Srvcast: Facilitating Host-transparent and Stateful Anycast for Computing-Aware Networks

Heyao Zhang[1], Bo Lei[2], Shuai Gao*[1], Weiting Zhang[1], Yuming Zhang[1], Cheng Chi[3], Hongke Zhang[1]
[1]Beijing Jiaotong University, The School of Electronic and Information Engineering, Beijing, China
[2]China Telecom Corporation Limited Research Institute, Beijing, China
[3]China Academy of Information and Communications Technology, Beijing, China
Email:[1]{23111018, shgao, wtzhang, ymzhang3, hkzhang}@bjtu.edu.cn, [2]leibo@chinatelecom.cn, [3]chicheng@caict.ac.cn

***Abstract*—6G-driven compute-intensive applications require the collaboration of communication and computing to achieve optimal performance. Such collaboration drives integrated sensing, communication, and computing to support service requirement sensing and on-demand computing task steering within the network. To this end, the Computing-Aware Network proposes to incorporate computing information into the network layer address called service identifier (SID), to implement service-oriented SID anycast. This integration aims to naturally support dynamic task steering using the anycast mechanism. However, SID represents an abstract service rather than a specific host, making SID anycast incompatible with the TCP communication patterns used by existing socket-based applications. To address this challenge, this paper introduces Srvcast, a host-transparent and stateful anycast solution. Srvcast consists of two stages: WAN routing and edge network forwarding. In WAN routing, it employs a novel service-oriented routing mechanism to ensure connection affinity for anycast. In the edge network forwarding, Srvcast presents ServiceNAT, a P4-based address translation mechanism that enhances SID compatibility with socket-based applications. To implement Srvcast, a prototype system is built in a practical WAN environment. The results demonstrate that Srvcast outperforms existing solutions in terms of system complexity and socket connection establishment time. Srvcast can maintain the flexibility of SID anycast while ensuring compatibility with TCP communication patterns at a lower cost.**

***Index Terms*—6G, integrated sensing, communication, and computing, computing-aware network, stateful anycast, P4**

I. Introduction

In the 6G outlook, compute-intensive applications, such as holographic communications and 3D virtual reality, require integrated sensing, communication, and computing to dynamically sense their service requirements and then steer the computing task flows to the appropriate places for optimal performance [1], [2]. To facilitate this, the IETF Computing-Aware Traffic Steering working group proposes the Computing-Aware Network (CAN) to build a general framework for the distribution of compute and network metrics and transport of traffic from network edge to computing entities [3], [4].

The CAN proposes embedding computing information into the network-layer address to construct a host-independent service identifier (abbreviated as SID), enabling the network to describe various computing services. Additionally, advanced forwarding and routing mechanisms can be developed using SID to steer the computing task flows to suitable places. Based on these proposals, several studies suggest combining SID with scalable anycast mechanism to achieve efficient and dynamic one-to-any service-oriented communication [5]–[7].

However, the dynamic feature provided by SID anycast creates conflicts with existing socket-based applications, as reflected in two issues: **1) SID lacks socket compatibility:** SID represents an abstract service rather than a specific host, making it incapable of establishing host-based TCP sockets. Yet, most compute-intensive applications rely on TCP for reliable data transmission. For example, distributed AI training requires lossless transmission to achieve model accuracy [8]. Similarly, cloud gaming demands stateful interactions between the host and the server. **2) Poor connection affinity for anycast:** Since the anycast mechanism selects destinations based on minimal metrics, combining this stateless communication paradigm with SID complicates the maintenance of stable connections, particularly when dealing with fluctuating computing resources. Previous studies have examined its potential negative through probabilistic analysis [9], [10].

To address these limitations, existing solutions mainly focus on unicast acquisition and proxy deployment. The unicast acquisition extends existing protocols to obtain server addresses for anycast destinations and then uses unicast to conduct stateful communication [11]–[13]. However, these approaches involve rewriting existing applications and protocols, making large-scale deployment challenging. Additionally, obtaining unicast addresses is difficult, as servers may be reluctant to share their addresses due to security, privacy, and other concerns. In contrast, proxy deployment offloads anycast functionality to proxies, which can be either servers or routers [9], [10], [14]. These proxies handle anycast queries and responses from nearby clients or servers, establishing proxy tunnels to forward anycast packets. However, such solutions are protocol-specific and require frequent configurations, limiting flexibility.

Therefore, this paper proposes *Srvcast*[1], a host-transparent and stateful anycast solution composed of two stages. The first stage focuses on wide area network (WAN) routing. Srvcast designs a service-oriented routing mechanism using anycast

This work was supported by the National Natural Science Foundation of China (No. 62394324 and No. 61972026), and the Beijing Natural Science Foundation (No. 4242008).

*Corresponding author.

