

Internet Engineering Task Force
Internet-Draft
Intended status: Informational
Expires: 9 August 2026

B. Zhang, Ed.
Pengcheng Laboratory
Y. Dai, Ed.
Sun Yat-sen University
B. Shen, Ed.
Harbin Institute of Technology
5 February 2026

Computing metrics as a service (CMAS) for facilitating traffic steering
in CATS framework
draft-zhangb-cats-cmas-01

Abstract

In the context of CATS applications, resource modeling and dynamic scheduling face core challenges: heterogeneous computing resources (e.g., CPUs, GPUs, FPGAs) with differentiated characteristics are difficult to unify through traditional coarse-grained metrics (e.g., virtual machine/container counts). Moreover, dynamically changing resource states (e.g., resource occupancy, service instance load cycles) complicate routing table maintenance in network nodes, creating bottlenecks for resources scheduling. This document provides a service-oriented computing capability modeling framework, abstracting heterogeneous resources into standardized service units for efficient resource modelling and traffic steering.

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 9 August 2026.

Copyright Notice

Copyright (c) 2026 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	2
2. Terminology	3
3. Computing Metrics As a Service (CMAS)	4
4. Service modelling with CMAS	6
5. Service Distribution Under CMAS	8
6. Service Consuming Under CMAS	10
7. References	11
7.1. Informative References	11
Authors' Addresses	11

1. Introduction

Computing-aware traffic steering (CATS) is a traffic engineering approach that takes into account the dynamic nature of computing resources and network state to optimize service-specific traffic forwarding towards a given service instance. As described in [I-D.ietf-cats-framework], the Computing-Aware Traffic Steering (CATS) framework assumes that there might be multiple service instances that are providing one given service, which are running in one or more service sites. Each of these service instances can be accessed via a service contact instance, which is a client-facing service function instance. A single service site may host one or multiple service contact instances. A single service site may have limited computing resources available at a given time, whereas the various service sites may experience different resource availability issues over time. Therefore, steering traffic among different service sites can address the issues of lacking resources in a specific service site. Base on this, [I-D.ietf-cats-framework] provides an architectural framework that aims at facilitating the making of compute- and network-aware traffic steering decisions in networking environments where computing service resources are deployed.

In CATS framework, C-SMA collects both computing-related capabilities and metrics, and associates them with a CS-ID that identifies the service. The C-SMA then advertises CS-IDs along with metrics to related C-PSes in the network. Computing metrics are very huge and

may change very frequently, which make them unsuitable for direct dissemination on the network. [I-D.ietf-cats-metric-definition] proposes to use normalized metrics in CATS. Level 1 and level 2 metrics are proposed to transfer on the network instead of the level 0 raw metrics. How to make level 1 and level 2 metrics to steer traffic is further to be studied.

Unlike electricity, computing metrics cannot be quantified simply in units like "kWh/kWh", especially considering the different types of CPUs, GPUs, FPGAs, ASICs and other chips, it is difficult to make a unified measurement. The concrete normalization method for computing metrics is very hard and is a key factor hindering the development of CATS. The normalization will face two challenges. The first is different service provider may use different normalization method, which will make C-PS hard to decide for a normalized metric. The second is a normalized value may lose important information of the concrete raw metrics.

To solve this problem, this draft proposes a service-oriented computing capability modeling framework, named Computing Metrics as a Service (CMAS). CMAS is a standardization approach that packages computing metrics (e.g., FLOPS, memory, latency) alongside services instead of pure resource aggregation. When deploying specific services, the service site allocates resources based on these bundled metric units, enabling efficient, service-oriented resource modelling across heterogeneous infrastructures.

2. Terminology

This document makes use of the terms defined in [I-D.ietf-cats-framework] and also makes use of the following terms:

- * Computing Metrics as a Service (CMAS): CMAS is a standardization approach that packages computing metrics (e.g., FLOPS, memory, latency) alongside services. When deploying specific services, the service site allocates resources based on these bundled metric units, enabling efficient, service-oriented resource allocation across heterogeneous infrastructures.
- * Public service platform: The public service platform hosts the complete set of CATS public services and acts as a bridge between clients and service sites. From it, service sites can download and deploy offerings, while clients can formulate and submit their service requests.

