffic dynamicity may be similar to that observed in the connected car scenario in the previous sub-section. Crucial here is high availability and bounded latency since a failure of the (overall) PLC functionality may lead to a production line stop, while boundary violations of the latency may lead to losing synchronization with other processes and, ultimately, to production faults, tool failures or similar.

Particular attention in Digital Twin scenarios is given to the problem of data storage. Here, decentralization, not only driven by the scenario (such as outlined in the connected car scenario for cases of localized reasoning over data originating from driving vehicles) but also through proposed platform solutions, such as those in [GAIA-X], plays an important role. With decentralization, endpoint relations between client and (storage) service instances may frequently change as a result.

Digital twin for networks[I-D.zhou-nmrg-digitaltwin-network-concepts] has also been proposed recently. It is to introduce digital twin technology into the network to build a network system with physical network entities and virtual twins, which can be mapped in real time. The goal of digital twin network will be applied not only to industrial Internet, but also to operator network. When the network is large, it needs real-time scheduling ability, more efficient and accurate data collection and modeling, and promote the automation, intelligent operation and maintenance and upgrading of the network.

4. Problems in Existing Solutions

There are a number of problems that may occur when realizing the use cases listed in the previous section. This section suggests a classification for those problems to aid the possible identification of solution components for addressing them.

4.1. Dynamicity of Relations

The mapping from a service identifier to a specific service instance that may execute the service for a client usually happens through resolving the service identification into a specific IP address at which the service instance is reachable.
Application layer solutions can be foreseen, using an application server to resolve binding updates. While the viability of these solutions will generally depend on the additional latency that is being introduced by the resolution via said application server, frequencies down to changing relations every few (or indeed EVERY) service requests is seen as difficult to be viable.

Message brokers, however, could be used, dispatching incoming service requests from clients to a suitable service instance, where such dispatching could be controlled by service-specific metrics, such as computing load. The introduction of such brokers, however, may lead to adverse effects on efficiency, specifically when it comes to additional latencies due to the necessary communication with the broker; we discuss this problem separately in the next subsection.

DNS[RFC1035] realizes an ‘early binding’ to explicitly bind from the service identification to the network address before sending user data, so the client creates an ‘instance affinity’ for the service identifier that binds the client to the resolved service instance address, which could also realize the load balancing.

However, we can foresee scenarios in which such ‘instance affinity’ may change very frequently, possibly even at the level of each service request. One such driver may be frequently changing metrics for the decision making, such as latency and load of the involved service instance. Also client mobility creates a natural/physical dynamicity with the result that ‘better’ service instances may become available and, vice versa, previous assignments of the client to a service instance may be less optimal, leading to reduced performance, such as through increased latency.

DNS is not designed for this level of dynamicity. Updates to the mapping between service identifier to service instance address cannot be pushed quickly enough into the DNS that takes several minutes updates to propagate, and clients would need to frequently resolve the original binding. If try to DNS to meet this level of dynamicity, frequent resolving of the same service name would likely lead to an overload of the it. These issues are also discussed in Section 5.4 of [I-D.sarathchandra-coin-appcentres].

A solution that leaves the dispatching of service requests entirely to the client may be possible to achieve the needed dynamicity, but with the drawback that the individual destinations, i.e., the network identifiers for each service instance, must be known to the client for doing so. While this may be viable for certain applications, it cannot generally scale with a large number of clients. Furthermore, it may be undesirable for every client to know all available service instance identifiers, e.g., for reasons of not wanting to expose this...
information to clients from the perspective of the service provider but also, again, for scalability reasons if the number of service instances is very high.

Existing solutions exhibit limitations in providing dynamic 'instance affinity', those limitations being inherently linked to the design used for the mapping between the service identifier and the address of the service instance, particularly when relying on an indirection point in the form of a resolution or load balancing server. These limitations may lead to 'instance affinity' to last many requests or even for the entire session between the client and the service, which may be undesirable from the service provider perspective in terms of best balance requests across many service instances.