[1]The source code is available at https://github.com/zhy1658858023/Srvcast

ICC 2025 - IEEE International Conference on Communications | 979-8-3315-0521-9/25/$31.00 ©2025 IEEE | DOI: 10.1109/ICC52391.2025.11161967

and segment routing (SR) to improve connection affinity for SID anycast [15]. The second stage targets edge networks (e.g., local area networks) forwarding. Srvcast designs a P4-based address translation mechanism for SID (called ServiceNAT), which ensures seamless compatibility between SID and socket-based applications without modifications [16]. Srvcast supports stateful anycast by incrementally deploying edge gateways, ensuring transparency with existing networks. Additionally, it uses Software-defined Networking (SDN) controllers to manage the gateway [17], enabling flexible configuration. Our contributions are summarized as follows:

- A service-oriented routing mechanism based on anycast and SR is designed to enable stateful SID task steering over the WAN. The mechanism utilizes anycast to route the first-initiated SID tasks to the optimal resource and then maintains the session until its end using SR tunnels.
- We propose ServiceNAT, which leverages the flexible and protocol-independent P4 for data plane functionality. ServiceNAT is capable of assigning tasks based on computing resources and translating SID to the server's host IP to transparently support socket connections.
- Srvcast is implemented and evaluated on a prototype system within a practical WAN environment using China Telecom's large-scale innovation infrastructure. Experimental results demonstrate that Srvcast outperforms existing solutions in terms of system complexity and socket connection establishment time at a lower cost.

The remainder of the paper is organized as follows: Section II outlines our motivation behind Srvcast and provides its architecture. Section III illustrates the design details. Section IV presents the implementations and evaluation of Srvcast. Section V presents our conclusions and future work.

II. Architecture Design

A. Our Motivations

Our goal is to improve the availability of SID anycast for service-oriented communication. We aim to achieve this by: 1) ensuring compatibility with existing socket-based applications to support TCP functionalities, and 2) preserving the dynamic nature of the SID anycast communication for the invocation of computing resources. However, achieving this is challenging, as it requires a system that meets the following requirements:

1) Backwards Compatibility: Maintaining transparency for clients and routers is crucial for the success of new anycast systems. Since Srvcast is a SID anycast enhancement mechanism based on existing protocols, it provides host-transparent without modifying current applications and routers.

2) Stateful Communication: SID anycast faces significant deficiencies in socket compatibility and connection affinity. Srvcast designs service-oriented routing and ServiceNAT to address these limitations and enable stateful communication.

3) Dynamic Network Steering: The primary goal of the CAN using SID anycast is to achieve dynamic computing task steering. Srvcast retains this advantage by utilizing a customized routing mechanism based on anycast and SR.

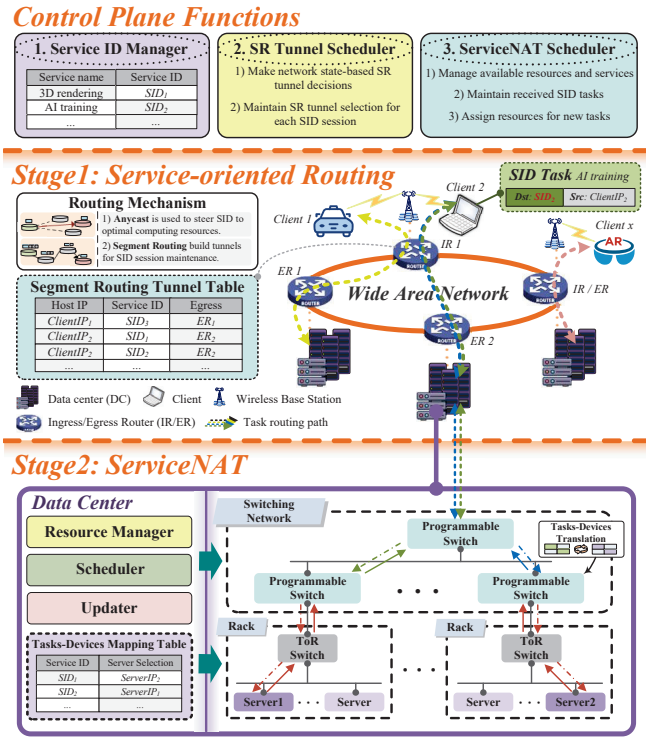


Fig. 1. Architecture of Srvcast. The blue and green lines describe the SID packet paths, which use SID as the packet source or destination address. The red lines indicate that the packet has been matched with a specific server and uses its host IP as the packet source or destination address.

4) Incremental Deployment: Given the high costs of deploying dedicated hardware in a WAN, Srvcast is designed to operate within the edge network, where deploying a limited number of devices can effectively support its functionality.

