

cats
Internet-Draft
Intended status: Informational
Expires: 29 June 2026

K. Yao
China Mobile
L. M. Contreras
Telefonica
H. Shi
Huawei Technologies
S. Zhang
China Unicom
Q. An
Alibaba Group
26 December 2025

Computing-Aware Traffic Steering (CATS) Problem Statement, Use Cases,
and Requirements
draft-ietf-cats-usecases-requirements-11

Abstract

Distributed computing is a computing pattern that service providers can follow and use to achieve better service response time and optimized energy consumption. In such a distributed computing environment, compute intensive and delay sensitive services can be improved by utilizing computing resources hosted in various computing facilities. Ideally, compute services are balanced across servers and network resources to enable higher throughput and lower response time. To achieve this, the choice of server and network resources should consider metrics that are oriented towards compute capabilities and resources instead of simply dispatching the service requests in a static way or optimizing solely on connectivity metrics. The process of selecting servers or service instance locations, and of directing traffic to them on chosen network resources is called "Computing-Aware Traffic Steering" (CATS).

This document provides the problem statement and the typical scenarios for CATS, which shows the necessity of considering more factors when steering traffic to the appropriate computing resource to better meet the customer's expectations.

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 29 June 2026.

Copyright Notice

Copyright (c) 2025 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1. Introduction	3
2. Definition of Terms	4
3. Problem Statement	4
3.1. Multi-deployment of Edge Sites and Service	4
3.2. Traffic Steering among Edges Sites and Service Instances	6
3.3. Discussions on Common Issues	9
4. Use Cases	10
4.1. Example 1: Computing-aware AR or VR	10
4.2. Example 2: Computing-aware Intelligent Transportation . .	13
4.3. Example 3: Computing-aware Digital Twin	15
4.4. Example 4: Computing-aware SD-WAN	15
4.5. Example 5: Computing-aware Distributed AI Training and Inference	17
4.5.1. Distributed AI Inference	17
4.5.2. Distributed AI Training	18
4.6. Discussion	20
5. Requirements	20
5.1. Support Dynamic and Effective Selection among Multiple Service Instances	20
5.2. Support Agreement on Metric Representation and Definition	21
5.3. Use of CATS Metrics	22
5.4. Support Instance Affinity	23
5.5. Preserve Communication Confidentiality	25

5.6. Correlation between Use Cases and Requirements	25
6. Security Considerations	27
7. IANA Considerations	28
8. References	28
8.1. Normative References	28
8.2. Informative References	28
Appendix A. Appendix A	29
A.1. Integrated Sensing and Communications (ISAC)	30
A.1.1. Requirements	32
Acknowledgements	32
Contributors	32
Authors' Addresses	34

1. Introduction

Network operators are extending their edge capabilities by deploying substantial computing resources, enabling support for services that require low latency, high reliability, and dynamic resource scaling. There are millions of home gateways, thousands of base stations, and hundreds of central offices in a city that could serve as compute-capable nodes to deliver a service. Note that not all of these nodes would be considered as edge nodes in some views of the network, but they can all provide computing resources for services.

It brings some key problems on service deployment and traffic scheduling to the most suitable computing resource in order to meet users' demands.

A service instance deployed at a single site might not provide sufficient capacity to fully guarantee the quality of service required by a customer. Especially at peak hours, computing resources at a single site can not handle all the incoming service requests, leading to longer response time or even dropping of requests experienced by clients. Moreover, increasing the computing resources hosted at each location to the potential maximum capacity is neither feasible nor economically viable in many cases. Offloading compute intensive processing to the user devices is not acceptable, since it would place pressure on local resources such as the battery and incur some data privacy issues if the needed data for computation is not provided locally.

Instead, the same service can be deployed at multiple sites for better availability and scalability. Furthermore, it is desirable to balance the load across all service instances to improve throughput. For this, traffic needs to be steered to the 'best' service instance according to information that may include current computing load, where the notion of 'best' may highly depend on the application demands.

This document describes sample usage scenarios that drive CATS requirements and will help to identify candidate solution architectures and solutions.

2. Definition of Terms

This document uses the terms defined in [I-D.ietf-cats-framework].

Service identifier: An identifier representing a service, which the clients use to access it.

Network Edge: The network edge is an architectural demarcation point used to identify physical locations where the corporate network connects to third-party networks.

Edge Computing: Edge computing is a computing pattern that moves computing infrastructures, i.e, servers, away from centralized data centers and instead places it close to the end users for low latency communication.

Relations with network edge: edge computing infrastructures connect to corporate network through a network edge entry/exit point.

Even though this document is not a protocol specification, it makes use of upper case key words to define requirements unambiguously.

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

3. Problem Statement

3.1. Multi-deployment of Edge Sites and Service

Since edge computing aims at a closer computing service based on the shorter network path, there will be more than one edge site with the same application in the city/province/state, a number of representative cities have deployed multi-edge sites and the typical applications, and there are more edge sites to be deployed in the future. Before deploying edge sites, there are some factors that need to be considered, such as:

- * The existing infrastructure capacities, which could be updated to edge sites, e.g. operators' machine room.
- * The amount and frequency of computing resource that is needed.

- * The network resource status linked to computing resource.

To improve the effectiveness of service deployment, the problem of how to choose the optimal edge node on which to deploy services needs to be solved. [I-D.contreras-alto-service-edge] introduces considerations for how to deploy applications or functions to the edge, such as the type of instance, optional storage extension, optional hardware acceleration characteristics, and the compute flavor of CPU/GPU, etc. More network and service factors may also be considered, such as:

- * Network and computing resource topology: The overall consideration of network access, connectivity, path protection or redundancy, and the location and overall distribution of computing resources in the network, and the relative position within the network topology.
- * Location: The number of users, the differentiation of service types, and the number of connections requested by users, etc. For edge nodes located in populous area with a large number of users and service requests, service duplication could be deployed more than in other areas.
- * Capacity of multiple edge nodes: Not only the capacity of a single node, but also the total number of requests that can be processed by the resource pool composed of multiple nodes.
- * Service category: For example, whether the service is a multi-user interaction, such as video conferencing, or games, or whether it just resource acquisition, such as video viewing. ALTO [RFC7285] can help to obtain one or more of the above pieces of information, so as to provide suggestions or formulate principles and strategies for service deployment.

This information could be collected periodically, and could record the total consumption of computing resources, or the total number of sessions accessed. This would indicate whether additional service instances need to be deployed. Unlike the scheduling of service requests, service deployment should follow the principle of proximity to place new service instances near to customer sites that will request them. If the resources are insufficient to support new instances, the operator can be informed to increase the hardware resources.

In general, the choice of where to locate service instances and when to create new ones in order to provide the right levels of resource to support user demands is important in building a network that supports computing services. However, those aspects are out of scope for CATS and are left for consideration in another document.

3.2. Traffic Steering among Edges Sites and Service Instances

This section describes how existing edge computing systems do not provide all of the support needed for real-time or near-real-time services, and how it is necessary to steer traffic to different sites considering mobility of people, different time slots, events, server loads, network capabilities, and some other factors which might not be directly measured, i.e., properties of edge sites(e.g., geographical location), etc.

