cats
Internet-Draft
Intended status: Informational
Expires: 17 July 2026

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

13 January 2026

Computing-Aware Traffic Steering (CATS) Problem Statement, Use Cases, and Requirements
draft-ietf-cats-usecases-requirements-12

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/.

Copyright Notice

Table of Contents

1. Introduction

Network operators are increasingly deploying computing resources, with a focus on enhancing edge capabilities to support services requiring low latency, high reliability, and dynamic resource scaling.

Diversified service demands have brought key challenges to service deployment and traffic scheduling. A single-site service instance often lacks sufficient capacity to guarantee the required quality of service, especially during peak hours when local computing resources may fail to handle all incoming requests, leading to longer response times or even request drops. Regular capacity expansion of a single site is neither practical nor economical. Additionally, relying solely on end-side computing enhancements cannot meet the computing requirements of all applications.

It is necessary to deploy services across multiple sites (either edge or central nodes) to improve availability and scalability. To this end, traffic should be steered to the "best" service instance based on factors like current computing load, where "best" is largely determined by application requirements.

However, existing routing schemes and traffic engineering methods fall short of addressing these challenges. The underlying networking infrastructures that include computing resources usually provide relatively static service dispatching or depend solely on connectivity metrics for traffic steering, failing to account for compute capabilities and resource status, which are critical for meeting the quality requirements of modern services.

To tackle this issue, the choice of service instance and network resources should further consider compute-oriented metrics beyond connectivity metrics. The process of selecting service instances and locations based on metrics that are oriented towards compute capabilities and resources, and of directing traffic to them on

chosen network resources is called "Computing-Aware Traffic Steering" (CATS). It should be noted that CATS is not limited to edge computing scenarios, however, Section 3 of this document will focus on edge computing scenarios for problem statement.

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

2. Definition of Terms

This document uses the terms defined in [I-D.ietf-cats-framework], including service site, service instance, CATS service identifier(CS-ID).

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.

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 Service Sites and Service

In edge computing environments, service instances typically adopt a multi-site deployment model. It should be clarified that specific service instance deployment strategies are not within the scope of CATS. However, it is undeniable that there is a close correlation between service instance deployment and traffic scheduling, especially in the definition and selection of core metrics such as computing capabilities and resources. 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.

Therefore, to present a clear and comprehensive problem statement, it is necessary to first introduce the relevant considerations for multi-edge service site deployment. This premise can better support the subsequent elaboration on CATS requirements and solutions.

The core goal of edge computing is to provide computing services closer to users through shorter network paths. Before deploying edge service sites, the following factors need to be considered:

* Geographic location: Including the number of users, differences in service types, and the number of connection requests from users. For edge service sites located in densely populated areas with a large number of users and service requests, more service replicas can be deployed compared to other areas.

* The type, scale, and usage frequency of required computing resources. For example, distributed AI inference services require the deployment of more GPU resources.

* The status of network resources associated with computing resources, such as network topology, network access methods, connectivity, link bandwidth, and path protection or redundancy information.

To improve the overall quality of service, during the service deployment phase, it is necessary to analyze the approximate network and computing resource requirements of the service, comprehensively form a reasonable network and computing resource topology, and clarify the location, overall distribution, and relative position of computing resources in the network topology. This process relies on standardized consensus on computing and network resources related metrics, which is also the point most closely related to the problem space addressed by CATS traffic scheduling.

3.2. Traffic Steering among Edges Service 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 changes in client distribution, different time slots, events, server loads, network capabilities, and some other factors which might not be directly measured, i.e., properties of edge service sites(e.g., geographical location), etc.

It’s assumed that service instances are multi-site deployed, and they are reachable through a network infrastructure.

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 a service instance from the set of candidates is performed using traffic steering methods.

