# Novel High-Speed Offloading Control Technique Enabled by Inter-Controller Cooperative Operation Towards Advanced Cloud Computing

Kota Asaka
*Faculty of Informatics*
*Tokyo City University*
Yokohama, Japan
asaka@tcu.ac.jp

Sota Natori
*Faculty of Informatics*
*Tokyo City University*
Yokohama, Japan

Yu Saruwatari
*Faculty of Informatics*
*Tokyo City University*
Yokohama, Japan

***Abstract*— This paper proposes a novel high-speed offloading control technique enabled by direct cooperation between a data center (DC) controller and an all-photonic network controller, eliminating the need for a central orchestrator. We design new architecture, control sequence flowchart, and resource selection algorithm to minimize offloading latency. Simulation results demonstrate that the proposed method reduces total processing time to 124 ms—about twenty times faster than conventional techniques. The effectiveness of a squeezing approach for optimizing DC and server allocation is also confirmed. These results indicate strong potential for application in time-critical services such as telesurgery.**

***Keywords—Offloading, Cloud computing, Distributed data center, All-photonic network, Inter-controller cooperative operation***

## I. Introduction

In recent years, cloud/edge computing—which allows users to remotely access computing resources via communication networks—has been rapidly spreading [1]. This technology, in which a portion of the tasks performed on user devices is offloaded to remote computing resources, is referred to as “offloading”. Fig 1 illustrates the offloading technology in conventional communication networks. From the user equipment (UE) *N*, (1) a portion of the tasks required for artificial intelligence (AI) development or high-resolution video processing is transmitted as offloaded data to high levels of performance graphics processing units (GPUs) or other processing units installed in the service provider’s data center (DC). (2) The computing resources such as GPUs in the DC then perform offload processing, and (3) the DC sends offload processing results back to the UE *N*. In this way, since the UE can delegate advanced tasks to remote computing resources, it has the advantage of not requiring high levels of performance GPUs or similar hardware locally. However, in conventional

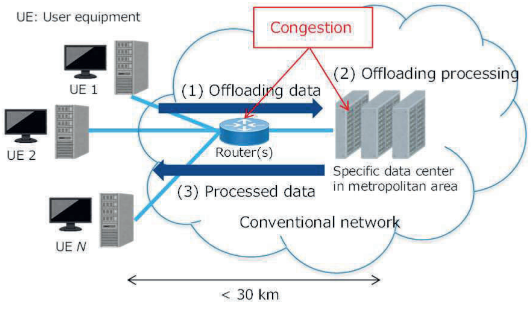


Fig. 1. Schematic view of a conventional offloading process in cloud computing

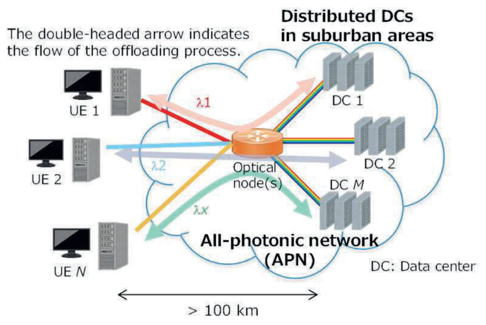

The double-headed arrow indicates the flow of the offloading process.
Distributed DCs in suburban areas
UE 1
λ1
DC 1
UE 2
λ2
Optical node(s)
DC 2
DC *M*
λ*x*
All-photonic network (APN)
UE *N*
DC: Data center
> 100 km

Fig. 2. Schematic view of an APN and distributed data centers in suburban areas for advanced cloud computing

communication networks, multiple routers are installed between the UE and the DC to handle tasks such as switching communication paths and avoiding data collisions. Since traditional routers temporarily store communication data in memory to perform collision avoidance control, when the volume of offloaded data is large or the number of UEs is high, congestion due to offload concentration may occur at the routers or a specific DC. This results in latency, leading to prolonged processing times for offloaded tasks.

In light of the above background, this paper proposes a novel high-speed offloading control technique in which the control unit of the all-photonic network (APN) and the centralized control unit of the distributed DCs are tightly coordinated to reduce offloading latency and shorten processing time for offloading tasks. The remainder of this paper is organized as follows. Section II reviews related work on offloading control and network congestion. Section III details the proposed technique: Section III.A provides an overview of the approach, Section III.B describes the sequence chart illustrating inter-controller cooperative operations, and Section III.C explains the algorithm for selecting appropriate DCs. Section IV outlines the performance evaluation conditions. Section V presents and analyzes the simulation-based evaluation results, demonstrating the effectiveness of the proposed method. Finally, Section VI concludes the paper.