In edge computing, the computing resources and network resources are considered when deploying edge sites and services. Traffic is steered to an edge site that is "closest" or to one of a few "close" sites using load-balancing. But the "closest" site is not always the "best" as the status of computing resources and of the network may vary as follows:

- * Closest site may not have enough resource, the load may dynamically change.
- * Closest site may not have related resource, heterogeneous hardware in different sites.
- * The network path to the closest site might not provide the necessary network characteristics, such as low latency or high throughput.

To address these issues some enhancements are needed to steer traffic to sites that can support the requested services.

We assume that clients access one or more services with an objective to meet a desired user experience. Each participating service may be deployed 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 edge router.

When a client issues a service request for a required service, the request is steered to one of the available service instances. 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 requested 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, latency experienced or similar, for a more dynamic selection of a suitable service instance.

It is important to note that clients may move. This means that the service instance that was "best" at one moment might no longer be best when a new service request is issued. This creates a (physical) dynamicity that will need to be catered for in addition to the changes in server and network load.

Figure 1 shows a common way to deploy edge sites in the metro. Edge sites are connected with Provider Edges(PEs). There is an edge data center for metro area which has high computing resource and provides the service to more User Equipments(UEs) at the working time. Because more office buildings are in the metro area. And there are also some remote edge sites which have limited computing resource and provide the service to the UEs close to them.

Applications to meet service demands could be deployed in both the edge data center in metro area and the remote edge sites. In this case, the service request and the resource are matched well. Some potential traffic steering may be needed just for special service request or some small scheduling demand.

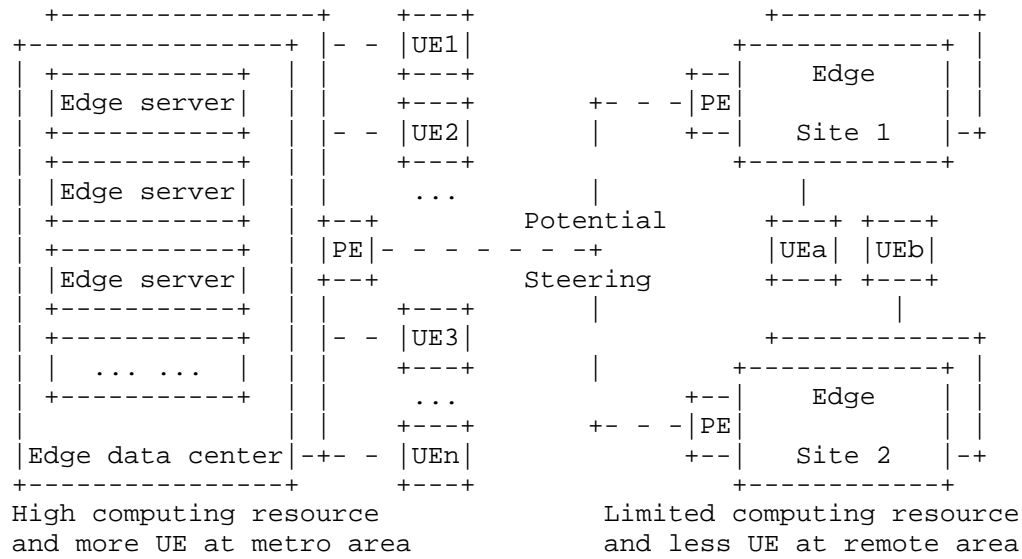


Figure 1: Common Deployment of Edge Sites

Figure 2 shows that during non-working hours, for example at weekend or daily night, more UEs move to the remote area that are close to their house or for some weekend events. So there will be more service request at remote but with limited computing resource, while the rich computing resource might not be used with less UE in the metro area. It is possible for many people to request services at the remote area, but with the limited computing resource, moreover, as the people move from the metro area to the remote area, the edge sites that serve common services will also change, so it may be necessary to steer some traffic back to the metro data center.

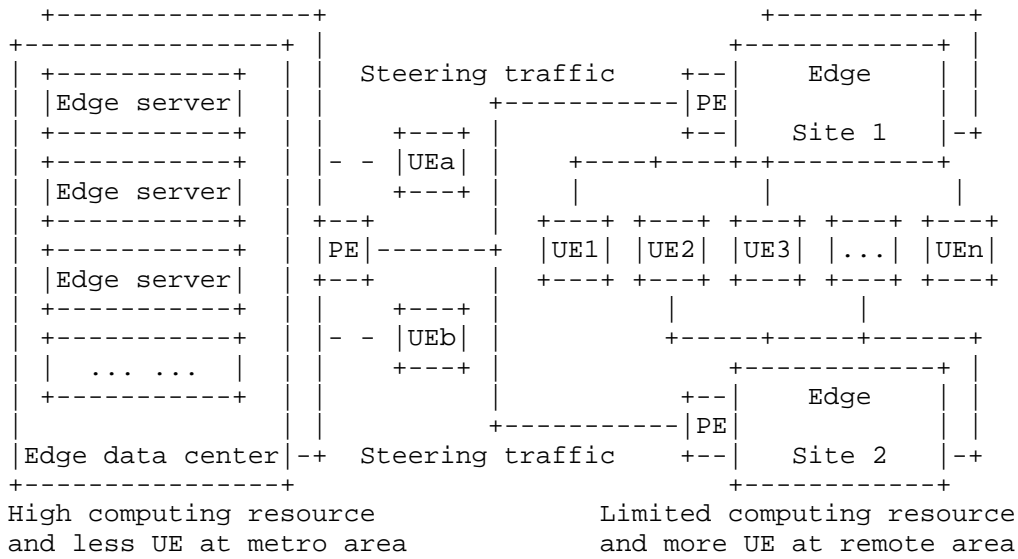


Figure 2: Steering Traffic among Edge Sites

There will also be the common variable of network and computing resources, for someone who is not moving but get a poor latency sometime. Because of other UEs moving, a large number of request for temporary events such as vocal concert, shopping festival and so on, and there will also be the normal change of the network and computing resource status. So for some fixed UEs, it is also expected to steer the traffic to appropriate sites dynamically.

Those problems indicate that traffic needs to be steered among different edge sites, because of the mobility of the UE and the common variable of network and computing resources. Moreover, some use cases in the following section require both low latency and high computing resource usage or specific computing hardware capabilities (such as local GPU); hence joint optimization of network and computing resource is needed to guarantee the Quality of Experience (QoE).

3.3. Discussions on Common Issues

Whether it comes to the deployment of service instances or traffic steering, two common issues need to be considered. First, how to define and select computing metrics; second, how to make decisions. The second issue pertains to the internal implementation of service or network systems and does not involve solution standardization. In contrast, the definition of computing metrics is related to protocol design and interoperability implementation, falling within the scope

of standardization.

