bmwg
Internet-Draft
Intended status: Informational
Expires: 26 November 2026

K. Yao
P. Liu
China Mobile
G. Zeng
Huawei
X. Yi
China Unicom
Q. Xiong
ZTE
M.-N. Tran
ETRI

25 May 2026

Benchmarking Methodology for Computing-aware Traffic Steering
draft-yl-bmwg-cats-04

Abstract

Computing-aware traffic steering (CATS) is a traffic engineering approach for steering service requests towards appropriate service instances based on the awareness of both computing and network information. This document proposes benchmarking methodologies for CATS.

Table of Contents

1. Introduction

Computing-aware traffic Steering(CATS) is a traffic engineering approach considering both computing and network metrics, in order to select appropriate service instances. Some of the latency-sensitive, throughput-sensitive applications or compute-intensive applications need CATS to guarantee effective instance selection, which are mentioned in [I-D.ietf-cats-usecases-requirements]. Considering there are many computing and network metrics that can be selected for traffic steering, as proposed in [I-D.ietf-cats-metric-definition], some benchmarking test methods are required to validate the effectiveness of different CATS metrics. Besides, there are also different deployment models as described in the CATS framework [I-D.ietf-cats-framework], i.e. the distributed model, the centralized model and the hybrid model, and there are also multiple objectives for instance selection, for example, instance with lowest end-to-end latency or the highest system utilization. The benchmarking methodology proposed in this document is essential for guiding CATS implementation.

2. Definition of Terms

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

* Computing-aware Traffic Steering (CATS)
* CATS Path Selector (C-PS)
* CATS-Forwarder
* CATS Service Metric Agent (C-SMA)
* CATS Network Metric Agent (C-NMA)

3. Test Methodology

3.1. Test Setup

The test setup in general is compliant with [RFC2544]. As is mentioned in the introduction, there are basically three models for CATS deployment. The centralized model, the distributed model, and the hybrid model.

The difference primarily sits in how CATS metrics are collected and distributed into the network and accordingly, where the CATS path selector(C-PS) is placed to make decisions, as is defined in [I-D.ietf-cats-framework].

3.1.1. Test Setup - Centralized Model

Figure 1 shows the test setup of the centralized model to implement CATS. The centralized test setup is similar to the Software Defined Networking(SDN) standalone mode test setup defined in [RFC8456]. The DUT locates at the same place with the SDN controller. In the centralized model, SDN controller takes the role of the decision making for instance selection as well as traffic steering. The application plane test emulator is connected with the forwarding plane test emulator via interface 2 (I2). The SDN controller is connected to Edge server manager via interface 4 (I4). The interface (I1) of the SDN controller is connected with the forwarding plane. Service request is sent from application to the ingress CATS-Forwarder through I2. CATS metrics are collected from Edge server manager via I4. The traffic steering policies are configured through I1.

In the forwarding plane, CATS-Forwarder 1 serves as the ingress node and is connected with the host which is an application plane emulator. CATS-Forwarder 2 and CATS-Forwarder 3 serve as the egress nodes and are connected with two edge servers respectively. Both of the edge servers are connected with edge server manager via I3. I3 is an internal interface for CATS metrics collection within edge sites.

To accommodate the CATS framework, the C-PS is placed at the SDN controller. The C-SMA is placed at the edge server manager. C-NMAs are placed at all CATS-Forwarders.