B. Architecture of Srvcast

In Srvcast, we focus on a scenario where multiple servers in different data centers provide various computing services to handle client SID tasks across the WAN. The architecture of Srvcast as shown in Fig.1 consists of two stages: *Service-oriented Routing* and *ServiceNAT*, which are guided by the control plane functionality. Before detailing each stage, we define the following terms: *1) Client:* A compute-intensive application within the edge network that uses SID to publish computing tasks. It is accessed by the wireless base station. *2) Ingress Routers (IRs):* Routers that receive packets directly from nearby *clients* (we assume each IR has optimal access by wireless base stations), supporting anycast and SR. *3) Egress Routers (ERs):* Gateways to data centers, with the same capabilities as the *IRs*. *4) Server:* A service entity within a data center that executes client computing tasks.

Service-oriented Routing: This stage focuses on the WAN. Srvcast steers SID tasks to the optimal data center and maintains the connection until the session ends. IRs and ERs are deployed at various locations within the WAN. To propagate available services, ERs regularly advertise SID reachability to their peers. IRs store entries from different ERs as potential anycast destinations. Srvcast introduces a service-oriented

routing mechanism that forwards first-initiated SID tasks using anycast to steer them to optimal computing resources. To support stateful anycast, it records the relationship between the client's tasks and their ER selections, then employs an SR-based tunnel mechanism to steer the ongoing tasks.

ServiceNAT: This stage targets the data center (DC). Srvcast introduces ServiceNAT to support server selection and host-oriented socket connections for SID tasks. Inside the DC, servers are connected via a ToR switch, and ToR switches are interconnected via the switching network. The switching network consists of programmable switches. ServiceNAT operates within these programmable switches, selecting the appropriate server for each new task based on available computing resources. To support stable TCP socket connections, ServiceNAT maintains task-to-server mappings and automatically performs address translation between SID and the server's IP.

Control Plane Functions: Srvcast employs three main functions for information management. The *Service ID Manager* records service information from DC and generates SID addresses. It periodically advertises SID reachability to IRs, ERs, and clients. The *SR-Tunnel Scheduler* operates in stage 1. It maintains the SID anycast entries and client's ER selections and makes routing decisions for SID tasks forwarding. The *ServiceNAT Scheduler* works in stage 2, which manages available resources and services within the DC. Built on this, it assigns the optimal server for tasks and then determines the Switching Network behavior based on the strategies.

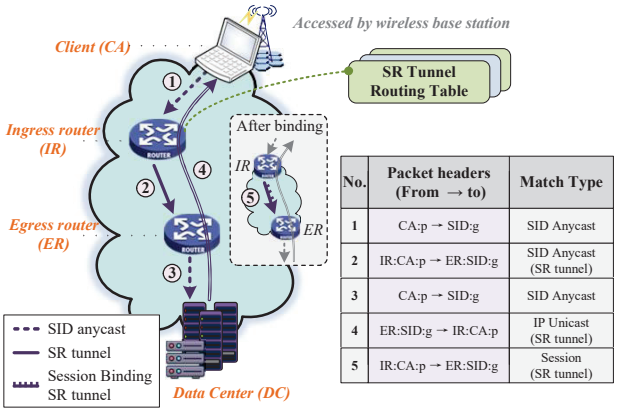


Fig. 2. Service-oriented Routing in WAN.

TABLE I
EXAMPLES OF THE SR TUNNEL ROUTING TABLE

Match Key	Tunnel Egress	Metric	Match type
SID_1, CA_1	ER_1	-	exact
SID_2, CA_1	ER_3	-	exact
...	...	-	exact
SID_i, CA_j	ER_n	-	exact
SID_1	ER_1	20	anycast
	ER_3	70	
	ER_n	35	
SID_i	...	...	anycast

III. DESIGN DETAILS

This section details the design of Srvcast. We begin with WAN service-oriented routing, explaining how Srvcast ensures stateful task steering while enabling dynamic SID anycast. Next, we introduce DC ServiceNAT, an address translation solution that transparently supports TCP communications.

A. Stage 1: Service-oriented Routing

In the WAN, the Srvcast overlay infrastructure consists of two node types: IRs and ERs, which are specialized routers with the capabilities outlined in Section II. These routers periodically synchronize SID routing information with peers. IRs and ERs advertise their proximity to the SID address range within the underlying routing system to ensure that packets destined for a SID address traverse the overlay network. When a client initiates a SID task, the nearest IR registers the task and selects the optimal ER to forward its packets. The ER then determines the most suitable data center to handle the task. Srvcast minimizes the impact of SID entries on the existing WAN by deploying IRs and ERs at the network edge and employing tunnels for WAN routing. Additionally, to meet the demands of load balancing and resource acquisition for service-oriented communication, Srvcast employs segment routing as a network tunneling mechanism.

Generally, the client's selection of IR remains constant throughout the entire service session. The ER considers the directly connected data center as optimal and does not redirect the tunnel due to fluctuations. Fig.2 illustrates the steps involved in setting up a stateful SID session between the client and the target data center based on these assumptions. The client initiates a session by sending a packet with the SID address, which is steered to the nearest IR (step 1). Leveraging the anycast mechanism, the IR matches the appropriate egress and transmits the packet through the SR tunnel (step 2). When the packet arrives at the ER, it is forwarded to the data center for processing (step 3). The return path (step 4) uses unicast IP, with the client address (CA) as the packet destination. However, the SR tunnel cannot be avoided on the return path because both the IR and ER need to monitor each session state. The SR tunnel is necessary in both directions (request and reply) for the IR and ER to maintain the session.