While computing metrics are primarily defined to support ingress nodes in making traffic steering decisions among existing service instances, some of these metrics may also be useful to service orchestration systems when making service instance placement or scaling decisions. In particular, metrics that reflect the availability, utilization, or performance of compute and network resources at service sites can provide valuable input to orchestration functions, even though the act of service placement itself remains outside the scope of CATS. This dual applicability allows a common set of metrics to inform both traffic steering and higher-level service management decisions, without requiring CATS to define orchestration behavior. Specific requirements for computing metrics will be introduced in subsequent sections.

4. Use Cases

This section presents a non-exhaustive set of use cases which would benefit from the dynamic selection of service instances and the steering of traffic to those service instances.

4.1. Example 1: Computing-aware AR or VR

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.

A Cloud VR service is delay-sensitive and influenced by both network and computing resources. Therefore, the edge node which executes the service has to be carefully selected to make sure it has sufficient computing resource and good network condition to guarantee the end-to-end service delay. 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. Further, the 20ms latency can be categorized as:

- (i) Sensor sampling delay(client), which is considered imperceptible by users is less than 1.5ms including an extra 0.5ms for digitalization and end device processing.

- (ii) Display refresh delay(client), which take 7.9ms based on the 144Hz display refreshing rate and 1ms extra delay to light up.
- (iii) Image/frame rendering delay(server), which could be reduced to 5.5ms.
- (iv) Round-trip network delay: The remaining latency budget is 5.1ms, calculated as $20 - 1.5 - 5.5 - 7.9 = 5.1\text{ms}$.

So the budgets for server(computing) delay and network delay are almost equivalent, which make sense to consider both of the delay for computing and network. And it could't meet the total delay requirements or find the best choice by either optimizing the network or computing resource.

Based on the analysis, here are some further assumption as Figure 3 shows, the client could request any service instance among 3 edge sites. The delay of client could be same, and the differences of edge sites and corresponding network path have different delays:

- * Edge site 1: The computing delay=4ms based on a light load, and the corresponding network delay=9ms based on a heavy traffic.
- * Edge site 2: The computing delay=10ms based on a heavy load, and the corresponding network delay=4ms based on a light traffic.
- * Edge site 3: The edge site 3's computing delay=5ms based on a normal load, and the corresponding network delay=5ms based on a normal traffic.

In this case, we can't get an optimal network and computing total delay if choosing the resource only based on either of computing or network status:

- * The edge site based on the best computing delay it will be the edge site 1, the E2E delay=22.4ms.
- * The edge site based on the best network delay it will be the edge site 2, the E2E delay=23.4ms.
- * The edge site based on both of the status it will be the edge site 3, the E2E delay=19.4ms.

So, the best choice to ensure the E2E delay is edge site 3, which is 19.4ms and is less than 20ms. The differences of the E2E delay is only 3~4ms among the three, but some of them will meet the application demand while the others don't.

The conclusion is that it requires to dynamically steer traffic to the appropriate edge to meet the E2E delay requirements considering both network and computing resource status. Moreover, the computing resources have a big difference in different edges, and the "closest site" may be good for latency but lacks GPU support and should therefore not be chosen.

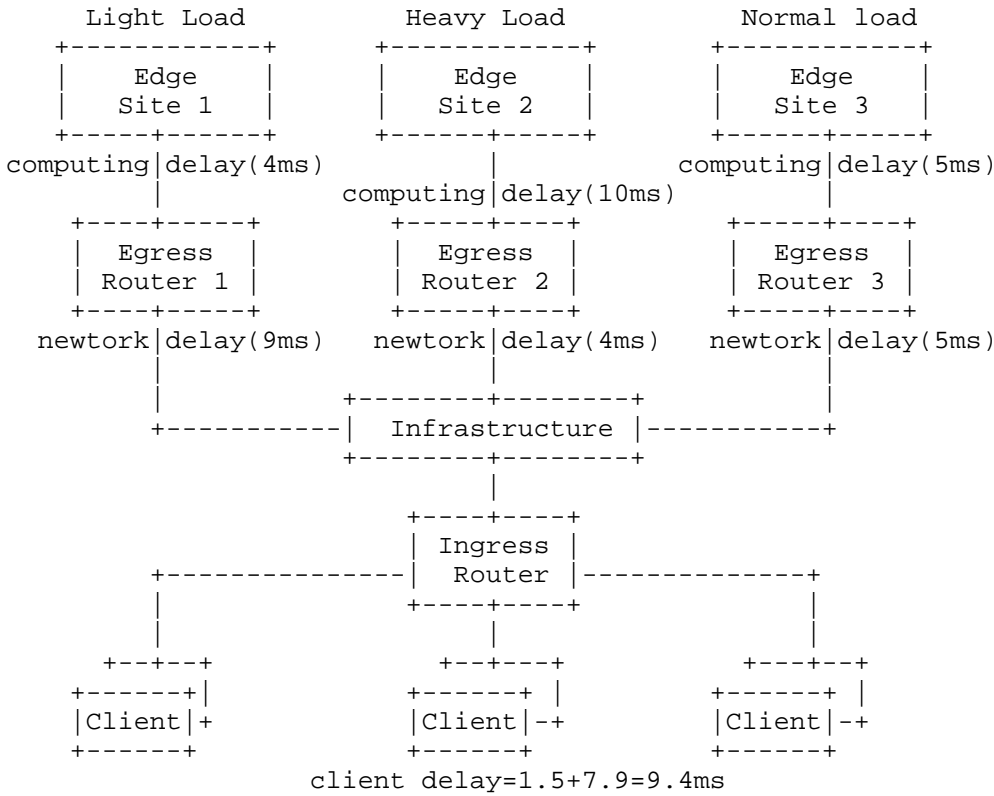


Figure 3: Computing-Aware AR or VR

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.

4.2. Example 2: Computing-aware Intelligent Transportation

For the convenience and safety of transportation, more video capture devices need to be deployed as urban infrastructure, and the better video quality is also required to facilitate the content analysis. Therefore, the transmission capacity of the network will need to be further increased, and the collected video data need to be further processed by edge nodes for edge computing, such as for pedestrian face recognition, vehicle tracking, and road accident prediction. This, in turn, also impacts the requirements for the video processing capacity of computing nodes and network capacity of network nodes in terms of network bandwidth and delay.

In auxiliary driving scenarios, to help overcome a non-line-of-sight problem due to blind spot (or obstacles) and an abrupt collision problem, the edge node can collect comprehensive road and traffic information around the vehicles' locations and perform data processing. Then the vehicles with high security risk could be warned accordingly in advance and provided with safe maneuver guide (e.g., left-lane change, right-lane change, speed reduction, and braking) . This could improve driving safety in complicated road conditions, such as at intersections and on highways, through the help from the edge node having a wider view. This scenario is also called "Extended Electronic Horizon" [HORITA], because the vehicles could extend their view range by exchanging their physical view information from their onboard camera and LiDAR with an adjacent edge node or other vehicles via Vehicle-to-Everything (V2X) communications. For instance, video images captured by an onboard camera in each vehicle are 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" vehicles, using relative location information among the vehicles. Furthermore, data privacy may lead to a requirement to process the data by an edge node (or an adjacent vehicle as a cluster node) as close to the source as possible to limit the data's spread across many network components in the network.