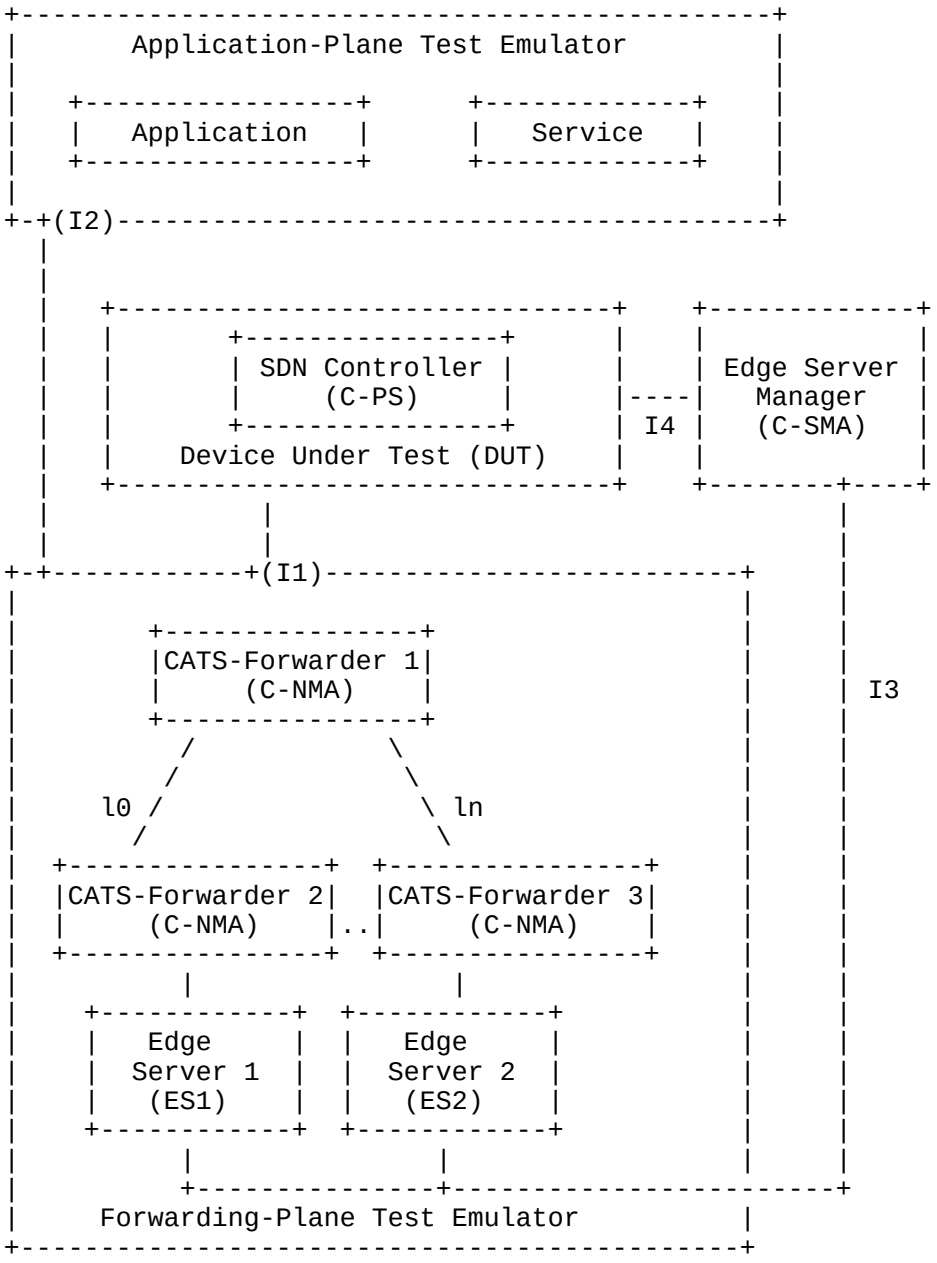

Figure 1: Centralized Test Setup

3.1.2. Test Setup - Distributed Model

Figure 2 shows the test setup of the distributed model to implement CATS. In the distributed test setup, The DUT is the group of CATS-Forwarders, since the decision maker is the CATS ingress node, namely CATS-Forwarder 1. CATS egress nodes, CATS-Forwarder 2 and CATS-Forwarder 3, take the role of collecting CATS metrics from edge servers and distribute these metrics towards other CATS-Forwarders. Service emulators from application plane is connected with the control-plane and forwarding-plane test emulator through the interface 1.