## II. Related Works

### *A. Edge Computing*

As a means to avoid congestion in routers during offloading and to achieve low latency, the use of edge computing, including mobile edge computing, has become increasingly widespread in recent years. Edge computing performs offloading processing by utilizing computational resources located at the edge of the network (i.e., closer to the UEs) rather than in centralized cloud DCs. This shorter distance between UEs and edge resources, compared to traditional cloud computing, eliminates the need to transmit offloaded data to the core of the cloud network. As a result, it can reduce network load (congestion) and latency in offloading processing. Due to these characteristics, edge computing is applied in various time-critical services requiring real-time control, such as factory production line control systems and control systems in autonomous vehicles. However, edge computing faces the following challenges: Since it requires the use of computing resources within DCs located near UEs, offloading processing through edge computing is not feasible in regions without nearby DCs. In particular, hyper-scale DCs tend to be concentrated in urban areas in recent years [2], making it difficult for UEs in suburban or rural areas to utilize edge computing. In Japan, the Watt-Bit Collaboration initiative promotes the regional decentralization of DCs to address urban power constraints and support carbon neutrality [3]. As part of this effort, APNs are expected to play a key role by providing ultra-high-capacity, low-latency optical connections not only between distributed DCs but also between UEs and these centers. Even when the DCs are physically located far from UEs, the high-speed and low-latency characteristics of APNs minimize the impact of geographical distance. This enables efficient and low-latency offloading of computational tasks, facilitating a more resilient and flexible digital infrastructure.

### B. *Offloading Control Technique*

Various offloading control techniques have been proposed to reduce the latency of offloading processes in cloud and edge computing environments [1, 4]. However, most of these studies focus on offloading algorithms or network architectures designed to reduce latency in mobile edge computing or hybrid edge–cloud computing systems, assuming the use of existing network and DC infrastructures. To the best of the authors' knowledge, there have been no prior studies that specifically investigate low-latency offloading in environments based on APNs and geographically distributed DCs, as described in the previous section.

## III. Proposed Offloading Control Technique

Fig. 2 illustrates the schematic view of advanced offloading technology in an APN with DCs distributed across suburban areas. In the APN, each UE is assigned a dedicated wavelength ($\lambda 1$-$x$), establishing a direct optical path between the UE and one of the multiple distributed DCs via optical nodes (optical routers and/or switches) that perform only optical path switching. Because each UE is allocated a unique wavelength, an end-to-end virtual dedicated optical link is provided for each UE. Since the optical nodes (ONs) do not perform optical-electrical and electrical-optical conversions or temporary data buffering in memory, there is no processing delay or congestion associated with such operations, enabling the realization of ultra-high-speed data transmission with low latency.

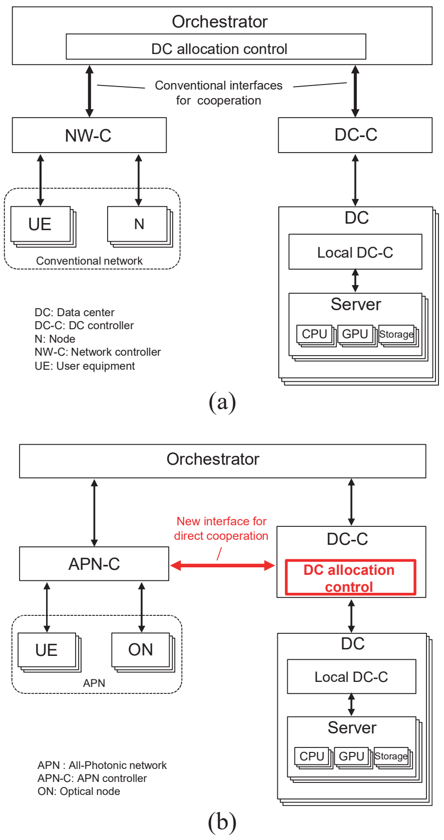


Fig. 3. (a) Conventional architecture for cooperation of network and DC controllers for offloading via an orchestrator, (b) Proposed architecture for inter-controller cooperative operation for high-speed offloading