In edge computing, traffic is steered to an edge service site that is "closest" or to one of a few "close" sites using load-balancing. Such traffic steering can be initiated either by the application layer or by the network layer: the application layer may actively query for the optimal node and guide traffic using mechanisms such as the ALTO protocol[RFC7285], while the network layer may leverage Anycast routing[RFC4786], where routering systems automatically distribute traffic according to routing tables in an application-transparent manner. However, regardless of whether the steering is performed by the application or the network, the core criteria for selecting "closest" or "close" sites often rely solely on communication metrics (such as physical distance, hop count, or network latency). This decision logic can easily lead to suboptimal choices, meaning that the "closest" site is not always the "best" one. This is because the computing resources and states of edge service sites can change in real time:

* The closest site may not have sufficient resources.

* The closest site may not have the specific computing resources required.

To address these issues, enhancements to traffic steering mechanisms are needed to direct traffic to sites that can adequately support the requested services. Steering decision may take into account more complex and possibly dynamic metric information, such as load of service instances, latency experienced or similar, for selection of a more 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. From a routing perspective, CATS is an application-transparent routing mechanism that can provide scheduling for both stateful and stateless services. However, in scenarios where clients move and the service is stateful, CATS requires the application to explicitly indicate whether it allows the routing system to enable CATS functionality. Otherwise, mid-session scheduling triggered by CATS may cause application context inconsistency among service sites or even service interruption.

Figure 1 shows a common way to deploy edge service sites in the metro. Edge service 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) (UE1 to UEn) at the working time. Because more office buildings are in the metro area. And there are also some remote edge service sites which have limited computing resource and provide the service to the UEs (UEa, UEb) close to them.

Applications to meet service demands could be deployed in both the edge data center in metro area and the remote edge service 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.

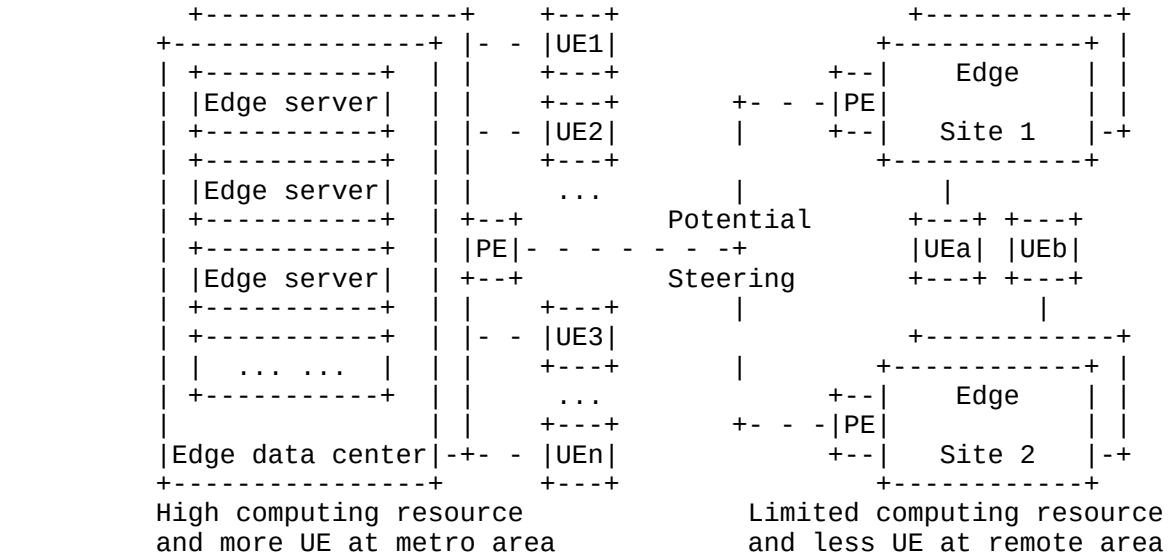

Figure 1: Common Deployment of Edge Service Sites

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 service sites that serve common services will also change, so it may be necessary to steer some traffic back to the metro data center.