To accommodate the CATS framework, the C-PS is placed at the ingress CATS-Forwarder 1. The C-NMAs and C-SMAs are placed at all egress CATS-Forwarders (CATS-Forwarder 2 and 3).

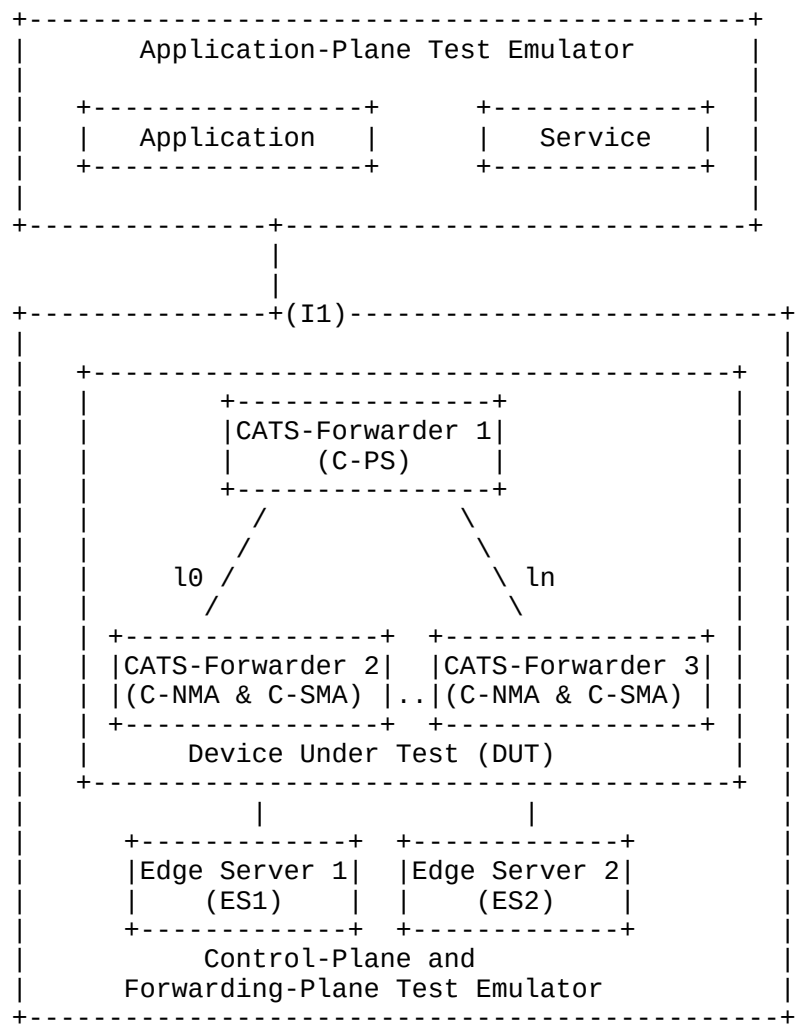

Figure 2: Distributed Test Setup

3.1.3. Test Setup - Hybrid Model

Figure 3 shows the test setup of the hybrid model to implement CATS. In hybrid model, some stable CATS metrics are distributed among involved network devices (i.e., CATS-Forwarders), while other frequent changing CATS metrics may be collected by a centralized SDN controller. At the mean time, Service scheduling function can be performed by a SDN controller and/or the ingress CATS-Forwarder. The entire or partial C-PS function may be implemented in the centralized control plane, depending on the specific implementation and

deployment.

To accommodate the CATS framework, the C-PSs are placed at the SDN controller and the ingress CATS-Forwarder 1. The C-SMAs are placed at the edge server manager and all egress CATS-Forwarders, The C-NMAs are placed at all egress CATS-Forwarders (CATS-Forwarder 2 and 3) too.

