PAPER

High-Throughput Exact Matching Implementation on FPGA with Shared Rule Tables among Parallel Pipelines

Xiaoyong SONG[†,††a)], **Zhichuan GUO**[†,††b)], **Xinshuo WANG**[†,††c)], ***and*** **Mangu SONG**[†,†††d)], ***Nonmembers***

SUMMARY In software defined network (SDN), packet processing is commonly implemented using match-action model, where packets are processed based on matched actions in match action table. Due to the limited FPGA on-board resources, it is an important challenge to achieve large-scale high throughput based on exact matching (EM), while solving hash conflicts and out-of-order problems. To address these issues, this study proposed an FPGA-based EM table that leverages shared rule tables across multiple pipelines to eliminate memory replication and enhance overall throughput. An out-of-order reordering function is used to ensure packet sequencing within the pipelines. Moreover, to handle collisions and increase load factor of hash table, multiple hash table blocks are combined and an auxiliary CAM-based EM table is integrated in each pipeline. To the best of our knowledge, this is the first time that the proposed design considers the recovery of out-of-order operations in multi-channel EM table for high-speed network packets processing application. Furthermore, it is implemented on Xilinx Alveo U250 field programmable gate arrays, which has a million rules and achieves a processing speed of 200 million operations per second, theoretically enabling throughput exceeding 100 Gbps for 64-Byte size packets.

key words: *field programmable gate arrays (FPGA), match-action table, exact matching, hash table, hash collision, CAM*

1. Introduction

In software defined network (SDN), most network functions are implemented based on match-action table (MAT) model. In MAT, the specific fields of data packets are extracted as key to probe the matching table, and the action instructions that should be executed are obtained after successful matching [1], [2]. Exact matching (EM) table plays an important role and is widely used in packet processing applications such as packet inspection [3], packet classification [4] and flow monitoring [5] etc. The processing speed of network packets and the scale of networks are increasing continuously, along with higher processing performance requirements for switch devices, which also demand higher performance and scalability to exact matching tables.

Manuscript received August 18, 2023.
Manuscript revised October 18, 2023.
Manuscript publicized January 30, 2024.

[†]The authors are with the National Network New Media Engineering Research Center, Institute of Acoustics, Chinese Academy of Sciences, Beijing 100190, China.
[††]The authors are with the University of Chinese Academy of Sciences, Beijing 100049, China.
[†††]The author is with Suzhou Haiwang Network Technologies Co., Ltd., China.
a) E-mail: songxy@dsp.ac.cn
b) E-mail: guozc@dsp.ac.cn (Corresponding author)
c) E-mail: wangxs@dsp.ac.cn
d) E-mail: songmg@dsp.ac.cn
DOI: 10.23919/transcom.2023EBP3140

Field programmable gate arrays (FPGA) has significant advantages in terms of programmable flexibility and parallel processing, and various network functions are being offloaded to FPGAs for accelerated processing [6]. However, neither EM nor content addressable memory (CAM) is on an FPGA. User needs to design and implement the matching table based on on-board resources. On FPGA, the mainly methods to implement exact matching table include hash-based methods and CAM-based methods. The exact matching table based on CAM consumes huge resource and has a low memory efficiency [7]. EM table based on hash has higher memory efficiency, but there are problems such as hash collision, insertion difficulty, and nondeterministic worst case latency [8], [9]. Moreover, both methods will face difficulties in achieving a large depth or large width EM on FPGA with a high speed.

In order to improve the throughput of the matching table, some designs employ multiple parallel channels. However, it also brings the problem of memory replication [10], resulting in huge on-chip storage consumption. Some multi-channel designs [11], [12] without storage replication also have the problems of low hash table load factor. Moreover, different from the out-of-order execution of a general Key-Value System (KVS) in database, it is necessary to maintain the sequence order of packets in most of network packet processing applications. Hence, the issue of uncertain processing latency or packets out-of-order should also be considered in network matching table application.

To implement a large-scale high-throughput exact matching table and solve the problems of hashing collision and out-of-order among multiple pipelines, this paper proposes a multi-channel exact matching table with shared rule table, which can improve the processing speed of the matching table without memory replication. Especially, it would rearrange the out-of-order matching results after processing to ensure a correct sequence, which avoids packets out-of-order or error in network. The main contributions of this work are as follows:

- This paper proposed an FPGA-based exact matching implementation that leverages shared rule tables across parallel pipelines to enhance overall throughput without memory replication. The implemented EM table based on FPGA could insert a million rules, which has good scalability and achieves a processing speed of 200 million operations per second, theoretically enabling throughput exceeding 100 Gbps for 64-Byte size pack-

ets.

- An out-of-order reordering function to recover the order of matching results within the pipelines to maintain packet sequence in network packet processing.
- A compact CAM-based exact match table is incorporated alongside the primary hash-based EM table within each pipeline to handle hash collisions, which ensures the important rules can be inserted into rule tables.

2. Exact Match Table Overview

2.1 Match-Action Model

As Fig. 1 shown, match-action model is the mainstream framework to process data packets in data plane of programmable devices. At each processing stage of data packets, the feature match filed in the packet is extracted as *Key* and used for MAT table lookup operation [2], [13]. The action engine then executes actions based on the result of the table lookup. Among the various types of MAT tables used in packet processing and pattern matching, the exact matching table is commonly employed.

There are two main ways to implement EM on FPGA, which is hash-based EM like [10] and CAM-based EM like [14]. Both hash-based and CAM-based exact matching tables have O(1) lookup performance. In contrast, hash-based EM has higher storage utilization efficiency, while the SRAM-based CAM has low storage utilization. However, CAM-based EM does not have the problem like hash collision or insert difficulty, etc.

2.2 Hash-Based Exact Match Table

The hash-based exact match table shown in Fig. 1(a) is a fast and efficient data structure that stores *Key-Value* pairs in the {*Vld, Key, Value*} data structure in each address space. During inserting or querying, the *Key* is hashed to generate the corresponding address index, and then the data structure is stored at the address or retrieved for comparison, ultimately yielding the corresponding value.

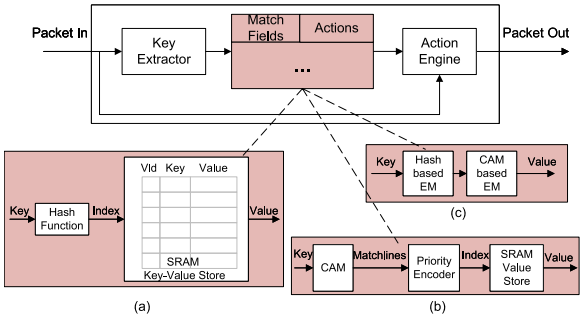


Fig. 1 Basic scheme of match-action model. (a) Architecture of hash-based exact match table. (b) Architecture of CAM-based exact match table. (c) Architecture of non-collision exact match table.

2.3 CAM-Based Exact Match Table

Content Addressable Memory (CAM) is a type of memory that enables fast content queries and has the advantage of fast search rate. As Fig. 1(b) shown, in CAM-based exact matching table, the *key* is entered into CAM to get the matching information *matchlines* which contains all match result of each address unit, and the match address index is encoder by *Priority Encoder*. Finally, this address is used to read the corresponding *Value* from the *Value Store*.

2.4 Non-Collision Exact Match Table

The probability of collision depends on the hash function, which is not possible to be perfect, especially in the case of random and frequently updatable data [7]. In order to solve the hash conflict, the cuckoo hash [9], multiple level hash table [10] or chaining [15], adding auxiliary storage [5], [12], [16], and other solutions have been proposed. Figure 1(c) shows a non-collision exact match table combined with hash-based EM and CAM-based EM. The rule entry is firstly inserted into hash-based EM table. If a collision occurs, the conflicted entry is then inserted into the CAM-based EM table. This hybrid structure leverages the benefits of both CAM and hash-based techniques to ensure efficient and collision-free matching.

3. Architecture

Although hash table has good scalability and high resource utilization, implementing a high-performance EM table on FPGA is still a challenge, especially when the size of table is large. It is difficult to perform matching with sufficient throughput for wire-speed processing. For instance, in a 100 Gbps high-speed network, at least 148.8 million of 64B size packets per second must be processed to meet processing speed requirements, which means the operation throughput of matching table should not be lower than 148.8 million.

The core idea of increasing the processing speed is to maximize the number of operations processed per clock cycle. Usually, the processing speed is increased by boosting the main frequency of system or utilization of multiple parallel pipelines [17]. It is not easy to improve the frequency on FPGA, especially when the entire system is complex and the table size is large. The problem faced by multiple parallel pipelines is that it requires multi-port memories or memory replication to store each rule several times, which consumes more storage resources. Due to the limited resources on FPGA, it is not feasible to use the method of memory replication when implementing a very large scale matching table.

3.1 Parallel Shared Hash Table with CAM Structure

To avoid storage replication and increase the number of entries processed in a single clock cycle, we optimize the multi-level hash pipeline structure to multiple parallel pipelines

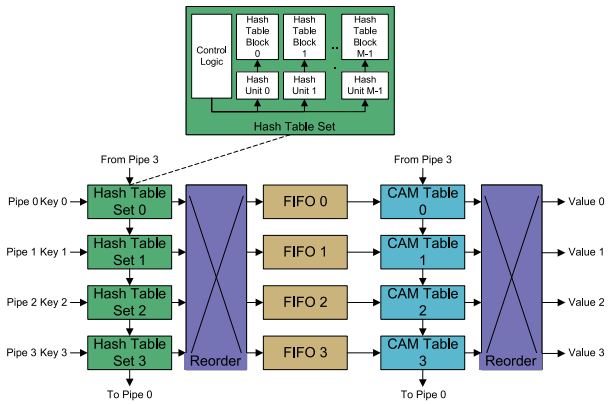


Fig. 2 Overall architecture of exact table with 4 parallel pipelines.

structure. Additionally, multiple CAM tables were adopted to handle hash conflicting.

All pipelines share all rules stored in the entire exact matching table, and each rule is only stored once time without backup. The table in each pipeline can be accessed by the operation from its neighbor pipeline if needed. For each key, there is a probability that they will be inserted or matched in the hash table set of each pipeline. If the operation succeeds in a hash table set, there is no need to access other tables in other pipelines. Meanwhile, the tables in other pipelines can process other operations. In the worst-case scenario, when all EM tables in all pipelines need to be accessed for each key, the throughput of the entire EM table is the same as that of a single pipeline. In the best scenario, all operations are succeed in the first hash table set they access, and the entire exact matching table could handle P operations in each clock cycle. However, for most cases, $1 \leq p \leq P$, where p is the number of operations EM can process in a clock cycle and P is the number of pipelines.

In the entire EM table, there are P parallel pipelines, each consisting of a set of hash table blocks as the main storage and a CAM-based EM table as auxiliary storage. Figure 2 shows the architecture of our EM table with four parallel pipelines. Multiple hash functions and hash table blocks are used in each hash table set to reduce the hash collision rate. To avoid the uncertainty in insertion time caused by cuckoo hashing, we select an empty address space from multiple alternative addresses in parallel, and the conflicted new entry would be inserted in other hash table sets instead of replacing existing entries if there is no empty alternative address in current hash table set. Entries failed inserted into all hash tables are eventually inserted into CAM. In order to maintain the sequence of packets and ensure a constant search latency, there is a *Reorder* module behind the hash table sets and CAM tables to return the searched key and matched result to its own input pipeline, and unify the latency of each search key according to its operation path.

3.2 Hash Table Block and Hash Table Set

In each pipeline, there is a set of hash-based EM table blocks, here called the *hash table set*. Each hash table set consists of M hash table blocks, and these hash table blocks are independent and store *Key-Value* pairs in their address units with the entry structure of {*Vld, Key, Value*}. If *Vld* is '1', it indicates the entry structure is valid and the slot in hash table block has been used.

3.2.1 Class *H3* Hash Function

The performance of a hashing scheme depends on the collision handling method and the hashing function chosen [18].

Class *H3* hash algorithm [8] was used to perform the hashing operation on the key, which has been demonstrated to be effective on distributing keys randomly [10]. Let i denotes the number of bits for input key, and j denotes the number of bits for hash index. Let Q denotes a $i \times j$ Boolean matrix. For a given $q \in Q$, let $q(m)$ be the bit string of the mth row of Q, and let $x(m)$ denote the mth bit of input key. The hashing function $h(x) : A \rightarrow B$ is defined as

$$h(x) = (x(1) \cdot q(1)) \oplus (x(2) \cdot q(2)) \oplus \ldots \oplus (x(i) \cdot q(i)). \quad (1)$$

Compared to other hashing algorithms like *Toeplitz* [19], the *H3* algorithm not only ensures uniformity and fast computation but also consumes fewer logic resources when implemented on an FPGA. The hardware which stores *H3* matrix can be organized in a bank of registers. The same hardware can realize any desired hashing function from this class and the hashing function can be changed dynamically by loading data into the bank of registers if needed [18]. To improve the clock frequency, the hashing operation is pipelined and completed within two clock cycles.

3.2.2 Hash Collision Handling

To reduce hash collisions, an independent *H3* hash matrix is set for each hash table block, and the entire EM table has $P \times M$ *H3* hash matrices and $P \times M$ hash function units. If a hash collision occurs during insertion, the new entry would select another empty slot to insert, instead of replacing the original entry like cuckoo hashing. In each hash table set, M hash function units perform hash calculations on the same key parallelly, and then choose an address without hash conflict from the M candidate addresses for insertion. With an increasing number of hash table blocks in each hash table set, the hash collision rate would be reduced significantly, which would be explained further in Sect. 4.1. If there is no empty candidate address in the current hash table set, the insertion operation proceeds to the hash table set of the next pipeline. If all hash tables fail to insert, the conflicting entries are stored in the CAM table finally.

3.2.3 Operations

- *Insert*: As illustrated in Fig. 3, for each hash table set, M hash function units in this set first generate M candidate addresses for the key during entry insertion. Then the

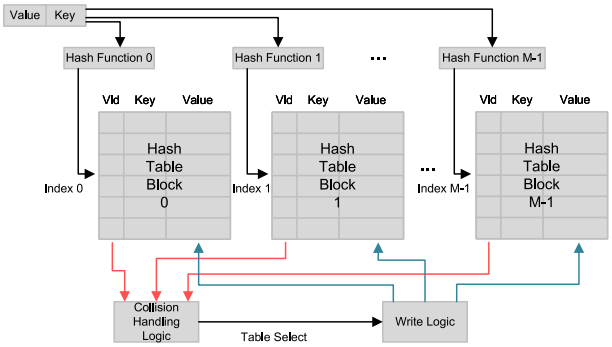


Fig. 3 Insert process of hash table set.

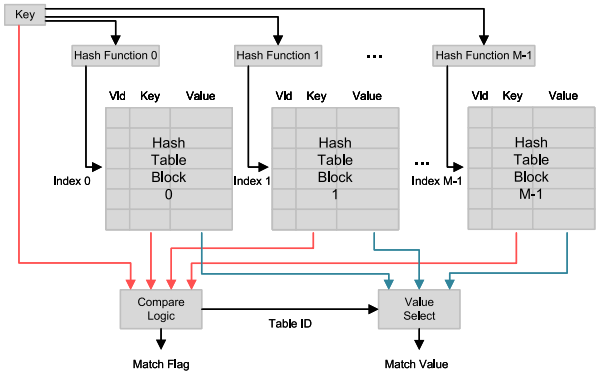


Fig. 4 Query process of hash table set.

entry structure stored in the candidate address of each hash table block is read. If the *Vld* bit in entry structure is '0', the address is empty, and '1' indicates that this address space has been already in use. The *Collision handling logic* selects a hash table block with empty candidate address to insert. *Write logic* writes the entry structure of new entry into the corresponding address of the selected table.

- *Query*: As illustrated in Fig. 4, after hashing calculation and entry structure reading, the queried key is compared with all keys in valid entry structures. After comparing, *Compare logic* encodes all comparison results and gets the matching address, and then the matching value is selected according to the matching address.
- *Delete*: Performs a query operation firstly. After the matching is successful, the content of the matching address is written to 0 to delete the entry.

3.3 Auxiliary CAM Tables

Even if we use multiple hash functions and multiple hash table blocks to reduce the probability of hash collisions, a perfect hash function does not exist. To avoid situations where important rules cannot be inserted into hash tables due to hash conflicts, we handle this problem by adding auxiliary storage, namely a small depth CAM-based EM table for storing entries that cannot be inserted into the hash table.

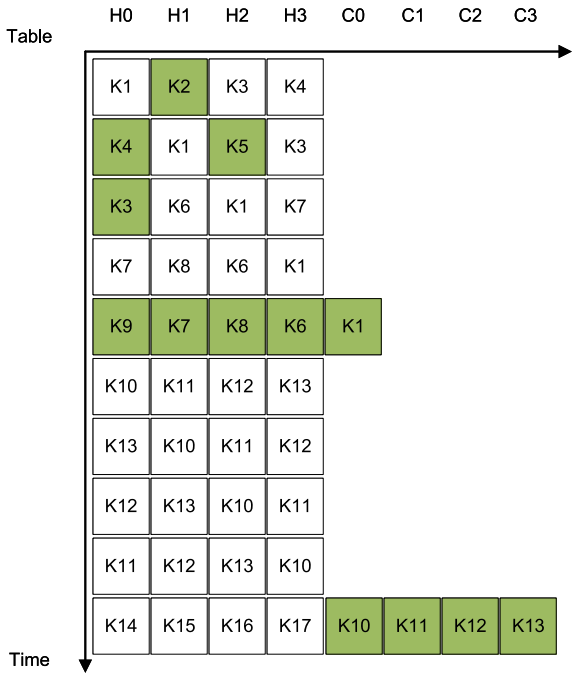


Fig. 5 Timeline of the operation (H: Hash Table Set. C: CAM Table. K: Operation Key. H0, H1, H2, H3 are the four hash table sets in Fig. 2, and C0, C1, C2, C3 are the four CAM tables in Fig. 2. K0, K1, $\cdots$, K13 are the operation keys into the tables).

The CAM here is implemented using the transposed SRAM method [20]. In the implementation of this method, key is used as write or read address, and *matchlines* containing entry address information are stored in SRAM. The width of *matchlines* is equal to the depth of CAM, with each address space corresponding to a single bit in *matchlines*. For a given key, if a particular bit in its corresponding *matchlines* is '1', it means that the address is a matching address.

Theoretically, the depth of CAM depends on the probability of hash collision, and a detailed analysis will be provided in Sect. 4.2. Here, we assume that the total CAM depth requirement is C_{depth}. Figure 5 illustrates the timeline of the operation in the EM table. When a key K enters the matching table, it will appear in the timeline and the grid in Fig. 5. The white grid indicates that the operation of K is not completed yet, and it needs to continue entering the next hash table set or CAM table to try its operation with the loop order of *Pipe* 0 $\rightarrow$ *Pipe* 1 $\rightarrow$ *Pipe* 2 $\rightarrow$ *Pipe* 3 $\rightarrow$ *Pipe* 0. The green grid indicates that the operation of K has been successfully completed or all tables have been accessed. For each key, if its operation is failed in all hash tables, it would further enter into CAM table. The key failed to do operation in a CAM table would access the next CAM table with the same loop order of *CAM* 0 $\rightarrow$ *CAM* 1 $\rightarrow$ *CAM* 2 $\rightarrow$ *CAM* 3 $\rightarrow$ *CAM* 0. For instance, the operation of *K1* is failed in all hash table sets and it finally completes its operation in CAM table 0. To avoid the worst-case scenario which shown in Fig. 5, when all hash table sets in multiple pipelines need to insert conflicting entries into CAM simultaneously (*K10*, *K11*, *K12*,

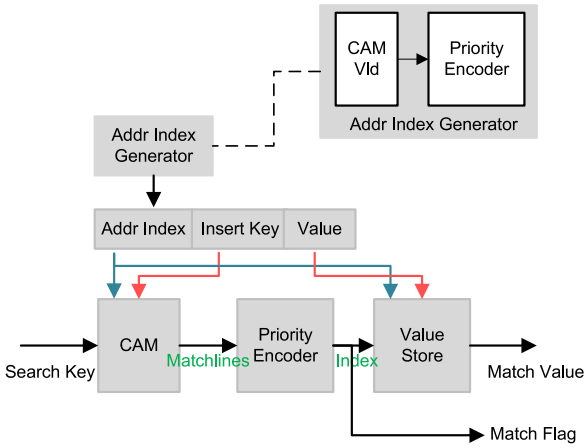


Fig. 6 Architecture of CAM-based EM table.

and *K13*), we place a CAM table after each hash table set in each pipeline. The key of *K10*, *K11*, *K12*, and *K13* are failed to do their operation after traversing all hash tables in four pipelines. After the first Reorder module behind the hash tables in Fig. 2, *K10* ~ *K13* enter into the CAM tables with their corresponding pipelines. The depth of the CAM in each pipeline $CD = \frac{C_{depth}}{P}$, where P is the number of pipelines or the number of hash table sets.

By combining CAM-based EM tables with hash-based EM tables, it can address potential hash conflicts and ensure that important rules are correctly inserted into the match table.

3.3.1 Address Spaces Management

As shown in Fig. 6, for each CAM table there is a bitmap *CAM Vld* vector which records the usage status of each CAM address space. '0' means that the space is already in use, while '1' means that it is available for use. When inserting an entry into CAM, the address index generator allocates an empty address space to this entry by the *vld* bitmap and a priority encoder. After an entry is deleted, the corresponding address *vld* bit corresponding would be set to '1' again, indicating this address can be reallocated.

3.3.2 Operations

- *Insert*: After generating the write index, the *new content* of the entry is generated based on the write index ($new\ content = 1 << write\ index$). In the same time, the original content in the address of writing key should be read out as *old matchiline*. The *new content* and *old matchiline* perform the ***or*** operation to generate the *new matchiline* written into CAM. *New matchiline* = *old matchlines* | *new content*. Use the entry key as write address, write this *new matchiline* into the CAM. At the same time, write the corresponding value into value store at the write index. Set the corresponding bit in the *vld* bitmap to '0' to indicate the address is now in use.

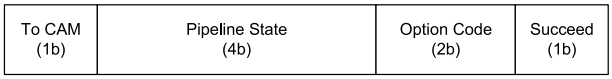


Fig. 7 State structure across EM pipelines.

- *Query*: During a query, the key is used as the read address to read the *matchlines* stored in CAM. A priority encoder is used to encode the *matchlines* and obtain the matching index. If the address space is in use, read the corresponding value stored in this address space from the value store.
- *Delete*: After completing the query operation, clear the content of corresponding bit in matchlines at the key's address in CAM and also clear the content at the corresponding index of the value store. Set corresponding bit in bitmap to '1', indicating that it can be used again.