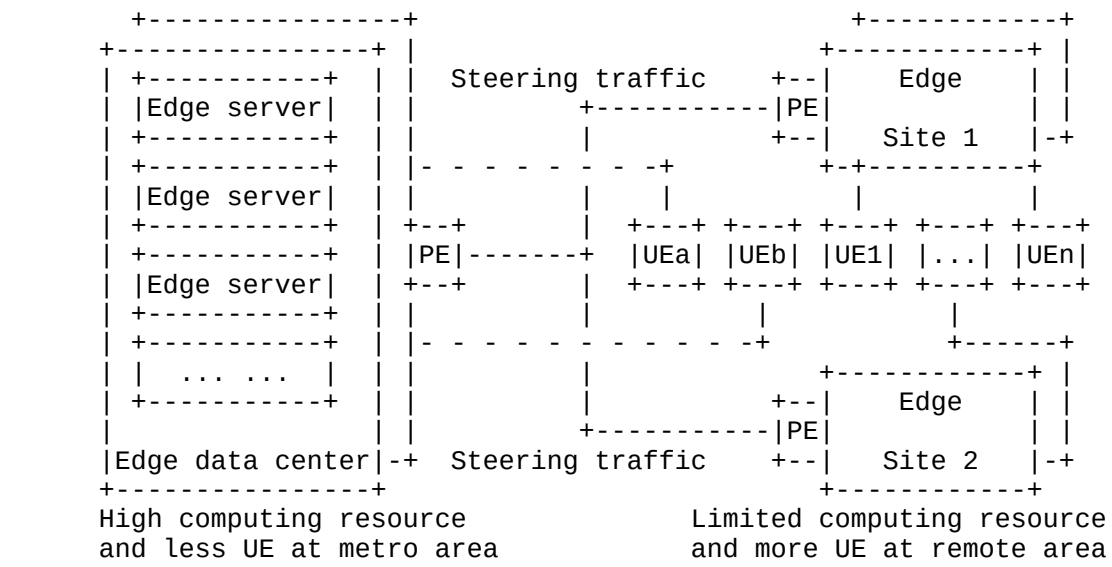

Figure 2: Steering Traffic among Edge Service Sites

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 service 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).

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 service site 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 categoried 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 5.1 ms, calculated as 20-1.5-5.5-7.9 = 5.1ms.

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 service sites. The delay of client could be same, and the differences of edge service sites and corresponding network path have different delays:

* Edge service site 1: The computing delay=4ms based on a light load, and the corresponding network delay=9ms based on a heavy traffic.

* Edge service site 2: The computing delay=10ms based on a heavy load, and the corresponding network delay=4ms based on a light traffic.

* Edge service site 3: The edge service 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 service site based on the best computing delay it will be the edge service site 1, the E2E delay=22.4ms.

* The edge service site based on the best network delay it will be the edge service site 2, the E2E delay=23.4ms.

* The edge service 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 service 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 E2E delay is a key factor for traffic steering, and it is also necessary to consider whether an edge service site has the required GPU resources. Even if a site offers good latency, it cannot provide the service if it lacks GPU support. For AR/VR scenarios, traffic should only be steered at the start of a service session, because mid-session steering involves significant context migration, which is costly and requires explicit application participation and approval.