Nevertheless, load at specifically "closest" nodes may greatly vary, leading to the possibility for the closest edge node becoming overloaded. This may lead to a higher response time and therefore a delay in responding to the auxiliary driving request. As a result, there will be road traffic delays or even vehicle accidents in the road networks. Thus, the selection of a "closest" node should consider the node's current workload and the network condition from the source vehicle to the "closest" edge node. Hence, in such cases, delay-insensitive services such as in-vehicle infotainment (e.g., online music playing, online video watching, and navigation service) should be dispatched to other lightly-loaded edge nodes instead of local edge nodes even though the lightly-loaded edge nodes are a little far away from the service user vehicle. On the other hand, delay-sensitive services are preferentially processed locally to ensure the service availability, Quality of Service (QoS), and Quality of Experience (QoE). Thus, according to delay requirements of services, the selection of appropriate edge nodes should be done. Also, since the vehicles keep moving along the roadways, the migration of contexts for the service user vehicles should be smoothly and proactively between the current edge node and the next edge node in a consistent way, considering the vehicles' service requirements.

In video recognition scenarios, as both the number of waiting people and that of vehicles increase, more computing resources are needed to process the video contents. Traffic congestion and weekend personnel flows from a city's edge to the city's center are huge. Thus, an efficient network and computing capacity scheduling is also required for scalable services according to the number of users. Those would cause the overload of the nearest edge sites (or edge nodes) to become much severer if there is no extra method used. Therefore, some of the service request flows might be steered to other appropriate edge sites (or edge nodes) rather than simply the nearest one.

4.3. Example 3: Computing-aware 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 subsection. 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.

4.4. Example 4: Computing-aware SD-WAN

SD-WAN is an overlay connectivity service that optimizes the transport of IP packets over one or more underlay connectivity services by recognizing applications and determining forwarding behavior through the application of policies [MEF70.2]. SD-WAN can be deployed by both service providers and enterprises to support connectivity across branch sites, data centers, and cloud environments. Applications or services may be deployed at multiple

locations to achieve performance, resiliency, or cost objectives.

In current SD-WAN deployments, forwarding decisions are primarily based on network-related metrics such as available bandwidth, latency, packet loss, or path availability. However, these decisions typically lack visibility into the computing resources available at the destination sites, such as CPU or GPU utilization, memory pressure, or other composite cost metrics.

CATS metrics can complement existing SD-WAN network metrics by providing information about the availability and condition of computing resources associated with service instances at edge or cloud sites. Such metrics may be consumed by a centralized SD-WAN controller when deriving policies or computing preferred paths, and/or by SD-WAN edge devices to make distributed, real-time traffic steering decisions among already-deployed service instances. In both cases, the goal is to enable application traffic to be steered towards service instances and sites that best satisfy application requirements by jointly considering network and computing conditions.

For the scenario of enterprises deploying applications in the cloud, SD-WAN provides enterprises with centralized control over Customer-Premises Equipments (CPEs) in branch offices and the cloudified CPEs (vCPEs) in the clouds. The CPEs connect the clients in branch offices and the application servers in clouds. The same application server in different clouds is called an application instance. Different application instances have different computing resource.

SD-WAN is aware of the computing resource of applications deployed in the clouds by vCPEs, and selects the application instance for the client to visit according to computing power and the network state of WAN.

Additionally, in order to provide cost-effective solutions, the SD-WAN may also consider cost, e.g., in terms of energy prices incurred or energy source used, when selecting a specific application instance over another. For this, suitable metric information would need to be exposed, e.g., by the cloud provider, in terms of utilized energy or incurred energy costs per computing resource.

Figure 4 below illustrates Computing-aware SD-WAN for Enterprise Cloudification.

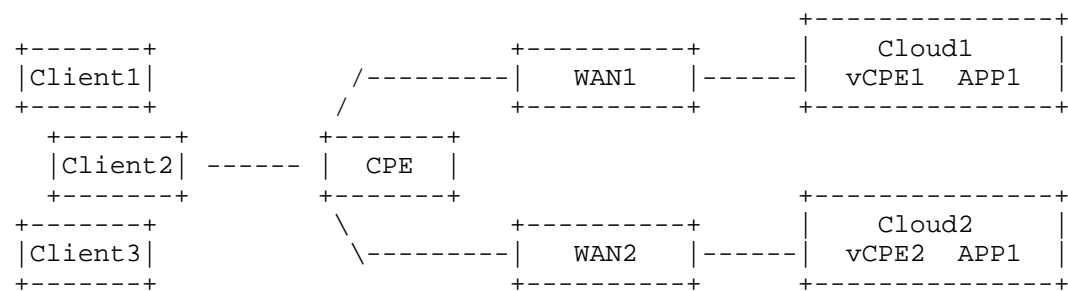


Figure 4: Illustration of Computing-aware SD-WAN for Enterprise Cloudification

The current computing load status of the application APP1 in cloud1 and cloud2 is as follows: each application uses 6 vCPUs. The load of application in cloud1 is 50%. The load of application in cloud2 is 20%. The computing resource of APP1 are collected by vCPE1 and vCPE2 respectively. Client1 and Client2 are visiting APP1 in cloud1. WAN1 and WAN2 have the same network states. Considering lightly loaded application SD-WAN selects APP1 in cloud2 for the client3 in branch office. The traffic of client3 follows the path: Client3 -> CPE -> WAN2 -> Cloud2 vCPE1 -> Cloud2 APP1

4.5. Example 5: Computing-aware Distributed AI Training and Inference

Artificial Intelligence (AI) large model refers to models that are characterized by their large size, high complexity, and high computational requirements. AI large models have become increasingly important in various fields, such as natural language processing for text classification, computer vision for image classification and object detection, and speech recognition.

AI large model contains two key phases: training and inference. Training refers to the process of developing an AI model by feeding it with large amounts of data and optimizing it to learn and improve its performance. On the other hand, inference is the process of using the trained AI model to make predictions or decisions based on new input data.

4.5.1. Distributed AI Inference

With the fast development of AI large language models, more lightweight models can be deployed at edge sites. Figure 5 shows the potential deployment of this case.

AI inference contains two major steps, prefilling and decoding. Prefilling processes a user's prompt to generate the first token of the response in one step. Following it, decoding sequentially generates subsequent tokens step-by-step until the termination token. These stages consume much computing resource. Important metrics for AI inference are processor cores which transform prompts to tokens, and memory resources which are used to store key-values and cache tokens. The generation and processing of tokens indicates the service capability of an AI inference system. Single site deployment of the prefilling and decoding might not provide enough resources when there are many clients sending requests (prompts) to access AI inference service.

More generally, we also see the use of cost information, specifically on the cost for energy expended on AI inferencing of the overall provided AI-based service, as a possible criteria for steering traffic. Here, we envision (AI) service tiers being exposed to end users, allowing to prioritize, e.g., 'greener energy costs' as a key criteria for service fulfilment. For this, the system would employ metric information on, e.g., utilized energy mix at the AI inference sites and costs for energy to prioritize a 'greener' site over another, while providing similar response times.