3.4 State Structure across Pipelines

In addition to entry structure stored in EM table, a custom data structure is maintained and transferred among pipelines to record the status and data path of each operation, here we call it *state structure*. As shown in Fig. 7, this state structure has a length of 8 bits, and each sub-field is defined as follows:

- [0]: *Succeed* – Indicates whether the operation of an entry is successful. It is set to 1 if the entry has been successfully inserted or if a query has been successful.
- [2:1]: *Option Code* – Specifies the operations to be performed on this entry. 2'b00 indicates no operation, 2'b01 indicates insertion, 2'b10 indicates deletion, and 2'b11 indicates query.
- [6:3]: *Pipeline State* – A 4-bit bitmap represents the status of whether pipelines or tables have been accessed. 1'b0 indicates that the table has been accessed, and 1'b1 indicates that table has not been accessed. For example, 4'b1011 indicates that the entry is entered from *Pipe* 2 and the table in *Pipe* 2 has been accessed. If further access is needed, the next step is to jump to *Pipe* 3 → *Pipe* 0 → *Pipe* 1 for access until operate succeed or all pipelines have been accessed. For table entries that have not been successfully operated in hash tables, this field will be restored to its initial state of 4'b1111 before entering CAM tables.
- [7]: *To CAM* - Determines whether to insert an entry into the CAM when insertion fails in all hash tables. 1'b0 means the entry can be directly discarded when insertion fails in hash tables, while 1'b1 means the entry should still be inserted into the CAM if needed.

As a key is queried or a *Key-Value* is inserted into the EM, the hash table or CAM table performs the operation based on its *Option Code*. If the current pipeline executes successfully, it directly jumps out of the table and enters the *Reorder* module. Otherwise, if there is another pipeline table

that has not been accessed according to the *Pipeline State*, it continues to jump to the next pipeline table for corresponding operation.

Once the hash table or CAM table of each pipeline completes its operation, it modifies the corresponding bit in the *Pipeline State* of the pipeline for that pipeline and updates the *Succeed* bit based on the operation success. Additionally, *Reorder* would control the exit time of each entry according to its *Pipeline State*, which ensures the processing latency of each entry is equal.

For a query entry, the state structure is propagated along the query entry until the matching result gets used. However, when inserting an entry, once the insertion is successful, the structure will not propagate backwards further.

3.5 Rearrange Out-of-Order

According to the previous design, when an entry completes the query operation in a hash table set or a CAM table, it will carry its state structure away from the table to allow more new entries to access the matching table for processing. Since each entry may not need to access all hash table sets or CAM tables, the operation latency vary from entry to entry. This would result in out-of-order and congestion at the out-ports of the EM table. Additionally, in network packet processing applications, it is usually necessary to maintain packet sequence.

To ensure that the sequence of packets entering and leaving the pipeline is not disrupted, and that the query latency of each entry is consistent, a *reorder* module is introduced to restore the order of processed entries and make corresponding delay for each entry.

As shown in Fig. 8, there are four channels in *reorder* module, and each corresponds to one channel in the EM table. Taking the hash table as an example, each queried entry carries its status structure from the current hash table set into the *reorder* module. The *parsing unit* parses its pipeline state field to find out which pipeline the entry enters the matching table from and how many hash table set it has been accessed. The module of *path select* dispatches the matching result back the pipeline it entered. Then *delay select* module selects an appropriate additional delay for this entry and outputs it from the corresponding outport of *reorder* module.

After being processed by the *reorder* module, the query latency of each entry is consistent and it is equal with the worst-case delay, and the corresponding matching results can still be returned to its own pipeline after cross-pipeline lookups, which ensures the sequence of packets in each pipeline in the later processing.

As illustrated in Fig. 9, the girds with the same color enter into the table simultaneously, and the colored grid means a key has completed its operation. For example, *K1*, *K2*, *K3* and *K4* are all green because they all entered into the EM table at the first time. *K1* completed its operation in its first table set *H0*, but *K2*, *K3* and *K4* not. They went through 2, 3 and 4 tables to complete the operation respectively. Then

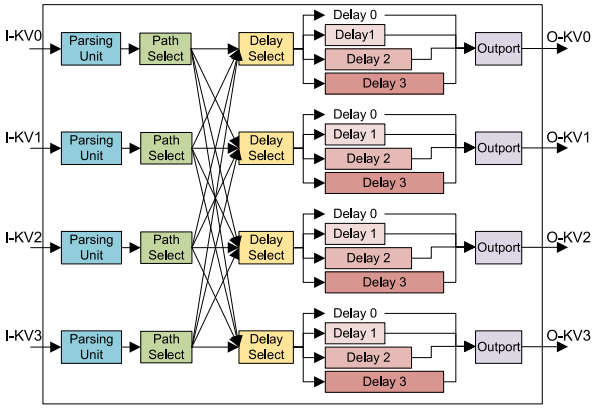


Fig. 8 Architecture of reorder.

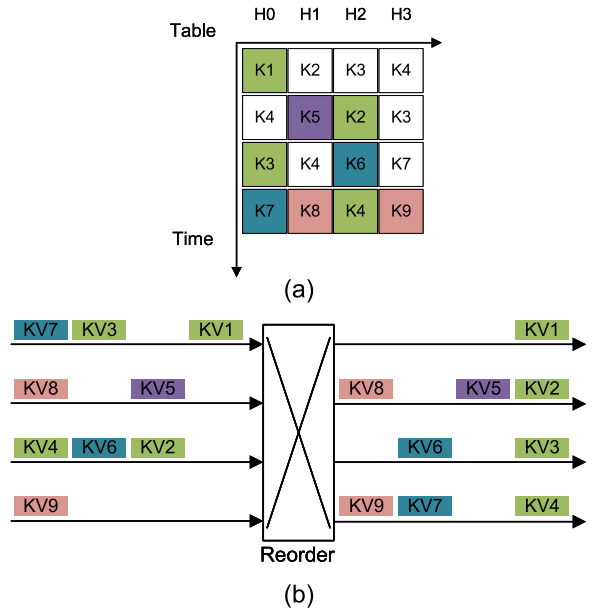


Fig. 9 Timeline of reorder operation.

they entered the reorder module from the channel which they complete operation. After reordering, these four keys exited the EM table simultaneously.

The keys entered from a same pipeline also keep their sequence after reordering. *K4*, *K7* and *K9* entered from *H3* table set at different time. Although they have different processing latency in hash tables, they still maintain their sequence after reordering.

4. Analysis

Here we analyze the hash collision rate, the capacity CAM required and the issue of consistency. Table 1 lists the variable abbreviations that will be used later and their meanings.

4.1 Hash Collision Rate

There are P hash table sets in the entire EM table, and each hash table set contains M hash table blocks with the depth of

Table 1 Tabel of abbreviation.

Abbr	Description
$r_{collision}$	Hash collision rate of a hash table set.
$R_{collison}$	Hash collision rate of the entire hash-based EM table.
$N_{collision}$	The number of conflicted entries of the entire hash-based EM table.
C_{depth}	The depth of entire CAM-based EM table.
H_{depth}	The depth of entire hash-based EM table.
K	Width of key.
V	Width of value.
HD	The depth of each hash table block.
M	The number of hash table blocks in each hash table set.
P	The number of pipelines.
CD	The depth of each CAM table in each pipeline.
$HTSUN$	The number of URAMs used by each hash table set.
$HTBUN$	The number of URAMs used by each hash table block.
$CAMBN$	The number of BRAM36Ks used by each CAM-based EM table.
TUN	The number of URAMs used by entire hash-based EM table.
TBN	The number of BRAM36Ks used by entire CAM-based EM table.
Ep	The expected hash table set number of each entry needs to query.

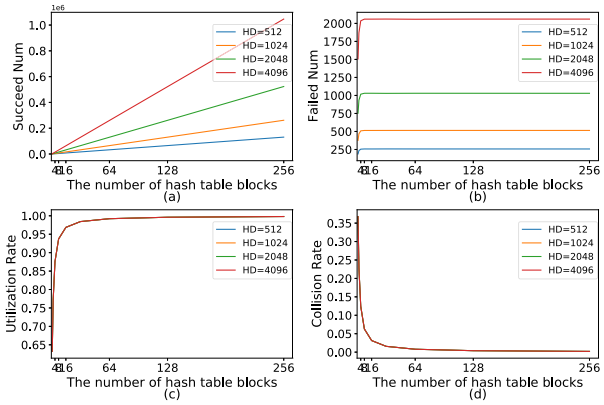


Fig. 10 Simulation results under different hash table block's number and depth. (a) The number of entries inserted into hash tables successfully. (b) The number of failed entries which are not inserted into hash tables. (c) The utilization rate of the entire hash table. (d) The collision rate of the entire hash table.

HD. Hence, there are $P \times M$ hash table blocks. We simulated the hash collision rate under different hash table block's number and depth. The probability of each key hashing to a particular location is uniform [21], so uniform random function acts as hash function unit to generate insert index in simulation. 1000 simulation experiments were conducted for each case and calculated the mean value. In each time, $P \times M \times HD$ entries are inserted into the hash tables.

The simulation results are shown in Fig. 10. The Fig. 10(a) shows the number of successful inserted entries and Fig. 10(b) shows the number of failed entries under different table numbers and different table depths, which provide us a reference to choose the depth of CAM under different cases. It can be seen from Fig. 10(c) and Fig. 10(d) that with the increase of the number of hash table blocks, the hash table utilization rate (load factor) would increase and the collision rate would decrease. When the number of entries inserted into all hash tables is the same as the total number of address spaces in hash tables, the collision rate of entire hash table is almost unaffected by the depth of each hash table block, but mainly determined by the number of hash table blocks. When the number of hash table blocks exceeds 64, the hash collision rate drops below 1%.

4.2 CAM Capacity

Assuming that the hash collision rate is $R_{collison}$, there are P hash table sets, and each hash table set has M hash table blocks with the depth of HD. Then there are H_{depth} ($= P \times M \times HD$) address spaces in the entire hash table. After inserting H_{depth} entries into hash tables, there would be $N_{collison}$ ($= R_{collison} \times H_{depth}$) entries cannot be inserted into the hash table finally.

Hence, the capacity of CAM C_{depth} required should be equal the number of conflicted entries.

$$C_{depth} = N_{collision} = R_{collision} \times (P \times M \times HD). \quad (2)$$

The CAM on each channel should be CD, and

$$CD = \frac{C_{depth}}{P}. \quad (3)$$

4.3 Consistency

Due to the latency in hash computation and SRAM read/write operations, there are two extreme scenarios where consistency issues may occur: (i) a queried key is the same as an inserting key; (ii) in a hash table set, a key is being inserted, the new inserted key takes the insert address of the inserting key as its candidate address.

For the former, if the same search key accesses the matching table during the writing process of an entry, it may result in incorrect query results. For the latter, the status of the inserted address is updated to the occupied state only after the entry is inserted completely. During this insertion period, the address space is still considered to be selected for subsequent insert entries, which may lead to a collision between two entries when selecting an address.

However, the probability and impact of these two scenarios are negligible. Firstly, compared with query operation, insertion operation is generally infrequent. Moreover, it is extremely rare for multiple collisions of a single address to occur in such a vast depth of table, especially within a very short period of time. The first case has been discussed to be negligible in prior work [10]. In the second case, entries can be inserted from different pipelines in the way of round-robin, or new entries can be inserted after confirming that the previous entry has completed the insertion operation.

5. Implementation and Evaluation

5.1 Implementation

We implement our design on a Alveo U250 [22] FPGA device, which has 1,728,000 LUTs, 3,456,000 Flip-flops, 2688 BRAM36K memory blocks and 1280 URAM memory blocks. In order to make a trade-off between latency and throughput, the EM table has a total of 4 channels in our implementation, each channel has a hash table set and an

auxiliary CAM based EM. Each hash table set has 64 hash table blocks with a depth of 4K. Based on the hash collision rate, it can be determined that 4K CAM entries are sufficient to store the collision entries of the hash table. Therefore, the CAM depth of each pipeline is 1K. The total EM can store 1048K (4*64*4096=1048576) entries.

5.2 Memory Utilization

The SRAM storage resources on FPGA are independent units, and each SRAM unit must be used by block. We utilized URAMs to implement hash table and transposed BRAMs to implement CAM. Here, each URAM block is configured as a 4K*72b SRAM and each BRAM36K block is configured as a 512*72b SRAM. The symbol of $\lceil * \rceil$ represents rounding up.

The number of URAMs used by each hash table block is

$$HTBUN = \left\lceil \frac{HD}{4096} \right\rceil \times \left\lceil \frac{K+V+1}{72} \right\rceil. \tag{4}$$

The number of URAMs used by each hash table set is

$$HTSUN = M \times \left(\left\lceil \frac{HD}{4096} \right\rceil \times \left\lceil \frac{K+V+1}{72} \right\rceil \right). \tag{5}$$

The number of BRAM36Ks used by each CAM based EM table is

$$CAMBN = \left\lceil \frac{K}{log_2^{512}} \right\rceil \times \left\lceil \frac{CD}{72} \right\rceil + \left\lceil \frac{CD}{512} \right\rceil \times \left\lceil \frac{V}{72} \right\rceil. \tag{6}$$

The total memory blocks consumed by the entire EM table is *TUN* URAM blocks and *TBN* BRAM36K blocks, in which

$$TUN = P \times \left(M \times \left(\left\lceil \frac{HD}{4096} \right\rceil \times \left\lceil \frac{K+V+1}{72} \right\rceil \right) \right), \tag{7}$$

and

$$TBN = P \times \left(\left\lceil \frac{K}{log_2^{512}} \right\rceil \times \left\lceil \frac{CD}{72} \right\rceil + \left\lceil \frac{CD}{512} \right\rceil \times \left\lceil \frac{V}{72} \right\rceil \right). \tag{8}$$

Table 2 shows the resource utilization for different widths of key and value in our implementation. In entire EM Tables, there are totally 1048K address spaces in hash tables and 4K address spaces to store conflicted entries in CAM-based EM tables. In general, the logical resource occupation remains within a reasonable range, which reserves enough resource and frequency space for the implementations of other on-board applications. The utilization of storage resources is directly proportional to the width of keys or values. However, in some cases, because of the SRAM must be used in the unit of an entire block, there may be storage waste, which is inevitable in FPGA implementations. In addition, when the depth of a matching table remains constant, increasing the width may lead to a decrease in achievable frequency. This is because increasing the width requires more on-board resources and may result in more complex routing and longer signal propagation paths. This increases wire delay and limits the operating frequency of the entire matching table.

Table 2 Resource utilization of 1048K entries exact matching table on U250 FPGA.

Value Width (bits)	Key Width (bits)		LUT (%)	FF (%)	BRAM (%)	URAM (%)	Frequency (MHz)
64	16	Total EM	3.45	1.66	4.61	40	200
		Hash Part	2.29	1.06	0	40	
		Cam Part	1.13	0.57	4.61	0	
	32	Total EM	4.09	1.96	8.93	40	200
		Hash Part	2.43	1.08	0	40	
		Cam Part	1.63	0.84	8.93	0	
	64	Total EM	5.69	2.28	17.56	40	192.308
		Hash Part	3.09	1.13	0	40	
		Cam Part	2.56	1.1	17.56	0	
	128	Total EM	7.86	2.45	32.66	60	163.934
		Hash Part	3.98	1.22	0	60	
		Cam Part	3.82	1.15	32.66	0	
128	16	Total EM	4.52	2.29	4.91	60	185.185
		Hash Part	3.23	1.64	0	60	
		Cam Part	1.18	0.6	4.91	0	
	32	Total EM	5.5	2.56	9.23	60	185.185
		Hash Part	3.69	1.62	0	60	
		Cam Part	1.76	0.88	9.23	0	
	64	Total EM	7.18	2.88	17.86	60	178.571
		Hash Part	4.39	1.66	0	60	
		Cam Part	2.74	1.15	17.86	0	
	128	Total EM	9.18	3.04	32.96	80	150.015
		Hash Part	5.28	1.74	0	80	
		Cam Part	3.84	1.21	32.96	0	

5.3 Performance Evaluation

For each hash table set, its collision rate is $r_{collison}$, which also means the probability of transferring from one hash table set to its adjacent channel after the insertion failure. As can be seen from Fig. 10(d), when there are a larger number of hash table blocks, almost all entries can be successfully inserted into its first hash table set. Taking an example when each hash table set has 64 hash table blocks (i.e., M=64), the collision rate $r_{collison}$ is approximately 0.00785, which means the entries rarely moves to other channels to insert. Hence, the speed of insertion would be effectively enhanced by multiple parallel pipeline channels.

For queries, the total number of address spaces in each hash table set is equal, so the depth of each hash table set is $\frac{1}{P}$ of the entire table (here do not consider the minimal number of entries in CAM), and the probability of a successful query for each table entry in the current table is $\frac{1}{P}$. On average, the expected number of hash table sets to be queried for each table entry can be denoted as Ep. Therefore, in average cases, it is equivalent to having $\frac{P}{Ep}$ channels working in parallel in our EM table.

The expected number of tables to be queried for each entry is

$$\begin{aligned} E_p = & \frac{1}{P} + 2\frac{1}{P}\left(1-\frac{1}{P}\right) + \cdots + i\frac{1}{P}\left(1-\frac{1}{P}\right)^{i-1} + \cdots \\ & + (P-1)\frac{1}{P}\left(1-\frac{1}{P}\right)^{P-2} + P\left(1-\frac{1}{P}\right)^{P-1}, \end{aligned} \tag{9}$$

$$= \sum_{i=1}^{P-1}\left(\frac{i}{P}\left(1-\frac{1}{P}\right)^{i-1}\right) + P\left(1-\frac{1}{P}\right)^{P-1}. \tag{10}$$

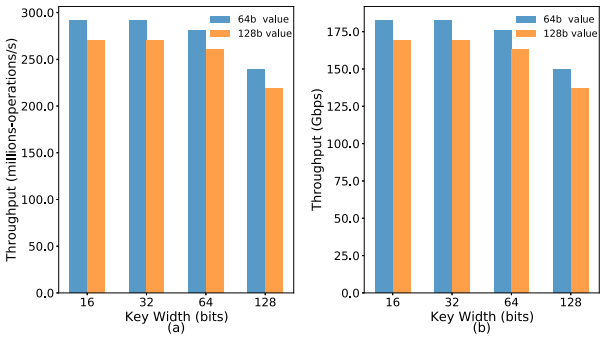


Fig. 11 Throughput under different EM configuration. (a) Throughput of operations. (b) Throughput of packets with length of 64B.

For the entire EM table, although each operation has a certain latency, both insert operations and query operations are pipelined. Therefore, the EM table can do $\frac{P}{Ep}$ operations per clock cycle on average. Therefore, the operation throughput of the EM table is

$$Operation\ Throughput = \frac{P}{Ep} * Freq. \tag{11}$$

For packets, the theoretical throughput that can be achieved is

$$Throughput = \frac{P}{Ep} * Freq * (Pkt\ Len + 20)B. \tag{12}$$

The additional 20 bytes are the extra overhead of packet transferring in network, which includes: 12 bytes inter frame gap (IFG) which is the minimum frame gap of Ethernet packets (IEEE 802.3), 7 bytes preamble for clock synchronization and 1 byte start of frame delimiter (SFD) for identifying the start of the frame.

In our implementation, the number of pipelines is 4. When P is 4, $Ep(P = 4) \approx 2.73$, and $\frac{P}{Ep} \approx 1.46$. Therefore, the EM table can handle 1.46 operations per clock cycle on average. According to the implementation frequency of EM in Table 2, we can calculate the operation throughput and supported packet throughput of EM table in different cases.

Figure 11 shows the number of query operations that EM can handle per second and the corresponding 64B packet throughput under different conditions. A smaller EM table is easy to achieve higher throughput because it could reach a higher working frequency. Overall, the entire EM table can process more than 200 million of operations per second, and can reach a throughput of about 125 Gbps for 64B packets.

6. Related Work and Discussion

Exact matching table is a research hotspot and is widely used in database, key-value store and packet classification etc., and hash-based exact matching table is a mainstream method on FPGA implementation. Researchers mainly focusing on scale expansion, hash collision handling and throughput enhancement of hash table on FPGA, which are also our main work in this paper.

With the continuous expansion of the network scale, the size of the matching table is also increasing. Although hash table is a storage efficient structure, the implementation of huge matching tables on FPGA would still encounter problems such as implement difficulties, resource constraints, and frequency reduction etc. Besides this, solving the hash collision problem is one of the key challenges to achieve accurate matching tables. Researchers use different methods to reduce collisions, such as using better hash functions, open addressing methods, chain methods, etc. Implementing collision resolution algorithm on FPGA needs to consider the balance between hardware resource utilization efficiency and throughput. To increase throughput, the researchers explored a variety of approaches. For example, parallel access is achieved by querying and manipulating multiple hash buckets in parallel.

Y.Z. Li [16] proposed a non-collision hash scheme using bloom filter (BF) and CAM to ensure that each lookup accesses memory at most once. An additional CAM is used to store the conflicting entries of hash table. And a bloom filter to pre-detect if an entry is in hash table ensures that each lookup accesses hash table or CAM at most once. It achieves better worst-case performance and has greater flexibility to quickly insert or query entries. However, bloom filter has some problems such as the difficulty of deleting, and it consumes a lot of resources when implementing a large matching table, which is not feasible in practice.

M. Sha [5] proposed to solve the cuckoo hash conflicts by using a set of distributed RAM as auxiliary storage, which is actually a small CAM implemented by distributed RAMs and registers. However, it has limited scalability especially under a bigger depth requirement of CAM when the matching table has huge depth. Additional, there may be uncertainty in the insertion time in this design because of the cuckoo hashing and the entries in extended table may be rewritten back into the hash table.

Yang et al. [10] proposed FASTHash to optimize hash table throughput through multiple parallel pipeline designs. In this design, it carries out memory replication on each pipeline, which means the tables in each pipeline are the same. Although it improves the throughput of the hash table, it consumes great storage resources to store the same rules multiple times, and it is infeasible to do memory replication on resource-limited FPGA when the size of the matching table is very huge. In addition, although the design reserves multiple slots for each address space to avoid hash collision, it does not solve the hash conflict more thoroughly. In some cases, there will still be entries that cannot be inserted successfully.