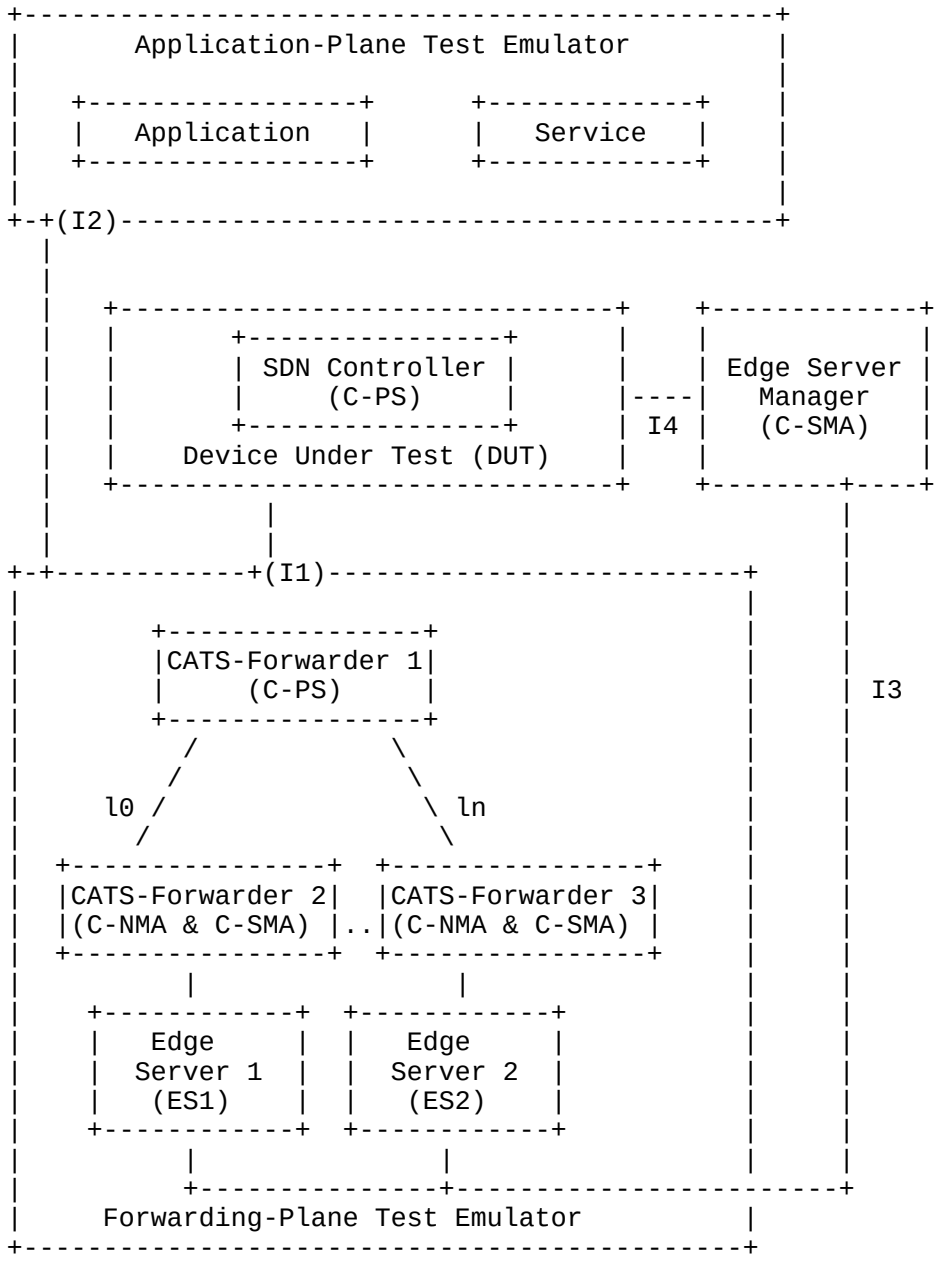

Figure 3: Hybrid Test Setup

3.2. Control Plane and Forwarding Plane Support

In the centralized model, Both of the control plane and forwarding plane follow Segment Routing pattern, i.e. SRv6[RFC8986]. The SDN controller configure SRv6 policies based on the awareness of CATS metrics and traffic is steered through SRv6 tunnels built between CATS ingress nodes and CATS egress nodes. The collection of CATS metrics in control plane is through Restful API or similar signalling protocols built between the SDN controller and the edge server manager.