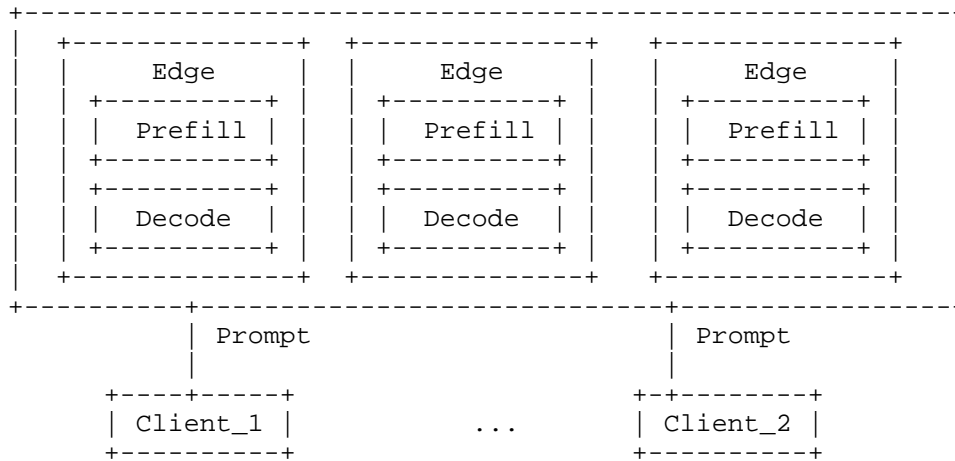


Figure 5: Illustration of Computing-aware AI large model inference

4.5.2. Distributed AI Training

Although large language models are nowadays confined to be trained with very large centers with computational, often GPU-based, resources, platforms for federated or distributed training are being positioned, specifically when employing edge computing resources.

While those approaches apply their own (collective) communication approach to steer the training and gradient data towards the various (often edge) computing sites, we also see a case for CATS traffic steering here. For this, the training clusters themselves may be multi-site, i.e., combining resources from more than one site, but acting as service instances in a CATS sense, i.e., providing the respective training round as a service to the overall distributed/federated learning platform.

One (cluster) site can be selected over another based on compute, network but also cost metrics, or a combination thereof. For instance, training may be constrained based on the network resources to ensure timely delivery of the required training and gradient information to the cluster site, while also computational load may be considered, particularly when the cluster sites are multi-homed, thus hosting more than one application and therefore become (temporarily) overloaded. But equally to our inferencing use case in the previous section, the overall training service may also be constrained by cost, specifically energy aspects, e.g., when positioning the service utilizing the trained model is advertising its 'green' credentials to the end users. For this, costs based on energy pricing (over time) as well as the energy mix may be considered. One could foresee, for instance, the coupling of surplus energy in renewable energy resources to a cost metric upon which traffic is steered preferably to those cluster sites that are merely consuming surplus and not grid energy.

Storage is also necessary for performing distributed/federated learning due to several key reasons. Firstly, it is needed to store model checkpoints produced throughout the training process, allowing for progress tracking and recovery in case of interruptions. Additionally, storage is used to keep samples of the dataset used to train the model, which often come from distributed sensors such as cameras, microphones, etc. Furthermore, storage is required to hold the models themselves, which can be very large and complex. Knowing the storage performance metrics is also important. For instance, understanding the I/O transfer rate of the storage helps in determining the latency of accessing data from disk. Additionally, knowing the size of the storage is relevant to understand how many model checkpoints can be stored or the maximum size of the model that can be locally stored.

4.6. Discussion

The five use cases mentioned in previous sections serve as examples to show that CATS are needed for traffic steering. Considering that these use cases are enough to derive common requirements, this document only includes the aforementioned five use cases in the main body, although there have been more similar use cases proposed in CATS working group[I-D.dcn-cats-req-service-segmentation]. CATS has raised strong interests in many other standardization bodies, such as ETSI, 3GPP. The applicability of CATS may be further extended in future use cases. At the mean time, the CATS framework may also need to be modified or enhanced according to new requirements raised by potential new CATS use cases. These potential use cases are not included in the current document main body, but are attached in the appendix A of this document.

5. Requirements

In the following, we outline the requirements for the CATS system to overcome the observed problems in the realization of the use cases above.

5.1. Support Dynamic and Effective Selection among Multiple Service Instances

The basic requirement of CATS is to support the dynamic access to different service instances residing in multiple computing sites and then being aware of their status, which is also the fundamental model to enable the traffic steering and to further optimize the network and computing services. A unique service identifier is used by all the service instances for a specific service no matter which edge site an instance may attach to. The mapping of this service identifier to a network locator is basic to steer traffic to any of the service instances deployed in various edge sites.

Moreover, according to CATS use cases, some applications require E2E low latency, which warrants a quick mapping of the service identifier to the network locator. This leads to naturally the in-band methods, involving the consideration of using metrics that are oriented towards compute capabilities and resources, and their correlation with services. Therefore, a desirable system

R1: MUST provide a dynamic discovery and resolution method for mapping a service identifier to one or more current service instance addresses, based on real-time system state.

R2: MUST provide a method to dynamically assess the availability of service instances, based on up-to-date status metrics (e.g., health, load, reachability).

5.2. Support Agreement on Metric Representation and Definition

Computing metrics can have many different semantics, particularly for being service-specific. Even the notion of a "computing load" metric could be represented in many different ways. 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 instance selection decision is important. Therefore, a desirable system

R3: The implementations MUST agree on using metrics that are oriented towards compute capabilities and resources and their representation among service instances in the participating edges, at both design time and runtime.

To better understand the meaning of different metrics and to better support appropriate use of metrics,

R4: A model of the compute and network resources MUST be defined. Such a model MUST characterize how metrics are abstracted out from the compute and network resources. We refer to this model as the Resource Model.

R5: The Resource Model MUST be implementable in an interoperable manner. That is, independent implementations of the Resource Model must be interoperable.

R6: The Resource Model MUST be executable in a scalable manner. That is, an agent implementing the Resource Model MUST be able to execute it at the required time scale and at an affordable cost (e.g., memory footprint, energy, etc.).

R7: The Resource Model MUST be useful. That is, the metrics that an agent can obtain by executing the Resource Model must be useful to make node and path selection decisions.

We recognize that different network nodes, e.g., routers, switches, etc., may have diversified capabilities even in the same routing domain, let alone in different administrative domains and from different vendors. Therefore, to work properly in a CATS system,

R8: Beyond metrics definition, CATS solutions MUST contain the staleness handling of CATS metrics and indicate when to refresh the metrics, so that CATS components can know if a metric value is valid or not.

R9: All metric information used in CATS MUST be produced and encoded in a format that is understood by participating CATS components. For metrics that CATS components do not understand or support, CATS components will ignore them. CATS SHOULD be applied in non-CATS network environments when needed, considering that CATS is designed with extensibility and could work compatibly with existing non-CATS network environments when the network components in these environments could be upgraded to know the meaning of CATS metrics.