The above mechanism does not support stateful sessions, since steps 2 and 4 cannot guarantee that subsequent packets from the same session arrive at the same destination. To address this issue, Srvcast introduces *Service-oriented Routing* mechanism, a function extension that combines anycast and SR to route SID packets based on sessions. Table I shows the two major entry types within the routing table of the mechanism. The first type is *session entries*, which are used to maintain tunnel selection for each SID session. During the initial session (i.e., steps 1 to 4), the IR and ER utilize (*SID*, *CA*) to describe a specific session and record its ER selection. When subsequent packets arrive at the routers, they are matched with session entries (step 5) to maintain a stateful session. The second type is *anycast entries*, which

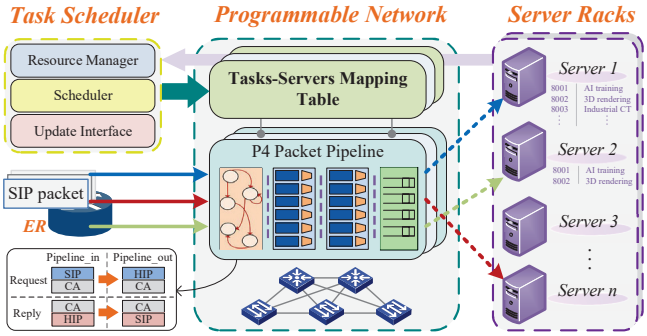


Fig. 3. ServiceNAT: DC task scheduler.

supports SID anycast routing. When a router receives SID packets, it first matches the session entries to guarantee stateful connection for ongoing sessions and then matches the anycast entries to steer the new SID task to the optimal DC.

Unlike normal IP entries, which only match the destination address, SID session entries also require the source address to differentiate clients and have a higher matching priority than anycast entries. To address these requirements, we integrate Srvcast into the SDN system. By leveraging southbound interfaces (e.g., OpenFlow [18]), the SDN controller can flexibly manage service-oriented routing. Additionally, to enhance SID routing table scalability, Srvcast uses hash indexes to match SID entries [19]. When the controller receives SID packets, it hashes the key (i.e., SID and CA) to generate an index and then queries the value in the corresponding address.

Scalability Discussion: Service-oriented routing only needs to be deployed in edge gateways, remaining transparent to the existing WAN infrastructure. In the control plane, the controller delegates functions such as session maintenance to the gateway, focusing on session assignment of new SID tasks and resource sharing between network regions. This deployment aligns with the traditional Internet approach for scaling and maintenance, supporting large-scale expansion.

Service-oriented routing enables stateful WAN routing of SID sessions while retaining the dynamic feature of anycast for resource acquisition. Although session entry matching adds complexity, experimental results demonstrate that the impact remains minimal, even as the number of entries increases.

B. Stage 2: ServiceNAT

When SID packets arrive at a data center, ServiceNAT assigns the optimal server for new tasks and steers the subsequent packet paths to maintain a stateful session. Additionally, it supports host-oriented socket connections through address translation between SID and the server's host IP. Fig.3 depicts the functionality and collaboration of ServiceNAT components in SID packet processing, which contains three parts.

Server Racks: The server racks are connected via the programmable network and contain multiple servers that handle client requests. These servers create sockets using their Host IP (HIP) and differentiate services by port number. For a given service, its listening port is consistent across different servers.

TABLE II
EXAMPLES OF THE TASKS-SERVERS MAPPING TABLE

Entry	Service ID	Client Address	Server HIP & Port
L_1	SID_1	CA_1	HIP_1 & $Port_2$
L_2	SID_2	CA_2	HIP_6 & $Port_1$
L_3	SID_3	CA_1	HIP_2 & $Port_3$
L_n	...	...	...

TABLE III
KEY NOTATIONS

Notations	Definition
$Dst/SrcIP$	the destination/source IP address in the packet
$Dst/SrcPort$	the destination/source TCP port in the packet
L_i	the i^{th} entry in the tasks-servers mapping table
P_i	a switch port connected to the L_i server
P_S	the switch ports connected to servers
P_R	a switch port that received the packet
P_C	a control port of the switch
P_G	the switch port that receives client requests
Psd	the pseudo-header in TCP checksum calculations

However, this consistency cannot be used to characterize services in our subsequent schemes, because different services possibly use the same port to listen to client requests.

Programmable Network: The programmable network executes the decisions from the task scheduler, forwarding SID packets to the assigned server for processing. To establish stateful SID connections, the programmable network utilizes a *tasks-servers mapping table* to maintain server selection for each session and enables host-oriented connections through network address translation.