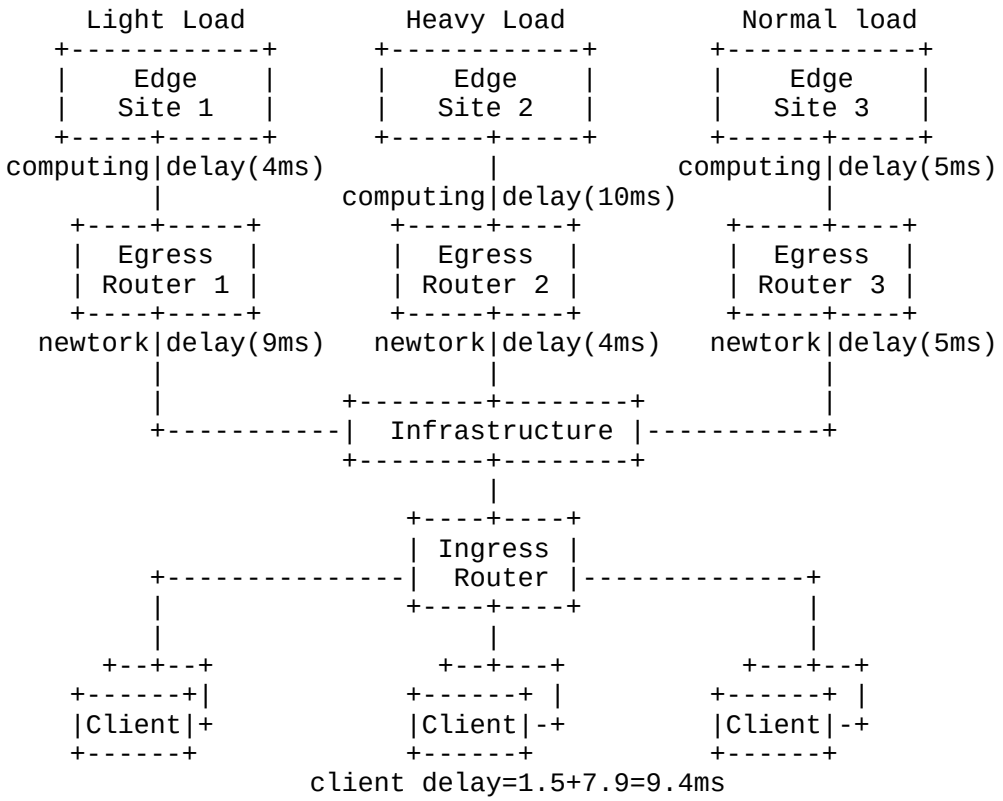

Figure 3: Computing-Aware AR or VR

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

Urban intelligent transportation relies on a large number of high-quality video capture devices, whose data needs to be processed at edge service sites (e.g., pedestrian flow statistics, vehicle tracking). This imposes stringent requirements on the computing capabilities of edge service sites (for video processing) and network performance (bandwidth, latency). CATS can address the issue by coordinating network and computing resources.

In auxiliary driving scenarios (for example, "Extended Electronic Horizon" [HORITA]), edge service sites collect road and traffic data via V2X to address blind-spot and collision risks, and provide real-time warnings and maneuver guidance. Requests are typically sent preferentially to the closest edge node. However, if the closest node becomes overloaded, it may lead to response delays and safety risks, which requires CATS to perform traffic steering.

Specifically, delay-insensitive services (e.g., in-vehicle entertainment) can be offloaded via CATS to edge service sites with lighter loads (even if they are farther away), while delay-sensitive assisted driving services are preferentially processed at local service sites. As mentioned in the problem statement section, CATS is an application-transparent network-layer solution. Unlike ALTO[RFC7285], it enables coordinated scheduling of network and computing resources without requiring application modifications. For moving vehicles, CATS supports smooth and proactive context migration between edge nodes, provided that the application allows it, to maintain service continuity. In addition, vehicle speed is a key factor: faster movement requires higher frequency of metric updates (to be detailed in the requirements section) to ensure that CATS steering decisions remain valid as vehicles switch services among base stations or edge service sites.

In video recognition scenarios, traffic surges (e.g., during rush hours or weekends) can easily overload the closest edge service sites. CATS addresses this scalability challenge by steering excess service requests to other appropriate sites, ensuring that processing capacity matches user demand.

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 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.

In this use case, CATS is required for selecting the optimal PLC instance and storage node, ensuring low latency and reliability for data processing in industrial scenarios, as well as low latency for data reading/writing during twin control processes.

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.

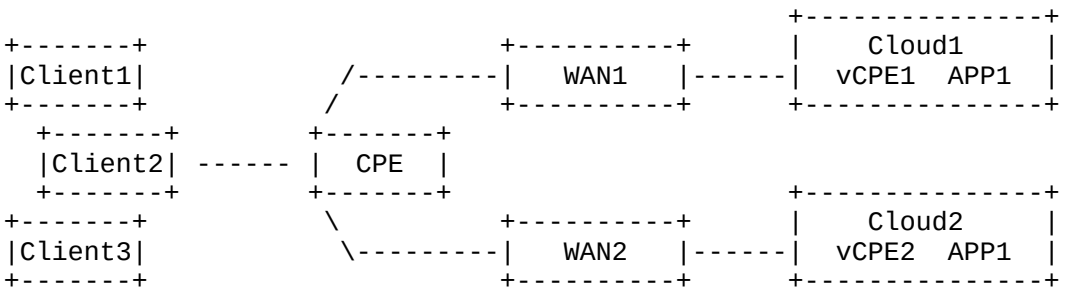

Figure 4: Illustration of Computing-aware SD-WAN for Enterprise Cloudification

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 service 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.