R10: CATS components SHOULD support a mechanism to advertise or negotiate supported metric types and encodings to ensure compatibility across implementations.

R11: The computation and use of metrics in CATS MUST be designed to avoid introducing routing loops or path oscillations when metrics are distributed and used for path selection.

5.3. Use of CATS Metrics

Network path costs in the current routing system usually do not change very frequently. Network traffic engineering metrics (such as available bandwidth) may change more frequently as traffic demands fluctuate, but distribution of these changes is normally damped so that only significant changes cause routing protocol messages.

However, metrics that are oriented towards compute capabilities and resources in general can be highly dynamic, e.g., changing rapidly with the number of sessions, the CPU/GPU utilization and the memory consumption, etc. Service providers must determine at what interval or based on what events such information needs to be distributed. Overly frequent distribution with more accurate synchronization may result in unnecessary overhead in terms of signaling.

Moreover, depending on the service related decision logic, one or more metrics need to be conveyed in a CATS domain. The problem to be addressed here may be the frequency of such conveyance, and which CATS component is the decision maker for the service instance selection should also be considered. Thereby, choosing appropriate protocols for conveying CATS metrics is important. While existing routing protocols may serve as a baseline for signaling metrics, for example, BGP extensions[RFC4760] and GRASP[RFC8990]. These routing protocols may be more suitable for distributed systems. Considering about some centralized approaches to select CATS service instances,

other means to convey the metrics can equally be chosen and even be realized, for example, ALTO protocol[RFC7285] which leverages restful API for publication of CATS metrics to a centralized decision maker. Specifically, a desirable system,

R12: MUST provide mechanisms for metric collection.

R13: MUST specify which entity is responsible for collecting metrics.

Collecting metrics from all of the services instances may incur much overhead for the decision maker, and thus hierarchical metric collection is needed. That is,

R14: SHOULD provide mechanisms to aggregate the metrics.

CATS components do not need to be aware of how metrics are collected behind the aggregator. The decision point may not be directly connected with service instances or metric collectors, therefore,

R15: MUST provide mechanisms to distribute the metrics.

There may be various update frequencies for different computing metrics. Some of the metrics may be more dynamic, while others are relatively static. Accordingly, different distribution methods may need to be chosen with respect to different update frequencies of different metrics. Therefore a system,

R16: MUST NOT be sensitive to the update frequency of the metrics, and MUST NOT be dependent on or vulnerable to the mechanisms used to distribute the metrics.

Sometimes, a metric that is chosen is not accurate for service instance selection, in such a case, a desirable system,

R17: SHOULD provide mechanisms to assess selection accuracy and re-select metrics if the selection result is not accurate.

5.4. Support Instance Affinity

In the CATS system, a service may be provided by one or more service instances that would be deployed at different locations in the network. Each instance provides equivalent service functionality to its respective clients. The decision logic of the instance selection is subject to the packet level communication and packets are forwarded based on the operating status of both network and computing resources. This resource status will likely change over time, leading to individual packets potentially being sent to different network locations, possibly segmenting individual service

transactions and breaking service-level semantics. Moreover, when a client moves, the access point might change and successively lead to the migration of service instances. If execution changes from one (e.g., virtualized) service instance to another, state/context needs to be transferred to the new instance. Such required transfer of state/context makes it desirable to have instance affinity as the default, removing the need for explicit context transfer, while also supporting an explicit state/context transfer (e.g., when metrics change significantly). So in those situations:

R18: CATS systems MUST maintain instance affinity for stateful sessions and transactions.

The nature of this affinity is highly dependent on the nature of the service, which could be seen as an 'instance affinity' to represent the relationship. 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 the manner of 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 the AR/VR use case, where the previous client input is needed to render subsequent frames.

However, a client, e.g., a mobile UE, may have many applications running. If all, or majority, of the applications request the CATS-based 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, and per-(sub)flow with regard to a service instance. Evidently, these fine-granular runtime states can potentially place a heavy burden on 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.

Besides, there might be the case that UE moves to a new (access) network or the service instance is migrated to another cloud, which cause the unreachable or inconvenient of the original service instance. So the UE and service instance mobility also need to be considered.

Therefore, a desirable system,

R19: Instance affinity MUST be maintained for service requests or transactions that belong to the same flow.

R20: MUST avoid maintaining per-flow states for specific applications in network nodes for providing instance affinity.

R21: MUST provide mechanisms to minimize client side states in order to achieve the instance affinity.

R22: SHOULD support service continuity in the presence of UE or service instance mobility.

5.5. Preserve Communication Confidentiality

Exposing CATS metrics to the network may lead to the leakage of application privacy. In order to prevent it, it is necessary to consider the methods to handle the sensitive information. For instance, using general anonymization methods, including hiding the key information representing the identification of devices, or using an index to represent the service level of computing resources, or using customized information exposure strategies according to specific application requirements or network scheduling requirements. At the same time, when anonymity is achieved, it is important to ensure that the exposed computing information remains sufficient to enable effective traffic steering. Therefore, a CATS system

R23: MUST preserve the confidentiality of the communication relation between a user and a service provider by minimizing the exposure of user-relevant information according to user's demands.

5.6. Correlation between Use Cases and Requirements

A table is presented in this section to better illustrate the correlation between CATS use cases and requirements, 'X' is for marking that the requirement can be derived from the corresponding use case.

		Use cases					
Requirements		AR/VR	ITS	DT	SD-WAN	AI	
Instance Selection	R1	X	X	X	X	X	
	R2	X	X	X	X	X	
	R3	X	X	X	X	X	

Metric Definition	R4	X	X	X	X	X
	R5	X	X	X	X	X
	R6	X	X	X	X	X
	R7	X	X	X	X	X
	R8	X	X	X	X	X
	R9	X	X	X	X	X
	R10	X	X	X	X	X
	R11	X	X	X	X	X
	R12	X	X	X	X	X
	R13	X	X	X	X	X
Use of Metrics	R14	X	X	X	X	X
	R15	X	X	X	X	X
	R16	X	X	X	X	X
	R17	X	X	X	X	X
	R18	X	X	X	X	X
	R19	X	X	X	X	X
Instance Affinity	R20	X	X	X	X	X
	R21	X	X	X	X	X
	R22	X	X			X
	R23	X	X	X	X	X
Confiden- -tiality						

Figure 6: Mapping between CATS Use Cases and Requirements

6. Security Considerations

CATS decision-making relies on real-time computing and network status as well as service information, requiring robust security safeguards to mitigate risks associated with dynamic service and resource scheduling, and cross-node data transmission.

Core Security Risks and Requirements include:

* User Privacy Leakage Risk

Description: CATS involves user-related data (e.g., access patterns, service requests) across edge nodes. Unauthorized disclosure of user identifiers or per-user behavior tracking risks profiling or identity theft, especially in use cases with personal/context-rich data (e.g., AR/VR, vehicle trajectories, AI prompts), violating regulations and eroding trust.