Salvatore Pontarelli Pedro Reviriego et al. [11] compared serial, parallel and parallel-pipeline hash table implementations, and proposed parallel d-pipeline implementation which increases the throughput by accessing the tables in parallel. However, the hash collision in cuckoo hashing does not further be solved in this design.

W.Q. Wu et al. [12] introduced CAM into d-Pipeline to address hash collision further and kept the high throughput

Table 3 Comparison with other methods.

Architecture	Collision Free	Constant latency	Auxiliary CAM/Memory	Multiple Pipelines	Memory Replication	Throughput	Load Factor of Hash Table	Out-of-Order Recovery
BF-Hash-CAM[16]	Yes	Yes	Yes	No	No	Low	Low	No need
FASTHash [10]	No	Yes	No	Yes	Yes	High	High	No need
d-Pipeline [11]	No	No	No	Yes	No	High	Mid	No
[12]	Yes	No	Yes	Yes	No	High	Mid	No
Proposed	Yes	Yes	Yes	Yes	No	High	High	Yes

of parallel hash tables. However, the structure has only one CAM unit after multiples hash tables. When load factor of hash table is high, the insertion of hash table is difficult and multiple conflicted entries need to access CAM simultaneously. Moreover, the CAM it used needs 16 clock cycles to finish a write operation. Limited by the writing speed of CAM, if there is entry to be written to CAM, the hash table needs to stall and wait its completion. The CAM cannot be adapted to the parallel hash tables with high throughput. In addition, the out-of-order problem is not considered in the design, and it is not applicable in some scenarios that require the sequence of network packets.

Table 3 shows the comparison of our work with existing methods. Based on resource considerations, we did not adopt the bloom filter in our design like BF-HASH-CAM [16]. Compared to [5], it does not replace the existing entry in insertion when collision occurs in our design, which avoids the uncertain insertion latency caused by cuckoo hashing. By sharing the rule matching table among multiple pipeline channels, our method avoids storage replication in FASTHash [10] and improves throughput. At the same time, the load factor is enhanced by increasing the number of hash table blocks, and hash conflicts are handled by auxiliary CAM units. In addition, for the network packet processing scenario, this paper specially considers the out-of-order recovery in the multiple parallel pipeline channels, which is not considered in the design of d-Pipeline [11] and [12].

Actually, there are several dedicated SmartNICs products or solutions to offload SDN packet processing these years, such as Nvidia's ConnectX series [23], Xilinx's Alveo U25 [24] and SN1000 [25] SmartNICs, Microsoft's Bluebird [26] etc. The proposed matching table is an platform-independent module, it could be embedded into these systems to support SDN packet processing as well.

7. Conclusions

In summary, this paper presented a large-scale high-throughput collision-free EM table which shares rule tables and with out-of-order recovery among multiple parallel pipelines for the network packet application based on FPGA. By multiple channels working parallelly and sharing their rule tables, the throughput of the entire table is increased by about 1.5 times without storage replication. All matching results would be reordered to ensure the operation sequence and the constant processing latency in each pipeline. Moreover, it reduces the collision rate through multiple hash table blocks, and stores conflicted entries into auxiliary CAM tables.The implemented exact match table supports 200 million query operations per second, which is enough to support exceeding 100 Gbps throughput even for 64B packets.

Acknowledgments

This research was funded by National Key Research and Development Program of China: Software-defined interconnecting chip and supporting software kit development (Project.No. 2022YFB2901004).

References

[1] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, and D. Walker, "P4: Programming protocol-independent packet processors," SIGCOMM Comput. Commun. Rev., vol.44, no.3, pp.87–95, July 2014.

[2] P. Bosshart, G. Gibb, H.-S. Kim, G. Varghese, N. McKeown, M. Izzard, F. Mujica, and M. Horowitz, "Forwarding metamorphosis: Fast programmable match-action processing in hardware for SDN," ACM SIGCOMM Comput. Commun. Rev., vol.43, no.4, pp. 99–110, 2013.

[3] R. Shubbar and M. Ahmadi, "Fast 2D filter with low false positive for network packet inspection," IET Networks, vol.6, no.6, pp.224–231, 2017.

[4] P. Reviriego, G. Levy, M. Kadosh, and S. Pontarelli, "Algorithmic tcams: Implementing packet classification algorithms in hardware," IEEE Commun. Mag., vol.60, no.9, pp.60–66, 2022.

[5] M. Sha, Z. Guo, K. Wang, and X. Zeng, "A high-performance and accurate FPGA-based flow monitor for 100 Gbps networks," Electronics, vol.11, no.13, p.1976, 2022.

[6] M. Sha, Z. Guo, and M. Song, "A review of FPGA's application in high-speed network processing," J. Network New Media, vol.10, pp.1–11, 2021.

[7] M. Irfan, A.I. Sanka, Z. Ullah, and R.C. Cheung, "Reconfigurable content-addressable memory (CAM) ON FPGAs: A tutorial and survey," Future Generation Computer Systems, vol.128, pp.451–465, 2022.

[8] J. Carter and M. Wegman, "Universal classes of hash functions (extended abstract)," Proc. ninth Annual ACM Symposium on Theory of Computing, STOC'77, pp.106–112, 1977.

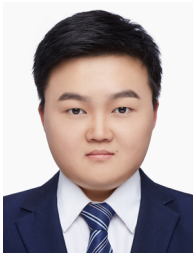
[9] R. Pagh and F.F. Rodler, "Cuckoo hashing," J. Algorithms, vol.51, no.2, pp.122–144, 2004.

[10] Y. Yang, S.R. Kuppannagari, A. Srivastava, R. Kannan, and V.K. Prasanna, "FASTHash: FPGA-based high throughput parallel hash table," Proc. 35th International Conference, ISC High Performance 2020, High Performance Computing, Frankfurt/Main, Germany, pp.3–22, Springer, June 2020.

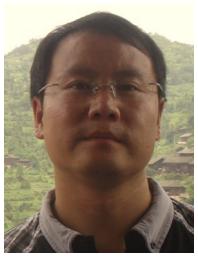
[11] S. Pontarelli, P. Reviriego, and J.A. Maestro, "Parallel d-pipeline: A cuckoo hashing implementation for increased throughput," IEEE Trans. Comput., vol.65, no.1, pp.326–331, 2015.

[12] W.-Q. Wu, M.-T. Xue, T.-Q. Zhu, Z.-G. Ma, and F. Yu, "High-throughput parallel SRAM-based hash join architecture on FPGA," IEEE Trans. Circuits Syst. II, Exp. Briefs, vol.67, no.11, pp.2502–2506, 2020.

[13] M. Kekely and J. Korenek, "Mapping of P4 match action tables to FPGA," 2017 27th International Conference on Field Programmable Logic and Applications (FPL), IEEE, pp.1–2, 2017.
[14] D.-H. Le, K. Inoue, and C.-K. Pham, "Design a fast CAM-based exact pattern matching system on FPGA and 0.18 μm CMOS process," IEICE Trans. Fundamentals, vol.E96-A, no.9, pp.1883–1888, Sept. 2013.
[15] Z. István, G. Alonso, M. Blott, and K. Vissers, "A flexible hash table design for 10 GBPS key-value stores on FPGAS," 2013 23rd International Conference on Field Programmable Logic and Applications, IEEE, pp.1–8, 2013.
[16] Y. Li, "Non-collision hash scheme using Bloom filter and CAM," 2009 Second Pacific-Asia Conference on Web Mining and Web-based Application, IEEE, pp.55–58, 2009.
[17] M. Kekely, L. Kekely, and J. Korenek, "Memory aware packet matching architecture for high-speed networks," 2018 21st Euromicro Conference on Digital System Design (DSD), IEEE, pp.1–8, 2018.
[18] M. Ramakrishna, E. Fu, and E. Bahcekapili, "Efficient hardware hashing functions for high performance computers," IEEE Trans. Comput., vol.46, no.12, pp.1378–1381, 1997.
[19] H. Krawczyk, "LFSR-based hashing and authentication," Proc. 14th Annual International Cryptology Conference, Advances in Cryptology — CRYPTO'94, Santa Barbara, California, USA, pp.129–139, Springer, Aug. 1994.
[20] W. Jiang, "Scalable ternary content addressable memory implementation using FPGAs," Architectures for Networking and Communications Systems, IEEE, pp.71–82, 2013.
[21] G.H. Gonnet, "Expected length of the longest probe sequence in hash code searching," J. ACM (JACM), vol.28, no.2, pp.289–304, 1981.
[22] Xilinx, "Alveo u200 and u250 data center accelerator cards data sheet (ds962)," Online, 2023, https://docs.xilinx.com/r/en-US/ds962-u200-u250
[23] Nvidia, "ConnectX-7 400G Adapters," Online, 2023, https://nvdam.widen.net/s/csf8rmnqwl.infiniband-ethernet-datasheet-connectx-7-ds-nv-us-2544471
[24] Xilinx, "Alveo U25 Product Brief," Online, 2023, https://www.xilinx.com/content/dam/xilinx/publications/product-briefs/alveo-u25-product-brief.pdf
[25] Xilinx, "Alveo SN1000 SmartNICs Data Sheet (DS989)," Online, 2023, https://docs.xilinx.com/v/u/en-US/ds989-sn1000
[26] M. Arumugam, D. Bansal, N. Bhatia, J. Boerner, S. Capper, C. Kim, S. McClure, N. Motwani, R. Narasimhan, U. Panchal, T. Pimpo, A. Premji, P. Shrivastava, and R. Tewari, "Bluebird: High-performance SDN for bare-metal cloud services," 19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22), Renton, WA, pp.355–370, USENIX Association, April 2022.



Xiaoyong Song received the B.S. degree from Beijing University of Technology, Beijing, China, in 2019. He is currently pursuing the doctor's degree with the School of Electronic, Electrical and Communication Engineering, University of Chinese Academy of Sciences, Beijing. His current research interest includes FPGA network acceleration, and matching table etc.



Zhichuan Guo received the B.S. degree from Wuhan University in 1996, and the Ph.D. degree from the University of Science and Technology of China in 2006. From 1996 to 2003, he served as an Electronics Engineer with the 13th Research Institute of China Electronics Technology Group Corporation and a SDH hardware R&D system engineer of optical networks at Huawei. In 2006, he joined with the Institute of Acoustics, Chinese Academy of Sciences, Beijing, China. Now he is a Professor of CAS engaging in field programmable gate array (FPGA)-based code acceleration, VLSI, and security.

Xinshuo Wang received the B.E. degree from Chongqing University of Posts and Telecommunications, Chongqing, China, in 2021. At present, he is studying for a doctorate degree in the school of electronic. electrical and communication engineering of the University of Chinese Academy of Sciences in Beijing, focusing on the field of FPGA network acceleration.

Mangu Song is currently working at the Institute of Acoustics, Chinese Academy of Sciences (IACAS), as a research assistant. She received her M.Sc degree in electronics and communication engineering from the School of Microelectronics, Chinese Academy of Science, Beijing, China in 2017. Her current research interests include FPGA-based code acceleration and network security.

IEICE TRANS. COMMUN., VOL.E107–B, NO.5 MAY 2024

PAPER

PopDCN: Popularity-Aware Dynamic Clustering Scheme for Distributed Caching in ICN*

Mikiya YOSHIDA[†a)], **Yusuke ITO**[††b)], **Yurino SATO**[†††c)], ***and*** **Hiroyuki KOGA**[††d)], ***Members***

SUMMARY Information-centric networking (ICN) provides low-latency content delivery with in-network caching, but delivery latency depends on cache distance from consumers. To reduce delivery latency, a scheme to cluster domains and retain the main popular content in each cluster with a cache distribution range has been proposed, which enables consumers to retrieve content from neighboring clusters/caches. However, when the distribution of content popularity changes, all content caches may not be distributed adequately in a cluster, so consumers cannot retrieve them from nearby caches. We therefore propose a dynamic clustering scheme to adjust the cache distribution range in accordance with the change in content popularity and evaluate the effectiveness of the proposed scheme through simulation.

key words: *ICN, distributed caching, dynamic clustering*

1. Introduction

Information-centric networking (ICN) [3], [4] has been attracting attention as a new architecture that uses network caching to satisfy the requirements (e.g., ultra-low latency, ultra-high reliability, and massive connectivity) for emerging services such as IoT-like automation, robotics, and industrial automation [5], [6]. In ICN, a consumer sends interest packets containing content names to request content. A content router (CR), which is an intermediate router receiving the interest packets, forwards the packets to a producer on the basis of a routing table called a forwarding information base (FIB). The producer then returns data packets of the requested content with the reverse path to consumers. The CRs cache data packets on their content store (CS) during forwarding, so they can return caches to consumers instead of the producer if they store the requested data. Namely, this in-network caching, which can satisfy the future requests of consumers, significantly reduces the network load and improves content delivery efficiency. To take full advantage of in-network caching, an efficient content caching scheme is needed, and various schemes have been proposed.

Simple content caching makes a cache decision on individual CRs, while distributed content caching distributes content by considering nearby caches to satisfy content requests. Distributed caching solves a cache efficiency problem that simple content caching causes cache duplication for a small amount of highly popular content over neighboring CRs. However, if the cached required content is distributed over a large range, delivery latency may increase.

Therefore, cluster-based distributed caching schemes have been proposed [7], [8] to control a distributed range. These schemes group CRs into clusters in a domain** and retain the main popular content in each cluster using a distributed caching manner. This aims to avoid cache duplication among CRs in the cluster, enabling the caches of each content to be distributed within it. As a result, the delivery latency can be controlled by cluster size, and it enables consumers to retrieve content efficiently from the originating clusters. However, a too-small distribution range decreases cache utilization and causes delivery delays due to the delivery from producers, while a too-large distribution range increases cache utilization but may cause delivery delays due to long cache delivery. Therefore, we believe that the adequate cache distribution range should be determined in accordance with content popularity on the basis of such trade-off factors. In a practical environment, the distribution of content popularity changes over time, so it is necessary to determine the distribution range depending on the situation.

We therefore propose a dynamic clustering scheme to adjust the cache distribution range, i.e., cluster size, in accordance with the change in content popularity, considering cache utilization and delivery latency. Our scheme controls the cluster size effectively using a simple threshold-based algorithm based on the number of cache updates on CRs in a cluster. Moreover, we evaluate the effectiveness of the proposed scheme compared with conventional schemes through simulation in a situation where content popularity changes.

The main contributions of this paper, updated from previous papers [1], [2], are as follows:

- We discuss recent studies that utilize clustering techniques in ICN.
- We evaluate the proposed scheme compared with conventional schemes in detail and discuss the effective

Manuscript received September 21, 2023.
Manuscript publicized January 30, 2024.

[†]The author is with the Center for Information Technology and Management, Okayama University, Okayama-shi, 700-8530 Japan.
[††]The authors are with the Graduate School of Environmental Engineering, The University of Kitakyushu, Kitakyushu-shi, 808-0135 Japan.
[†††]The author is with the Department of Control Engineering, National Institute of Technology (KOSEN), Sasebo college, Sasebo-shi, 857-1193 Japan.

*Earlier version of this paper was presented at ACM ICN2022 and APSIPA ASC2022 [1], [2].

a) E-mail: m-yoshida@okayama-u.ac.jp
b) E-mail: y-ito@kitakyu-u.ac.jp
c) E-mail: y-sato@sasebo.ac.jp
d) E-mail: h.koga@kitakyu-u.ac.jp

DOI: 10.23919/transcom.2023EBP3152

**In this paper, the term 'domain' refers to a large-scale network consisting of one or more ISPs.

threshold settings.
- We investigate the effectiveness of the proposed scheme in practical network topologies.

The rest of this paper is organized as follows. In Sect. 2, we describe our motivation for this study through a discussion of related works. In Sect. 3, we describe our scheme. In Sect. 4, we describe the simulation model and evaluation details. In Sect. 5, we evaluate the performance of our scheme in comparison with conventional schemes. Finally, in Sect. 6, we summarize our findings and conclude the paper.

2. Related Work

In this section, we describe an issue of this study through discussions of various content caching schemes to improve content delivery efficiency.

2.1 Distributed Caching

Simple caching schemes such as LCE [3] and Prob(p) [9] make a cache decision on individual CRs. This may cause cache duplication for a few high-popular contents over neighboring CRs because more frequently requested content is likely to be cached. This becomes useless for other content requests. Therefore, distributed caching schemes have been proposed such as MCD [10], WAVE [11], MuNCC [12], and Hash-routing [13], which distribute various content considering nearby caches to satisfy various content requests. The key idea of MCD and WAVE schemes is that each CR moves requested caches to downstream CRs. Namely, the CR caching the requested content sends its cache to the downstream CR and removes it from itself, so that each cache is not duplicated on the default path, i.e., the shortest path to consumers. However, it is unable to avoid cache duplication among neighboring CRs outside the default path.

In contrast, in MuNCC and proposed in [14] schemes, each CR shares cache summaries that are formed using a Bloom filter among neighboring CRs to avoid cache duplication. When a data packet arrives, a CR determines whether it caches it or not depending on the cache summaries of neighboring CRs. The Hash-routing scheme distributes content to CRs using a hash function that maps content identifiers to each CR of the domain, without additional functionality such as shared cache summaries. In particular, when an edge router in the domain receives a request, it calculates the hash value from the received content identifier and forwards it to the responsible CR. Similarly, each CR caches the responsible content whose hash value matches its identifier during forwarding. As a result of this approach, since the cache location of each content is limited to one CR over a domain, it can avoid cache duplication among CRs. However, if the cached required content is distributed over a large range, delivery latency may increase.

2.2 Cluster-Based Distributed Caching

To control the cache distribution range considering delivery latency, network clustering-based distributed caching schemes for ICN have been proposed [7], [8], [15], [16]. These schemes group CRs into clusters in a domain and retain the main popular content in each cluster using a Hash-routing-like distributed caching manner. The delivery latency can thus be controlled by cluster size, enabling consumers to retrieve content efficiently from the originating clusters. As a scheme similar to the aforementioned ones, the HCC [17] scheme has also been proposed. It centrally manages the distributed caches by a cluster header constructed in each cluster. The cluster header calculates the content popularity and importance of each node on the basis of information collected from the cluster, and then assigns the more popular content to the more important node to improve cache efficiency and delivery latency.

However, the amount of content that can be cached in the cluster depends on the cluster size. In other words, a smaller cluster size is insufficient to reduce delivery latency since it cannot cache necessary content sufficiently in the cluster. As mentioned before, in this study, we believe that it is necessary to determine the adequate cache distribution range in accordance with content popularity on the basis of the following trade-off factors. A too-small cache distribution range against the amount of main popular content will not retain sufficient caches, so it decreases cache utilization and causes delivery delays due to the delivery from producers. A too-large cache distribution range can satisfy most user requests within the cluster but causes delivery delays due to the delivery from widely distributed caches. In a practical environment, the distribution of content popularity, i.e., the amount of main popular content, will change over time [18], so it is necessary to determine the distribution range adequately depending on the situation.

3. Proposed Scheme

We propose a dynamic clustering scheme to adjust the cache distribution range in accordance with the change in content popularity. This scheme is an extended version of our previous work [7] that formed a fixed size of clusters. In this section, we explain the operation of the proposed scheme. We first explain the main points of the previous work in Sect. 3.1, and then explain the extension in detail in Sect. 3.2.

3.1 Cluster-Based Cache Distribution Scheme

To improve delivery latency and cache efficiency, we have proposed the cluster-based cache distribution scheme. It groups CRs into clusters in a domain and retains the main popular content in each cluster using a distributed caching manner, enabling consumers to retrieve content from the originating clusters. Furthermore, it can also retrieve caches from closer CRs by advertising cache information among CRs. In the following, we explain two functions of cluster-based distributed caching and advertisement-based routing.

The distributed caching approach uniformly distributes chunks of individual content to all CRs in each cluster, as

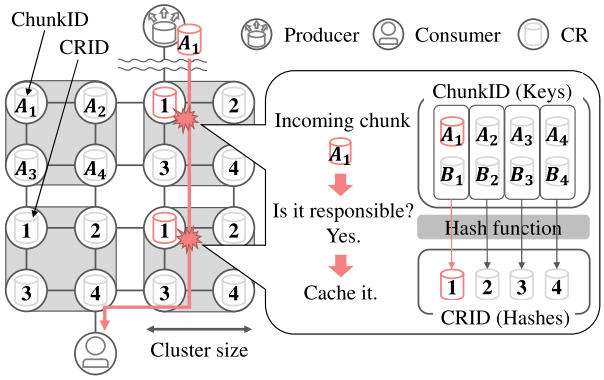


Fig. 1 Cache placement.

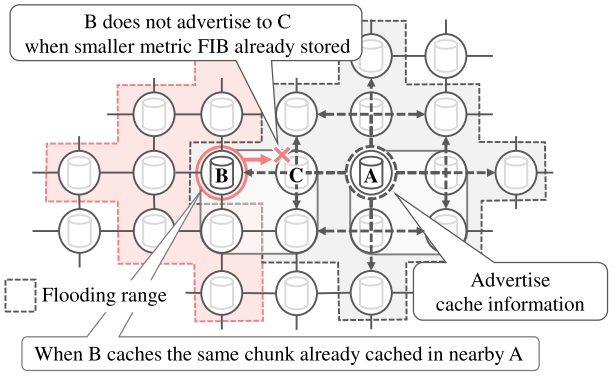


Fig. 2 Cache information advertisement.

shown in Fig. 1. This approach improves cache efficiency by avoiding cache duplication in the cluster, leading to more cached content in it. Furthermore, transmission efficiency can also be improved by multi-path cache delivery from multiple CRs (i.e., load balancing).

To uniformly distribute chunks in a cluster, this scheme partitions a domain into clusters of the same size and assigns unique identifiers (CRIDs) to each CR in advance. To avoid cache redundancy among CRs in a cluster, it uses a hash function that maps chunk identifiers to CRIDs. Specifically, when a CR receives a chunk, it caches it as a responsible one if the hash value calculated from the received chunk identifier matches its own CRID. The Least Recently Used (LRU) cache replacement algorithm is used for spaces on the CS. In this study, each cluster is assumed to be a square shape. The cluster size is defined as the number of CRs on one side (as shown in Fig. 1 is 2), which affects the cache efficiency and distance from consumers. Note that the shape of clusters is not important because the main popular content will be retained in clusters if CRIDs are properly assigned within clusters in any topologies. For example, it can be accomplished by defining the cost as the distance between a CR and the nearest CR with a different CRID and solving the problem of minimizing the total cost. This will ensure that each CRID is assigned almost uniformly without bias according to the cluster size, i.e., the number of CRIDs to be assigned. Since each CR is neighboring to CRs with a CRID different from its own, each cluster is nearly a circle shape.