4.2. Efficiency

The use of external resolvers, such as application layer repositories in general, also affects the efficiency of the overall service request. Additional signaling is required between client and resolver, either through the application layer solution, which not only leads to more messaging but also to increased latency for the additional resolution. Accommodating smaller instance affinities increases this additional signaling but also the latencies experienced, overall impacting the efficiency of the overall service transaction.

As mentioned in the previous subsection, broker systems could be used to allow for dispatching service requests to different service instances at high dynamicity. However, the usage of such broker inevitably introduces 'path stretch' compared to the possible direct path between client and service instance, increasing the overall flow completion time.

Existing solutions may introduce additional latencies and inefficiencies in packet transmission due to the need for additional resolution steps or indirection points, and will lead to the accuracy problems to select the appropriate edge.

4.3. Complexity and Accuracy

As we can see from the discussion on efficiency in the previous subsection, the time when external resolvers collect the necessary information and deal with it to select the edge nodes, the network and computing resource status may change already. So any additional control decision on which service instance to choose for which incoming service request requires careful planning to keep potential inefficiencies, caused by additional latencies and path stretch, at a minimum. Additional control plane elements, such as brokers, are
usually neither well nor optimally placed in relation to the data path that the service request will ultimately traverse.

Existing solutions require careful planning for the placement of necessary control plane functions in relation to the resulting data plane traffic to improve the accuracy; a problem often intractable in scenarios of varying service demand.

4.4. Metric Exposure and Use

Some systems may use the geographical location, as deduced from IP prefix, to pick the closest edge. The issue here may be that edges may not be far apart in edge computing deployments, while it may also be hard to deduce geo-location from IP addresses. Furthermore, the geo-location may not be the key distinguishing metric to be considered, particularly if geographic co-location does not necessarily mean network topology co-location. Also, "closer geographically" does not consider the computing load of possible closer yet more loaded nodes, consequently leading to possibly worse performance for the end user.

Solutions may also perform 'health checks' on an infrequent base (>1s) to reflect the service node status and switch in fail-over situations. Health checks, however, inadequately reflect an overall computing status of a service instance. It may therefore not reflect at all the decision basis a suitable service instance, e.g., based on the number of ongoing sessions as an indicator of load. Infrequent checks may also be too coarse in granularity, e.g., for supporting mobility-induced dynamics such as the connected car scenario of Section 3.2.

Service brokers may use richer computing metrics (such as load) but may lack the necessary network metrics.

Existing solutions lack the necessary information to make the right decision on the selection of the suitable service instance due to the limited semantic or due to information not being exposed across boundaries between, e.g., service and network provider.

4.5. Security

Resolution systems opens up two vectors of attack, namely attacking the mapping system itself, as well as attacking the service instance directly after having been resolved. The latter is particularly an issue for a service provider who may deploy significant service infrastructure since the resolved IP addresses will enable the client to directly attack the service instance but also infer (over time) information about available service instances in the service
infrastructure with the possibility of even wider and coordinated Denial-of-Service (DoS) attacks.

Broker systems may prevent this ability by relying on a pure service identifier only for the client to broker communication, thereby hiding the direct communication to the service instance albeit at the expense of the additional latency and inefficiencies discussed in Section 4.1 and 4.2. DoS attacks here would be entirely limited to the broker system only since the service instance is hidden by the broker.

Existing solutions may expose control as well as data plane to the possibility of a distributed Denial-of-Service attack on the resolution system as well as service instance. Localizing the attack to the data plane ingress point would be desirable from the perspective of securing service request routing, which is not achieved by existing solutions.

4.6. Changes to Infrastructure

Dedicated resolution systems, such as the DNS or broker-based systems, require appropriate investments into their deployment. While the DNS is an inherent part of the Internet infrastructure, its inability to deal with the dynamicity in service instance relations, as discussed in Section 4.1, may either require significant changes to the DNS or the establishment of a separate infrastructure to support the needed dynamicity. In a manner, the efforts on Multi-Access Edge Computing [MEC], are proposing such additional infrastructure albeit not solely for solving the problem of suitably dispatching service requests to service instances (or application servers, as called in [MEC]).