Task Scheduler: The task scheduler periodically detects available services and resources of the servers. Based on this, it utilizes classification methods (e.g., linear classifier, artificial neural network) to assign optimal servers for new tasks. Additionally, the task scheduler generates and updates the entries in *task-server mapping table* to control the actions of the programmable network.

The tasks-servers mapping table, as depicted in Table II, maintains the server assignments for SID tasks. Each entry contains three main parameters—Service ID (SID), Client Address (CA), and Server's Host IP (HIP)— which are all IP addresses but serve distinct roles. Firstly, the entries characterize each task based on SID and CA. The SID, as the service label at the network layer, represents the computing service required by the task, while the CA identifies the client's host address. Secondly, the HIP, along with the Port, describes the address and port information of the assigned server.

SID task scheduling differs significantly from traditional task scheduling, necessitating data plane programmability for customized packet processing. Therefore, ServiceNAT leverages the protocol-independent P4 language to design the SID task processing logic. To simplify the description, Table III summarizes the additional notations used in ServiceNAT.

ServiceNAT utilizes user-defined metadata and actions to facilitate SID header processing. Algorithm 1 illustrates the

Algorithm 1 A P4-based address mapping algorithm

Define: based on the notations in Table II and Table III;
P4 Pipeline: following the PISA Architecture;
// **Parser**
if (the packet is Valid) **then**
  extract Ethernet, IP, TCP header; get $P_R \leftarrow$*ingress.port*;
  get IP header.$\langle Dst/SrcIP \rangle$, TCP header.$\langle SrcPort \rangle$;
  if (IP header $\in$ SID) **then**
    performs stateful SID match-action pipeline;
  else
    performs normal IP forwarding;
  end if
else
  drop the packet;
end if
// **Stateful SID Match-Action Pipeline**
if ($P_R == P_G$, a request from client) **then**
  match ($L_i \in L$) **do**
  if ($\exists i$, $\langle SID_i, CA_i \rangle == \langle DstIP, SrcIP \rangle$) **then**
    set $DstIP \leftarrow HIP_i$, *egress.port* $\leftarrow P_i$; break;
  else
    set *egress.port* $\leftarrow P_C$ to trigger update; break;
  end if
else if ($P_R \in P_S$, a reply from server) **then**
  match ($L_j \in L$) **do**
  if ($\exists j$, $\langle HIP_j, Port_j \rangle == \langle SrcIP, SrcPort \rangle$) **then**
    set $SrcIP \leftarrow SID_j$, *egress.port* $\leftarrow P_j$; break;
  else
    no-action or drop; break;
  end if
end if
// **Checksum**
set $Psd \leftarrow$ new IP.$\langle DstIP, SrcIP \rangle$, TCP.length;
TCP.checksum $\leftarrow$ Hash.csum16(Psd, TCP hdr, payload);
// **Deparser**
reassemble the new packet; emit to *egress.port*;

P4 pipeline design, which includes the following functions:

Packet Parsing and Deparsing: When the programmable network receives SID packets, the P4 parser extracts the required fields and relevant metadata from the header to support user-defined matches and actions. The P4 deparser reassembles essential headers into the packet for output format declaration. The parser determines the subsequent behavior of the switch based on the IP header type (i.e., sensing task). Specifically, we set a 32-bit fixed prefix for the SID to assist the switch in SID matching. This approach identifies the SID while restricting its address space. For non-SID addresses, the switch performs normal IP packet forwarding. Additionally, to simplify the description, the above mechanism only presents the stateful SID processing pipeline. Stateless SID uses normal IP forwarding and differentiates based on the header field.

Task-based Matching: ServiceNAT introduces a task-based matching mechanism to maintain the SID task communications. This mechanism selects the *egress.port* for each packet based on the task. Due to differences between request and reply packet headers, ServiceNAT customizes the matching method for each type. Given that the connected devices on switch ports are usually constant, the mechanism determines the packet type based on P_R. Firstly, since the IP header of request packets carries complete task information, the mechanism utilizes $\langle DstIP, SrcIP \rangle$ as the key to match $\langle SID, CA \rangle$ in the entries, and then forwards the packet based on the matched entry's egress port. Additionally, new tasks that cannot be matched are forwarded to *task scheduler* to trigger entry updates. Secondly, the mechanism uses $\langle SrcIP, SrcPort \rangle$ of reply packets to match $\langle HIP, Port \rangle$ in the entries. Since the switch forwards reply packets through a fixed egress port, this match primarily serves to differentiate services (i.e. SID).

SID Address Mapping: Establishing stateful connections, such as TCP, requires host-specific descriptions, which SID lacks. To address this, ServiceNAT designs SID address mapping to translate the SID address of packets. Specifically, this mapping performs SID to HIP translation for the $DstIP$ in request packets and HIP to SID translation for the $SrcIP$ in reply packets. As a result, the mapping enables clients to transparently establish stateful connections with servers through asymmetric sockets, where the client uses (SID, CA) and the server uses (HIP, CA).