### A. *Proposed Network Architecture for High-Speed Offloading Technique*

Based on the networks illustrated in Figs. 1 and 2, Fig. 3 shows (a) the conventional network architecture and (b) the proposed network architecture, respectively. In Fig. 3(a), the network controller (NW-C) is responsible for managing and controlling multiple UEs and nodes (e.g., routers) under its domain, while the DC controller (DC-C) manages and controls multiple DCs in metropolitan areas. The orchestrator is a network element positioned at a higher hierarchical level than both the NW-C and the DC-C, enabling coordination between the two controllers through the orchestrator. An offloading request transmitted from a UE is sent to the orchestrator via the NW-C. At this time, the DC allocation control function within the orchestrator selects the optimal DC from a list of DCs with low offloading load, which is periodically provided by the DC-C. The selected DC is then notified to both the NW-C and the UE as the offloading destination. Upon receiving this notification, the NW-C determines the communication path between the selected DC and the UE, and accordingly configures the routing paths on the nodes along the determined route. Once the routing is configured, the UE promptly transmits the offloading data to the designated DC. At this time, if the number of offloading

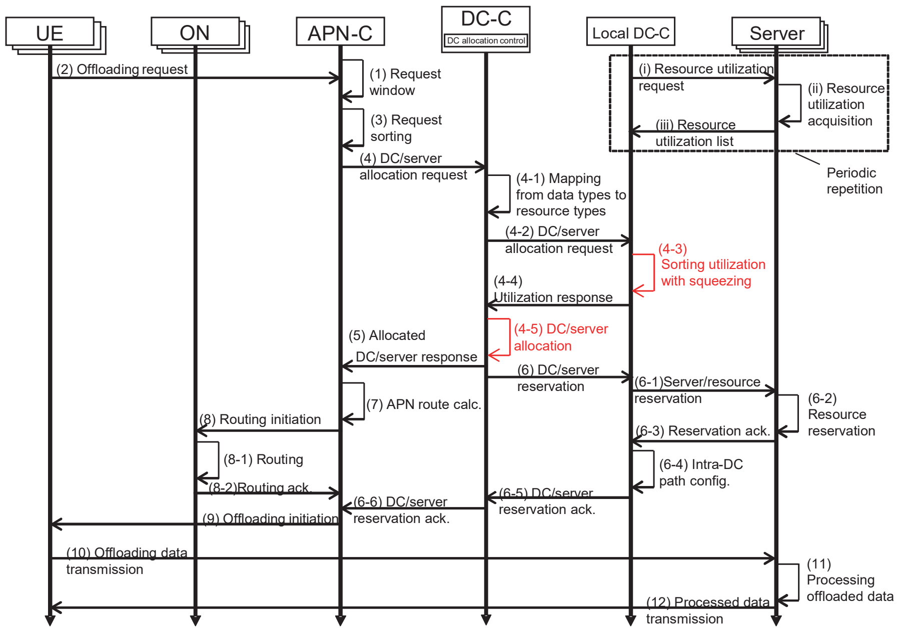

UE
ON
APN-C
DC-C
DC allocation control
Local DC-C
Server
(2) Offloading request
(1) Request window
(3) Request sorting
(4) DC/server allocation request
(4-1) Mapping from data types to resource types
(4-2) DC/server allocation request
(4-3) Sorting utilization with squeezing
(4-4) Utilization response
(4-5) DC/server allocation
(5) Allocated DC/server response
(6) DC/server reservation
(i) Resource utilization request
(ii) Resource utilization acquisition
(iii) Resource utilization list
Periodic repetition
(6-1)Server/resource reservation
(6-2) Resource reservation
(6-3) Reservation ack.
(6-4) Intra-DC path config.
(6-5) DC/server reservation ack.
(7) APN route calc.
(8) Routing initiation
(8-1) Routing
(8-2)Routing ack.
(6-6) DC/server reservation ack.
(9) Offloading initiation
(10) Offloading data transmission
(11) Processing offloaded data
(12) Processed data transmission

Fig. 4. Proposed sequence chart of high-speed offloading control technique utilizing inter-controller cooperation

requests from UEs is high, or if there is heavy background traffic on the network path between UEs and DCs concentrated in urban areas, significant delays may occur due to congestion at intermediate nodes or within specific DCs. This can result in prolonged processing times for offloading, posing a critical challenge. Furthermore, it is known that processing time for coordination between two controllers via the orchestrator generally takes a few seconds [5, 6]. Therefore, under the current typical network architecture as shown in Figs 1 and 3(a), achieving low-latency offloading processing cannot be practical.