3. Computing Metrics As a Service (CMAS)

CMAS is realized through ingeniously building a public service platform, which hosts the complete set of CATS public services and acts as a bridge between clients and service sites. From it, service sites can download and deploy offerings, while clients can formulate and submit their service requests. Most importantly, in this platform, services and computing metrics are bundled, and the service site allocates resources according to the computing metric units bundled with services when deploying specific services. The lightweight service-oriented resource model can be easily build with CMAS, making it possible for CATS to be widely deployed on the Internet. The public service platform provides all public services of the CATS framework and serves as a bridge between clients and service sites, from which the service sits can download and deploy some services to provide service for clients and the clients can build their service requests. Table 1 illustrates a typical public service table in an openly searchable and browsable registry for both clients and service sites.

Service Code	Service ID	Service Name	Input	Service	Se Runi
	Computing	Storage	Computing Time	Software	
	Requirement	Requirement		Description Dependency	
hub Link			Motion Capture	This service	Git
			Voice Tracking	receives multiple	
	multi-thread CPUs	16GB DRAM	Eye Tracking	Unity,	
	AR1	AR/VR	蚬、 1ms	inputs from sensors	
	with minimum	256GB SSD	Environmental	Unreal Engine,	
	2.0GHz; Higher		Sensing	and generate scenes	
hub Link				etc.	Git
	than RTX 4060				
			Transport standard	Automation Driving	
	CPU: 蚬 · 4.0GHz,	64GB DDR5 DRAM		Apache Kafka	
	TP1	Intelligent	datas, transport	Sensing Enviroment	
	蚬 · 24MB L3 Cache,	蚬 · 1TB NVMe SSD	蚬、 20ms	Apollo	
hub Link			transportation	traffic info, etc.	Git
	GPU: 蚬 · 200 TOPS			CUDA	
			Video input source	Video Game Live	
	CPU: 蚬 · 4.5GHz, 12	蚬 · 32GB DDR5 DRAM	depending on	OBS Studio	
	LB1	Live	Audio input source	Interaction Live	
	cores; GPU: NVENC	蚬 · 5TB NVMe SSD	pecific scene	WebRTC	
hub Link			Interaction input	Sport Live	Git
	encoder	broadcase	0.5s - 3s	FFmpeg	
			speech input	real-time caption	
	CPU: 蚬 · 3.5GHz, 16	蚬 · 32GB DDR5 DRAM		CUDA/cuDNN	
	ST1	Simultaneous	(optional) action cap	conf. translation	
	threads; GPU:	蚬 · 1TB NVMe SSD	蚬、 1s	Apache Kalfa	
hub Link			interaction input		Git
	RTX 4090, FP16	蚬 · 16GB GPU DRAM			

Table 1: examples of the service table in the public service platform. The service ID represents the service.

The service name is the name of the service. The input describes the concrete information and format of the input data for the service.

The service description details the concrete function of the service. The service code contains the location of the service code.

The computing requirement lists the basic computing resources demands of the service such as CPU/GPU/NPU detailed information.

The storage requirement lists the basic storage demands of the service such as memory and disk detailed information.

The computing time describes the computing delay of the service when computing a basic data sample.

The software dependency describes the software environment for deploying the service.

The service ID is empowered to indicate a kind of service ability by the service table. The client can query the service table to find the service he may interest, build his service request using the service ID, and send the service request to its Ingress CATS-Forwarder to get the service. The service site can query the service table and find services interested, allocates resouces based on the computing and storage requirements of a service and deploy these services as service instances. A Service contact instance can be run on the service site for these service instances who provide the same service.

By ingeniously designing the public-service platform, clients can formulate their requirements in plain service-language, while service sites normalize their heterogeneous compute and storage into service-specific units only. A unified abstraction across resource types is required. The service table spells out a common resource recipe (CPU, memory, runtime) for one logical service unit. A site may therefore:

- * allocate 3 that recipe and run three AR1 service instances, or
- * allocate 4x and run four TP1 service instances, all according to its own capacity and business goals.