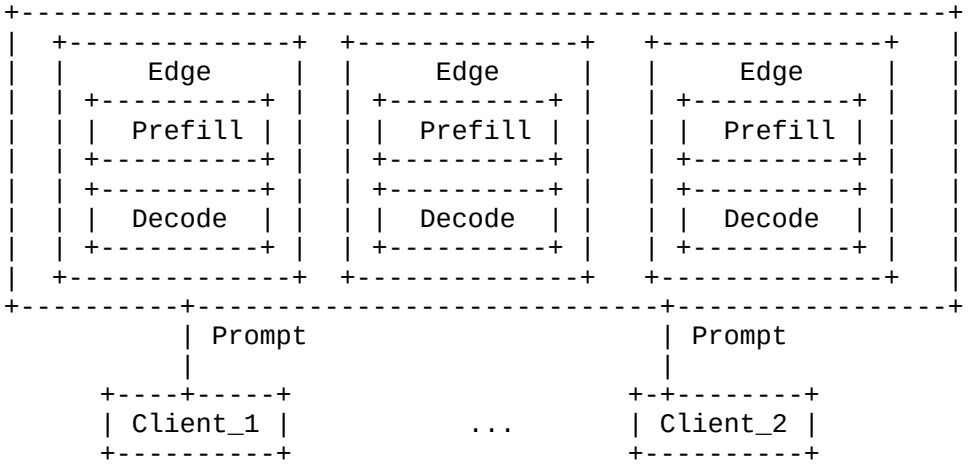

Figure 5: Illustration of Computing-aware AI large model inference

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 CATS service identifier (CS-ID) is used by all the service instances for a specific service no matter which edge service site an instance may attach to. The mapping of this CS-ID to a network locator is basic to steer traffic to any of the service instances deployed in various edge service 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 CS-ID to one or more current service instance addresses, based on up-to-date 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).

Note: The term "up-to-date" herein refers to the latest metric information collected by the system in accordance with the preset metric update cycle. The principle for setting the cycle is generally pre-determined by the network. For example, based on historical statistical data, a relatively appropriate update cycle (either second-level or millisecond-level) is selected for a specific type or certain types of services.

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: An information 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 information model as the Resource Model.

R5: The Resource Model MUST be implementable in an interoperable manner. That is, metrics generated by this resource model MUST be understood and interoperable across independent CATS implementations.

R6: The Resource Model MUST be executable in a scalable manner. That is, the Resource Model MUST be capable of being executed at the required time scale and at an affordable cost (e.g., memory footprint, energy, etc.).

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,

R7: CATS systems MUST support staleness handling for CATS metrics and provide indications of when metrics should be refreshed, so that CATS components can know if a metric value is valid or not.

R8: 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.

R9: CATS MAY 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.

Compute metrics can change rapidly, which may lead to path oscillation if metrics are updated too frequently or become stale if updated too infrequently. R10 ensures that CATS components can negotiate metric types for consistent interpretation, while R11 requires that metrics be used in a way that avoids routing loops and path instability. Together, they balance responsiveness with stability.

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, leveraging restful API for publication of CATS metrics to a centralized decision maker. Specifically, a desirable system,

R12: MUST provide mechanisms for metric collection, including specifying the responsible entity for collection.

Collecting metrics from all of the services instances may incur much overhead for decision makers. Hierarchical aggregation helps reduce this burden by consolidating metrics at intermediate nodes, providing a more scalable and efficient view of resource conditions.

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,

R13: 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,

For example, In highly mobile scenarios, such as fast-moving vehicles mentioned in Section 4.2, compute metrics can quickly become outdated as the UE moves across base stations and edge service sites, potentially requiring more frequent updates. However, updates should remain stable and avoid excessive overhead.