On the other hand, in the proposed architecture shown in Fig. 3(b), the network infrastructure is replaced by the APN, and the APN Controller (APN-C) is responsible for managing and controlling multiple UEs and ONs under its domain. In addition, the DC-C manages and controls multiple DCs distributed across suburban areas. Here, the differences from the conventional architecture shown in Fig. 3(a) are explained. First, the DC allocation control function, which was conventionally placed in the orchestrator, is relocated to the DC-C. Furthermore, by introducing a new interface between the APN-C and the DC-C, offloading can be performed without going through the orchestrator. As a result, faster offloading processing can be expected. The detailed operation procedure of the proposed architecture shown in Fig. 3(b) is described in the next section (III.B).

## B. *Proposed Sequece Chart for High-Speed Offloading Technique*

This section describes the DC/server selection procedure in the proposed fast offloading control technique. Fig. 4 illustrates the sequence chart tailored for our proposal. As a prerequisite of this method, the Local DC-C periodically collects the utilization status of each resource (CPU, GPU, and storage) in the servers under its management (as indicated by the dashed box in Fig. 4). (i) The Local DC-C sends a resource utilization request to each server under its control. (ii) Upon receiving this request, each server acquires the utilization information of its internal resources. (iii) Each server then replies to the Local DC-C with the collected resource utilization information. Then, the Local DC-C stores the received utilization data in its internal database. By periodically storing and updating this information, the Local DC-C ensures that up-to-date resource utilization data is always available. This enables rapid and correct selection of DCs and resources when an offloading request is received from a UE.

Based on the above prerequisite, the following describes the procedural flow illustrated in the sequence chart in Fig. 4. (1) The APN-C opens a request window, a configurable time period during which it accepts multiple offloading requests from UEs. During this window, the APN-C remains in a standby state awaiting requests from UEs. (2) When a UE has processing tasks to be offloaded, it sends an offloading request to the APN-C. This request includes three key pieces of information: UE identifier (ID), data type, and offloading data size. In this study, we define four types of offloadable data for the data type field, ranked in descending order of priority: (a) AI inference, (b) video processing, (c) general computing, and (d) data storage. Note that while Fig. 4 illustrates the sequence flow for a single UE for simplicity, the proposed scheme is designed to support requests from multiple UEs. (3) The APN-C extracts the priority level from each request received during the request window and sorts the requests in descending order of priority. In the case of multiple requests with the same priority, they are processed on a first-come, first-served basis. (4) The APN-C sequentially sends DC/server allocation requests to the DC-C, starting from the request with the highest priority determined in step (3). (4-1) Upon receiving each request, the DC allocation control function within the DC-C maps the specified data type to the appropriate types and quantities of resources required by the processing. (4-2) The DC-C then sends a DC/server allocation request to each Local DC-C, requesting information about available DCs and resources that can fulfill the offloading request. This request

**Algorithm 1**: *sel_dc*

```
Input: $c\_req$, $g\_req$, $s\_req$
Output: $res$

01. Set $ws$ based on $s\_req$ thresholds
02. $tc \leftarrow c\_req + g\_req$
03. If $tc > 0$ then
04.     $wc \leftarrow (c\_req \,/\, tc) \times (1 - ws)$
05.     $wg \leftarrow (g\_req \,/\, tc) \times (1 - ws)$
06. else
07.     $wc \leftarrow 0,\ wg \leftarrow 0$
08. end if
09. $best \leftarrow \infty$, $res \leftarrow$ {dc: "def", id: $-1$}
10. For each $dc$ in $dc\_list$:
11.     For each $sv$ in $dc.svrs$:
12.         $sc \leftarrow wc \times sv.cpu + wg \times sv.gpu + ws \times sv.sto$
13.         If $sc < best$ then
14.             $best \leftarrow sc$,  $res \leftarrow$ {dc: dc.nm, id: sv.id}
15.         end if
16.     end for
17. end for
18. Return $res$
```