R24: User activity privacy **MUST** be preserved by anonymizing identifying information. Per-user behavior pattern tracking is prohibited.

* Service Instance Identity Spoofing and Traffic Hijacking

Description: Attackers may spoof legitimate service instance identities or tamper with "service identifier-instance address" mappings (per R1), diverting traffic to malicious nodes. This undermines CATS' core scheduling logic, causing service disruptions, data leaks, and potential physical harm in safety-critical scenarios.

R25: Service instances **MUST** be authenticated. "Service identifier - instance address" mapping results **MUST** be encrypted.

* Tampering and False Reporting of CATS Metrics

Description: Attackers may tamper with core scheduling metrics or submit false data (per R3-R17), misleading traffic steering decisions. This leads to node overload, link congestion, or "resource exhaustion attacks," directly degrading Quality of Experience (QoE).

R26: Metric collection and distribution **MUST** employ encryption. Mechanisms for secondary validation and traceability of abnormal metrics **MUST** be supported, avoiding over-reliance on single-node reports.

* Security of Cross-Node Context Migration Data

Description: During user or terminal mobility, session states and computing context (e.g., AR rendering progress, vehicle status) may be intercepted or tampered with during cross-node migration (per R18-R22). This impairs service continuity, leaks sensitive data, or causes state inconsistency.

R27: Migration data MUST use end-to-end encryption, accessible only to authorized target instances. Migration instructions MUST include integrity check codes.

7. IANA Considerations

This document makes no requests for IANA action.

8. References

8.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC4760] Bates, T., Chandra, R., Katz, D., and Y. Rekhter, "Multiprotocol Extensions for BGP-4", RFC 4760, DOI 10.17487/RFC4760, January 2007, <<https://www.rfc-editor.org/info/rfc4760>>.
- [RFC7285] Alimi, R., Ed., Penno, R., Ed., Yang, Y., Ed., Kiesel, S., Previdi, S., Roome, W., Shalunov, S., and R. Woundy, "Application-Layer Traffic Optimization (ALTO) Protocol", RFC 7285, DOI 10.17487/RFC7285, September 2014, <<https://www.rfc-editor.org/info/rfc7285>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.
- [RFC8990] Bormann, C., Carpenter, B., Ed., and B. Liu, Ed., "GeneRic Autonomic Signaling Protocol (GRASP)", RFC 8990, DOI 10.17487/RFC8990, May 2021, <<https://www.rfc-editor.org/info/rfc8990>>.

8.2. Informative References

- [GAIA-X] Gaia-X, "GAIA-X: A Federated Data Infrastructure for Europe", 2021.

- [HORITA] Horita, Y., "Extended electronic horizon for automated driving", Proceedings of 14th International Conference on ITS Telecommunications (ITST), 2015.
- [I-D.contreras-alto-service-edge]
Contreras, L. M., Randriamasy, S., Ros-Giralt, J., Perez, D. A. L., and C. E. Rothenberg, "Use of ALTO for Determining Service Edge", Work in Progress, Internet-Draft, draft-contreras-alto-service-edge-10, 13 October 2023, <<https://datatracker.ietf.org/doc/html/draft-contreras-alto-service-edge-10>>.
- [I-D.dcn-cats-req-service-segmentation]
Ng 盼皇, T. M. and Y. Kim, "Additional CATS requirements consideration for Service Segmentation-related use cases", Work in Progress, Internet-Draft, draft-dcn-cats-req-service-segmentation-02, 1 July 2025, <<https://datatracker.ietf.org/doc/html/draft-dcn-cats-req-service-segmentation-02>>.
- [I-D.ietf-cats-framework]
Li, C., Du, Z., Boucadair, M., Contreras, L. M., and J. Drake, "A Framework for Computing-Aware Traffic Steering (CATS)", Work in Progress, Internet-Draft, draft-ietf-cats-framework-19, 20 November 2025, <<https://datatracker.ietf.org/doc/html/draft-ietf-cats-framework-19>>.
- [Industry4.0]
Industry4.0, "Details of the Asset Administration Shell, Part 1 & Part 2", 2020.
- [MEF70.2] MEF, Ed., "SD-WAN Service Attributes and Service Framework", 2023.
- [TR22.874] 3GPP, "Study on traffic characteristics and performance requirements for AI/ML model transfer in 5GS (Release 18)", 2021.

Appendix A. Appendix A

This section presents an additional CATS use case, which is not included in the main body of this document. Reasons are that the use case may bring new requirements that are not considered in the initial charter of CATS working group. The requirements impact the design of CATS framework and may need further modification or enhancement on the initial CATS framework that serves all the existing use cases listed in the main body. However, the ISAC use

case is promising and has gained industry consensus. Therefore, this use case may be considered in future work of CATS working group.

A.1. Integrated Sensing and Communications (ISAC)

Integrated Sensing and Communications (ISAC) enables wireless networks to perform simultaneous data transmission and environmental sensing. In a distributed sensing scenario, multiple network nodes --such as base stations, access points, or edge devices-- collect raw sensing data from the environment. This data can include radio frequency (RF) reflections, Doppler shifts, channel state information (CSI), or other physical-layer features that provide insights into object movement, material composition, or environmental conditions. To extract meaningful information, the collected raw data must be aggregated and processed by a designated computing node with sufficient computational resources. This requires efficient coordination between sensing nodes and computing resources to ensure timely and accurate analysis, making it a relevant use case for Computing-Aware Traffic Steering (CATS) in IETF.

This use case aligns with ongoing efforts in standardization bodies such as the ETSI ISAC Industry Specification Group (ISG), particularly Work Item #5 (WI#5), titled 'Integration of Computing with ISAC'. WI#5 focuses on exploring different forms of computing integration within ISAC systems, including sensing combined with computing, communications combined with computing, and the holistic integration of ISAC with computing. The considerations outlined in this document complement ETSI's work by examining how computing-aware networking solutions, as developed within CATS, can optimize the processing and routing of ISAC sensing data.

As an example, we can consider a network domain with multiple sites capable of hosting the ISAC computing "service", each with potentially different connectivity and computing characteristics. Figure 7 shows an exemplary scenario. Considering the connectivity and computing latencies (just as an example of metrics), the best service site is #n-1 in the example used in the Figure. Note that in the figure we still use the old terminology in which by ICR we mean Ingress CATS-Forwarder [I-D.ietf-cats-framework], and by ECR we mean Egress CATS-Forwarder.