R14: 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.

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).

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,

R15: CATS systems MUST maintain instance affinity for stateful sessions and transactions on a per-flow basis.

R16: MUST avoid maintaining per-flow states for specific applications in network nodes for providing instance affinity.

R17: 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

R18: 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.

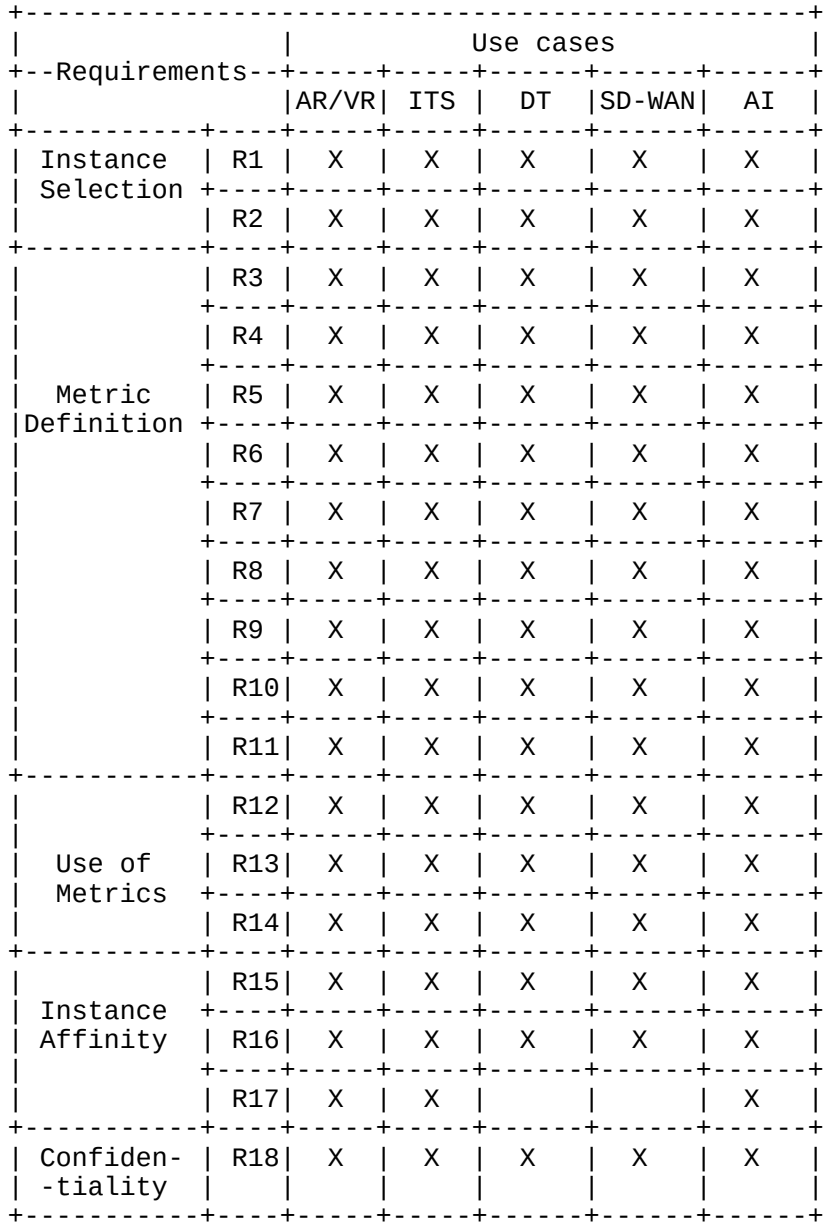

Figure 6: Mapping between CATS Use Cases and Requirements

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 service sites. 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.

R19: 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 "CS-ID-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.

R20: Service instances MUST be authenticated. "CS-ID - 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).

R21: 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.

R22: 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>.

[RFC4786] Abley, J. and K. Lindqvist, "Operation of Anycast Services", BCP 126, RFC 4786, DOI 10.17487/RFC4786, December 2006, <https://www.rfc-editor.org/info/rfc4786>.