In the distributed model, in terms of the control plane, EBGP[RFC4271] is established between CATS egress nodes and edge servers. IBGP[RFC4271] is established between CATS egress nodes with CATS ingress nodes. BGP is chosen to distribute CATS metrics in network domain, from edge servers to CATS ingress node. Carrying CATS metrics is implemented through the extension of BGP, and [I-D.ietf-idr-5g-edge-service-metadata] provides some examples by leveraging sub-TLVs extensions.

In the hybrid model, the metric distribution follows the control plane settings in both centralized and distributed model, according to the actual choices in what metrics are required to be distributed centrally or distributedly.

In terms of the forwarding plane, SRv6 tunnels are enabled between CATS ingress nodes with CATS egress nodes.

Service flows are routed towards service instances by following anycast IP addresses in all of the models.

3.3. Topology

In terms of all of the approaches to test CATS performance in laboratory environments, implementors consider only single domain realization, that is all CATS-Forwarders are within the same AS. There is no further special requirement for specific topologies.

3.4. Device Configuration

Before implementation, there are some pre-configurations need to be settled.

** Application plane Setup

Application plane functionalities must be setup in edge servers before the implementation, and hosts that send service requests must also be setup.

** CATS Metrics Collector Setup

In the centralized model and the hybrid model, the CATS metrics collector need to be first setup in the edge server manager. A typical example of the collector can be the monitoring components of Kubernetes. It can periodically collect different levels of CATS metrics. Then the connecton between the edge server manager and the SDN controller must be established, one example is to set restful API or ALTO protocol for CATS metrics publication and subscription.

In the distributed model and the hybrid model, the CATS metrics collector need to be setup in each edge site. In this benchmark test, the collector is setup in each edge server which is directly connected with a CATS egress node. Implementors can use plugin software to collect CATS metrics. Then each edge server must set BGP peer with the CATS egress node that’s directly connected. In each edge server, a BGP speaker is setup.

** Control Plane and Fordwarding Plane functionality Setup

In the centralized model and the hybrid model, the SDN controller need to be pre-configured and the interface between the SDN controller and CATS-Forwarders must be tested to validate if control plane policies can be correctly downloaded and it metrics from network side can be correctly uploaded. In the distributed model and the hybrid model, the control plane setup is the iBGP connections between CATS-Forwarders. For both models. the forwarding plane functions, SRv6 tunnels must be pre-established and tested.

4. Reporting Format

CATS benchmarking tests focus on data that can be measured and controllable.

* Control plane configurations:
  - SDN controller types and versions;
  - northbound and southbound protocols.
* Forwarding plane configurations:
  - forwarding plane protocols (e.g., SRv6);
  - the number of CATS-Forwarders;
  - the number of edge servers;

- the number of links;

- edge server types, versions.

* Application plane configurations:

  - Traffic types and configurations.

* CATS Metrics: Each test must clearly state what CATS metrics it use for traffic steering, according to the CATS metrics definition in [I-D.ietf-cats-metric-definition].

  - For Level 0 metrics, benchmarking tests must declare metric types, units, statistics(e.g, mean, max, min), format, and metric sources(e.g, nominal, estimation, aggregation).

  - For Level 1 metrics, benchmarking tests must declare metric types, statistics, units, format, aggregation functions, and metric sources.

  - For the Level 2 metric, benchmarking tests must declare metric type, normalization functions, and metric source.

