dyncast
Internet-Draft
Intended status: Informational
Expires: August 1, 2021

P. Liu
China Mobile
P. Willis
BT
D. Trossen
Huawei

February 1, 2021

Dynamic-Anycast (Dyncast) Use Cases and Problem Statement
draft-liu-dyncast-ps-usecases-00

Abstract

Service providers are exploring the edge computing to achieve better response time, control over data and carbon energy saving by moving the computing services towards the edge of the network in 5G MEC (Multi-access Edge Computing) scenarios, virtualized central office, and others. Providing services by sharing computing resources from multiple edges is an emerging concept that is becoming more useful for computationally intensive tasks. Ideally, services should be computationally balanced using service-specific metrics instead of simply dispatching the service in a static way, e.g., to the geographically closest edge since this may cause unbalanced usage of computing resources at edges which further degrades user experience and system utilization. This draft provides an overview of scenarios and problems associated with realizing such scenarios.

The document identifies several key areas which require more investigations in terms of architecture and protocol to achieve balanced computing and networking resource utilization among edges providing the services.

This Internet-Draft will expire on July 22, 2021.

Table of Contents

1. Introduction 2
2. Definition of Terms 4
3. Use Cases 4
3.1. Cloud Virtual Reality (VR) or Augmented Reality (AR) . . . 4
3.2. Connected Car 6
3.3. Digital Twin 6
4. Shortcomings of Existing Solutions 7
5. Desirable System Characteristics and Requirements 9
5.1. Anycast-based Service Addressing Methodology 9
5.2. Instance Affinity 10
5.3. Encoding Metrics 10
5.4. Signaling Metrics 11
5.5. Using Metrics in Routing Decisions 11
5.6. Supporting Service Dynamism 12
6. Conclusion 13
7. Security Considerations 13
8. IANA Considerations 13
9. Informative References 13
Acknowledgements 14
Authors' Addresses 14

1. Introduction

Edge computing aims to provide better response times and transfer rate, with respect to Cloud Computing, by moving the computing towards the edge of the network. Edge computing can be built on industrial PCs, embedded systems, gateways and others, all being

located close to the end user. There is an emerging requirement that multiple edge sites (called "edges" too in this document) are deployed at different locations to provide the 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 hosting service nodes. Depending on the location of the edge and its capacity, each edge has different computing resources to be used for a service. At peak hour, computing resources attached to a client's closest edge site 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 site to the potential maximum capacity is neither feasible nor economical in many cases.

Some user devices are purely battery-driven. 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.

DNS-based load balancing usually configures a domain in Domain Name System (DNS) such that client requests to the domain are distributed across a group of servers. It usually provides several IP addresses for a domain name. 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.

There are some dynamic ways which attempt to distribute the request to the server that best fits a service-specific metric, such as the best available resources and minimal load. They usually require L4-L7 handling of the packet processing. It is not an efficient approach for a large number of short connections. At the same time, such approaches can 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 consideration of both computing and network metrics.

Distributing a service request to a specific service having multiple

instances attached to multiple edge computing sites, while taking into account computing as well as service-specific metrics in the distribution decision, can be seen as a dynamic anycast (or "dyncast" for short) routing problem. This draft describes usage scenarios, problem space and key areas of investigation for this dyncast problem.

2. Definition of Terms

Anycast: An addressing and routing methodology that assign an "anycast" address for one or more network locations to which requests to an "anycast" address could be routed.

Dyncast: Dynamic Anycast, taking the dynamic nature of computing resource metrics into account to steer an anycast routing decision.

Service: A service represents a defined endpoint of functionality encoded according to the specification for said service.

Service instance: One service can have several instances running on different nodes. Service instance is a running environment (e.g., a node) that makes the functionality of a service available.

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.

3. Use Cases

This section presents several typical scenarios which require multiple edge sites to interconnect and to co-ordinate at the network layer to meet the service requirements and ensure user experience. The scenarios here are exemplary only for the purpose of this document and not comprehensive.

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

Cloud VR/AR introduces the concept and technology 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.

Furthermore, techniques may be employed that 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. Connected Car

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.

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 advocate 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 loosing 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.

4. Shortcomings of Existing Solutions

Given that the current state of the art for routing is based on the network cost, computing resource and/or load information as well as other service-specific metrics are not available nor distributed at the network layer. At the same time, computing resource metrics are not well defined and understood by the network. Furthermore, although we have focused in our examples on computing load and networking latency, metrics that decide the selection of the most appropriate service instance may not be limited to those. Proximity, even of physical nature, as well as capabilities of computing and network resources may be other metrics used for the selection of an appropriate service instance, while a "computing metric" itself can include aspects such as CPU/GPU capacity and load, number of sessions currently serving, latency of service process expected, possibly applying weights to each metric. Overall, we observe that given the service-specific nature of the notion of "best instance", it is hard to make the best choice of the edge based on both computing and network metrics at the same time, leading to the problems observed in the following when realizing the use cases in Section 3.

As a key takeaway from Section 3, we observe that a service request should be dynamically routed to the most suitable service instance in real time among the multiple edges in which service instances have been deployed. Existing mechanisms use one or more of the following ways and each of them has issues associated.

- Use the least network cost as metric to select the edge. Issue: Computing information, among other metrics, is key to be considered in edge computing, and it is not included here. In our scenarios of Section 3, this may lead to service requests routed to closer albeit possibly overloaded edge-based service instances, degrading the service quality.