Even if caches are uniformly distributed within a cluster, consumers may not efficiently retrieve all chunks of the requested content from the originating cluster. This is because not all chunks will be cached due to the limitation of total cache capacity in a cluster, or there may be caches on closer CRs in neighboring clusters than those in the originating cluster. Therefore, requests should be forwarded to the nearest caches even those not in the originating clusters regardless of cluster boundaries for efficient content delivery, so the advertisement-based routing approach is used, which forwards interest packets to nearby caches on the basis of the advertised cache information.

To achieve this behavior, each CR informs neighboring CRs of their own responsible cache status. Specifically, CRs that newly cache or discard responsible chunks advertise the cache information (newly cached/discarded) in the flooding manner shown in Fig. 2. The CR receiving the advertised packet updates its FIB entry with the received cache information. Considering the overhead of this operation, the flooding range should be limited but would affect the content retrieval efficiency, which is defined as the flooding limit parameter (as shown in Fig. 2 is 2). This operation is performed only when responsible chunks are cached or discarded, thereby reducing the overhead compared with conventional schemes flooded for all cached chunks such as proposed in [19]. Moreover, to reduce the load caused by flooding, our scheme simply discards and does not forward the flooding packets when it can be determined that neighboring CRs do not need to update their FIB. Let us explain this process using the example shown in Fig. 2. When CR A caches responsible chunks, it advertises its cache information to neighboring CRs (gray-colored range). After that, when CR B caches the same chunk, it can decide not to flood to CR C and advertises the cache information to neighboring CRs except it (red-colored range). This is because CR B has an FIB entry with metric of 2 hops for the chunk by advertised information from CR C and it indicates that CR C already has a valid metric of 1 hop that does not need updating. Namely, if the CRs already have FIB entries of plus 2 hops or fewer metrics than the flooding one, it does not need to advertise it in that direction. Note that this scheme increases overheads including cache information sharing and FIB entry increases to improve acquisition efficiency compared to on-path routing schemes as an inherent issue of off-path routing schemes. To resolve this issue (overheads caused by off-path extension), several solutions (e.g., a Bloom filter approach [12], [14], [20]) have been proposed, while we focus on reducing delivery latency by adjusting cache distribution range while considering only communication overheads caused by flooding in this study so that we will leave this issue for future work.

3.2 Popularity-Aware Dynamic Clustering Scheme

As previously mentioned, the cluster size, i.e., cache distribution range, should be adequately determined in accordance

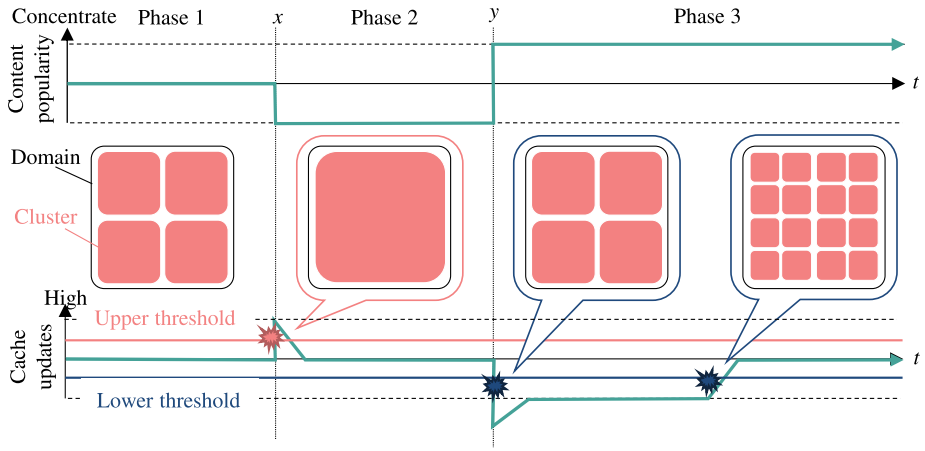


Fig. 3 Operation of dynamic clustering.

with content popularity. In a practical environment, the distribution of content popularity changes over time, so it is necessary to determine the distribution range depending on the situation. We therefore propose a dynamic clustering scheme to adjust the cache distribution range in accordance with the change in content popularity, considering cache utilization and delivery latency. Our scheme controls the cluster size effectively using a simple threshold-based algorithm based on the number of cache updates in the cluster.

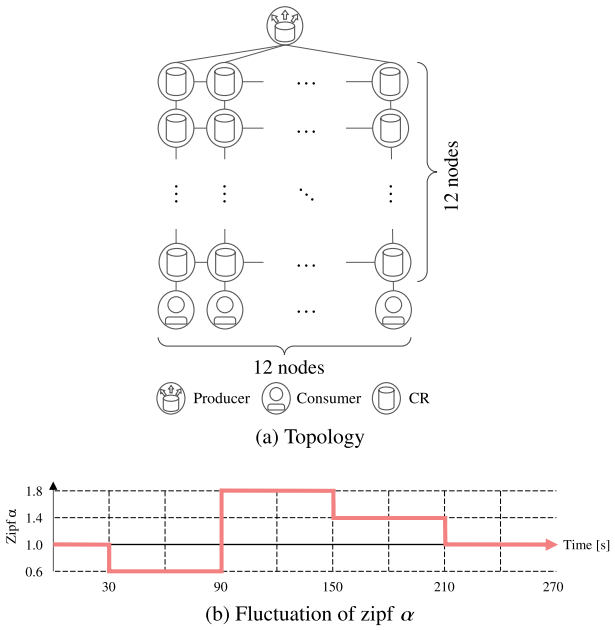
To discuss the adequate cluster size, we focus on the frequency of cache updates in a cluster. This is because this metric is useful to estimate whether the current cluster size is suitable to cache the main popular content. When the frequency of cache updates is high, it indicates that caches are updated by incoming data packets from outside the cluster. Namely, requested content cannot be retrieved inside the cluster as well as the cluster size is too small. A low frequency of cache updates indicates that caches are not updated since requested content can be retrieved inside the cluster. Namely, the cluster size may be decreased to reduce delivery latency. Thus, we consider that the frequency of cache updates in a cluster would fall into a certain range with the appropriate cluster size.

From the aforementioned strategy, the proposed scheme adjusts the cluster size using a simple threshold-based algorithm based on the frequency of cache updates. Specifically, it uses the number of cache updates in a cluster as a metric, and decreases/increases the cluster size when the metric falls below or exceeds lower/upper thresholds. Figure 3 explains how the proposed scheme migrates to the adequate cluster size in accordance with the change in content popularity. Let us consider a t-second scenario when the content popularity will disperse after x seconds, and then heavily concentrate after y seconds. In phase 1 until x seconds, we assume that each cluster, which represents the domain divided into four parts, can store most of the popular content, so the frequency of cache updates fits between the upper and lower thresholds. Namely, the current cluster size is adequate. In phase 2 from x to y seconds when the content popularity disperses, the frequency of cache updates increases and exceeds the upper threshold because the current cluster size cannot retain the popular content sufficiently. Therefore, the cluster size is increased by one level to store them, and therefore the frequency of cache updates decreases and falls within the upper and lower thresholds. In phase 3 after y seconds when the content popularity is heavily concentrated, the cache update frequency decreases and falls below the lower threshold because the current large cluster size has exceeded the sufficient cache capacity compared with the amount of main popular content. Therefore, it attempts to improve the delivery latency by decreasing the cluster size by one level. However, this cluster size still has an excessive cache capacity, so the frequency of cache updates remains below the lower threshold. Therefore, the cluster size is decreased by one more level, and therefore the frequency of cache updates increases and falls within the upper and lower thresholds. Through these procedures, the cluster size can be migrated to the adequate cluster size in accordance with the change in content popularity.

To achieve this function, we assume that a controller is located in a domain and each CR notifies the controller with the number of cache updates. The controller calculates the total number of cache updates separately in each cluster by the information received from each CR. When at least one of the calculated values falls below or exceeds lower/upper thresholds, it reassigns a new CRID and hash function to each CR to decrease/increase cluster size. The cluster size is not changed for a certain period, which is defined as the reclustering interval parameter, immediately after reclustering to mitigate the effect of the heavy fluctuation of cache updates. We believe that such information sharing between the controller and CRs can be achieved by a mechanism like software defined networking (SDN) and the detailed design of the scheme will be left as future work.

4. Simulation Model

We evaluated the proposed scheme focusing on the effectiveness of retrieving content from nearby clusters/caches in a large domain environment where content popularity



(a) Topology

(b) Fluctuation of zipf α

Fig. 4 Simulation model.

Table 1 Simulation parameters.

Link bandwidth	1 [Gb/s]
Propagation delay time	5 [ms]
Communication protocol	UDP
Amount of Contents	128
Content size	64 [chunk]
Chunk size	1000 [Byte]
CS size on CR	128 [chunk]
Zipf α	0.6, 1.0, 1.4, 1.8
Cluster size	2, 3, 4, 6, 12
Upper threshold	70–280
Lower threshold	10–150
Reclustering interval	1–24 [s]

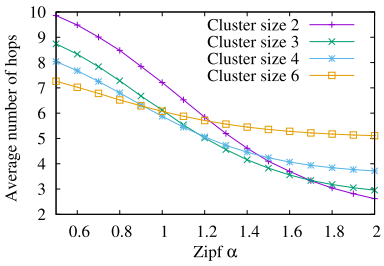
changes through simulations using Network Simulator ns-3 ver. 3.30.1 [21] with the implementation of our scheme. We used a simple grid topology with multiple paths to eliminate the effects of cluster shape and content cache placement within clusters as shown in Fig. 4(a) to enable us to focus on the essential effect of dynamically changing cluster size. One producer and 12 consumers were located on the upper and lower sides of the grid (12×12) of CRs, respectively. The parameters used in the simulation are summarized in Table 1. The default path is the shortest path to the producer (13 hops from each consumer) and it was set to the FIB of each CR. The ratio of the CS size on CR to the amount of content was set to approximately 1.5% on the basis of comparative papers [8], [22]. The flooding limit was set to 6, which was the best value in terms of cost performance between overhead and efficiency in a preliminary evaluation. As mentioned before, the proposed scheme needs to share information among CRs via the controller, which can be achieved by a number of mechanisms like SDN, and we ignore its effect in this simulation since the exchange of shared information is very infrequent and small compared with data delivery. Each CR notifies the controller of the number of cache updates at 1 second intervals.

Each consumer sent interest packets to request content toward the producer at normal distribution intervals with an average value of 0.3 seconds. The requested content was determined on the basis of the content popularity, in which P2P content was generally known to follow a Zipf-mandelbrot distribution [23]. In this distribution, the degree of bias depends on the parameters α and q. α is the skewness factor that controls the slope of the curve, while $q(\geq 0)$ is known as the plateau factor that determines the flatness of the curve. In this simulation, we gave q a fixed value of 5 and changed the content popularity with α to avoid the complexity of the discussion. Furthermore, we assumed no packet loss occurs so we can focus on the fundamental characteristics of the dynamic clustering approach. The simulation was performed for 270 seconds. We set the Zipf parameter α to 1.0 at the start of the simulation as shown in Fig. 4(b). α changed to 0.6 at 30 seconds after the simulation started, in which a wider range of content is requested, to 1.8 at 90 seconds, to concentrate on the requested content, and after that, it decreases by 0.4 every 60 seconds back to 1.0.

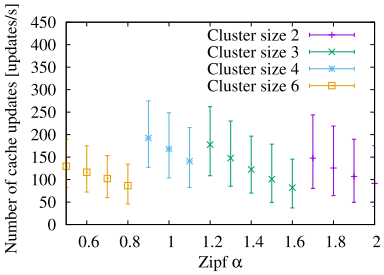
In this simulation, we compared and evaluated the effectiveness of five representative schemes: LCE (LRU), Hash-routing [13], Hash-routing + cluster [8], conventional (Static) [7], and proposed (Dynamic). Furthermore, the average number of hops needed to retrieve content, cache hit rate, and advertisement rate were used as evaluation indices to discuss the effectiveness of our scheme. Note that the Hash-routing + cluster scheme uses the k-split algorithm with the number of hops as similarity metrics for clustering and forms k clusters. The average number of hops focused on content retrieval time, which was defined as the total number of hops during the time when all consumers retrieved content divided by the total number of requests for all consumers. The cache hit rate focused on cache efficiency, which was defined as the total number of cache hits on all CRs divided by the total number of requests for all consumers. The advertisement rate focused on communication overhead, which was defined as the amount of advertisement packets divided by the total amount of traffic. In this study, we assumed the average name length is 30 bytes, and the size of the advertisement packet which includes the content name, the flooding limit, and the flag bit that indicates the cache information (newly cached/discarded), is the same as the Interest packet.

5. Simulation Results

In this section, we first show the effectiveness of our scheme compared with the conventional schemes. Then, we investigate how each parameter including the lower/upper thresholds and reclustering interval affects our scheme. Finally, we investigate the effect of the change interval of Zipf α and network topology to reveal the environmental tolerance and practicality of our scheme.



(a) Average number of hops



(b) Number of cache updates

Fig. 5 Estimation of adequate thresholds.

5.1 Evaluation of Effectiveness Based on Estimation of Adequate Thresholds

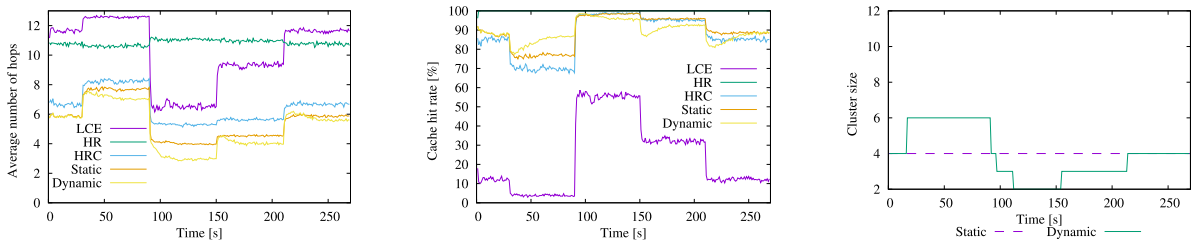
In this section, we first discuss the basis for determining threshold values of the proposed scheme through quantitative evaluations and estimate the effective lower/upper threshold values, which is a key point of the proposed scheme. As mentioned in Sect. 3.2, given an adequate cluster size, the number of cache updates in the cluster falls into a certain range. We believe that the adequate cluster size can be determined in accordance with the distribution of content popularity. Figure 5 shows the average number of hops and cache updates in the cluster when α varies from 0.5 to 2.0. From Fig. 5(a), we can see that the adequate cluster size is 6 when α is less than 0.9, 4 for α of 1.0–1.1, 3 for α of 1.2–1.6, and 2 for α of 1.7 or larger, respectively, since these cluster size achieve the smallest number of hops for each content popularity. Correspondingly, the number of cache updates in the cluster falls into a certain range when the adequate cluster size is given as shown in Fig. 5(b). Specifically, it is approximately 50 or more for the adequate cluster size of 6 (α = 0.9 or less), 90–280 for the size of 4 (α = 1.0–1.1), 50–270 for the size of 3 (α = 1.2–1.6), and 250 or less for the size of 2 (α = 1.7 or above), respectively. From the aforementioned results, if the number of cache updates in the cluster is approximately 50 and more or 280 and less, the given cluster size will be adequate. Namely, the lower/upper threshold values can be set on the basis of the number of cache updates.

On the basis of the aforementioned discussion, we now show the simulation results and discuss the effectiveness of the proposed scheme as compared with the conventional schemes. Here, the lower/upper threshold values were set to 50/280, the reclustering interval was set to 3 seconds, and the initial cluster size of Hash-routing + cluster (HRC), conventional (Static), and proposed (Dynamic) schemes was set to 4, which was the appropriate value for an α of 1.0 at the start of the simulation. Figure 6 shows the average number of hops, cache hit rate, and cluster size as a function of time. From Figs. 6(a) and (b), the LCE scheme shows the worst performance among the other schemes because it causes duplicate caches on nearby CRs. The Hash-routing (HR) scheme improves the performance, especially cache efficiency, compared with the LCE scheme due to no duplicate cache occurrences, but the average number of hops, i.e., delivery latency is not good because the caches are distributed widely. The Hash-routing + cluster (HRC) scheme improves the performance compared with the HR scheme due to controlling the cache distribution range at the cost of a little cache efficiency. The conventional (Static) scheme using advertisement-based routing improves the performance compared with the HRC scheme due to the avoidance of detour routing caused by the false-positive problem with the HR scheme as well as the effect of retrieving nearby caches regardless of cluster boundaries. The proposed (Dynamic) scheme further improves the delivery latency while maintaining the cache hit rate compared with the conventional (Static) scheme in almost all ranges of time because it adjusts the cluster size to an adequate value.

Next, let us take a look at adjusting the cluster size of the Dynamic scheme focusing on three periods where the content popularity changes. First, in the period of 30–90 seconds, a wider range of content becomes to be requested, so that the cluster size is adjusted to a larger value (it is 6, which is an adequate value when α = 0.6 (Fig. 5(a)) due to the high frequency of cache updates as shown in Fig. 6(c). It improves the cache hit rate as well as delivery latency, although it takes time to distribute new caches in the cluster. Second, in the period of 90–150 seconds, the requested content becomes to be concentrated, so that the cluster size is adjusted to a smaller value (it is 2, which is an adequate value when α = 1.8) due to the low frequency of cache updates. It improves the delivery latency, although it takes time to discard unnecessary caches from the cluster, and comes at the cost of a slight decrease in cache hit rate. Finally, in the period of 150–270 seconds, similar to 30–90 seconds the requested content becomes to be a wider range gradually, so that the cluster size is adjusted to larger values (they are 3 and 4, which are adequate values when α = 1.4 and 1.0, respectively). It improves delivery latency while maintaining a high cache hit rate. This adjustment of cluster size is performed by searching for the cluster size that keeps the number of cache updates in the range of 50 to 280. Therefore, the proposed scheme can adapt effectively to the environment where content popularity changes.

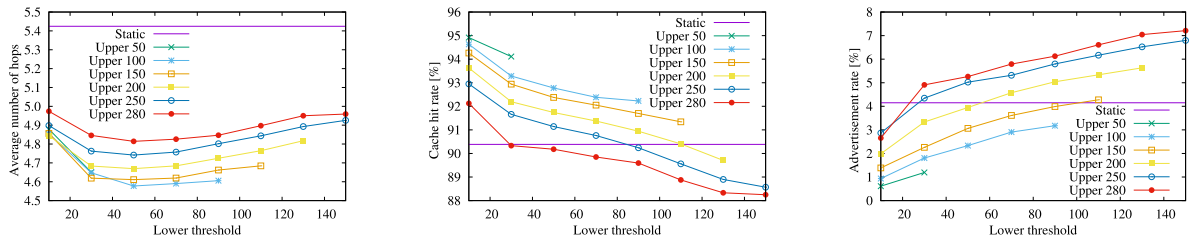
5.2 Effect of Thresholds

Next, we investigate the effect of the thresholds. Figures 7(a), (b), and (c) show the average number of hops, cache hit rate,



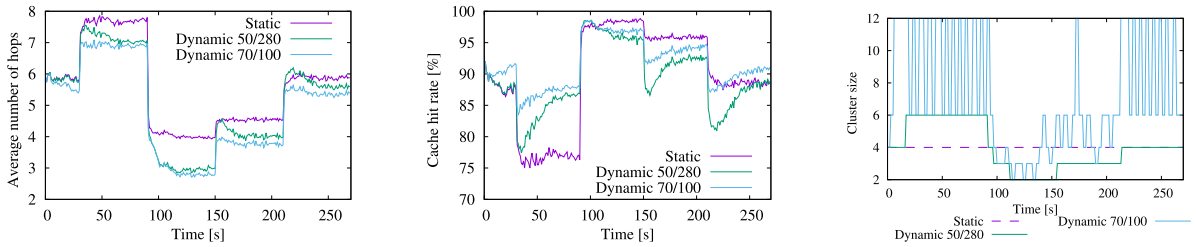
(a) Average number of hops (b) Cache hit rate (c) Cluster size

Fig. 6 Effectiveness of our scheme.



(a) Average number of hops (b) Cache hit rate (c) Advertisement rate

Fig. 7 Effect of thresholds.

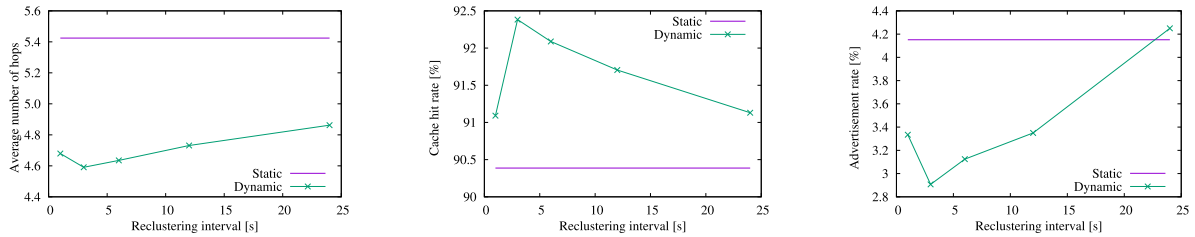


(a) Average number of hops (b) Cache hit rate (c) Cluster size

Fig. 8 Estimated and adequate thresholds.

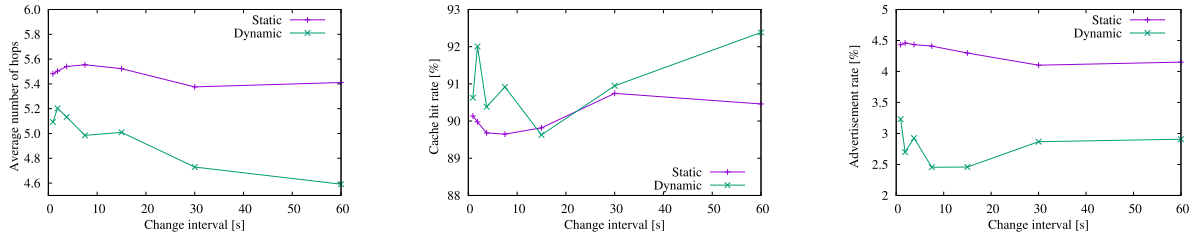
and advertisement rate, respectively, when the lower/upper thresholds vary. Here, the reclustering interval was set to 3 seconds. Figures 7(a) and (b) indicate that the upper and lower threshold values should be set to an appropriate range (neither too large nor too small) to reduce delivery latency and maintain the high cache hit rate. When the upper threshold value is too large, it is difficult to migrate to a larger cluster size despite high frequent cache updates. As a result, it worsens cache efficiency as well as delivery latency. When the upper threshold value is too small, it is easy to migrate to a larger cluster size despite low frequent cache updates. As a result, it improves cache efficiency but increases delivery latency because the caches are widely distributed. The lower threshold observes a similar trend. Consequently, the adequate threshold values should be determined on the basis of the delivery latency and cache hit rate considering these trade-offs. The adequate lower/upper thresholds are 70/100 in this simulation environment, which achieves the lowest number of hops (Fig. 7(a)) and the high cache efficiency (Fig. 7(b)). Furthermore, the adequate cluster size does not cause frequent cache updates and reduces the flooding of advertisement packets for dynamic FIB updates, so the proposed (Dynamic) scheme with adequate thresholds also improves the advertisement rate, i.e., communication overhead, to approximately 2% of the total amount of traffic (Fig. 7(c)).