[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.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 initail charter of CATS working group. The requirements impact the design of CATS framework and may need further modificaition 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 stilluse 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.

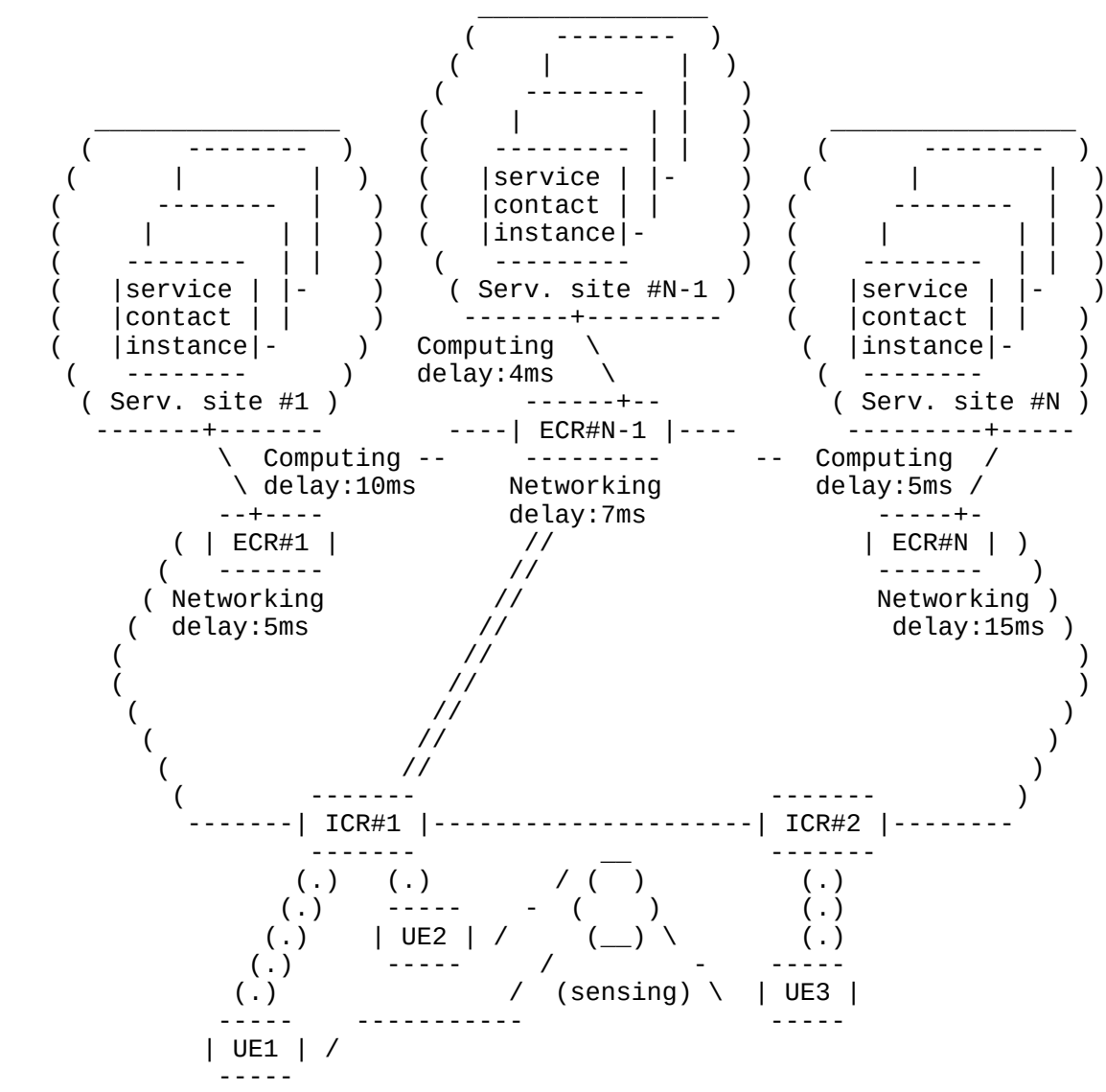

Figure 7: Exemplary ISAC Scenario

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. 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:

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).

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_hy1@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