includes the required resource quantities as calculated in (4-1). (4-3) Each Local DC-C calculates the sum of the utilization rates of all resources for the servers in the list (Step (iii)) and sorts them in ascending order. Note that the sorting is performed after being squeezed to the top 1% or 0.1%, aiming to reduce the processing time. (4-4) Each Local DC-C returns a list of candidate servers and associated resource utilization rates within its DC to the DC-C. (4-5) The DC-C aggregates the candidate information received from all Local DC-Cs and selects the most suitable DC and servers for the offloading request. To reduce processing latency, it limits the candidate pool to the top ten responses received in order of arrival. (5) The DC-C replies to the APN-C with the selected DC and server allocation information. (6) Immediately after step (5), the DC-C sends a request to the relevant Local DC-C to reserve the selected server and prepare for offloading. (6-1) The designated Local DC-C instructs the corresponding server to reserve the required resources and enter a standby state. (6-2) The server reserves the specified resources and transitions to a wait state. (6-3) The server sends a confirmation response to the Local DC-C, indicating successful reservation. (6-4) Upon receiving this response, the Local DC-C performs internal path configuration within the DC. (6-5) The Local DC-C then sends a confirmation response to the DC-C indicating that the reservation has been successfully completed. (6-6) The DC-C forwards this confirmation to the APN-C, notifying that the offloading destination is now in a ready state. (7) Upon receiving the allocation information from the DC-C, the APN-C immediately performs path computation within the APN and determines the route to the selected DC. (8) Based on the computed path, the APN-C sends path control commands to multiple ONs along the route. (8-1) Each ON configures its input/output ports based on the received command. (8-2) Each ON sends a path setup confirmation back to the APN-C. (9) After receiving both the DC/server reservation confirmation (Step (6-6)) and the path setup confirmations (Step (8-2)), the APN-C notifies the UE to initiate the offloading process to the selected DC and resources. (10) The UE begins transmitting the offloading data to the designated DC and resources. (11) Upon receiving the data, the server in the selected DC performs the offloading task using the reserved server and resources. (12) Once the offloading task is complete, the server promptly returns the processing results to the UE. The above outlines the procedure of the proposed high-speed offloading control scheme. Note that, for the sake of simplicity, post-processing steps following offloading completion are omitted from the above description.

### C. DC/Servers Allocation Algorithm

This section explains the DC/server allocation algorithm, which corresponds to step (4-5) in Section III. B, in the proposed fast offloading control technique. Algorithm 1 shows the proposed algorithm to allocate the optimal DC and server based on requested computational resources (CPU, GPU) and storage. The algorithm begins by setting the storage weight *ws* based on *s_req* (line 1) and calculating the total number of logical cores *tc* (line 2). Depending on the value of *tc*, the CPU and GPU weights (*wc* and *wg*) are computed accordingly (lines 3–8). Subsequently, the algorithm iterates through all DCs and their servers (lines 10–17), computes the score *sc* for each server (line 12), and selects the one with the lowest score (lines 13–15). The result *res* is then returned (line 18).

The key feature of this algorithm lies in its ability to select a well-balanced server by dynamically weighting the importance of CPU, GPU, and storage resources based on the UE's specific requirements. Instead of treating all resource types equally, the algorithm adjusts the influence of each resource type in the scoring process, allowing it to favor servers that best match the demand profile. This ensures that the selected server is neither over-provisioned nor underutilized in any particular dimension, leading to more efficient and context-aware resource allocation.

## IV. Performance Evaluation

This section describes the various conditions and verification environment used for the performance evaluation. The emulated functions of APN-C, multiple UEs, ONs, DC-C, Local DC-Cs, individual DCs, and intra-DC servers were distributed across two computers. The APN-C, UEs, and ON emulators were implemented on a computer with an Intel Core i5-1155G7 processor (2.50 GHz, 4 cores / 8 threads), 16 GB of memory, and Windows 11 OS. The DC-C and emulators of