Here, it is noted that the estimated threshold values and adequate ones are largely different. This indicates that it should aggressively migrate to various sizes of clusters with the setting of larger/smaller lower/upper threshold values to maintain cache hit rates in the environment where the content popularity changes significantly. Figure 8 shows the average number of hops, cache hit rate, and change of cluster size in the conventional (Static) scheme and proposed (Dynamic) scheme with the estimated (50/280) and adequate (70/100) threshold values. Figure 8(c) clearly shows that the proposed scheme with adequate thresholds can more frequently migrate closer to the appropriate cluster size than that with estimated thresholds. Moreover, Fig. 8(a) and (b) show that such migration quickly improves the delivery latency and cache hit rate when the content popularity changes. Consequently, although the proposed scheme with estimated thresholds achieves good performance, it can be further improved by setting adequate thresholds on the basis



(a) Average number of hops (b) Cache hit rate (c) Advertisement rate

Fig. 9 Effect of reclustering intervals.



(a) Average number of hops (b) Cache hit rate (c) Advertisement rate

Fig. 10 Effect of change intervals of Zipf α.

of the aforementioned trade-offs as well as detecting sensitive changes in content popularity to quickly adjust the cluster size with appropriate cache distribution. However, the adequate threshold values may need to be adjusted dynamically in accordance with network conditions (the topology, frequency of requests, etc.), which will be tackled in future work.

5.3 Effect of Reclustering Intervals

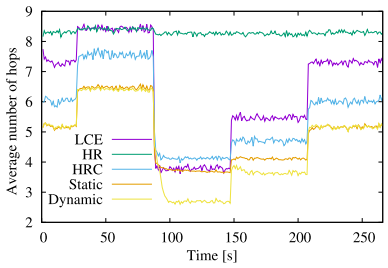
We investigate the effect of reclustering intervals. Figure 9 shows the average number of hops, cache hit rate, and advertisement rate when the reclustering interval varies. Here, the lower/upper thresholds were set to 70/100 (adequate values in this environment). Figure 9(a) shows that shorter reclustering intervals improve delivery latency except for too-short ones. This is because the shorter intervals can quickly migrate to the adequate cluster size and improve cache hit rates as shown in Fig. 9(b). However, too-short intervals inhibit migration to the adequate cluster size due to heavy cache updates immediately after reclustering. In addition, Fig. 9(c) shows that shorter reclustering intervals improve the overhead. This is because unnecessary cache updates are reduced by quickly migrating to the adequate cluster size. Consequently, the reclustering interval should be set to an adequately short value, which is 3 seconds in this environment.

5.4 Effect of Change Intervals of Zipf

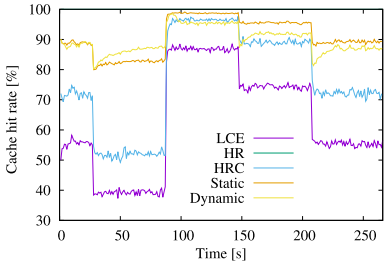
We investigate the effect of change intervals of Zipf α to show the environmental tolerance. For example, when the change intervals are set to 20 seconds, 30 seconds after the simulation starts with Zipf α of 1.0 and a cluster size of 4, Zipf α sequentially changes to 0.6, 1.4, 1.8, and 1.0 every 20 seconds, and these changes are repeated for 240 seconds (until the end of simulation). Figure 10 shows the average number of hops, cache hit rate, and advertisement rate when the change intervals of Zipf α vary. Here, the thresholds of lower/upper were set to 70/100, and the reclustering interval was set to 3 seconds (adequate values in this environment). From Fig. 10, the proposed (Dynamic) scheme always improves the delivery latency, cache hit rate, and overhead compared with the conventional (Static) scheme in a wide range of change intervals. This is because the proposed scheme can adapt cluster sizes smoothly to environments where the content popularity changes frequently.

5.5 Effect of Network Topology

Finally, we evaluate the proposed and conventional schemes comparatively in a practical network topology. We used the Interoute topology of 110 nodes from the Internet Topology Zoo [24] on the basis of comparative paper [8]. Since the dataset shows the relationship of pop-level routers, we defined each node as CR and placed producers and consumers on each CR. Content was randomly placed on each producer. Each consumer sent interest packets requesting content toward the producer at normal distribution intervals with an average value of 1.0 seconds. The Dynamic scheme used the clustering algorithm described in Sect. 3.1 and its initial cluster size was set to approximately 4 (16 CRs in each cluster). The HRC scheme formed 6 clusters by the k-split algorithm. These settings were the appropriate value for an α of 1.0 at the start of the simulation. The Dynamic scheme can migrate the cluster size, which consists of 4, 9, 16, 36, or 110 CRs in each cluster, during the simulation. The reclustering interval was set to 3 seconds, the flooding limit was set



(a) Average number of hops



(b) Cache hit rate

Fig. 11 Effect of network topology.

to 6, and the lower/upper threshold values were set to 70/130, which were the appropriate values in this simulation. Other simulation parameters shall conform to Table 1.

Figure 11 shows the average number of hops and cache hit rate as a function of time. It indicates that the trend is almost the same as the results for the grid topology shown in Sect. 5.1, and the Dynamic scheme always maintains the high cache hit rates and reduces the average number of hops. In addition, regarding communication overheads, the Dynamic scheme achieves smaller advertisement rates of 4.37% than the Static scheme of 6.07%. Therefore, the proposed scheme is effective even in practical network topologies.

6. Conclusion

We proposed a dynamic clustering scheme to adjust the cache distribution range in accordance with the change in content popularity. Our scheme adjusts the cluster size effectively using a simple threshold-based algorithm based on the number of cache updates in the cluster. Simulation evaluations have indicated that the proposed scheme can reduce the delivery latency while consistently maintaining a high cache hit rate in a large domain environment where content popularity changes. In future work, we will investigate more flexible clustering schemes considering content attributes in practical topologies.

Acknowledgments

This work was supported in part by JSPS KAKENHI Grant-in-Aid for Scientific Research (C) Number 21K11872.

References

[1] M. Yoshida, Y. Ito, Y. Sato, and H. Koga, “Popularity-aware dynamic-clustering scheme for distributed caching in ICN,” Proc. ACM ICN2022, pp.192–193, Sept. 2022. DOI: 10.1145/3517212.3559482

[2] M. Yoshida, Y. Ito, Y. Sato, and H. Koga, “Performance evaluation of popularity-aware dynamic-clustering scheme for distributed caching in ICN,” Proc. APSIPA ASC2022, pp.185–190, Nov. 2022.

[3] V. Jacobson, D.K. Smetters, J.D. Thornton, M. Plass, N. Briggs, and R. Braynard, “Networking named content,” Commun. ACM, vol.55, no.1, pp.117–124, Jan. 2012. DOI: 10.1145/2063176.2063204

[4] L. Zhang, A. Afanasyev, J. Burke, V. Jacobson, K.C. Claffy, P. Crowley, C. Papadopoulos, L. Wang, and B. Zhang, “Named data networking,” ACM SIGCOMM Comput. Commun. Rev., vol.44, no.3, pp.66–73, July 2014. DOI: 10.1145/2656877.2656887

[5] S. Arshad, M.A. Azam, M.H. Rehmani, and J. Loo, “Recent advances in information-centric networking-based internet of things (ICN-IoT),” IEEE Internet Things J., vol.6, no.2, pp.2128–2158, April 2019. DOI: 10.1109/JIOT.2018.2873343

[6] M. Meddeb, A. Dhraief, A. Belghith, T. Monteil, K. Drira, and S. Al-Ahmadi, “Named data networking: A promising architecture for the internet of things (IoT),” International Journal on Semantic Web and Information Systems, vol.14, no.2, pp.86–112, April 2018. DOI: 10.4018/IJSWIS.2018040105

[7] M. Yoshida, Y. Ito, Y. Sato, and H. Koga, “A cluster-based cache distribution scheme in content-centric-networking,” Proc. ACM ICN2018, pp.196–197, Sept. 2018. DOI: 10.1145/3267955.3269012

[8] V. Sourlas, I. Psaras, L. Saino, and G. Pavlou, “Efficient hash-routing and domain clustering techniques for information-centric networks,” Elsevier Computer Networks, vol.103, pp. 67–83, July 2016. DOI: 10.1016/j.comnet.2016.04.001

[9] N. Laoutaris, H. Che, and I. Stavrakakis, “The LCD interconnection of LRU caches and its analysis,” Elsevier Performance Evaluation, vol.63, no.7, pp.609–634, July 2006. DOI: 10.1016/j.peva.2005.05.003

[10] N. Laoutaris, S. Syntila, and I. Stavrakakis, “Meta algorithms for hierarchical web caches,” Proc. IEEE IPCCC2004, pp.445–452, April 2004. DOI: 10.1109/PCCC.2004.1395054

[11] K. Cho, M. Lee, K. Park, T.T. Kwon, Y. Choi, and S. Pack, “Wave: Popularity-based and collaborative in-network caching for content-oriented networks,” Proc. IEEE INFOCOM2012 Workshops, pp.316–321, May 2012. DOI: 10.1109/INFCOMW.2012.6193512

[12] T. Mick, R. Tourani, and S. Misra, “MuNCC: Multi-hop neighborhood collaborative caching in information centric networks,” Proc. ACM ICN2016, pp.93–101, Sept. 2016. DOI: 10.1145/2984356.2984375

[13] L. Saino, I. Psaras, and G. Pavlou, “Hash-routing schemes for information centric networking,” Proc. ACM ICN2013, pp.27–32, Aug. 2013. DOI: 10.1145/2491224.2491232

[14] J.H. Mun and H. Lim, “Cache sharing using Bloom filters in named data networking,” Journal of Network and Computer Applications, vol.90, pp.74–82, July 2017. DOI: 10.1016/j.jnca.2017.04.011

[15] C. Li and K. Okamura, “Cluster-based in-networking caching for content-centric networking,” International Journal of Computer Science and Network Security, vol.14, no.11, pp.1–9, 2014. http://paper.ijcsns.org/07_book/201411/20141101.pdf

[16] B. Alahmri, S. Al-Ahmadi, and A. Belghith, “Efficient pooling and collaborative cache management for NDN/IoT networks,” IEEE Access, vol.9, pp.43228–43240, March 2021. DOI: 10.1109/ACCESS.2021.3066133

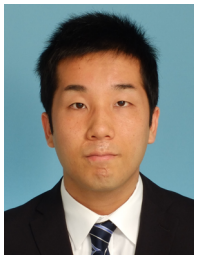
[17] H. Yan, D. Gao, W. Su, C.H. Foh, H. Zhang, and A.V. Vasilakos, “Caching strategy based on hierarchical cluster for named data networking,” IEEE Access, vol.5, pp.8433–8443, March 2017. DOI: 10.1109/ACCESS.2017.2694045

[18] S. Traverso, M. Ahmed, M. Garetto, P. Giaccone, E. Leonardi, and S. Niccolini, “Temporal locality in today’s content caching: Why it matters and how to model it,” SIGCOMM Comput. Commun. Rev., vol.43, no.5, pp.5–12, Oct. 2013. DOI: 10.1145/2541468.2541470

[19] W. Wong, L. Wang, and J. Kangasharju, “Neighborhood search

and admission control in cooperative caching networks," Proc. IEEE GLOBECOM2012, pp.2852–2858, Dec. 2012. DOI: 10.1109/GLOCOM.2012.6503549
[20] S. Nayak, R. Patgiri, and A. Borah, "A survey on the roles of Bloom filter in implementation of the named data networking," Elsevier Computer Networks, vol.196, art. no.108232, Sept. 2021. DOI: 10.1016/j.comnet.2021.108232
[21] G.F. Riley and T.R. Henderson, "The ns-3 network simulator," Modeling and Tools for Network Simulation, K. Wehrle, M. Güneş, and J. Gross, eds., pp.15–34, Springer, Berlin, Heidelberg, 2010. DOI: 10.1007/978-3-642-12331-3_2
[22] A. Ioannou and S. Weber, "A survey of caching policies and forwarding mechanisms in information-centric networking," IEEE Commun. Surveys Tuts., vol.18, no.4, pp.2847–2886, May 2016. DOI: 10.1109/COMST.2016.2565541
[23] M. Hefeeda and O. Saleh, "Traffic modeling and proportional partial caching for peer-to-peer systems," IEEE/ACM Trans. Netw., vol.16, no.6, pp.1447–1460, March 2008. DOI: 10.1109/TNET.2008.918081
[24] S. Knight, H. Nguyen, N. Falkner, R. Bowden, and M. Roughan, "The internet topology zoo," IEEE J. Sel. Areas Commun., vol.29, no.9, pp.1765–1775, Oct. 2011. DOI: 10.1109/JSAC.2011.111002

Mikiya Yoshida received the B.E. degree in Information Science and Electrical Engineering from the Kyushu Sangyo University, Japan, in 2017, and his M.E. degree in Information Engineering from the University of Kitakyushu, Japan, in 2019. He is now a doctoral student at the University of Kitakyushu, Japan. His research interests include network architecture and information-centric networking.



Yusuke Ito received the B.E., M.E., and D.E. degrees in Information and Media Engineering from the University of Kitakyushu, Japan in 2014, 2016, and 2019, respectively. From 2019 to 2022, he was an Assistant Professor at the Tokyo University of Science. Currently, he is a Lecturer in the Department of Information Systems Engineering, Faculty of Environmental Engineering, The University of Kitakyushu, Japan. His research interests include network architecture and edge cloud computing.

Yurino Sato received the B.E., M.E., and D.E. degrees in Information and Media Engineering from the University of Kitakyushu, Japan in 2012, 2014, and 2019, respectively. She has been an assistant professor since April 2018 in the Department of Control Engineering, National Institute of Technology (KOSEN), Sasebo College, Japan. Her research interests include network architecture, transport protocol, and forward error correction.

Hiroyuki Koga received the B.E., M.E., and D.E. degrees in Computer Science and Electronics from the Kyushu Institute of Technology, Japan in 1998, 2000, and 2003, respectively. From 2003 to 2004, he was a postdoctoral researcher in the Graduate School of Information Science, Nara Institute of Science and Technology. From 2004 to 2006, he was a researcher in the Kitakyushu JGN2 Research Center, National Institute of Information and Communications Technology. From 2006 to 2009, he was an assistant professor in the Department of Information and Media Engineering, Faculty of Environmental Engineering, The University of Kitakyushu, and has been an associate professor in the same department since April 2009. His research interests include performance evaluation of computer networks, mobile networks, and communication protocols.

IEICE TRANS. COMMUN., VOL.E107–B, NO.5 MAY 2024

PAPER

Traffic Reduction for Speculative Video Transmission in Cloud Gaming Systems

Takumasa ISHIOKA[†a)], **Tatsuya FUKUI**[††], **Toshihito FUJIWARA**[††], **Satoshi NARIKAWA**[††], ***Nonmembers***, **Takuya FUJIHASHI**[†], **Shunsuke SARUWATARI**[†], ***and*** **Takashi WATANABE**[†], ***Members***

SUMMARY Cloud gaming systems allow users to play games that require high-performance computational capability on their mobile devices at any location. However, playing games through cloud gaming systems increases the Round-Trip Time (RTT) due to increased network delay. To simulate a local gaming experience for cloud users, we must minimize RTTs, which include network delays. The speculative video transmission pre-generates and encodes video frames corresponding to all possible user inputs and sends them to the user before the user's input. The speculative video transmission mitigates the network, whereas a simple solution significantly increases the video traffic. This paper proposes tile-wise delta detection for traffic reduction of speculative video transmission. More specifically, the proposed method determines a reference video frame from the generated video frames and divides the reference video frame into multiple tiles. We calculate the similarity between each tile of the reference video frame and other video frames based on a hash function. Based on calculated similarity, we determine redundant tiles and do not transmit them to reduce traffic volume in minimal processing time without implementing a high compression ratio video compression technique. Evaluations using commercial games showed that the proposed method reduced 40–50% in traffic volume when the SSIM index was around 0.98 in certain genres, compared with the speculative video transmission method. Furthermore, to evaluate the feasibility of the proposed method, we investigated the effectiveness of network delay reduction with existing computational capability and the requirements in the future. As a result, we found that the proposed scheme may mitigate network delay by one to two frames, even with existing computational capability under limited conditions.

key words: *cloud gaming, low latency, speculative video transmission, traffic reduction*

1. Introduction

As networks become more sophisticated, there is a growing interest in cloud gaming services [1]. A cloud gaming service enables high-end games on low-end devices by running games on a cloud server equipped with a GPU to replace the graphics rendering process [2]. High-end games require complex processing with dedicated GPU cards installed in high-performance PCs or game consoles. Using cloud gaming services, users can play high-end games from anywhere with mobile terminals like smartphones.

Cloud gaming systems necessitate interactive operation over the network. Typical video transmission for cloud gaming services is request-and-response. Specifically, a user device sends input to the cloud server, and the server sends back video frames corresponding to the received input. This generated video frame is subsequently compressed using video coding technology and transmitted to the user. The user decodes the received video frame and displays the video frame. However, a significant problem with the cloud gaming system is an increase in the round-trip time (RTT). RTT is the delay from sending the user's input to displaying the corresponding video frame on the user's device. The cloud gaming server must wait for the input to return the video frames, increasing the network delay. The network delay is the sum of the packet delivery time from the user's device to the server and vice-versa, and it is a part of the RTT. Here, the packet delivery time represents the delay when the first bit leaves the sender until the last is received. Any increase in the RTT negatively impacts the quality of the user experience, thus posing a major challenge in gameplay [3].

Many discussions have addressed the issue of RTT on cloud gaming systems. The RTT has a significant effect on the gaming experience [4]. It's a crucial factor in some games, and the demands can change based on the target game's features [5]. Beyond reducing propagation delay, several aspects, like reducing transmission delay through video quality optimization, application optimization, and more, have been explored to cut down on system-wide latency. Table 1 lists major research topics related to cloud gaming response time and traffic issues. Certain studies have used platform techniques, like mobile edge computing, to shrink response time [6]. Numerous studies focused on platform optimization, which includes dynamically adjusting video quality [7], optimizing cloud resources [8], [9], and creating cloud-native games [10], [11]. However, simply reducing latency does not provide an experience identical to playing a game locally. We need to eliminate any network delays to make the user's experience match that of playing a game locally.

Some studies suggest speculative video transmission methods to minimize the RTT. Speculative video transmission is an approach where video data is transmitted efficiently by utilizing input predictions. This approach predicts the next potential input based on the user's past inputs or general input patterns and accordingly pre-renders and transmits the frames. For example, in [12], the cloud server creates in advance and encodes video frames corresponding to all potential user inputs within one frame, then sends them all

Manuscript received June 26, 2023.
Manuscript revised October 12, 2023.
Manuscript publicized January 30, 2024.

[†]The authors are with Graduate School of Information Science and Technology, Osaka University, Suita-shi, 565-0871 Japan.
[††]The authors are with NTT Access Network Service Systems Laboratories, NTT Corporation, Musashino-shi, 180-8585 Japan.
a) E-mail: ishioka.takumasa@ist.osaka-u.ac.jp

DOI: 10.23919/transcom.2023EBP3108

Table 1 Major research topics on response time and traffic challenges in cloud gaming.

Research Topics	Description
Low-latency Network Techniques	Developing and applying high-speed, low-latency network technologies (5G/6G) to mitigate latency and improve the cloud gaming experience.
Edge Computing	Processing games on devices closer to the edge of the network to mitigate response time and traffic issues.
Dynamic Video Quality Adjustment	Dynamically adjusting video quality according to bandwidth constraints to mitigate response time and traffic issues.
Data Compression Techniques	Developing and applying efficient compression techniques for images and audio to reduce data transfer volume, thereby mitigating response time and traffic issues.
Cloud Resource Allocation Optimization	Optimizing cloud resource allocation and developing algorithms and methods to mitigate response time and traffic issues.
Network Protocol Optimization	Developing and applying new network protocols and techniques to mitigate response time and traffic issues.
Cloud-native Game Development	Developing games optimized for the cloud environment to address response and traffic issues and improve the cloud gaming experience.
Latency Mitigation using Deep Reinforcement Learning	Optimizing cloud gaming experience using deep reinforcement learning to mitigate response time issues.
Speculative Execution on Cloud Gaming Servers **(A topic of focus in this paper)**	Pre-rendering game frames corresponding to all potential user input patterns and transmitting the frames in advance to mitigate response time issues, ideally achieving zero latency.

to the user. The speculative video transmission aims to mitigate the network delay. In other words, we aim for the perceived network delay to be zero. The perceived network delay is essentially how a user experiences the network delay. By transmitting video frames speculatively, the proposed scheme makes the user perceive the network delay as zero, even if there's an inherent delay.

A key issue in realizing speculative video transmission is significant video traffic when the number of the user's potential inputs in each frame is large. For example, when the network delay is d [s] and the frame rate is f [fps], the server lists the potential inputs in future $n = \lceil d \cdot f \rceil$ frames. The server then renders the video frames according to the potential inputs and transmits the video frames to the user. Since the video frames corresponding to the listed potential inputs are buffered on the user's device before his/her input, the network delay can be regarded as zero. The server then renders the video frames according to the listed potential inputs and transmits the video frames to the user. Current request-and-response transmissions commonly use differential coding methods such as H.264/Advanced Video Coding (AVC) [13] and H.265/High Efficiency Video Coding (HEVC) [14]. However, each speculative video frame contains the effects of prediction errors and uncertainty in prediction data. It is desirable to mitigate the impact of accumulated prediction errors and the uncertainty of prediction data by processing each frame independently. Therefore, in speculative video transmission, intra-coding, a coding method that processes each frame independently, is a more appropriate approach than the existing differential coding methods. If one simply uses intra-coding alone, it would be challenging to achieve sufficient compression rates to counteract the increase in traffic volume caused by speculative processing. The proposed method aims to reduce traffic volume by focusing on the similarities between speculative frames at the same time slot, utilizing intra-coding as a base.