Checksum Recalculation: The basic TCP checksum requires the inclusion of the packet payload. However, the P4 data plane only supports fixed-format packet processing and cannot parse variable-length packets. Therefore, ServiceNAT updates the checksum through incremental computation, i.e.,

$$HC' = (HC^{\sim} + m^{\sim} + m')^{\sim}. \tag{1}$$

Equation 1 shows the calculation, where the complement of the new checksum HC' is equal to the complement of the old checksum HC plus the complement of the pre-modification field m, and the modified field m'. ServiceNAT achieves this by accumulating header fields in self-created *metadata* during the Parser and Checksum processing separately.

$$TCP'_{cs} = Dst' + Src' - TCP_{cs} - Dst - Src. \tag{2}$$

Specifically, ServiceNAT calculates the TCP checksum as shown in Equation 2. Since P4 pipes are unidirectional, we define metadata to accumulate the checksum. It *subtracts* the old TCP checksum, $DstIP$, and $SrcIP$ in the parser, and *updates* the new $DstIP$ and $SrcIP$ in the deparser.

ServiceNAT designs a novel P4-based scheme that transparently supports stateful session connections and socket establishment for SID tasks. This scheme employs simple matching operations and minimizes the utilization of complex processing logic to guarantee efficient task processing.

IV. Implementation & Evaluation

A. Prototype Implementation

Supported by *China Telecom*'s large-scale innovation research infrastructures, our prototype system is deployed in 3 data center networks (located in *Beijing*, *Guangzhou* and *Chengdu*), and connected via the practical WAN environment.

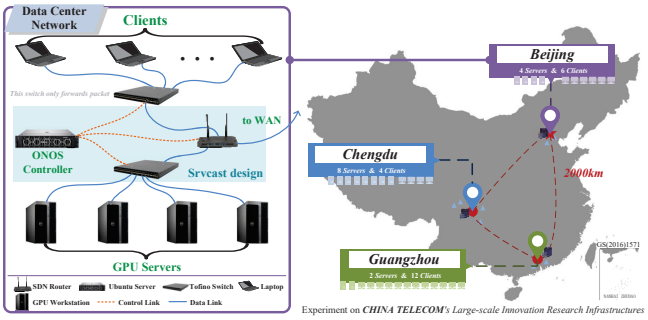
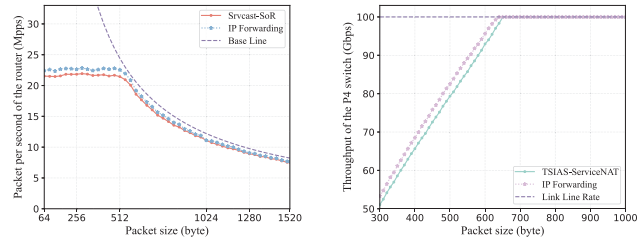


Fig. 4. The topology of the prototype system.



(a) Service-oriented Routing (SoR) (b) ServiceNAT

Fig. 5. Srvcast data plane efficiency vs. packet sizes (In subfigure (a), the baseline represents the packet processing speed trajectory at the line rate).

The prototype system topology is illustrated in Fig.4. Each data center network consists of 2 programmable Tofino switches, 1 SDN (Software Defined Network) router, 1 controller, 2-8 servers, and 4-12 laptops. The servers and laptops are connected to Tofino switches, responsible for initiating client tasks and providing services, respectively. The Tofino switches are connected to the SDN router, enabling P4-based *ServiceNAT*. The SDN router performs *Service-oriented Routing*. Additionally, we employ the Open Network Operation System (ONOS) [20] as the controller.

Hardware Parameters: The Tofino switch is equipped with an Intel(R) Xeon(R) CPU 1527 and 100Gbps port. Each GPU workstation is equipped with an Intel(R) Xeon(R) CPU w7-3445, RTX 3080 GPU, and 10Gbps of NIC. Each laptop is equipped with an Intel(R) CPU Ultra 125H and 1000Mbps of NIC. The SDN router supplied by China Telecom provides 100Gbps port. The controller is equipped with an Intel(R) Xeon(R) Silver CPU 4214 and 62.6GB of memory.

B. System Performance Evaluation

In the Srvcast performance evaluation, we conduct experiments in terms of processing complexity, system resilience, and performance comparison.

Processing Complexity: To evaluate whether the extra processing introduced by Srvcast is acceptable, we measure the packet processing speed (in stage 1, LAN to WAN) of the SDN router and the throughput of the P4 switch (in stage 2). We introduced SID traffic with timestamps into the SDN router (P4 switch) at a rate of 100 Gbps, utilizing the processing time of a single packet to calculate the throughput (packet processing speed). The size of the SID packet was increased by 16 (10) bytes, from 64 (300) bytes to 1520 (1000) bytes. For comparison, we also performed a similar test based on IP.