Existing solutions may expose control as well as data plane to the possibility of a distributed Denial-of-Service attack on the resolution system as well as service instance. Localizing the attack to the data plane ingress point would be desirable from the perspective of securing service request routing, which is not achieved by existing solutions.

5. Conclusion

This document presents use cases in which we observe the demand for considering the dynamic nature of service requests in terms of requirements on the resources fulfilling them in the form of service instances. In addition, those very service instances may themselves be dynamic in availability and status, e.g., in terms of load or experienced latency.
As a consequence, the problem of satisfying service-specific metrics to allow for selecting the most suitable service instance among the pool of instances available to the service throughout the network is a challenge, with a number of observed problems in existing solutions. The use cases as well as the categorization of the observed problems may start the process of determining how they are best satisfied within the IETF protocol suite or through suitable extensions to that protocol suite.

6. Security Considerations

Section 4.5 discusses some security considerations.

7. IANA Considerations

This document does not make any IANA request.

8. Contributors

The following people have substantially contributed to this document:

Peter Willis
BT

9. Informative References


Acknowledgements

The author would like to thank Yizhou Li, Luigi IANNONE, Christian Jacquenet, Kehan Yao and Yuexia Fu for their valuable suggestions to this document.

Authors' Addresses

Peng Liu
China Mobile
Email: liupengyjy@chinamobile.com

Philip Eardley
British Telecom
Email: philip.eardley@bt.com

Dirk Trossen
Huawei Technologies

Dynamic-Anycast (Dyncast) Requirements
draft-liu-dyncast-reqs-02

Abstract

This draft provides requirements for an architecture addressing the problems outlined in the use case and problem statement draft for Dyncast[I-D.liu-dyncast-ps-usecases].

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."

This Internet-Draft will expire on 8 September 2022.

Copyright Notice

Copyright (c) 2022 IETF Trust and the persons identified as the document authors. All rights reserved.
1. Introduction

Computing service instances instantiated at multiple geographical edge sites are used to better realize an edge computing service in edge computing use cases, as shown in [I-D.liu-dyncast-ps-usecases]. To optimally deliver the service request to the most appropriate service instance is the fundamental requirement in such deployments. As shown in [I-D.liu-dyncast-ps-usecases], choosing the most appropriate service instance should take both, the computing resources available and the network path quality, into consideration. "Optimal" here additionally means the architecture and overall mechanism should be efficient, support high dynamism, while maintaining instance affinity, as shown in [I-D.liu-dyncast-ps-usecases].

This draft provides the requirements to realize the potential dynamic anycast architecture by alleviating the problems of existing solutions outlined in [I-D.liu-dyncast-ps-usecases].
2. Definition of Terms

Service: A monolithic functionality that is provided by an endpoint according to the specification for said service. A composite service can be built by orchestrating monolithic services.

Service instance: Running environment (e.g., a node) that makes the functionality of a service available. One service can have several instances running at different network locations.

Service identifier: Used to uniquely identify a service, at the same time identifying the whole set of service instances that each represent the same service behaviour, no matter where those service instances are running.

Anycast: An addressing and packet sending methodology that assign an "anycast" identifier for one or more service instances to which requests to an "anycast" identifier could be routed, following the definition in [RFC4786] as anycast being "the practice of making a particular Service Address available in multiple, discrete, autonomous locations, such that datagrams sent are routed to one of several available locations".

Dyncast: Dynamic Anycast, taking the dynamic nature of computing resource metrics into account to steer an anycast-like decision in sending an incoming service request.

3. Desirable System Characteristics and Requirements

In the following, we outline the desirable characteristics of a system to overcome the observed problems in [I-D.liu-dyncast-ps-usecases] for the realization of the use cases described in that document.

3.1. Anycast-based Service Addressing Methodology

A unique service identifier is used by all the service instances for a specific service no matter which edge it attaches to. An anycast like addressing and routing methodology among multiple edges makes sure the data packet can potentially reach any of the edges with the service instance attached. At the same time, each service instance has its own unicast address to be used by the attaching edge to access the service. Since a client will use the service identifier as the destination addressing, mapping of the service identifier to the unicast address will need to happen in-band, considering the metrics for selection to make this selection service-specific. From an addressing perspective, a desirable system for the realization of the use cases described in that document.
o MUST provide a discovery and mapping methodology for the in-band mapping of the service identifier (an anycast address) to a specific unicast address.