As a method focusing on the similarities between speculative frames within the same time slot, this paper proposes a tile-wise delta detection approach. In this approach, the cloud server generates the video frames based on the user's potential input and selects one video frame as the reference frame. It then partitions these generated video frames into multiple tiles. Instead of encoding and transmitting all tiles uniformly, the system sends only those that have notable differences from the tiles of the reference frame. Furthermore, this tile-wise delta detection approach utilizes a hash function to estimate the similarity between the tiles of the reference frame and the generated videos. This approach achieves a low computational cost while maintaining certain similarity estimation accuracy.

To evaluate the effectiveness of the proposed method, we adopted several commercial games, including first-person, third-person, and omniscient games, for comparison. From the evaluations, the proposed method reduces the video traffic for speculative video transmission irrespective of the game genre. Depending on the game genres, we achieve high reductions especially. In addition, we discuss the feasibility of the proposed method considering the computational capability of the cloud server.

The contributions of this study are as follows:

1. We design tile-wise delta detection for cloud gaming services to simultaneously achieve network delay mitigation and traffic reduction while keeping video quality.
2. We adopt a hash-based similarity calculation to estimate the similarity between the tiles with a low computational cost.
3. We develop a speculative video transmission system to empirically evaluate the effectiveness and feasibility of the proposed method.
4. We use three genres of commercial games for evaluations, i.e., first-person, third-person, and omniscient

games, to further discuss the effectiveness of the proposed method in practical game videos.
5. We simulated the computational capability required to implement the proposed method and discussed the feasibility.

2. Proposed Method

2.1 Overview

Figure 1 shows an overview architecture of the proposed method. The cloud gaming server generates all potential input patterns for the user according to the number of speculative frames and input patterns per frame. The system renders the video frames according to the potential input patterns and the current game state. It divides each rendered video frame into multiple tiles and selects one video frame as the reference to calculate the similarity between other video frames using a hash function. It calculates the similarity for each tile. Based on the similarity, the tiles with high similarity are not encoded and transmitted to the user. The user uses the same tiles in the reference video frame, whereas the tiles with low similarity are encoded and transmitted to the user. Each user decodes the video frame corresponding to the actual input and displays it on the screen. The cloud gaming server begins the transmission of the video frames, considering the network delay. The network delay appears to be zero from the user's perspective. In addition, each user sequentially sends his/her input to the cloud gaming server to update the current game state.

Figure 2 shows the data flow between the cloud gaming server and the user's device. We consider the number of speculative frames n to be two based on the network delay and the frame rate. At the time instance t_0, the user's device sends an input to the cloud gaming server. The cloud gaming server has already received the input for t_{-1} from the user's device and updates the game state based on the received

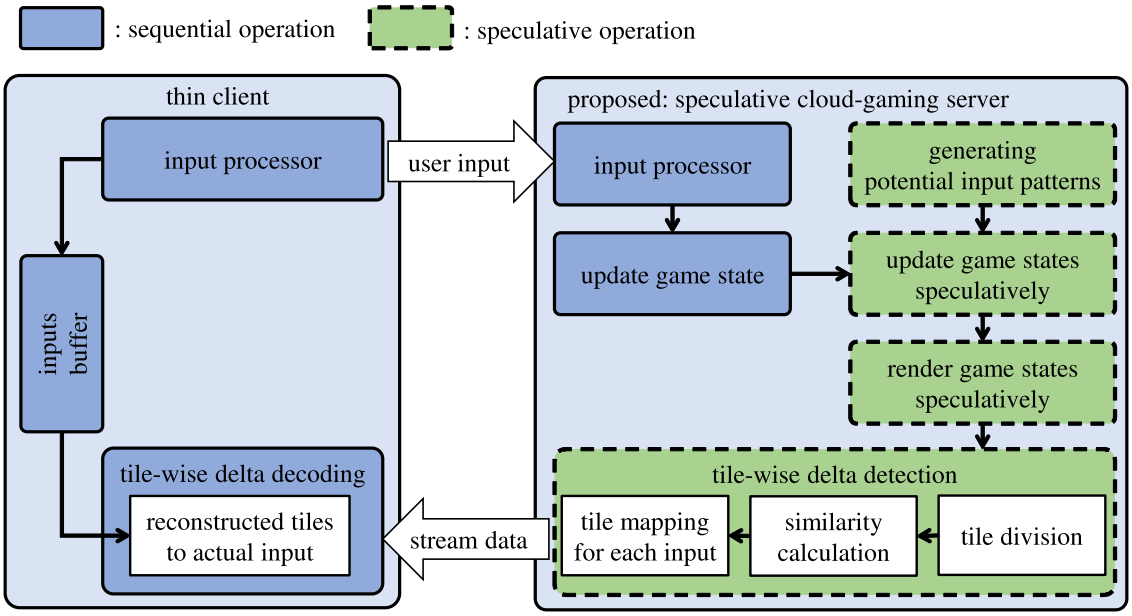


Fig. 1 Proposed cloud gaming system architecture.

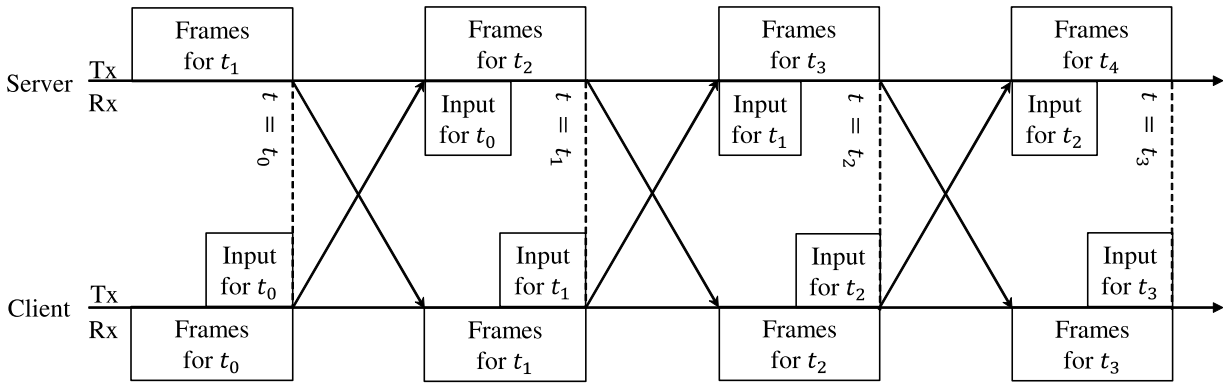


Fig. 2 Data flow between server and client.

input. Let A be the number of the potential input patterns in each frame. In this case, the cloud gaming server generates A^2 potential input patterns and prepares multiple game states for t_1 by speculatively updating the game state for each input. It then renders video frames of the game states for t_1 and sends the video frames to the user's device, followed by the proposed tile-wise delta detection. The user's device displays the video frames with a negligible RTT based on the user's true input at t_1. The above operations are repeated in every frame.

2.2 Tile-Wise Delta Detection

Figure 3 shows the overview of the proposed tile-wise delta detection. The cloud gaming server with speculative video transmission generates video frames for user potential input patterns. The speculative video transmission can mitigate the network delay by pre-sending the generated video frames to the user, whereas it will cause large video traffic. In the proposed method, we detect redundant frames with respect to the reference frame. It then calculates the similarity between the reference video frame and the other tiles. For the similarity calculation, the cloud gaming server partitions the reference video and other video frames into M vertical tiles and N horizontal tiles. Figure 3 presents an example for the case where $M = 4$ and $N = 4$. We identify redundant tiles relative to the reference frame to reduce the video traffic generated by video frames.

Our proposed method employs perceptual hash (pHash) [15] to calculate the similarity based on Discrete Cosine Transform (DCT) between tiles. The pHash is a widely utilized image similarity estimation algorithm. It prioritizes extracting the features in the low-frequency components easily perceived by humans, thereby enabling the extraction of perceptible differences. Despite requiring longer computational time compared to average hash (aHash) and difference hash (dHash), pHash demonstrates robustness against image reduction and blur [16]. Considering that certain games may generate high-quality images or motion blur, we argue that pHash is our method's most appropriate image similarity estimation algorithm. The procedure to compute the pHash value for each tile is as follows.

1. Converts a video frame to a grayscale image with the same luminance as the original color image.
2. Resize the grayscale image to 32×32 [pixels].
3. Perform a discrete cosine transform (DCT) for the resized tiles and extract the DCT coefficients over an 8×8 [pixels] region. It selects 64 low-frequency DCT coefficients.
4. Obtain the median of the extracted DCT coefficients and convert them to binary based on the average value. Finally, we obtain a 64-bit hash value.

The system then uses the pHash values corresponding to each tile in the reference video frame and the hash values of tiles in other video frames to determine similar and dissimilar tiles according to Hamming distance. The maximum possible Hamming distance is 64.

2.3 Number of Speculative Frames Relative to Computational Capability

The speculative video transmission can mitigate the network

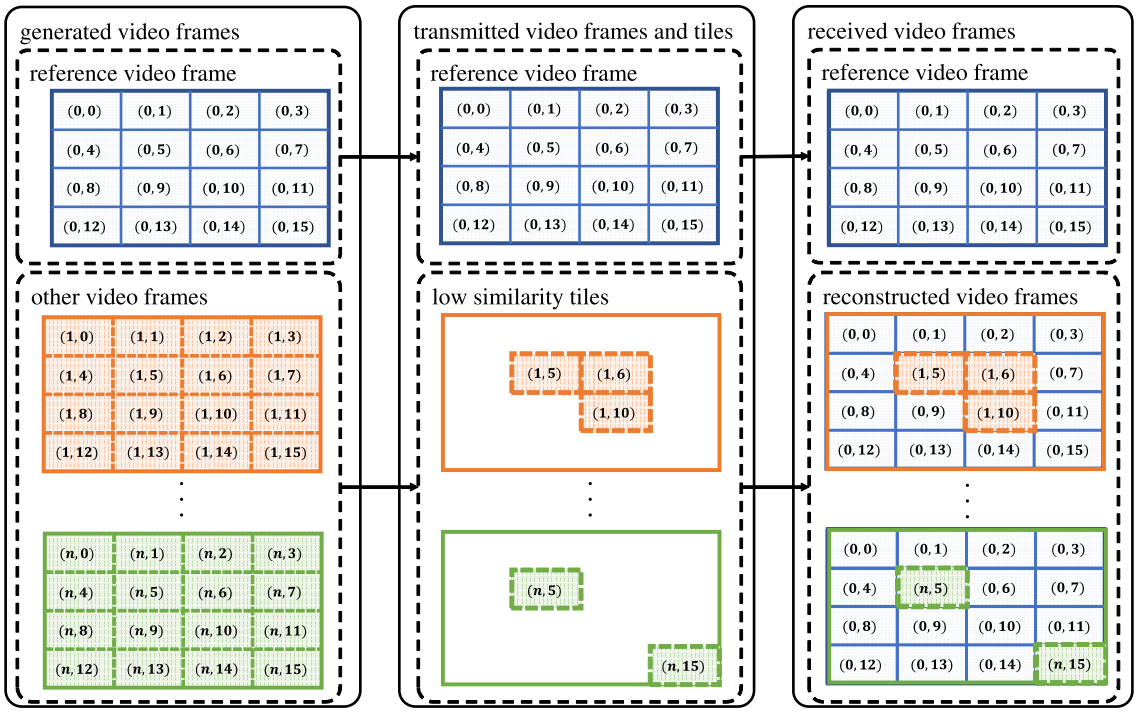


Fig. 3 Overview of the proposed tile-wise delta detection.

delay from the user's perspective by sending the video frames of possible user's future input patterns. In the proposed scheme, the cloud gaming server lists the user's potential inputs in future frames and renders the video frames. The cloud gaming server generates the video frame and performs tile-wise delta detection on a frame-by-frame basis. Accordingly, the latency requirement depends on the frame rate of cloud gaming services. We consider the frame rate of 60 fps; thus, the requirement is 16.6 ms.

Let C_S [Hz] be the total operating frequency of the central processing unit (CPU) in the cloud gaming server, P_S [cycles] be the total number of cycles of speculative game execution processes, and f [fps] be the frame rate. Here, A denotes the number of the potential input patterns in each frame. In this case, the possible number of speculative frames $\hat{n}$ [frames] that can be speculatively processed is subject to the following restrictions:

$$\hat{n} = \left\lfloor \log_A \left(\frac{C_S}{f \cdot P_S} \right) \right\rfloor \tag{1}$$

Let P_G be the number of game execution process cycles per frame, P_S can be determined as follows:

$$P_S = P_G \cdot A^{\hat{n}} \tag{2}$$

Therefore, $\hat{n}$ relative to the total operating frequency of the CPU is as follows:

$$\hat{n} = \left\lfloor \frac{1}{2} \log_A \left(\frac{C_S}{f \cdot P_G} \right) \right\rfloor \tag{3}$$

Here, d [s] and t_p [s] denote the network delay and the perceived network delay. When the server lists the potential inputs for future $\hat{n}$ [frames] within each frame, the perceived network delay t_p is as follows:

$$t_p = \begin{cases} 0 & \text{if } \left(\frac{\hat{n}}{f} \geq d \right), \\ d - \frac{\hat{n}}{f} & \text{else} \end{cases} \tag{4}$$

The proposed method reduces the perceived network delay by rendering a large number of future input patterns frame-by-frame. However, as defined in Eq. (3), significant computation power is required to mitigate the perceived network delay as the value of $\hat{n}$ increases.

3. Rate-Distortion Optimized Speculative Frame Coding

In this section, we introduce the notion of transition probability $P(F_k)$ to each frame at time t. F_k denotes the k-th frame in the group of frames ($\mathcal{F}_t$) to be speculatively processed at time t. The frames the user requests are independent and therefore $\sum_k P(F_k) = 1$. The transition probabilities to each frame are assumed to be obtainable in advance. We can determine transition probabilities using time series analysis of user operation logs or machine learning-based analysis. This idea draws influence from the discussion in [17] on improving input prediction based on the assumption that the input continues seamlessly from the immediately preceding input. Note that, depending on the game genre and the input method of the operations, determining the exact transition probabilities might not always be practical. However, this limitation does not restrict the generality of our approach, as individuals can modify the probabilistic model without affecting the proposed system.

When the input pattern per frame A is speculatively processed for $\hat{n}$ frames, $\mathcal{F}_t$ is shown as follow:

$$\mathcal{F}_t = \{F_k | k \leq A^{\hat{n}}\} \tag{5}$$

Here, we assume that the server has determined the transition probabilities $P(F_k|\mathcal{F}_t)$ for all possible transition frames. Allocating more bits to frames more likely to be displayed by the user improves the user experience.

The rate allocation algorithm is implemented to minimize the distortion expected at the decoder, according to the transition probability $P(F_k|\mathcal{F}_t)$. Assuming $r(k)$ bits are assigned to k-th frames, the distortion is shown as follows:

$$D = \sum_{k=1}^{A^{\hat{n}}} D(r(k)) P(F_k|\mathcal{F}) \tag{6}$$

In the proposed method, the difference from the reference frame is determined for each tile of M×N and transmits only those with a difference. τ represents the hash threshold, $r(i,j,k)$ indicates the bitrate of (i,j)-th tile in k-th frame, and $x(i,j,k)$ is a binary variable that determines whether the tile is transmitted or not. Specifically, the proposed scheme assigns $x(i,j,k) = 0$ when the Hamming distance of the tile between the reference video frame and another video frame is below the hash threshold τ and vice-versa. It means that setting a lower hash threshold increases the number of transmission tiles. In this case, the distortion is defined as:

$$\begin{aligned} D &= \sum_{k=1}^{A^{\hat{n}}} \sum_{i=1}^{\mathrm{M}} \sum_{j=1}^{\mathrm{N}} D(r(i,j,k)) P(F_k|\mathcal{F}) x(i,j,k) \\ &s.t. \quad \sum_{i=1}^{\mathrm{M}} \sum_{j=1}^{\mathrm{N}} \sum_{k=1}^{A^{\hat{n}}} r(i,j,k) x(i,j,k) \leq R_{limit} \\ &s.t. \quad x(i,j,k) = \begin{cases} 1 & \text{(if send)} \\ 0 & \text{(else)} \end{cases} \end{aligned} \tag{7}$$

Here, $r(i,j,k)x(i,j,k)$ denotes the rate distribution limited by the given rate R_{limit}, $D(r(i,j,k))$ denotes the distortion for each tile encoded with $\mathbf{r}(i,j,k)$ bits. Since $P(F_k|\mathcal{F})$ does not depend on r, the rate distribution optimisation problem is solved with Lagrange multipliers $\lambda > 0$, the cost J is as follow:

$$\begin{aligned} J = \min_{\mathbf{r},\mathbf{x}} \Bigg\{ &\sum_{k=1}^{A^{\hat{n}}} \sum_{i=1}^{\mathrm{M}} \sum_{j=1}^{\mathrm{N}} D(r(i,j,k)) P(F_k|\mathcal{F}) x(i,j,k) \\ &+ \lambda ||\mathbf{rx}||_1 \Bigg\} \end{aligned} \tag{8}$$

By solving the optimization problem, the proposed scheme

can find the optimal hash threshold τ and bit assignment for each tile $r(i, j, k)$ to minimize the distortion under the given rate R_{limit}.

4. Evaluation

4.1 Traffic Reduction in Commercial Games

We measured the video traffic and the corresponding structural similarity (SSIM) index [18] of the speculative video transmission with tile-wise delta detection. Table 2 presents the environments of this evaluation. SSIM index is a better objective metric for predicting the perceptual similarity between original and processed video frames. Larger values of the SSIM index close to 1 indicate higher perceptual similarity between these frames.

In this evaluation, we classify commercial games into first-person, third-person, and omniscient. First-person games are displayed from the viewpoint of the user-controlled character. First-person shooting (FPS) is a typical first-person game. The video frame of an FPS game is characterized by the display of the user interface (UI) and weapons, such as guns and knives held by the characters at certain coordinates. The objects contained within the viewpoint change rapidly in response to the user's inputs.

Third-person games are displayed from the rear view of the user's character. A typical third-person game is an action game. The video frame of an action game has the characteristic of displaying the user's character at certain coordinates. As in FPS games, the objects contained within the viewpoint change rapidly in response to the user's inputs.

Omniscient games are displayed so that the user overlooks the controlling character. A typical omniscient game is a fighting game. The video frame of a fighting game displays a fixed viewpoint such that the user's character and the entire field of play are contained. The user-controlled character moves within a fixed viewpoint in response to the user's inputs.

We used the following commercial games for comparison: Valorant [19], Overwatch2 (OW2) [20], and Borderlands3 [21] as first-person games; Genshin Impact [22], Monster Hunter World (MHW) [23], and ELDENRING [24] as third-person games; and Street Fighter V (SFV) [25], Hearthstone [26], and Cuphead [27] as omniscient games. To facilitate comparison, we replicated a game state for each game and obtained video frames corresponding to various inputs from that state. Specifically, we recorded 10 video frame patterns for each game, from which one frame was chosen randomly to serve as the reference frame. All frames, also divided tiles, were processed without compression. Accordingly, in Eq. (8), $P(F_k|\mathcal{F})$ remains constant regardless of the values of k, and $D(\mathbf{r}(i, j, k))$ equals zero.

Table 3 shows each commercial game's SSIM index and traffic reduction ratio on the proposed system. In this evaluation, the proposed method divided each video frame into 3×3 tiles, and the Hamming distance threshold is $\tau = 0$, i.e., only tiles with perfect matches are not transmitted. The SSIM index was measured with reference to the original video frame. As a result, all omniscient games had a reduction efficiency of 40–50% with an SSIM index of approximately 0.98. Since the region in which the user input effect tends to be limited, the tiles with zero Hamming distances are more likely to appear. Therefore, high reduction efficiency can be achieved. On the other hand, some first-person and third-person games had video quality degradation and low reduction ratio compared to omniscient games. In first-person and third-person games, the entire video frame may change owing to user input. Therefore, the reduction efficiency is reduced when the Hamming distance threshold is set to $\tau = 0$. Optimizing the Hamming distance threshold and the number of tile divisions may improve reduction efficiency while limiting quality degradation.

Figures 4(a) through 4(f) show the snapshots of the games. We select Valorant, Genshin Impact, and SFV as the first-person, third-person, and omniscient games. Figures 4(a) through 4(c) show the original video frames and Figs. 4(d) through 4(f) show the proposed video frame. Here, we divided each uncompressed frame into 3×3 tiles and set τ=0 for the first-person, third-person, and omniscient games, respectively. From the snapshots, the proposed method can reduce video traffic irrespective of the game types. In addition, quality degradation due to the tile-wise delta detection does not significantly impact the visual quality.

Finally, Figs. 5(a) through 5(c) show the reduction ratio for the SSIM index of the first-person, third-person, and omniscient games, respectively. In this evaluation, the proposed method divided each video frame into 2×2 to 5×5 tile divisions and varied the Hamming distance threshold τ from 0 to 64. As the value of τ increases, the number of tiles regarded as similar also increases, leading to higher video quality and larger SSIM indexes. Here, no tile division means that the similarity calculation is performed on

Table 2 PC specs used for evaluation.

	configuration
OS	Ubuntu Desktop 20.04.2.0 LTS
Motherboard	Z490 1200 DDR4 ATX
CPU	Intel Core i9-10850K
RAM-size	128 GB
GPU	NVIDIA GeForce RTX 3080

Table 3 The SSIM index and traffic reduction ratio in each commercial game. The proposed method divided each video frame into 3×3 tiles and set τ=0 for tile-wise delta detection.