The computing time listed in the table is the delay measured when the basic recipe processes the basic data sample (Table 1). If a client wants faster turnaround, he simply requests more service instances (higher Gas); CATS will pick the site/instance combination whose real computing time requested delay, while keeping cost close to his stated budget.

4. Service modelling with CMAS

A service site as a public service provider or contributor can publish some specific services to the public service platform based on the fields in the service table. A service site—such as a regional cloud-computing pool—deploys services by browsing the public service platform's catalogue and applying for the specific services it intends to host. The service site models its available compute and storage resources in terms of the services it can actually deliver. Table 2 illustrates such a service model table: each row describes one service type (e.g., AR1), while the columns expose the site's current capacity, economics and contact points.

- * GAS (Global Available Slots) the total number of identical service instances that the site is willing to run concurrently. Example: GAS = 3 for AR1 means the site will keep three AR1 service instances alive, so three clients can be served simultaneously.
- * Cost per instance the site-declared price for one such slot. Rule of thumb: an edge site with scarce GPUs may set higher cost than a central cloud with abundant resources.
- * CSCI-ID a tiny proxy VM created per service, whose public IP is published as the CSCI-ID. Role: handles concrete hand-over (token exchange, redirect, health ping) so that the main service instances remains shielded from direct client traffic.

Thus, the service site realizes the service-oriented computing metrics modeling by: 1.inserts one row per service into its service model table; 2.spawns GAS identical service instances; 3.starts one proxy per service and stores its CSCI-ID into the service model table; 4.updates cost and GAS in real time as local load or hardware changes occur. This allows CATS to rank and select the most economical or closest instance for each client request, while the service site retains full control over its own pricing and capacity policies.

Service ID	Gas	Cost	CSCI-ID
AR1	3	4	IP address
TP1	6	5	IP address
LB1	2	7	IP address
ST1	1	2	IP address

Table 2: example of the service-oriented modelling table of a service site.

In this way, a service site turns its raw compute and storage capacity into service-oriented offers without ever exposing internal computing metrics to the network. Instead of publishing FLOPS, memory sizes or utilization curves, the site simply maintains and

distributes its service model table—a concise, standardized summary of how many instances of each service type it can run and at what cost.

* Initial state: the site sends the entire table to the C-SMA.

- * Subsequent changes: only the delta (new deployments, added/removed instances, price adjustments) is transmitted, keeping updates lightweight and avoiding the complex normalization of raw computing metrics.

CMAS turns the traditional flood of raw computing metrics into a single, lightweight service model table. Because the table contains only service counts and cost values, the information volume shrinks dramatically and is immediately understandable to the C-PS. Resource management inside the site is equally simplified:

- * To allocate resources for a service, the site simply increments its GAS counter by one.
- * To free resources, it decrements GAS by one.

No normalization, no complex resource aggregation, no metric flooding—just “+1” or “-1” against its own service model table.

5. Service Distribution Under CMAS

[I-D.ietf-cats-framework] describes that a C-SMA collects both computing-related capabilities and metrics, and associates them with a CS-ID that identifies the service, then advertises CS-IDs along with metrics to related C-PSes in the network. With CMAS mechanism, the C-SMA only needs to collect a minimal service-metric tuple—(Service ID, CSCI-ID, GAS, cost)—from each site. Meanwhile, the C-NMA gathers purely network-related metrics such as delay, jitter, and bandwidth; no raw computing figures are ever distributed, keeping both data collection and cross-domain orchestration lightweight and standardized.

Figure 1 illustrates how CATS metrics are disseminated under the CMAS mechanism. A client reaches the network through “CATS-Forwarder 1”. For the service identified by CS-ID “1”, two contact instances exist:

- * Instance CSCI-ID “1” at Service Site 2 (reachable via CATS-Forwarder 2)
- * Instance CSCI-ID “3” at Service Site 3 (reachable via CATS-Forwarder 3)

Additionally, two separate services (CS-ID “2” and CS-ID “3”) each have one contact instance located at Service Site 2 and Service Site 3, respectively.