TABLE I. Parameters for Numerical Analysis

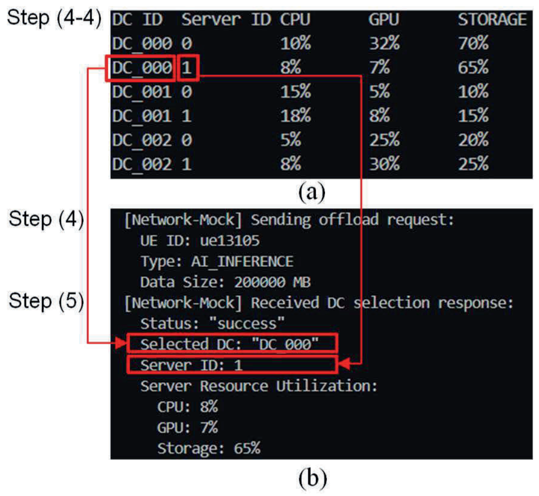
| Simulation Parameter | Value |
|---|---|
| Number of DCs | 200 |
| Number of servers in each DC | 10,000 |
| Number/Size of CPUs/GPUs/Storages in each server | CPU: 64-128 cores<br>GPU: 64-128 x $10^3$ cores<br>Storage: 10-100 TB |
| Data types of offloading | (a) AI inference<br>(b) Video processing<br>(c) General computing<br>(d) Data storage |
| Max. required number/size of resources (CPU/GPU and Storage) for offloading | (a) 32/48 x $10^3$ cores and 10% of data size<br>(b) 16/48 x $10^3$ cores and 10% of data size<br>(c) 48/0 cores and 10% of data size<br>(d) 16/0 cores and 200 GB |
| Data size of offloading | 1 MB - 200 GB |
| Max. distance between UE and DC | 500 km |

Local DC-Cs, DCs, and servers were implemented on another computer with an Intel Core i7-8700K processor (4.70 GHz, 6 cores / 12 threads), 24 GB of memory, and Windows 11 OS. For the APN-C, we newly developed C/C++ programs to execute its main steps—(1) Request window, (3) Request sorting, and (7) APN route calculation—as described in Section III.B and Fig. 4, as well as the signal transmission and reception processes with UEs, ONs, and the DC-C. Similarly, the DC-C was implemented with newly developed C/C++ programs to perform its core functions — (4-1) Mapping from data types to resource types and (4-5) DC/server allocation—along with communication with the Local DC-Cs and APN-C. To further reduce the overall processing time for resource allocation, Step (4-3) Sorting utilization with squeezing was also implemented within the Local DC-Cs.

The two computers described above were connected via a 1 Gigabit Ethernet link, and communication between them was conducted using socket-based messaging. While APN-C and DC-C are originally intended to be deployed at physically separated locations—a few hundred kilometers apart or more—the initial evaluation in this study was conducted within a simplified test environment on the same local area network (LAN). Table 1 summarizes the parameters used in numerical analysis. In this analysis, we assumed that all DCs are geographically distributed across Japan and considered as candidate destinations for offloading. In line with current domestic deployment trends, we set the number of DCs to 200 and the number of servers per DC to 10,000 [7]. Accordingly, the maximum transmission distance between a UE and a DC for offloading purposes was set to 500 km, based on the approximate distance between Tokyo and Osaka. Each server was assumed to be equipped with three types of resources: CPU, GPU, and storage. For each server, the number of resources implemented for each type is defined within a range—64–128 cores for CPU, 64–128 × $10^3$ cores for GPU, and 10–100 TB for storage devices—and is randomly assigned within that range. As described in section III. B, we considered four types of offloading data: (a) AI inference, (b) video processing, (c) general computing, and (d) data storage. The maximum required resources for each type were defined as follows: (a) and (b) require both CPU and GPU resources—32 CPU cores and 48 x $10^3$ GPU cores in maximum for (a), and 16 CPU cores and 48 x $10^3$ GPU cores for (b). (c) requires only 48 CPU cores, while (d) requires 16 CPU cores and 200 GB of storage. The actual number and size of resource allocations are randomly assigned for each offloaded data item, with the aforementioned maximum value serving as the upper bound. Note that (a), (b), and (c) require storage equal to 10% of the size of the offloaded data, which ranges from 1 MB to 200 GB. This storage is used for the temporary buffering of data during the offloading process.

## V. Results and Discussion

This section presents the simulation results for verifying the proposed technique, as well as the discussion based on these results. Fig. 5 shows the verification results of the DC/server allocation algorithm proposed in Section III.C. Fig. 5 (a) illustrates the list of DC/server candidates received by the DC-C from the Local DC-Cs, corresponding to step (4-4) in Fig. 4. For explanatory purposes, the number of DCs and servers per DC in this list are simplified to three DCs and two servers, respectively; however, in the actual list, the variables defined in Table I were used. Figure 5 (b) shows the terminal



```
DC ID   Server ID CPU       GPU      STORAGE
DC_000 0          10%       32%      70%
DC_000 1          8%        7%       65%
DC_001 0          15%       5%       10%
DC_001 1          18%       8%       15%
DC_002 0          5%        25%      20%
DC_002 1          8%        30%      25%
```

```
[Network-Mock] Sending offload request:
  UE ID: ue13105
  Type: AI_INFERENCE
  Data Size: 200000 MB
[Network-Mock] Received DC selection response:
  Status: "success"
  Selected DC: "DC_000"
  Server ID: 1
  Server Resource Utilization:
    CPU: 8%
    GPU: 7%
    Storage: 65%
```