Types	first-person game			third-person game			omniscient game		
Titles	Valorant	OW2	Borderlands3	Genshin Impact	MHW	ELDENRING	SFV	Hearthstone	Cuphead
SSIM	0.9848	0.9815	0.9609	0.9802	0.9682	0.9533	0.9755	0.9863	0.9782
Reduction ratio [%]	35.29	4.17	23.18	17.24	7.99	24.97	52.10	54.14	44.00

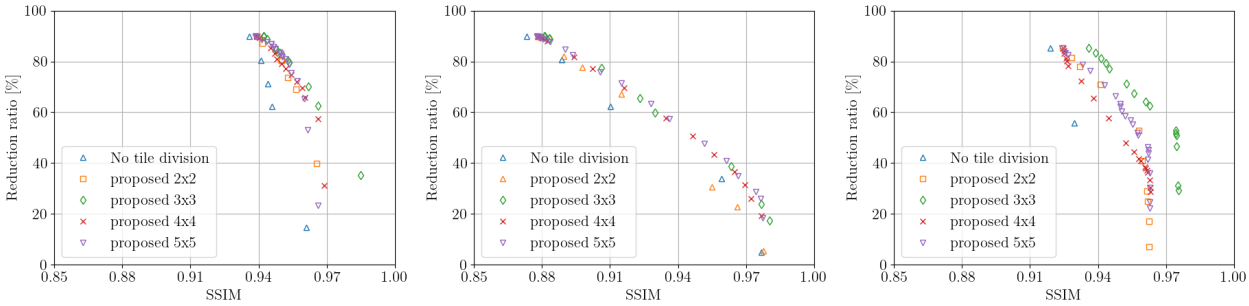
(a) Original (Valorant) (b) Original (Genshin Impact) (c) Original (SFV)

(d) Proposed (Valorant)
SSIM index is 0.9848
Reduction ratio is 35.29 %

(e) Proposed (Genshin Impact)
SSIM index is 0.9767
Reduction ratio is 23.77 %

(f) Proposed (SFV)
SSIM index is 0.9741
Reduction ratio is 52.95 %

Fig. 4 Snapshots of the games. (a) and (d) show the first-person game's video frames. (b) and (e) shows the third-person game's video frames. (c) and (f) shows the omniscient game's video frames.



(a) first-person game (Valorant) (b) third-person game (Genshin Impact) (c) omniscient game (SFV)

Fig. 5 Traffics relative to SSIM index. To evaluate traffic based on the SSIM index, we measured the traffic and average SSIM index for each tiling by varying the Hamming distance threshold τ value from 0 to 64 for the original video frame.

the whole frame without tiling it.

We can see the following results:

- Raising the threshold τ value enhances the traffic reduction ratio across all games by classifying more tiles as similar. Nevertheless, this also leads to a significant decrease in the SSIM index.
- Figures 5(a) to (c) show that performing similarity estimation on each tile after division performs better compared to the scenario where similarity estimation is performed without division. All games produced the highest quality video, especially when split into 3×3 tiles.
- Figure 5(c) indicates that optimizing the number of tiles increases the reduction ratios. For instance, with an SSIM index of 0.9628, the reduction ratio registers 62.55% for 3×3 tile division and 36.14% for 4×4 tile division.
- Figure 5(c) shows that tile-wise delta detection works well in omniscient games. In such games, the background tiles are static, irrespective of the user's inputs, and this can lead to a large reduction in traffic.
- Figure 5(a) indicates the potential effectiveness in other genres, depending on each game's characteristics. In other words, the proposed method works well when the game field has minimal transitions for each input, such as when the background is monotonous.

Considering the rate-distortion optimization discussed in Sect. 3, the results from Figs. 5(a) to 5(c) suggest that regardless of the number of tile divisions, setting a smaller τ increases the number of transmitted tiles, i.e., there is a higher proportion of instances where $x = 1$. In such cases, it becomes necessary to reduce the quality of each tile through rate-distortion optimization, thereby decreasing the value of r. Conversely, setting larger τ decreases the number of transmitted tiles, i.e., there is a higher proportion of instances where $x = 0$. The value of r can be increased

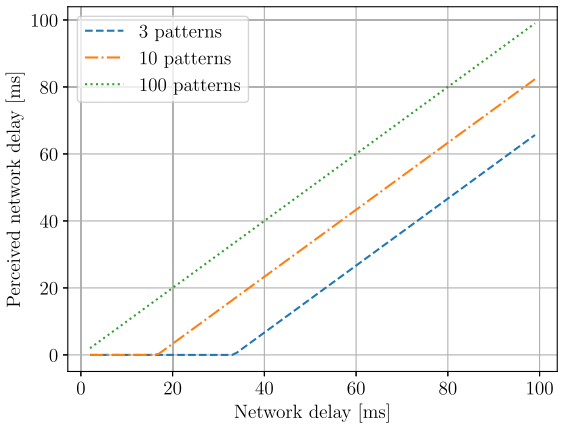


Fig. 6 The perceived network delay as a function of network delays with the different number of potential input patterns in each frame.

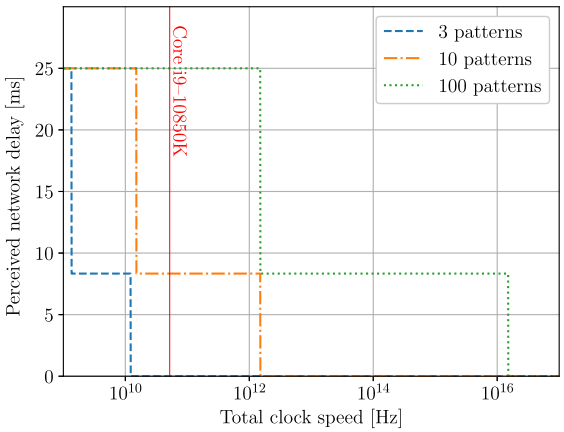


Fig. 7 The perceived network delay as a function of the total clock speed, where the network delay is assumed to be 25 ms.

through rate-distortion optimization.

4.2 Feasibility

In this section, we carry out experiments to discuss the perceived network delay under the different computation resources and the number of potential input patterns in each frame. Table 2 shows the experimental setup. Intel Core i9-10850K has ten unlocked cores and hyper-threading, and each core can turbo up to an operating frequency of 5.2×10^9 Hz. Here, we consider the average number of game execution process cycles per frame P_G to be 2.5×10^6 cycles based on the measurement during running SFV.

Figure 6 shows the perceived network delay as a function of network delays with the different number of potential input patterns in each frame. We assume the same computation capability as the Intel Core i9-10850K CPU used in our implementation. From the evaluation results, the cloud gaming server can decrease the number of video frames needed to be rendered in a short network delay environment and a limited number of potential inputs. In this case, the cloud gaming server can complete the required operations in every frame, and the perceived network delay becomes zero. In a long network delay and a large number of the user's potential inputs, the cloud gaming server does not complete the required operations within one frame, and thus, the perceived network delay becomes longer. However, the proposed scheme can reduce the perceived network delay because the cloud gaming server sends the rendered video frames once the required operations are finished. A lower perceived network delay, i.e., RTT, also contributes to the improvement of the cloud gaming experience.

Figure 7 shows the perceived network delay as a function of the total clock speed, where the network delay is assumed to be 25 ms [28]. When the number of potential input patterns is three, the proposed scheme can eliminate the network delay using an off-the-shelf CPU, e.g., Intel Core i9-10850K. On the other hand, the proposed scheme needs 100 and 10000 times the computational capabilities of the current experimental setup to achieve zero perceived network delay when the number of potential input patterns in each frame is 10 and 100, respectively.

5. Related Works

This research is related to studies on speculative video transmission, network delay reduction in cloud gaming, and differential coding.

5.1 Network Delay Reduction in Cloud Gaming

In cloud gaming systems, the network delay will be a critical issue since a long network delay damages the user's experience depending on the characteristics of the target game [29]. Some studies aim to reduce network delay to enhance user experience quality in cloud gaming systems. We can classify the existing methods into individual delay reduction and zero-delay solutions. For individual delay reduction, Suzujevic [30] proposed an adaptive video coding method to reduce transmission and processing delays. Specifically, the cloud gaming server degrades game video frames and their frame rate to reduce the video traffic and corresponding transmission delay. In [31], they aim to reduce propagation delay by decreasing the distance between the cloud gaming server and the user terminal. Specifically, each user connects to a physically nearby cloud gaming server and exchanges packets, thus achieving an experienced quality closer to a local gaming system. Zhang [6] utilized edge networks to reduce network delay and bandwidth consumption. They form edge networks to a data center for cloud gaming and perform computationally demanding operations, such as video rendering, on the edge networks. Other studies [10], [11] adjust the game system to reduce the effect of a long network delay on the quality of experience.

For zero-delay solutions, speculative video transmission has been designed in recent years. Outatime [32] is proposed to eliminate the network delay in cloud gaming systems by utilizing speculative execution. Outatime transmits speculatively generated multiple video frames as early as the network delay of the server-user network. Each user outputs

a game video frame from the received multiple video frames by applying the appropriate image synthesis technique according to the user's input. In contrast, the user device requires a high computation cost for running the sophisticated image synthesis technique. CloudHide [12] is another method for eliminating network delay in cloud gaming systems through speculative execution. CloudHide transmits all speculatively generated video frames to the user in advance for network delay reduction, whereas it significantly increases video traffic. In existing methods, while issues related to video traffic are mentioned, specific solutions are not provided.

Our study is one of the zero-delay solutions. We aim to eliminate the network delay by transmitting all speculatively generated game video frames to the user in advance while reducing traffic. To reduce video traffic without an additional computation cost, we propose hash function-based tile-wise delta detection. Specifically, it detects redundant tiles between game video frames using a hash function, i.e., low computation cost, and skips encoding and transmission of the redundant tiles for traffic reduction.

5.2 Differential Coding

Video coding techniques such as H.264/AVC and H.265/HEVC are used to encode video frames in cloud gaming. In H.264/AVC and H.265/HEVC, traffic reduction is achieved by encoding and transmitting the difference between the current frames and the previously encoded frame. For cloud gaming, traffic reduction methods based on users' perspectives have been proposed to prevent the degradation of the user's perceptual quality. Sabat [33] determines the video quality of the game video frames based on gazing points at objects in the game. Hegazy et al. [34] improve the perceptual quality by maximizing the video quality of the regions of interest within each video frame. Illahi [35] proposed a foveated video encoding (FVE) to determine the quality within each game video frame based on the user's gaze information. FVE can reduce traffic by encoding the periphery of the field of view at a lower resolution by taking advantage of the property that the resolution becomes lower as one moves outward from the field of view.

Other than cloud gaming, there have been studies on traffic reduction in immersive video applications. In [36], they proposed adaptive tile-based virtual reality (VR) video transmission. The tile-based video transmission schemes [37]–[40] divide the entire video frame into multiple tiles. There are two advantages of tile-based schemes. The first advantage is to realize video coding parallelization to decrease coding delay. For example, in [38], they realize parallel video coding in H.265/HEVC considering adaptive tile patterns and decrease the coding delay by approximately 20–40%. The second advantage is fine-grained quality control. Given that many areas in VR video are outside the user's viewport, the video quality can be improved by transmitting only the viewport. Another study in [41] aims for a traffic reduction for multi-view video. Multi-view video coding requires efficient encoding to transmit multiple viewpoint videos over band-limited networks. For this purpose, they utilize disparity prediction and compensation to remove the inter-view redundancy between the viewpoints.

The proposed tile-wise delta detection is designed for coding multiple game video frames at the same time instant. The tile-wise delta detection is a similar way to the disparity prediction, whereas the computational complexity of the tile-wise delta detection is low because of a hash function-based similarity prediction. Although the similarity prediction is a simple way, the proposed method can yield traffic reduction in commercial games from evaluation results.

6. Conclusion

In this paper, we propose a traffic reduction method for speculative video transmission in cloud gaming systems to mitigate the network delay. The concept of the proposed method can be applied to all genres of commercial games. Evaluations on Valorant, Genshin Impact, and SFV showed that the proposed method reduced the traffic by approximately 35%, 24%, and 53%, respectively, without video compression techniques when the average SSIM index was approximately 0.98. In addition, we discussed the feasibility of the proposed method based on the experimental environment. The proposed speculative execution for two frames may be possible when there are three input patterns in each frame.

In the future, we will extend the proposed scheme to accommodate multiple users. In this case, the cloud gaming server lists future inputs for each user and renders the video frames for each user. A key issue is reducing the traffic increment with an increase in the number of users. A potential solution is to employ a tile-wise delta detection for the video frames across the users to remove redundant information. How to efficiently remove the redundant information to accommodate multiple users is left as the future work.

Acknowledgments

This work was supported by JSPS KAKENHI Grant Numbers JP19H01101, JP22H03582, and NTT Access Network Service Systems Laboratories, Japan.

References

[1] W. Cai, R. Shea, C.Y. Huang, K.T. Chen, J. Liu, V.C.M. Leung, and C.H. Hsu, "A survey on cloud gaming: Future of computer games," IEEE Access, vol.4, pp.7605–7620, Aug. 2016.

[2] C.Y. Huang, K.T. Chen, D. Chen, H.J. Hsu, and C. Hsu, "GamingAnywhere: The first open source cloud gaming system," ACM Trans. Multimedia Comput. Commun. Appl., vol.10, no.1s, pp.1–25, Jan. 2014.

[3] Y.T. Lee, K.T. Chen, H.I. Su, and C.L. Lei, "Are all games equally cloud-gaming-friendly? An electromyographic approach," 2012 11th Annual Workshop on Network and Systems Support for Games (NetGames), pp.1–6, Nov. 2012.

[4] S.F. Lindström, M. Wetterberg, and N. Carlsson, "Cloud gaming: A QoE study of fast-paced single-player and multiplayer gaming," 2020 IEEE/ACM 13th International Conference on Utility and Cloud

Computing (UCC), pp.34–45, Dec. 2020.
[5] S.S. Sabet, S. Schmidt, S. Zadtootaghaj, C. Griwodz, and S. Moller, "Towards the impact of gamers strategy and user inputs on the delay sensitivity of cloud games," 2020 Twelfth International Conference on Quality of Multimedia Experience (QoMEX), pp.1–3, May 2020.
[6] X. Zhang, H. Chen, Y. Zhao, Z. Ma, Y. Xu, H. Huang, H. Yin, and D.O. Wu, "Improving cloud gaming experience through mobile edge computing," IEEE Wireless Commun., vol.26, no.4, pp.178–183, April 2019.
[7] A. Alhilal, T. Braud, B. Han, and P. Hui, "Nebula: Reliable low-latency video transmission for mobile cloud gaming," Proc. ACM Web Conference 2022, WWW'22, pp.3407–3417, April 2022.
[8] I. Slivar, L. Skorin-Kapov, and M. Suznjevic, "QoE-Aware resource allocation for multiple cloud gaming users sharing a bottleneck link," 2019 22nd Conference on Innovation in Clouds, Internet and Networks and Workshops (ICIN), pp.118–123, Feb. 2019.
[9] L. De Giovanni, D. Gadia, P. Giaccone, D. Maggiorini, C.E. Palazzi, L.A. Ripamonti, and G. Sviridov, "Revamping cloud gaming with distributed engines," IEEE Internet Computing, vol.26, no.6, pp.88–95, May 2022.
[10] S.S. Sabet, S. Schmidt, S. Zadtootaghaj, B. Naderi, C. Griwodz, and S. Möller, "A latency compensation technique based on game characteristics to mitigate the influence of delay on cloud gaming quality of experience," Proc. 11th ACM Multimedia Systems Conference, pp.15–25, May 2020.
[11] R. Salay and M.L. Claypool, "A comparison of automatic versus manual world alteration for network game latency compensation," Extended Abstracts of the 2020 Annual Symposium on Computer-Human Interaction in Play, pp.355–359, Nov. 2020.
[12] B. Anand and P. Wenren, "CloudHide: Towards latency hiding techniques for thin-client cloud gaming," Proc. Thematic Workshops of ACM Multimedia 2017, Thematic Workshops'17, pp.144–152, Oct. 2017.
[13] T. Wiegand, G.J. Sullivan, G. Bjontegaard, and A.K. Luthra, "Overview of the H.264/AVC video coding standard," IEEE Trans. Circuits Syst. Video Technol., vol.13, no.7, pp.560–576, Aug. 2003.
[14] J.R. Ohm, G.J. Sullivan, H. Schwarz, T.K. Tan, and T. Wiegand, "Comparison of the coding efficiency of video coding standards — Including high efficiency video coding (HEVC)," IEEE Trans. Circuits Syst. Video Technol., vol.22, no.12, pp.1669–1684, Oct. 2012.
[15] C. Zauner, "Implementation and benchmarking of perceptual image hash functions," 2010.
[16] W. Hua, M. Hou, Y. Qiao, X. Zhao, S. Xu, and S. Li, "Similarity index based approach for identifying similar grotto statues to support virtual restoration," Remote Sensing, vol.13, no.6, p.1201, 2021.
[17] A. Ehlert, "Improving input prediction in online fighting games," Master's thesis, 2021.
[18] Z. Wang, A.C. Bovik, H.R. Sheikh, and E.P. Simoncelli, "Image quality assessment: From error visibility to structural similarity," IEEE Trans. Image Process., vol.13, no.4, pp.600–612, April 2004.
[19] "VALORANT official webcite," URL: https://playvalorant.com/ [Accessed 28 June 2022].
[20] "Overwatch official webcite," URL: https://playoverwatch.com/ [Accessed 28 June 2022].
[21] "Borderlands official webcite," URL: https://borderlands.com/ [Accessed 28 June 2022].
[22] "GenshinImpact official webcite," URL: https://genshin.hoyoverse.com/ [Accessed 28 June 2022].
[23] "MONSTER HUNTER WORLD official webcite," URL: https://www.monsterhunterworld.com/ [Accessed 28 June 2022].
[24] "ELDENRING official webcite," URL: https://www.eldenring.com/ [Accessed 28 June 2022].
[25] "STREET FIGHTER V CHAMPION EDITION official webcite," URL: https://www.capcom.co.jp/sfv/ [Accessed 28 June 2022].
[26] "Hearthstone official webcite." URL: https://playhearthstone.com/ [Accessed 28 June 2022].
[27] "Cuphead official webcite," URL: https://cupheadgame.com/ [Accessed 28 June 2022].
[28] M. Carrascosa and B. Bellalta, "Cloud-gaming: Analysis of google stadia traffic," Computer Communications, vol.188, pp.99–116, March 2022.
[29] R. Shea, J. Liu, E.C.H. Ngai, and Y. Cui, "Cloud gaming: Architecture and performance," IEEE Netw., vol.27, no.4, pp.16–21, Aug. 2013.
[30] M. Suznjevic, I. Slivar, and L. Skorin-Kapov, "Analysis and QoE evaluation of cloud gaming service adaptation under different network conditions: The case of NVIDIA geforce NOW," 2016 Eighth International Conference on Quality of Multimedia Experience (QoMEX), pp.1–6, June 2016.
[31] S. Choy, B. Wong, G. Simon, and C. Rosenberg, "The brewing storm in cloud gaming: A measurement study on cloud to end-user latency," 2012 11th Annual Workshop on Network and Systems Support for Games (NetGames), pp.1–6, Nov. 2012.
[32] K. Lee, D. Chu, E. Cuervo, J. Kopf, S. Grizan, A. Wolman, and J. Flinn, "Outatime: Using speculation to enable low-latency continuous interaction for mobile cloud gaming," MobiSys 2015 - Proc. 13th Annual International Conference on Mobile Systems, Applications, and Services, pp.151–165, June 2015.
[33] S.S. Sabet, M.R. Hashemi, S. Shirmohammadi, and M. Ghanbari, "A novel objective quality assessment method for perceptually-coded cloud gaming video," 2018 IEEE Conference on Multimedia Information Processing and Retrieval (MIPR), pp.75–79, April 2018.
[34] M. Hegazy, K.M. Diab, M. Saeedi, B. Ivanovic, I. Amer, Y. Liu, G. Sines, and M. Hefeeda, "Content-aware video encoding for cloud gaming," Proc. 10th ACM Multimedia Systems Conference, MMSys'19, pp.60–73, June 2019.
[35] G.K. Illahi, T.V. Gemert, M. Siekkinen, E. Masala, A. Oulasvirta, and A. Ylä-Jääski, "Cloud gaming with foveated video encoding," ACM Trans. Multimedia Computing, Communications, and Applications, vol.16, no.1, pp.1–24, Feb. 2020.
[36] J.V. der Hooft, M.T. Vega, S. Petrangeli, T. Wauters, and F. de Turck, "Tile-based adaptive streaming for virtual reality video," ACM Trans. Multimedia Comput. Commun. Appl., vol.15, no.4, pp.1–24, Dec. 2019.
[37] C.C. Chi, M. Alvarez-Mesa, B. Juurlink, G. Clare, F. Henry, S. Pateux, and T. Schierl, "Parallel scalability and efficiency of hevc parallelization approaches," IEEE Trans. Circuits Syst. Video Technol., vol.22, no.12, pp.1827–1838, Dec. 2012.
[38] I. Storch, D. Palomino, B. Zatt, and L. Agostini, "Speedup evaluation of HEVC parallel video coding using tiles," J. Real-Time Image Proc., vol.17, no.5, pp.1469–1486, Oct. 2020.
[39] T. Amestoy, W. Hamidouche, C. Bergeron, and D. Menard, "Quality-driven dynamic VVC frame partitioning for efficient parallel processing," 2020 IEEE International Conference on Image Processing (ICIP), pp.3129–3133, Oct. 2020.
[40] M. Saldanha, G. Sanchez, C. Marcon, and L. Agostini, "Tile adaptation for workload balancing of 3D-HEVC encoder in homogeneous multicore systems," IEEE Trans. Circuits Syst. I, Reg. Papers, vol.67, no.5, pp.1704–1714, March 2020.
[41] T. Li, L. Yu, H. Wang, and Z. Kuang, "A bit allocation method based on inter-view dependency and spatio-temporal correlation for multi-view texture video coding," IEEE Trans. Broadcast., vol.67, no.1, pp.159–173, March 2021.

Takumasa Ishioka received the B.E. and M.E. degrees from Osaka University, Japan, in 2019 and 2021. He is currently an assistant professor at the Faculty of Engineering, Kyoto Tachibana University, Japan, since April 2023. He is a member of IEEE and IPSJ. His research interests include wireless networks.

Tatsuya Fukui received the B.E. and M.E. degrees from Waseda University, Faculty of Science and Engineering, Japan in 2008 and 2010. He is now working for NTT Access Network Service Systems Laboratories. His current research interests include research and development of carrier networks such as wide-area Ethernet systems.

Toshihito Fujiwara received the B.E., M.E., and Ph.D. degrees in engineering from the University of Tsukuba, Ibaraki, Japan, in 2002, 2004, and 2011, respectively. In 2004, he joined the NTT Access Network Service Systems Laboratories, Japan, where he has been involved in the research and development of optical video transmission system, passive optical network system, content delivery network system and ultralow latency video system.