CS-ID 1, CSCI-ID 1, gas, cost

CS-ID 2, CSCI-ID 2, gas, cost

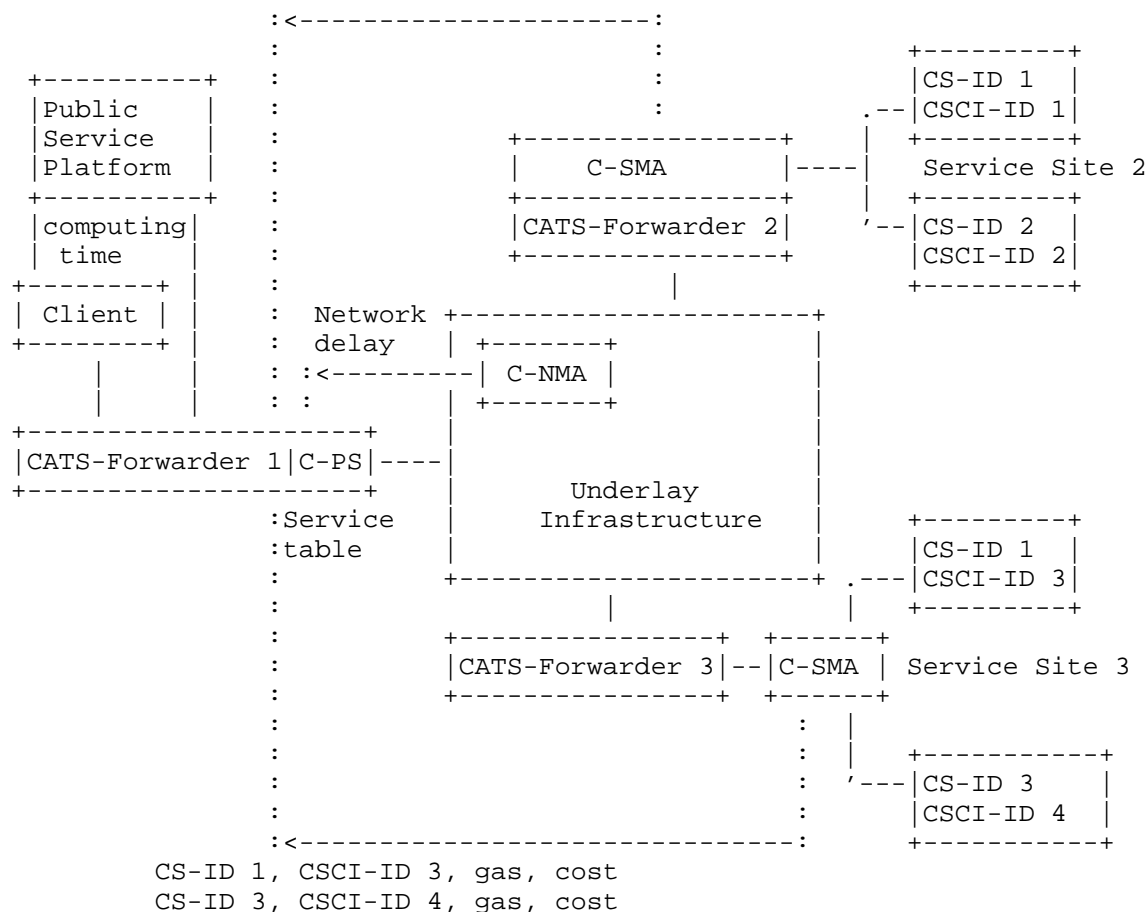


Figure 1: An Example of CATS Service Metric Dissemination Under CMAS.