3.2. Instance Affinity

A routing relation between a client and a service exists not at the packet but at the service request level in the sense that one or more service requests, possibly consisting of one or many more routing-level packets, must be ensured to be sent to said service. Each service may be provided by one or more service instances, each providing equivalent service functionality to their respective clients, while those service instances may be deployed at different locations in the network. With that, the routing problem becomes one between the client and a selected service instance for at least the duration of the service-level request, but possibly more than just one request.

This relationship between the client and the chosen service instance is described as "instance affinity" in the following, where the "affinity" spans across the aforementioned one or more service requests. This impacts the routing decision to be taken in that the normal packet level communication, i.e., each packet is forwarded individually based on the forwarding table at the time, will need extending with the notion of instance affinity since otherwise individual packets may be sent to different places when the network status changes, possibly segmenting individual requests and breaking service-level semantics.

The nature of this affinity is highly dependent on the nature of the specific service. The minimal affinity of a single request represents a stateless service, where each service request may be responded to without any state being held at the service instance for fulfilling the request. Providing any necessary information/state in-band as part of the service request, e.g., in the form of a multi-form body in an HTTP request or through the URL provided as part of the request, is one way to achieve such stateless nature. Alternatively, the affinity to a particular service instance may span more than one request, as in our VR example in [I-D.liu-dyncast-ps-usecases], where previous client input is needed to render subsequent frames. Therefore, a desirable system

o MUST maintain "instance affinity" which MAY span one or more service requests, i.e., all the packets from the same flow MUST go to the same service instance.
3.3. Proper Runtime-state Granularity and Keeping

The instance affinity, as outlined in Section 3.2, requires a client and the chosen service instance to keep persistent relationship across one or more service requests. For a multi-request session, this determines that the mapping logic has to consistently pick up the same service instance. This type of affinity can be normally achieved by deploying a mapping device to keep in-place all the necessary states. However, a client, e.g., a mobile UE, has generally many applications running. If all, or majority, of the applications request the dyncast-like services, then the runtime states that need to be created and accordingly maintained would require high granularity. In the extreme scenario, this granular requirement could reach the level of per-UE per-APP per-(sub)flow with regard to a service instance.

Evidently, these fine-granular runtime states can potentially become heavy burden for network devices if they have to dynamically create and maintain them. On the other hand, it is not appropriate either to place the state-keeping task on clients themselves. Therefore, a desirable system

- MUST avoid keeping fine runtime-state granularity in network nodes in order to achieve instance affinity.
- MUST provide mechanism to free clients from maintaining granular runtime-states in order to achieve instance affinity.

3.4. Encoding Metrics

As outlined in the scenarios in [I-D.liu-dyncast-ps-usecases], metrics can have many different semantics, particularly if considered to be service-specific. Even the notion of a "computing load" metric may be computed in many different ways. What is crucial, however, is the representation and encoding of that metric when being conveyed to the routing fabric in order for the routing elements to act upon those metrics. Such representation may entail information on the semantics of the metric or it may be purely one or more semantic-free numerals. Agreement of the chosen representation among all service and network elements participating in the service-specific routing decision is important. Specifically, a desirable system

- MUST agree on the service-specific metrics and their representation between service elements in the participating edges in the network and network elements acting upon them.
3.5. Signaling Metrics

The aforementioned representation of metrics needs conveyance to the network elements that will need to act upon them. Depending on the service-specific decision logic, one or more metrics will need to be conveyed. Problems to be addressed here may be that of loop avoidance of any advertisement of metrics as well as the frequency of such conveyance and therefore the overall load that the signaling may add to the overall network traffic. While existing routing protocols may serve as a baseline for signaling metrics, other means to convey the metrics can equally be realized. Specifically, a desirable system