- Use of geographical location, as deduced from IP prefix, is used to pick the closest edge. Issue: Edges are not so far apart in edge computing scenario. Either hard to be deduced from IP address or the location is not the key distinguishing metric to be considered., particularly since geographic co-location does not necessarily mean network topology co-location. Furthermore, "closer geographically" does not consider the computing load of possible closer yet more loaded nodes, similar to the previous point.

- Health check on an infrequent base (>1s) to reflect the service node status, and switch when fail-over. Issue: Health check is very different from computing status information of service instance and may not reflect at all the decision basis for the scenarios, e.g., the number of ongoing sessions as an indicator of load. It may also be too coarse in granularity, e.g., for supporting mobility-induced dynamics such as the connected car scenario of Section 3.2.

- Application layer randomly picks or uses round-robin mechanism to pick a service instance. Issue: It may share the load across multiple service instances in terms of the computing capacity all while assuming equal resource capability for each service instance, the network cost variance is barely considered. Edges can be deployed in different cities which are not equal cost paths to a client. Therefore network status is also a major concern. Also, in our scenarios of Section 3, the choice of "nearest" or "closer" is required for a better choice.

- Global resolver and early binding (DNS-based load balancing): Client queries a global resolver or load balancer first and gets the exact server's address. And then steer traffic using that address as binding address. It is called early binding because an explicit binding address query has to be performed before sending user data. Issue: Firstly, it clashes with the service dynamism

across all scenarios in Section 3. Also, the resolver does not have the capability of such high frequent change of indirection to new instance based on the frequent change of each service instance. Secondly, edge computing flow can be short. One or two round trips would be completed, requiring very frequent bindings with accompanying DNS resolutions, while an out-of-band query for specific server address has high overhead as it takes one more round trip. Lastly, to avoid DNS resolution for every request, out-of-band signaling would also be required to the client to remove any previous DNS resolution from the client-local DNS cache to avoid the use of stale DNS entries. These issues are also discussed in section 5.4 of [I-D.sarathchandra-coin-appcentres], outlining the significant challenges for the flexible re-routing to appropriate service instances out of an available pool when utilizing DNS for this purpose.

- Traditional anycast. Issue: Only works for single request/reply communication. No instance affinity (see Section 5.2) guaranteed nor is any service-specific metric, such as load and latency, being considered. This would cause significant issues in all our use cases in Section 3

- EIGRP [RFC7868]: Although allowing a number of metrics, EIGRP has no notion of computing load as a metric, while it also does not enforce instance affinity, which may lead to problems in our use cases of Section 3 in service requests being sent mid-request to other service instances.

5. Desirable System Characteristics and Requirements

In the following, we outline the desirable characteristics of a system to overcome the observed problems in Section 4 for the realization of the use cases in Section 3.

5.1. 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

- MUST provide a discovery and mapping methodology for the in-band mapping of the service identifier (an anycast address) to a specific unicast address.

5.2. 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 Section 3.1, where previous client input is needed to render subsequent frames. Therefore, a desirable system

- 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.

5.3. Encoding Metrics

As outlined in our scenarios in Section 3, 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.

- MAY obfuscate the specific semantic of the metric to preserve privacy of the service provider information towards the network provider.

- MAY include routing protocol metrics

5.4. 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

5.5. Using Metrics in Routing Decisions

Metrics being conveyed, as outlined in Section 5.4, in the agreed manner, as outlined in Section 5.3, 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

- MUST specify a default action to be taken, if more than one action possible
- SHOULD enable other alternative actions to be taken.
    - 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 5.4).
    - Any solution SHOULD allow associating the desired action to a specific service identifier.

5.6. 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 5.2. With this in mind, a desirable system

- 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 5.4 and 5.5).

6. Conclusion

This document presents use cases in which we observe the demand for consideration of the dynamic nature of service requests as well as the availability of network resources so as to satisfy service-specific metrics to allow for selecting the most suitable service instance among the pool of instances available to the service throughout the network. These use cases and the observed problems with existing solutions motivate the outline for a desirable system that may provide a solution for realizing the use cases outlined in this document; we call this system Dyncast due to its anycast-based addressing methodology.

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

7. Security Considerations

TBD

8. IANA Considerations

No IANA action is required so far.

9. Informative References

[RFC7868] D. Davage et al. , "Cisco's Enhanced Interior Gateway Routing Protocol (EIGRP)", RFC 7868, May 2016, https://tools.ietf.org/html/rfc7868

[I-D.sarathchandra-coin-appcentres] Trossen, D., Sarathchandra, C., and M. Boniface, "In-Network Computing for App-Centric Micro-Services", draft-sarathchandra-coin-appcentres-03 (work in progress), October 2020.

[TR22.874] 3GPP, "Study on traffic characteristics and performance requirements for AI/ML model transfer in 5GS (Release 18)", TR 22.874 V0.2.0, November 2020

[Industry4.0] Industry4.0, "Details of the Asset Administration Shell, Part 1 & Part 2", November 2020, https://www.plattform-i40.de/PI40/Redaktion/EN/Standardartikel/specification-administrationshell.html.

[GAIA-X] Gaia-X, "GAIA-X: A Federated Data Infrastructure for Europe", accessed January 2021, https://www.data-infrastructure.eu/GAIAX/Navigation/EN/Home/home.html

[HORITA] Y. Horita et al., "Extended electronic horizon for automated driving", Proceedings of 14th International Conference on ITS Telecommunications (ITST), 2015

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

Peter Willis
BT
Email: peter.j.willis@bt.com

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