The service table formed in C-PS in this example is:

(CS-ID 1, CSCI-ID 1, gas, cost, Computing time, Network delay)

(CS-ID 1, CSCI-ID 3, gas, cost, Computing time, Network delay)

(CS-ID 2, CSCI-ID 2, gas, cost, Computing time, Network delay)

(CS-ID 3, CSCI-ID 4, gas, cost, Computing time, Network delay)

In Figure 1, the C-SMA co-located with "CATS-Forwarder 2" advertises the CMAS metrics for both service-contact instances it covers—namely (CS-ID 1, CSCI-ID 1, gas, cost) and (CS-ID 1, CSCI-ID 2, gas, cost). Likewise, the C-SMA agent at "Service Site 3" publishes the metrics for the two services hosted by that site. All these service-metric advertisements are received and processed by the C-PS hosted on "CATS-Forwarder 1", which also handles the network-metric advertisements sent by the C-NMA.

Under CMAS, the C-PS can effortlessly build and maintain a unified service view for every offering. It simply collects all service metrics from the sites, appends the network metrics (here, delay) advertised by the C-NMA, and fetches each service's computing time from the public service platform. This yields a comprehensive service table in the form (Service ID, CSCI-ID, Gas, Cost, Computing time, Network delay), which we call it the whole service table.

Using this table, the C-PS selects the most suitable path to the egress CATS-Forwarder by evaluating:

- * the client's initial service request ("CS-ID 1" or "CS-ID 2"),
- * the real-time state of each service-contact instance (gas, cost, computing time), and
- * the current network state (delay and other metrics).

6. Service Consuming Under CMAS

In the example of Figure 1, the client first queries the public service platform to build its request from Table 1. The request is expressed as a 4-tuple: (Service ID, Gas, Cost, Delay). It is injected into the network through "CATS-Forwarder 1" (ingress role), which forwards it to the C-PS. The C-PS (co-located or centralised) selects the CSCI-ID that best matches the tuple by comparing:

- * Gas requested Gas,
- * Cost closest to requested Cost,
- * Real Delay requested Delay, where Real Delay = Computing Time + Network Delay taken from the whole service table.

The C-PS returns a 6-tuple response: (Service ID, CSCI-ID, Gas, Real-Cost, Real-Delay, success). When this response reaches "CATS-Forwarder 1", the forwarder notifies the client that the service is ready with a simplified acknowledgement: (Service ID, Gas, Real-Cost, Real-Delay, success). The client then:

- * pays the Real-Cost,
- * assembles input data according to the "input" schema in Table 1,
- * sends (Service ID, data) back to "CATS-Forwarder 1".

Using the CSCI-ID returned in the response, the forwarder establishes a direct data-plane tunnel between the client and the selected service-contact instance. Client data and subsequent computing results flow through this tunnel; no further routing decisions are needed.

Immediately after the tunnel is set up, the forwarder signals the C-PS to decrement the corresponding GAS value in the global service table. When the service completes, the contact instance itself sends a "service-finished" heartbeat to the C-PS, which increments GAS again, keeping the table accurate in real time.

7. References

7.1. Informative References

- [I-D.ietf-cats-usecases-requirements]
Yao, K., "Computing-Aware Traffic Steering (CATS) Problem Statement, Use Cases, and Requirements", June 2025, <<https://datatracker.ietf.org/doc/draft-ietf-cats-usecases-requirements/>>.
- [I-D.ietf-cats-metric-definition]
Kumari, W. and K. Yao, "CATS Metrics Definition", July 2025, <<https://datatracker.ietf.org/doc/draft-ietf-cats-metric-definition/>>.
- [I-D.ietf-cats-framework]
Li, C., "A Framework for Computing-Aware Traffic Steering (CATS)", July 2025, <<https://datatracker.ietf.org/doc/draft-ietf-cats-framework/>>.

Authors' Addresses

Bin Zhang (editor)
Pengcheng Laboratory
Sibilon Street
Shenzhen
518055
China
Email: bin.zhang@pcl.ac.cn

Yina Dai (editor)
Sun Yat-sen University
Sun Yat-sen Street
Guangzhou
510080
China
Email: daiyn5@mail2.sysu.edu.cn

Bowen Shen (editor)
Harbin Institute of Technology
Taoyuan Street
Shenzhen
518055
China
Email: shenbowen@stu.hit.edu.cn