- MUST provide mechanisms to signal the metrics for using in routing decisions
- MUST realize means for rate control for signaling of metrics
- MUST implement mechanisms for loop avoidance in signaling metrics, when necessary

3.6. Using Metrics in Routing Decisions

Metrics being conveyed, as outlined in Section 3.4, in the agreed manner, as outlined in Section 3.3, will ultimately need suitable action in the routers of the network. Routing decisions can be manifold, possibly including (i) min or max over all metrics, (ii) extending previous action with a random or first choice when more than one min/max entry found, (iii) weighted round robin of all entries, among others. It is important for the proper work of the service-specific routing decision, that it is understood to both network and service provider, which action (out of a possible set of supported actions) is to be used for a particular set of metrics. Specifically, a desirable system

Further, different network nodes, e.g., routers, switches, etc., bear diversified capabilities even in the same routing domain, let alone in different administrative domains. So, the service-specific metrics that have been adopted by some nodes might not be supported by others, either due to technical reasons, administrative reasons, or something else. There could be some scenario that a node supporting service-specific metrics might prefer some type of metrics...
to others [3GPP-TR22.847], or, in another scenario, even not utilize any at all. Therefore, there must exist flexibility in term of metrics handling and routing decisions in a network.

- MUST specify a default action to be taken, if more than one action possible
- MUST allow a network node not supporting service-specific metrics to interoperate with the supporting ones, i.e., providing backward compatibility.
- SHOULD allow the prioritization of using the service-specific metrics when compared to the currently widely-used networking metrics, like bandwidth, delay, loss, etc.
- SHOULD enable other alternative actions to be taken. (1) Any solution MUST provide appropriate signaling of the desired action to the router. For this, the action MAY be signaled in combination with signaling the metric (see Section 3.4). (2) Any solution SHOULD allow associating the desired action to a specific service identifier.

3.7. Supporting Service Dynamism

Network cost in the current routing system usually does not change very frequently. However, computing load and service-specific metrics in general can be highly dynamic, e.g., changing rapidly with the number of sessions, CPU/GPU utilization and memory space. It has to be determined at what interval or events such information needs to be distributed among edges. More frequent distribution of more accurate synchronization may result in more overhead in terms of signaling.

Choosing the least path cost is the most common rule in routing. However, the logic does not work well when routing should be aware of service-specific metrics. Choosing the least computing load may result in oscillation. The least loaded edge can quickly be flooded by the huge number of new computing demands and soon become overloaded with tidal effects possibly following.
Generally, a single instance may have very dynamic resource availability over time in order to serve service requests. This availability may be affected by computing resource capability and load, network path quality, and others. The balancing mechanisms should adapt to the service dynamism quickly and seamlessly. With this, the relationship between a single client and the set of possible service instances may possibly be very dynamic in that one request that is being dispatched to instance A may be followed by a request that is being dispatched to instance B and so on, generally within the notion of the service-specific service affinity discussed before in Section 3.2. With this in mind, a desirable system

o MUST support the dynamics of metrics changing on, e.g., a per flow basis, without violating the metrics defined in the selection of the specific service instance, while taking into account the requirements for the signaling of metrics and routing decision (see Section 3.4 and 3.5).

4. Conclusion

This document presents high-level requirements for solutions to Dyncast, where the architecture should address how to distribute the resource information and how to assure instance affinity in an anycast based service addressing environment, while realizing appropriate routing actions to satisfy the metrics provided.

5. Security Considerations

TBD

6. IANA Considerations

No IANA action is required so far.

7. Contributors

The following people have substantially contributed to this document:

Peter Willis
BT

8. Informative References


Acknowledgements

The author would like to thank Yizhou Li, Luigi IANNONE and Geng Liang for their valuable suggestions to this document.

Authors’ Addresses

Peng Liu
China Mobile
Email: liupengyjy@chinamobile.com

Tianji Jiang
China Mobile
Email: jiangtianji@chinamobile.com

Philip Eardley
British Telecom
Email: philip.eardley@bt.com

Dirk Trossen
Huawei Technologies
Email: dirk.trossen@huawei.com

Cheng Li
Huawei Technologies
Email: c.l@huawei.com