  For all metric levels, benchmarking tests must report the Measurement_Window parameter used for metric collection.

Detailed normalization functions and aggregation functions are listed in appendix A.

The recommended metric levels are listed in Figure 4 for the benchmark tests that are described in Section 5:

Test Objective	Recommended Metric Level	Example Metric_Type
CATS Metrics Collection and Distribution	All Metric Levels	compute_CPU_util or level1_composed or level2_global
Session Continuity	Level 1 or Level 2	level1_composed or level2_global
End-to-end Service Latency	Level 1 or Level 2	level1_composed or level2_global
System Utilization	Level 1	level1_computing
Load Balancing Variance	Level 1 or Level 2	level1_computing or level2_global

Figure 4: Mapping between Benchmarking Tests and Metric Levels

5. Benchmarking Tests

5.1. CATS Metrics Collection and Distribution

* Objective: To determine that CATS metrics can be correctly collected and distributed to the DUTs which are the SDN controller in the centralized model and the CATS ingress node in the distributed model, as anticipated within a pre-defined time interval for CATS metrics update.

* Procedure:

In the centralized model and the hybrid model, the edge server manager periodically grasp CATS metrics from every edge server that can provide CATS service. Then it passes the information to the SDN controller through publish-subscription methods. Implementors then should log into the SDN controller to check if it can receive the CATS metrics from the edge server manager.

In the distributed model and the hybrid model, the collectors within each edge server periodically grasp the CATS metrics of the edge server. Then it distributes the metrics to the CATS egress node it directly connected. Then Each CATS egress node further distributes the metrics to the CATS ingress node. Implementors then log into the CATS ingress node to check if metrics from all edge servers have been received.

For all of these approaches, to test whether metrics can be received within the pre-defined time interval, implementors could compare the timestamp when receiving the current metric and the timestamp when the last metric arrives from the logs. If the time difference is exactly equal to the pre-defined metric update time interval, then CATS metrics collection is correct.

* Expected results:

** All CATS metrics are correctly received without loss or error.

** The interval between consecutive metric updates matches the predefined value.

5.2. Session continuity

* Objective: To determine that traffic can be correctly steered to the selected service instances and TCP sessions are maintained for specific service flows.

* Procedure: Enable several hosts to send service requests. In distributed model, log into the CATS ingress node to check the forwarding table that route entries have been created for service instances. Implementors can see that a specific packet which hits the session table, is matched to a target service intance. Then manually increasing the load of the target edge server. From the host side, one can see that service is going normally, while in the interface of the CATS-Forwarder, one can see that the previous session table aging successfully which means CATS has steer the service traffic to another service instance.

In the centralized model and the hybrid model, implementors log into the management interface of the SDN controller and can check routes and sessions.

* Expected results:

** Traffic is correctly forwarded to the selected edge server without blackholing.

** Session entries are dynamically created and properly aged.

** When the target server is overloaded, traffic is smoothly switched to another instance.

** No session drop or service interruption occurs during steering.

5.3. End-to-end Service Latency

* Objective: To determine that CATS works properly under the pre-defined test condition and prove its effectiveness in service end-to-end latency guarantee.

* Procedure: Pre-define the CATS metrics distribution time to be T_1 seconds. Enable a host to send service requests. In distributed model, log into the CATS ingress node to check if route entries have been successfully created. Suppose the current selected edge server is ES1. Then manually increase the load of ES1, and check the CATS ingress node again. The selected instance has been changed to ES2. CATS works properly. Then print the logs of the ingress CATS-Forwarder to check the time it updates the route entries. The time difference delta_T between when the new route entry first appears and when the previous route entry last appears should equals to T_1. Then check if service SLA can be satisfied.

In the centralized model and the hybrid model, implementors log into the management interface of the SDN controller and can check routes and sessions.