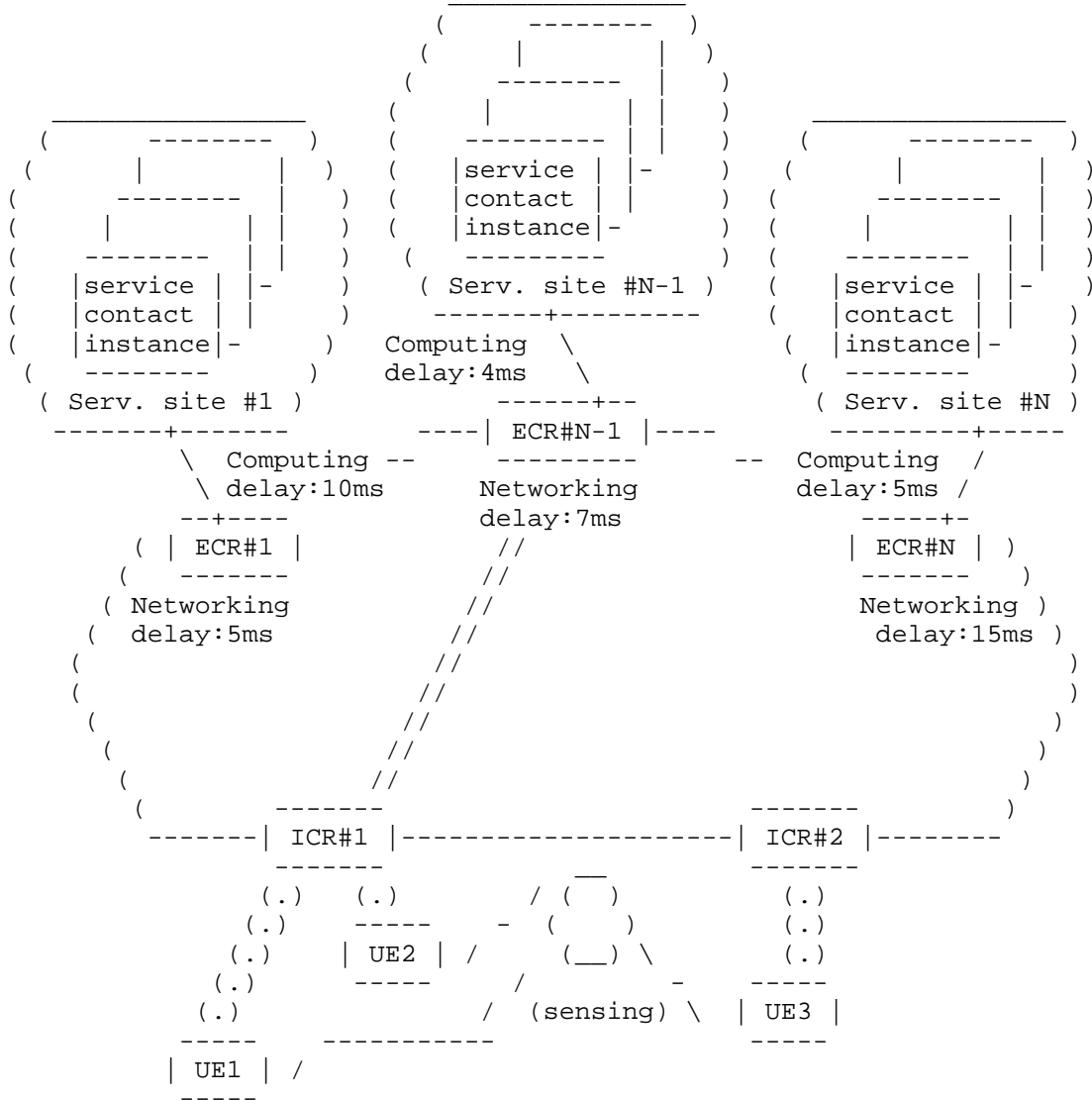


Figure 7: Exemplary ISAC Scenario

In the distributed sensing use case, the sensed data collected by multiple nodes must be efficiently routed to a computing node capable of processing it. The choice of the computing node depends on several factors, including computational load, network congestion, and latency constraints. CATS mechanisms can optimize the selection of the processing node by dynamically steering the traffic based on

computing resource availability and network conditions. Additionally, as sensing data is often time-sensitive, CATS can ensure low-latency paths while balancing computational demands across different processing entities. This capability is essential for real-time applications such as cooperative perception for autonomous systems, industrial monitoring, and smart city infrastructure.

A.1.1.1. Requirements

In addition to some of the requirements already identified for CATS in the main body of this document, there are several additional challenges and requirements that need to be addressed for efficient distributed sensing in ISAC-enabled networks:

R-ISAC-01: CATS systems should be able to select an instance where multiple nodes can steer traffic to simultaneously, ensuring that packets arrive within a maximum time period. This is required because there are distributed tasks in which there are multiple nodes acting as sensors that produce sensing data that has to be then processed by a sensing processing function, typically hosted at the edge. This implies that there is a multi-point to point kind of direction of the traffic, with connectivity and computing requirements associated (which can be very strict for some types of sensing schema).

R-ISAC-02: CATS systems should provide mechanisms that implement per node/flow security and privacy policies to adapt to the nature of the sensitive information that might be exchanged in a sensing task.

Acknowledgements

The authors would like to thank Adrian Farrel, Peng Liu, Joel Halpern, Jim Guichard, Cheng Li, Luigi Iannone, Christian Jacquenet, Yuexia Fu, Erum Welling, Ines Robles, Linda Dunbar, and Jim Reid for their valuable suggestions to this document.

The authors would like to thank Yizhou Li for her early IETF work of Compute First Network (CFN) and Dynamic Anycast (Dyncast) which inspired the CATS work.

Contributors

The following people have substantially contributed to this document:

Yizhou Li
Huawei Technologies
Email: liyizhou@huawei.com

Dirk Trossen
Email: dirk@trossen.tech

Mohamed Boucadair
Orange
Email: mohamed.boucadair@orange.com

Carlos J. Bernardos
UC3M
Email: cjbc@it.uc3m.es

Peter Willis
Email: pjw7904@rit.edu

Philip Eardley
Email: ietf.philip.eardley@gmail.com

Tianji Jiang
China Mobile
Email: tianjijiang@chinamobile.com

Minh-Ngoc Tran
ETRI
Email: mipearlska@etri.re.kr

Markus Amend
Deutsche Telekom
Email: Markus.Amend@telekom.de

Guangping Huang
ZTE
Email: huang.guangping@zte.com.cn

Dongyu Yuan
ZTE
Email: yuan.dongyu@zte.com.cn

Xinxin Yi
China Unicom
Email: yixx3@chinaunicom.cn

Tao Fu
CAICT
Email: futao@caict.ac.cn

Jordi Ros-Giralt
Qualcomm Europe, Inc.
Email: jros@qti.qualcomm.com

Jaehoon Paul Jeong
Sungkyunkwan University
Email: pauljeong@skku.edu

Yan Wang
Migu Culture Technology Co.,Ltd
Email: wangyan_hyl@migu.chinamobile.com

Authors' Addresses

Kehan Yao
China Mobile
Email: yaokehan@chinamobile.com

Luis M. Contreras
Telefonica
Email: luismiguel.contrerasmurillo@telefonica.com

Hang Shi
Huawei Technologies
Email: shihang9@huawei.com

Shuai Zhang
China Unicom
Email: zhangs366@chinaunicom.cn

Qing An
Alibaba Group
Email: anqing.aq@alibaba-inc.com