Fig. 5. (a) List of DC/server candidates, (b) Terminal screen of the APN-C.

screen of the APN-C. The upper part displays the content of the offloading request sent to the DC-C (Step (4)), while the lower part presents the response received from the DC-C indicating the selected destination DC and server (Step (5)). Since the offloading request corresponds to data type (a) AI inference, which requires the largest number of CPU and GPU cores as shown in Table I, it was confirmed that the DC allocation algorithm successfully selected the server (Server_ID = 1 in DC_000) with the lowest combined utilization rate of GPU and CPU resources. The algorithm, which assigns weights to each resource according to the data type, proved effective. Instead of merely selecting another server (Server_ID = 0 in DC_001) with the lowest GPU utilization, it achieved a selection that also takes CPU utilization into account.

Table II shows the simulated results of processing times for each step related to DC/server allocation, which is the most critical component in our proposed method. In the DC/server allocation process, each server within every DC is sorted in ascending order based on utilization, creating a list (*dc_list*). Then, in Step (4-5) DC/server allocation, the target DC/server for offloading is determined based on Algorithm 1 using this list. If the utilization rates of all servers across all domestic DCs are included in the list, the number of entries can reach up to two million, leading to significant processing delays. Therefore, in Step (4-3) Sorting utilization with squeezing, we limited the sorting to only the top 1% or 0.1% of servers in each DC (corresponding to 1,000 and 100 servers, respectively). As shown in Table II, the sorting process took 963 ms without squeezing, whereas it was dramatically reduced to 5.6 ms for the top 1% and 1.3 ms for the top 0.1%. This substantial reduction in sorting time was achieved by performing a partial sort—sorting only the top 1% or 0.1% of the list, while leaving the rest unsorted—thus greatly reducing computation time. Moreover, we confirmed that this squeezing also contributed to reducing the processing time in the subsequent step, Step (4-4) Utilization response. As shown in Table II, the processing time was 1,798 ms without squeezing but was significantly reduced to 91.0 ms for the top 1% and 5.0 ms for the top 0.1%. This improvement is attributed to the fact that the utilization response signals sent from the 200 local DC-Cs to the DC-C become smaller in data

TABLE II. SIMULATED RESULTS OF PROCESSING TIMES OF DC/RESOURCE ALLOCATION

| **Process (Step)** | **w/o Squeezing (ms)** | **w/ Squeezing (ms)** | |
|---|---|---|---|
| | | **Top 1%** | **Top 0.1%** |
| Sorting utilization (4-3) | 963.0 | 5.6 | 1.3 |
| Utilization response (4-4) | 1,798.0 | 91.0 | 5.0 |
| DC/Server allocation (4-5) | 4.2 | 0.021 | 0.008 |
| Total | 2,765.2 | 96.6 | 6.3 |

size when the list contains fewer elements, thereby shortening the total time required for all 200 responses to be received. As a result of significantly reducing the number of elements in the list in advance, the processing time for Step (4-5) DC/server allocation was as short as 0.021 ms (top 1% squeezing) and 0.008 ms (top 0.1% squeezing), which is less than 0.5% of the time required without squeezing (4.2 ms). Finally, as shown in Table II, the total processing time for all steps related to DC/server allocation was 2,765.2 ms without squeezing. In contrast, it was reduced to 96.6 ms with top 1% squeezing and further to 6.3 ms with top 0.1% squeezing, demonstrating the significant overall effectiveness of the squeezing technique.