* Expected results:

** C-PS updates routing entries within delta_T, which is approximately equal to T1.

** End-to-end latency remains within the predefined SLA threshold.

** Latency after steering is not higher than before steering.

** Service performance is stable during path switching.

5.4. System Utilization

* Objective: To determine that CATS can have better load balancing effect at server side than simple network load balancing mechanism, for example, Equal cost multi-path routing (ECMP).

* Procedure: Enable several hosts to send service requests and enable ECMP at network side. Then measure the bias of the CPU utilization among different edge servers in time duration delta_T_2. Stop services. Then enable the same number of service requests and enable CATS at network side(the distributed model, the centralized model, and the hybrid model are tested separately.). Measure the bias of the CPU utilization among the same edge servers in time duration delta_T_2. Compare the bias value from two test setup.

* Expected results:

** CATS reduces CPU utilization bias compared to ECMP, and edge server resources are used more evenly.

** No server is overloaded while others are underutilized.

** Overall system resource utilization is improved.

5.5. Load Balancing Variance

* Objective: To test the load balancing variance under different path selection algorithms, which could evaluate the traffic steering effectiveness of these algorithms. Low variance value means the algorithm performs better for traffic steering. Algorithms that are compared include ECMP, global-min, and Proportional-Integral-Derivative (PID).

* Procedure: Three different path selection algorithm are tested one-by-one. In distributed model, pre-configure the control plane function C-PS in the ingress CATS-Forwarder, while in the centralized and hybrid model, the path selection function is configured in the SDN controller. For each test round, implementors initiate the same number of service flows to multiple service edge sites. For example, the number of service flows are set to M, while the number of service edge sites is N. Implementors need to record the number of service flows at each site, and calculate the load balancing variance, according to the following equations:

** n_avg = (n_s1 + n_s2 + .. + n_sN) / N **

** var_alg = sqrt(sum_{i=1 to N} (n_si - n_avg)^2 / N) **

Where 'n_s1', 'n_s2', ..., 'n_sN' refer to the number of service flows that are steered to the corresponding edge site, while 'n_avg' refers to the average number of service flows per site. 'var_alg' is the standard deviation of service flows across all N edge sites, which is used to evaluate the load balancing effectiveness of each algorithm. A lower standard deviation indicates better load balancing, as traffic is more evenly distributed among edge sites.

* Expected Results:

** CATS algorithms (global-min, PID) show lower variance than ECMP.

** Traffic is distributed evenly without flow loss or service degradation.

6. Security Considerations

The benchmarking characterization described in this document is constrained to a controlled environment (as a laboratory) and includes controlled stimuli. The network under benchmarking MUST NOT be connected to production networks. Beyond these, there are no specific security considerations within the scope of this document.

7. IANA Considerations

This document has no IANA actions.

8. Acknowledgements

9. References

9.1. Normative References

[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-24, 2 April 2026, <https://datatracker.ietf.org/doc/html/draft-ietf-cats-framework-24>.

[I-D.ietf-cats-metric-definition]
Yao, K., Li, C., Contreras, L. M., Ros-Giralt, J., and G. Zeng, "CATS Metrics Definition", Work in Progress, Internet-Draft, draft-ietf-cats-metric-definition-08, 15 May 2026, <https://datatracker.ietf.org/doc/html/draft-ietf-cats-metric-definition-08>.

[RFC2544] Bradner, S. and J. McQuaid, "Benchmarking Methodology for Network Interconnect Devices", RFC 2544, DOI 10.17487/RFC2544, March 1999, <https://www.rfc-editor.org/rfc/rfc2544>.

[RFC4271] Rekhter, Y., Ed., Li, T., Ed., and S. Hares, Ed., "A Border Gateway Protocol 4 (BGP-4)", RFC 4271, DOI 10.17487/RFC4271, January 2006, <https://www.rfc-editor.org/rfc/rfc4271>.