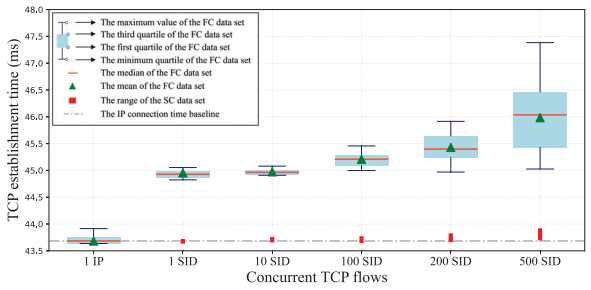


Fig. 6. Srvcast system performance at different concurrent flow densities.

The results are as shown in Fig.5.(a) and Fig.5.(b). Srvcast forwarding performance is slightly lower than legacy IP forwarding (especially for small packets) because each SID packet requires extra header processing. The results show that SID forwarding quickly reaches line-rate performance as the packet size increases. The threshold sizes for *Service-oriented Routing* and *ServiceNAT* line-rate SID forwarding are 543 bytes and 664 bytes (522 bytes and 638 bytes for IP forwarding), respectively. Srvcast can be implemented at the network layer with minimal cost over standard IP forwarding.

System Resilience: To examine the Srvcast performance in handling concurrent requests, we use *Iperf* on the *Guangzhou* clients to initiate SID flows with TCP requests, then utilize *Wireshark* and user logs to record data. We test the First Connection (FC) latency for 1, 10, 100, 200, and 500 concurrent SID flows to assess the Srvcast speed in handling new tasks. Besides, we measure the Subsequent Connection (SC) latency for these flows, showing the efficiency of Srvcast in repetitive processing. The experiment uses IP flow as the control group. Each flow density measurement was conducted 10 times (100 times for IP flow) and presented with a box plot.

The results are depicted in Fig.6. We calculate that the average FC of SID groups increases by $2.9\% \sim 5.27\%$ compared to the control group, with their median and variance of FC following the same trend. The controller updates the flow tables in the data plane to handle new tasks, but its limited processing power creates a bottleneck in the scheduling queue, leading to a deterioration in FC performance. In contrast, the range of SC latency increases only slightly (by 0.5% at 500 SID) since the lack of interaction with the controller significantly improves Srvcast efficiency. To summarize, while intensive SID flows increase the FC latency of Srvcast, they have little impact on SC latency. This could affect bursty requests, but extended experiments demonstrate that the resource efficiency gain from SID anycast fully compensates for the additional cost introduced by Srvcast.

Performance Comparison: We selected typical solutions to compare their performance with Srvcast, including unicast IP (control group), five-way handshake [11] (unicast acquisition series), and ASTAS [10] (proxy deployment series). Our experiments reproduced the workflows of these compared solutions using customized P4 processing logic. Given the impact of

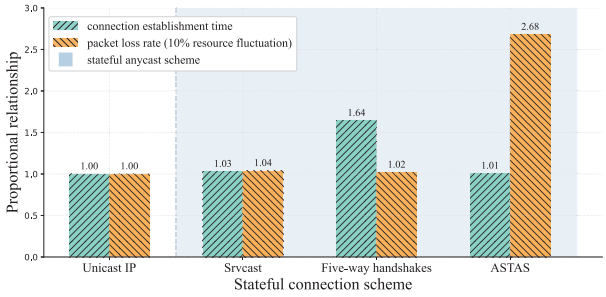


Fig. 7. Performance comparison. The schemes within the blue area represent typical stateful anycast solutions. We normalize the performance of these schemes across different DC choices (for legibility), using unicast IP as a base, and show the relative fraction of the compared schemes.

dynamic resources on SID network routing, we adjusted the selected *anycast entries* for SID tasks at each round (5 seconds per round) with a 10% probability during the test. To conduct the tests, we repeatedly introduced SID flows that initiated TCP data transmission targeting the prototype system multiple times and then measured their average connection establishment times and packet loss rates during the transmission.

The results are as shown in Fig.7. Due to the extra cost of the control plane, the connection establishment time of Srvcast and ASTAS is slightly higher than that of the control group. However, the five-way handshake requires an additional interaction to obtain the unicast address, increasing the time by about 64%, which is unacceptable for frequent transmission with a small amount of data. Srvcast and the five times handshake support stateful anycast, so the packet loss rate is mainly caused by the network. In contrast, ASTAS lacks the ability to maintain sessions. Changes in computing resources directly affect SID routing decisions, resulting in a significantly higher packet loss rate that increases with resource dynamics.

V. Conclusion

In this paper, we have introduced Srvcast, a novel host-transparent and stateful anycast solution for CAN. Srvcast has ensured seamless compatibility between SID anycast and TCP communication patterns while preserving the dynamic advantages of anycast for efficient resource acquisition. A prototype system has been implemented to evaluate its performance, with experimental results demonstrating that Srvcast has achieved line-rate packet delivery and has significantly outperformed existing solutions in terms of system complexity and socket connection establishment time. In future work, we will explore control plane strategies to optimize the SID task scheduling and reduce route maintenance costs within CAN.