This subsection discusses the total processing time of the proposed scheme. Based on the APN-C simulation, the processing time related to Step (1) Request window (window size = 10 ms), Step (3) Request sorting, and Step (7) APN route calculation was 28 ms in total when two UEs were involved. In addition, Step (8-1) Routing assumes an optical switch port switching delay of less than 50 ms [8]. For Steps (10) and (12), corresponding to the transmission and reception of offloaded data, a delay of 7 ms (round-trip distance of 1,000 km) was adopted, referring to reported results from APN transmission experiments [9]. In the simulation environment, APN-C and DC-C, each with their respective subordinate elements (UEs, ONs, and DCs) implemented, were connected back-to-back within the same LAN. Therefore, the delay associated with multiple exchanges of control signals among controllers/elements that would be physically separated by more than 100 km—such as those in Steps (2), (4), (5), (6), and others—could not be directly evaluated. To account for this, it was assumed that APN is also utilized for transmitting these control signals, and the total delay was estimated to be 33 ms (3.67 ms per 500 km for one-way, and 33 ms for nine-way) [9]. The processing time of Steps (6-1)–(6-4) within the Local DC-C and servers was assumed to be negligibly small. By adding 6 ms for the DC/server allocation process presented in Table II, the total processing time associated with the proposed mechanism was estimated to be 124 ms, excluding Step (11) Processing offloaded data.

If the proposed inter-controller cooperation mechanism is not employed, the coordination between APN-C and DC-C via the orchestrator would take a few seconds [5, 6]. This result indicates that the proposed method can reduce the processing time required for offloading to less than one-twentieth of that without inter-controller cooperation. Thus, the proposed technique can provide high-speed offloading processing to a remote DC located hundreds of kilometers away from the UE. For instance, in remote surgery applications where the end-to-end latency requirement is less than 320 ms [10], the proposed method can be applied if the offloaded tasks can be completed within 196 ms, indicating its high practicality and effectiveness. Further evaluations with an increased number of UEs are left for future work , and some parameters were set to assumed values whose refinement will also be addressed in future studies.

## VI. CONCLUSION

Assuming a future infrastructure composed of distributed DCs and APNs, we newly proposed a control technique that enables high-speed offloading by allowing controllers at each site to communicate directly without relying on a central orchestrator. In this paper, we designed a new architecture, sequence chart, and DC/server selection algorithm to realize this technique and conducted simulation-based validation. The results revealed that, compared to conventional technologies, the proposed method can accelerate offloading processes by up to twenty times. In particular, the effectiveness of the squeezing approach in reducing computation time for DC/server allocation was clearly demonstrated. Our proposed technique suggests high applicability to time-critical services such as telesurgery.

## ACKNOWLEDGMENT

The research work was supported by JSPS KAKENHI Grant Number 24K23871.

## REFERENCES

[1] C, Jiang, X. Cheng, H. Gao, X. Zhou, and J. Wan, “Toward computation offloading in edge computing: A survey,” IEEE Access, vol. 7, pp. 131543– 131558, 2019.

[2] T. P. Fang and S. Greenstein. "Where the cloud rests: The cconomic geography of data centers," Harvard Business School Working Paper, No. 21-042, September 2020. (Revised August 2025.)

[3] Minitstry of Econoy, Trade and Industiry of Japan, “Report1.0 of the public-private advisory council on Watt-Bit collaboration published,”, News release, June 12, 2025.

[4] Y. Toyoshima, T. Hatano, T. Shimada, and T. Yoshida, "Dynamic hardware accelerator Sslection achieving optimal utilization of resources," in IEICE Communications Express, vol. 13, no. 12, pp. 504-508, December 2024.

[5] E. Kosmatos, C. Matrakidis, D. Uzunidis, A. Stavdas, S. Horlitz, and T. Pfeiffer, "Real-time orchestration of QoS-aware end-to-end slices across a converged metro and access network exploiting burst-mode technology," Journal of Optical Communications and Networking, vol. 15, no. 1, pp. 1-15, January 2023.

[6] H. Ou, K. Asaka, T.Shimada, and T. Yoshida, “A real-time optical path control scheme across optical and wireless network domains toward latency-critical services,” IEICE Transactions on Communications, in press.

[7] Impress Research Institute, Data center research report 2025 (in Japanese), Impress Corporation, 2025.

[8] Specification sheet, Huber+Suhner Polatis 6000 Series, Single mode optical switch up to 192x192 ports, https://www.redhelix.com/products/hubersuhner-polatis-6000-series/?utm_source=chatgpt.com

[9] NTT Corporation, “Demonstration shows IOWN APN's low-latency capability can be used for real-time diagnosis and treatment on a remote server to realize world's first cloud endoscopy system”, News release, Nov. 19, 2024.

[10] Y. Wang, Q. Ai, T. Shi, Y. Gao, B. Jiang, W. Zhao *et al.*, “Influence of network latency and bandwidth on robot-assisted laparoscopic telesurgery: A pre-clinical experiment,” Chinese Medical Journal, Vol. 138, No. 3, pp. 325-331, 2025