[RFC8456] Bhuvaneswaran, V., Basil, A., Tassinari, M., Manral, V., and S. Banks, "Benchmarking Methodology for Software-Defined Networking (SDN) Controller Performance", RFC 8456, DOI 10.17487/RFC8456, October 2018, <https://www.rfc-editor.org/rfc/rfc8456>.

[RFC8986] Filsfils, C., Ed., Camarillo, P., Ed., Leddy, J., Voyer, D., Matsushima, S., and Z. Li, "Segment Routing over IPv6 (SRv6) Network Programming", RFC 8986, DOI 10.17487/RFC8986, February 2021, <https://www.rfc-editor.org/rfc/rfc8986>.

9.2. Informative References

[I-D.ietf-cats-usecases-requirements] Yao, K., Contreras, L. M., Shi, H., Zhang, S., and Q. An, "Computing-Aware Traffic Steering (CATS) Problem Statement, Use Cases, and Requirements", Work in Progress, Internet-Draft, draft-ietf-cats-usecases-requirements-14, 2 February 2026, <https://datatracker.ietf.org/doc/html/draft-ietf-cats-usecases-requirements-14>.

[I-D.ietf-idr-5g-edge-service-metadata] Dunbar, L., Majumdar, K., Li, C., Mishra, G. S., and Z. Du, "BGP Extension for 5G Edge Service Metadata", Work in Progress, Internet-Draft, draft-ietf-idr-5g-edge-service-metadata-32, 27 April 2026, <https://datatracker.ietf.org/doc/html/draft-ietf-idr-5g-edge-service-metadata-32>.

Appendix A. Aggregation and Normalization Functions

This appendix specifies the detailed aggregation functions (categorized by time-scale, spatial, and cross-category) and normalization functions for CATS metrics, aligned with [I-D.ietf-cats-metric-definition].

A.1. Time-scale Aggregation

Time-scale aggregation consolidates metric values over a specified time window to reflect temporal trends (e.g., average computing load over 1 minute). The following functions are defined for CATS benchmarking:

** Time Average

Time_Average = (Sample_1 + Sample_2 + ... + Sample_n) / n

** Time Maximum

Time_Max = Maximum value of all samples in the time window

** Time Minimum

Time_Min = Minimum value of all samples in the time window

A.2. Spatial Aggregation

Spatial aggregation consolidates metric values across multiple compute entities (e.g., multiple edge servers) to reflect regional/ global status.

** Spatial Sum

Spatial_Sum = Entity_1_Value + Entity_2_Value + ... + Entity_m_Value

** Spatial Average

Spatial_Average = (Entity_1_Value + Entity_2_Value + ... + Entity_m_Value) / m

** Spatial Median

Spatial_Median = Middle value of sorted entity values

A.3. Cross-category Aggregation

Cross-category aggregation combines metrics from different categories (e.g., type 1: CPU load; type 2: GPU memory) to generate metrics with level up for CATS decision-making. This documents adopts linear combination as the cross-category aggregation function.

** Linear Combination

Linear_Combination = alpha * Metric_1 + beta * Metric_2 + ... (alpha + beta + ... = 1)

A.4. Normalization

Normalization scales CATS metrics to a uniform range (0-10) using min-max scaling, ensuring comparability while ensuring simplicity. The functions are defined as:

** Standard Normalization

Normalized_Value = 10 * (Raw_Value - Min_Value) / (Max_Value - Min_Value)

** Inverse Normalization

Inverse_Normalized_Value = 10 * (Max_Value - Raw_Value) / (Max_Value - Min_Value)

Authors' Addresses

Kehan Yao
China Mobile
Email: yaokehan@chinamobile.com

Peng Liu
China Mobile
Email: liupengyjy@chinamobile.com

Guanming Zeng
Huawei
Email: zengguanming@huawei.com

Xinxin Yi
China Unicom
Email: yixx3@chinaunicom.cn

Quan Xiong
ZTE
Email: xiong.quan@zte.com.cn

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