References

[1] W. Yuan, Z. Wei, S. Li, J. Yuan, and D. W. K. Ng, "Integrated sensing and communication-assisted orthogonal time frequency space transmission for vehicular networks," *IEEE Journal of Selected Topics in Signal Processing*, vol. 15, no. 6, pp. 1515–1528, 2021.

[2] T. Zhang, G. Li, S. Wang, G. Zhu, G. Chen, and R. Wang, "ISAC-Accelerated Edge Intelligence: Framework, Optimization, and Analysis," *IEEE Transactions on Green Communications and Networking*, vol. 7, no. 1, pp. 455–468, 2023.

[3] C. Li, Z. Du, M. Boucadair, L. M. Contreras, and J. Drake, "A Framework for Computing-Aware Traffic Steering (CATS)," Internet Engineering Task Force, Internet-Draft draft-ietf-cats-framework-02, Apr. 2024, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/draft-ietf-cats-framework/02/

[4] J. Cao, S. Zhang, Q. Chen, H. Wang, M. Wang, and N. Liu, "Computing-aware routing for LEO satellite networks: A transmission and computation integration approach," *IEEE Transactions on Vehicular Technology*, 2023.

[5] X. Zhang, T. Sen, Z. Zhang, T. April, B. Chandrasekaran, D. Choffnes, B. M. Maggs, H. Shen, R. K. Sitaraman, and X. Yang, "Anyopt: Predicting and Optimizing IP Anycast Performance," in *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, 2021, pp. 447–462.

[6] B. Liu, J. Mao, L. Xu, R. Hu, and X. Chen, "CFN-dyncast: Load Balancing the Edges via the Network," in *2021 IEEE Wireless Communications and Networking Conference Workshops (WCNCW)*. IEEE, 2021, pp. 1–6.

[7] S. Peng, J. Mao, R. Hu, and Z. Li, "Demo abstract: Apn6: Application-aware IPv6 Networking," in *IEEE INFOCOM 2020-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE, 2020, pp. 1330–1331.

[8] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard *et al.*, "TensorFlow: A System for Large-Scale Machine Learning," in *12th USENIX symposium on Operating Systems Design and Implementation (OSDI 16)*, 2016, pp. 265–283.

[9] H. Ballani and P. Francis, "Towards a Global IP Anycast Service," *ACM SIGCOMM Computer Communication Review*, vol. 35, no. 4, pp. 301–312, 2005.

[10] T. Stevens, M. De Leenheer, C. Develder, F. De Turck, B. Dhoedt, and P. Demeester, "ASTAS: Architecture for Scalable and Transparent Anycast Services," *Journal of Communications and Networks*, vol. 9, no. 4, pp. 457–465, 2007.

[11] S. Weber and L. Cheng, "A Survey of Anycast in IPv6 Networks," *IEEE Communications Magazine*, vol. 42, no. 1, pp. 127–132, 2004.

[12] J. Jiang, "Enhancing IP Anycast with Location Redirection for Stateful Communication," in *Computer, Intelligent Computing and Education Technology*. CRC Press, 2014, pp. 86–91.

[13] S. Doi, S. Ata, H. Kitamura, and M. Murata, "IPv6 anycast for simple and effective service-oriented communications," *IEEE Communications Magazine*, vol. 42, no. 5, pp. 163–171, 2004.

[14] Y. Li, Z. Han, S. Gu, G. Zhuang, and F. Li, "Dyncast: Use Dynamic Anycast to Facilitate Service Semantics Embedded in IP address," in *2021 IEEE 22nd International Conference on High Performance Switching and Routing (HPSR)*, 2021, pp. 1–8.

[15] Y. Liu, Y. Xia, W. Zhang, W. Jia, and J. Wu, "Sfant: A SRv6-based flexible and active network telemetry scheme in programming data plane," *IEEE Transactions on Network Science and Engineering*, vol. 11, no. 3, pp. 2415–2425, 2023.

[16] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese *et al.*, "P4: Programming Protocol-independent Packet Processors," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 3, pp. 87–95, 2014.

[17] D. Kreutz, F. M. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, 2014.

[18] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: enabling innovation in campus networks," *ACM SIGCOMM computer communication review*, vol. 38, no. 2, pp. 69–74, 2008.

[19] Z. Huang, J.-K. Peir, and S. Chen, "Approximately-perfect hashing: Improving network throughput through efficient off-chip routing table lookup," in *2011 Proceedings IEEE INFOCOM*. IEEE, 2011, pp. 311–315.

[20] P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz, B. O'Connor, P. Radoslavov, W. Snow *et al.*, "ONOS: Towards an Open, Distributed SDN OS," in *Proceedings of the third workshop on Hot topics in software defined networking*, 2014, pp. 1–6.