Satoshi Narikawa received the B.E., M.E. and Ph.D. degrees from Tokyo Institute of Technology, Department of Electrical and Electronics Engineering, Japan in 2001, 2003 and 2012, respectively. He is now working for NTT Access Network Service Systems Laboratories. His current research interests include research and development of optical access systems.

Takuya Fujihashi received the B.E. degree in 2012 and the M.S. degree in 2013 from Shizuoka University, Japan. In 2016, he received Ph.D. degree from the Graduate School of Information Science and Technology, Osaka University, Japan. He is currently an assistant professor at the Graduate School of Information Science and Technology, Osaka University since April, 2019. He was research fellow (PD) of Japan Society for the Promotion of Science in 2016. From 2014 to 2016, he was research fellow (DC1) of Japan Society for the Promotion of Science. From 2014 to 2015, he was an intern at Mitsubishi Electric Research Labs. (MERL) working with the Electronics and Communications group. His research interests are in the area of video compression and communications, with a focus on immersive video coding and streaming.

Shunsuke Saruwatari received the Dr. Sci. degree from the University of Tokyo in 2007. From 2007 to 2008, he was a visiting researcher at the IllinoisGenetic Algorithm Laboratory, University of Illinois at Urbana-Champaign. From 2008 to 2012, he was a research associate at the RCAST, the University of Tokyo. From 2012 to 2016, he was an assistant professor at Shizuoka University. He has been an associate professor at Osaka University Since 2016. His research interests are in the areas of wireless networks, sensor networks, and system software.

Takashi Watanabe is a Professor of Graduate School of Information Science and Technology, Osaka University, Japan. He received his B.E., M.E. and Ph.D. degrees from Osaka University, Japan, in 1982, 1984 and 1987, respectively. He joined Faculty of Engineering, Tokushima University as an assistant professor in 1987 and moved to Faculty of Engineering, Shizuoka University in 1990. He was a visiting researcher at University of California, Irvine from 1995 through 1996. He has served on many program committees for networking conferences, IEEE, ACM, IPSJ, IEICE (The Institute of Electronics, Information and Communication Engineers, Japan). His research interests include mobile networking, ad hoc networks, sensornetworks, ubiquitous networks, intelligent transport systems, specially MAC and routing. He is a member of IEEE, IEEE Communications Society, IEEE Computer Society as well as IPSJ and IEICE.

PAPER

Estimation of Drone Payloads Using Millimeter-Wave Fast-Chirp-Modulation MIMO Radar

Kenshi OGAWA[†a)], ***Student Member***, **Masashi KUROSAKI**[†], ***Nonmember***, ***and*** **Ryohei NAKAMURA**[†b)], ***Member***

SUMMARY With the development of drone technology, concerns have arisen about the possibility of drones being equipped with threat payloads for terrorism and other crimes. A drone detection system that can detect drones carrying payloads is needed. A drone's propeller rotation frequency increases with payload weight. Therefore, a method for estimating propeller rotation frequency will effectively detect the presence or absence of a payload and its weight. In this paper, we propose a method for classifying the payload weight of a drone by estimating its propeller rotation frequency from radar images obtained using a millimeter-wave fast-chirp-modulation multiple-input and multiple-output (MIMO) radar. For each drone model, the proposed method requires a pre-prepared reference dataset that establishes the relationships between the payload weight and propeller rotation frequency. Two experimental measurement cases were conducted to investigate the effectiveness of our proposal. In case 1, we assessed four drones (DJI Matrice 600, DJI Phantom 3, DJI Mavic Pro, and DJI Mavic Mini) to determine whether the propeller rotation frequency of any drone could be correctly estimated. In case 2, experiments were conducted on a hovering Phantom 3 drone with several payloads in a stable position for calculating the accuracy of the payload weight classification. The experimental results indicated that the proposed method could estimate the propeller rotation frequency of any drone and classify payloads in a 250 g step with high accuracy.

key words: *millimeter-wave MIMO radar, fast chirp, radar imaging, drone detection, payload weight estimation*

1. Introduction

Drones have advanced rapidly and are widely used in various fields, such as security, surveying, delivery, photography, disaster response, and agriculture in recent years [1]. However, along with their growing use, concerns have arisen about the possibility that drones can be equipped with payloads of explosives, biological and chemical weapons, and illicit materials for terrorism and other crimes [2], [3]. Therefore, antidrone systems must be able to detect the presence or absence of payloads and deal with these drones on a priority basis. Drone detection technologies, including cameras, microphones, and radars, are being actively studied and developed. Radars are attracting significant attention as an effective drone detection technology because they are not affected by weather conditions, unlike cameras and microphones [4].

Most studies on drone detection using radars rely on the micro-Doppler signatures generated by the rotation of drone propellers [5]–[9]; in these cited studies, drone models were classified based on differences in their micro-Doppler signatures. In addition, several studies have been conducted recently on detection of drones carrying payloads using micro-Doppler signatures [10]–[13]. In [10], the micro-Doppler signatures of two types of drones with different payloads were obtained using W-band, C-band, and S-band frequency-modulated continuous-wave (FMCW) radars. Different micro-Doppler signatures were observed with an increase in payload weight, and a payload weight classification algorithm based on micro-Doppler signatures was proposed. In particular, the W-band was found to be the preferred frequency band for payload classification using the FMCW radar. In [11], the micro-Doppler signatures of a drone with a payload were obtained using an S-band multistatic pulsed Doppler radar. In [12], a convolutional neural network was applied to the data acquired in [11], and payload weights were classified well. Of particular interest is a study about drones equipped with heavy payloads and dynamic payloads generating inertial forces, such as guns [13]. In this study, the micro-Doppler signatures of two types of drones were obtained using a K-band FMCW radar and a W-band continuous-wave radar. The authors discussed the effects of payloads on micro-Doppler signatures and showed that these signatures were inconsistent and not unique to the drones carrying the target payloads. [12] used micro-Doppler signatures for achieving a highly accurate payload classification, similar to [10] and [11]. Furthermore, [13] reported that no unique micro-Doppler signatures could clearly distinguish between drones with and without a payload. Hence, the robust discrimination between payload and no payload is challenging. These results show that depending on the radar specifications and measurement environments, the payload estimation using micro-Doppler signatures may be difficult. Therefore, methods for estimating payload weights that do not rely on micro-Doppler signatures should be explored. [13] and [14] revealed that the rotation frequency of a propeller increases with the payload weight due to the need for additional thrust. The increase trend of the propeller rotation frequency depends on the drone model. Therefore, combined with existing algorithms for classifying drone models, such trends can be used as a reference dataset for estimating payload weights.

In this paper, we propose a method for classifying the payload weight of a drone by estimating its propeller rotation frequency from radar images obtained using a millimeter-

Manuscript received June 21, 2023.
Manuscript revised October 1, 2023.
Manuscript publicized January 30, 2024.

[†]The authors are with National Defense Academy of Japan, Yokosuka-shi, 239-8686 Japan.
a) E-mail: ed22007@nda.ac.jp
b) E-mail: r.nakamura@ieee.org

DOI: 10.23919/transcom.2023EBP3104

wave fast-chirp-modulation multiple-input and multiple-output (mmW FCM MIMO) radar. The proposed method requires a pre-prepared reference dataset that relates the payload weight to the propeller rotation frequency for each drone model. To the best of our knowledge, the proposed method is the first report of a payload estimation method that does not rely on micro-Doppler signatures when investigating the radar-based payload classification. We studied the radar imaging of a drone using an mmW FCM MIMO radar in [15]. The results showed that the propeller rotation produced periodic variations in the signal intensity of the pixels corresponding to the propeller in radar images. The sampling period of an mmW FCM MIMO radar is fast enough for observing a drone's propeller rotation. The use of the millimeter-wave radar in W-band is the preferred choice for the payload estimation, as revealed in [10], and the radar is considered to reflect off small components, such as drone propellers, due to its wavelength characteristics. The rotation frequency of a propeller can be estimated by applying fast Fourier transform (FFT) to the signal intensity variations. To demonstrate the estimation of the rotation frequency of propellers, we conducted measurement experiments on four drones: DJI Matrice 600, DJI Phantom 3, DJI Mavic Pro, and DJI Mavic Mini. Additionally, we performed experiments on a drone with several payloads in a stable position to investigate the effectiveness of the proposed method for estimating payload weights from estimated rotation frequencies.

The rest of this paper is organized as follows. Section 2 is an explanation of the mmW FCM MIMO radar, radar imaging, and the payload weight estimation method. Section 3 shows our measurement results and a discussion of the effectiveness of our proposal. Finally, we summarize this paper in Sect. 4.

2. Payload Weight Estimation

2.1 mmW FCM MIMO Radar

Figure 1 shows a diagram of the mmW FCM MIMO radar. The FCM radar transmits and receives a sinusoidal signal called chirp, whose frequency is modulated over an ultrawide bandwidth with time. The modulation and observation times of a chirp are called fast and slow times, respectively. A received chirp is mixed with a transmitted chirp to measure the intermediate-frequency (IF) signal. The IF signal is sampled using an analog-to-digital converter for each receive antenna and stored in memory as multiple-input, multiple-output (MIMO) channel data. The MIMO channel data, consisting of the IF signals of the radio channels between the transmit and receive antennas, are reconstructed into single-input, multiple-output channel data of a contiguous virtual array (MIMO virtual array) [16]. The received matrix $\boldsymbol{R}(n,m,l)$ obtained using the radar is a 3D data matrix (MIMO virtual array × fast time × slow time) that includes the propagation delay time, direction of arrival (DOA), and Doppler frequency. Here $N(n = 1,2,\cdots,N)$ is the number of fast-time samples, $M(m = 1,2,\cdots,M)$ is the number of

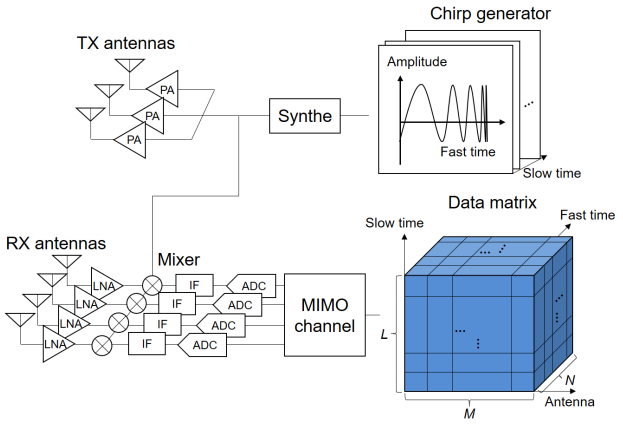


Fig. 1 mmW FCM MIMO radar.

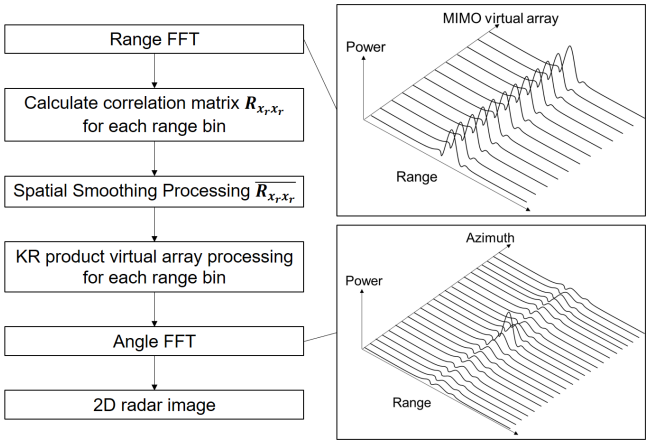


Fig. 2 Flow of digital signal processing.

MIMO virtual array elements, and $L(l = 1,2,\cdots,L)$ is the number of slow-time samples.

2.2 Radar Imaging Procedure

Figure 2 shows the signal processing flow for 2D radar image generation. A 2D FFT process is performed on the received matrix $\boldsymbol{R}(n,m,l)$ to generate a 2D radar image (range-angle map). The distance from the radar to the object is estimated by performing FFT (range FFT) on the IF signal obtained by each element constituting the MIMO virtual array. The data matrix $\boldsymbol{R}_{range}(r,m,l)$ after range FFT is as follows:

$$\boldsymbol{R}_{range}(r,m,l) = \frac{1}{N}\sum_{n=1}^{N}\boldsymbol{R}(n,m,l)e^{-j2\pi f_n \frac{2r}{c}}, \quad (1)$$

where r and c are the range bin and the speed of light, respectively. f_n represents the frequency of the kernel of the Fourier transform.

A drone has many scattering points from its components, such as its body and propellers. The spatial resolution must be improved to obtain clear radar images. As shown in Fig. 2, we apply the Khatri-Rao (KR) product virtual array processing to the MIMO virtual array elements in

each range bin [17]–[19] to improve the angular resolution. Here, assuming that K waves are observed using M uniform linear array (ULA) elements, the MIMO virtual array data $\boldsymbol{R}_{range}(r_b,m,l)$ in a certain range bin r_b are as follows:

$$\begin{aligned}\boldsymbol{R}_{range}(r_b,m,l) &= \sum_{k=1}^{K}\boldsymbol{a}(\theta_k)s_k(l)+\boldsymbol{n}(l)\\ &= \boldsymbol{A}\boldsymbol{s}(l)+\boldsymbol{n}(l) \quad (2)\\ \boldsymbol{A} &= [\boldsymbol{a}(\theta_1),\boldsymbol{a}(\theta_2),\cdots,\boldsymbol{a}(\theta_k)] \quad (3)\\ \boldsymbol{s}(l) &= [s_1(l),s_2(l),\cdots,s_k(l)]^T, \quad (4)\end{aligned}$$

where $\boldsymbol{a}(\theta_k) \in \mathbb{C}^M$ and $s_k(l)$ denote the mode vector and complex amplitude of the k-th wave, respectively; $\boldsymbol{A} \in \mathbb{C}^{M\times K}$ is the mode matrix; and $\boldsymbol{n}(l)$ is the noise vector. The correlation matrix $\boldsymbol{R}_C$ of the MIMO virtual array data $\boldsymbol{R}_{range}(r_b,m,l)$ is as follows:

$$\begin{aligned}\boldsymbol{R}_C &= E[\boldsymbol{R}_{range}(r_b,m,l)\boldsymbol{R}^H_{range}(r_b,m,l)]\\ &= \boldsymbol{A}\boldsymbol{S}\boldsymbol{A}^H+\boldsymbol{R}_N, \quad (5)\end{aligned}$$

where $E[]$ and ^{H} denote ensemble averaging and the complex conjugate transpose, respectively; $\boldsymbol{S}$ is the source correlation matrix; and $\boldsymbol{R}_N$ is the noise correlation matrix. We also apply spatial smoothing processing (SSP) to this correlation matrix before the KR product virtual array processing to suppress the signal coherence of incident waves [20] because the correlation of incident waves leads to errors in virtual array signals [21]. The vectorization $\boldsymbol{y}$ of the spatially smoothed correlation matrix $\overline{\boldsymbol{R}_C}$ is as follows:

$$\begin{aligned}\boldsymbol{y} &= vec[\overline{\boldsymbol{R}_C}]\\ &= vec[\boldsymbol{A}\bar{\boldsymbol{S}}\boldsymbol{A}^H]+vec[\bar{\boldsymbol{R}}_N]\\ &= (\boldsymbol{A}^*\odot\boldsymbol{A})s'+vec[\bar{\boldsymbol{R}}_N], \quad (6)\end{aligned}$$

where $vec[]$ and * are the vectorization operator and the complex conjugate, respectively; $\odot$ denotes the KR product operator; $\boldsymbol{s}' \in \mathbb{C}^K$ is the diagonal element of $\bar{\boldsymbol{S}}$; $(\boldsymbol{A}^*\odot\boldsymbol{A}) \in \mathbb{C}^{M^2\times K}$ is the KR product virtual array response matrix; and the vector $\boldsymbol{y}$ contains repeated elements that do not help increase the aperture length. The nonrepeating elements of vector $\boldsymbol{y}$ are extracted to obtain the KR virtual array data of $2M-1$ elements, so the aperture length is virtually increased.

The DOA of reflected signals is estimated by performing a second FFT (angle FFT) over the indexes of the KR virtual array elements on all range bins of the data matrix $\boldsymbol{R}_{KR}(r,m',l)$ after KR product virtual array processing. The radar image at the l-th slow time $Image(r,a,l)$ generated after angle FFT is as follows:

$$Image(r,a,l) = \frac{1}{2M-1}\sum_{m'=1}^{2M-1}\boldsymbol{R}_{KR}(r,m',l)e^{-j\frac{2\pi(m'-1)}{2M-1}a}, \quad (7)$$

where a is the angle bin and $m'(=1,2,\cdots,2M-1)$ is the index of the virtual antennas after KR product virtual array processing.

2.3 Proposed Method

We investigated the effect of payload weight on the propeller rotation frequency of a drone (Sect. 2.3.1) and developed a payload weight estimation method using the results of this investigation (Sect. 2.3.2).

2.3.1 Reference Dataset for Payload Weight Estimation

The proposed method requires a reference dataset of the relationship between payload weight and propeller rotation frequency. Therefore, to show an example, we created a reference dataset for a hovering Phantom 3.

Figure 3 shows the environment for measuring the rotation frequency of the drone's propeller. The hovering Phantom 3 drone was suspended in the air using guide ropes and connected to a spring scale. A payload weight was applied to the drone because the tension between a drone and a spring scale increases with the drone's propeller rotation frequency. We measured the rotation frequency of the drone using a digital tachometer for 10 s when the spring scale showed values of 0, 250, 500, 750, and 1000 g. In this study, we consider that it is sufficient to detect a threatening payload by estimating rough weight. Therefore, measurement data were collected in a 250 g step.

Figure 4 shows the measured relationship between the payload weight and rotation frequency of the Phantom 3.

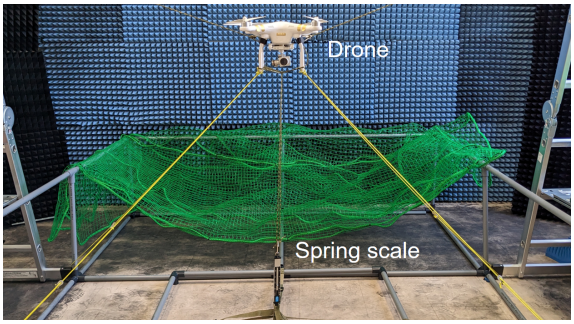


Fig. 3 Measurement environment for generating reference dataset.

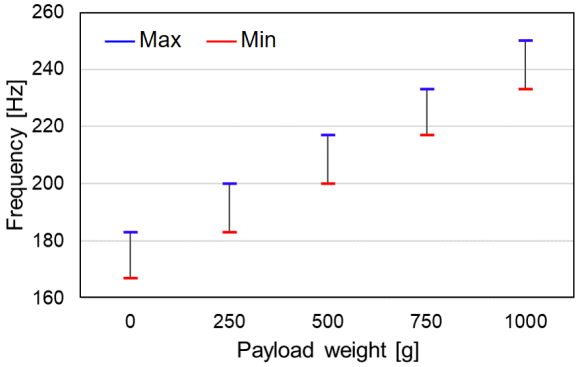


Fig. 4 Relationship between payload weight and rotation frequency of Phantom 3.

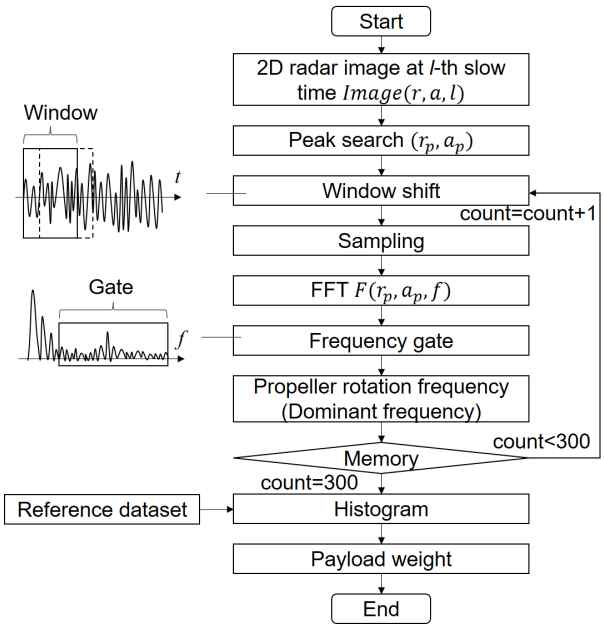


Fig. 5 Flowchart of payload weight estimation method.

The figure indicates an increase in the propeller rotation frequency with the payload. When focused on each payload, it is clear that the frequency is not constant and varies between 16 and 17 Hz due to the drone's attitude control. The frequency variations do not overlap for payloads with 250 g steps, indicating that the payload can be uniquely determined if the rotation frequency is estimated using the radar. However, these frequency variations overlap for steps below 250 g and may cause errors in the payload estimation. The measurement results obtained with 250 g steps were defined as the reference dataset for the payload weight estimation in this study.

2.3.2 Signal Processing

Figure 5 shows a flowchart of our proposed payload weight estimation method. The basis of this method is to find a pixel in a drone's radar image that corresponds to the propeller and analyze the temporal variation of its signal intensity.

First, a 2D radar image of a drone is acquired by the mmW FCM MIMO radar. As an example, the 2D radar image of the Phantom 3 is shown in Fig. 6. The characteristic shape of the drone could be imaged; specifically, the maximum peak at (0.1, 1.8 m) was an echo from the drone's body, and the peaks at (−0.2, 1.7 m) and (0.15, 1.6 m) were the echoes from the left and right propellers, respectively. Thus, a drone's propeller is the reflection point with the largest reflection intensity after that of the body. Therefore, the initial sampling point (r_p, a_p) for the propeller is the pixel of the peak of the second-largest reflection intensity in the radar image. The second and subsequent sampling points were obtained from the same coordinates. Since the reflection intensity of the pixel corresponding to the propeller fluctuates periodically with the propeller rotation, the propeller rotation

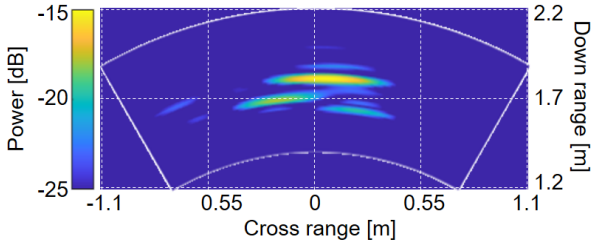


Fig. 6 Example of 2D radar image of Phantom 3.

frequency is estimated by performing FFT on the reflection intensity fluctuation. With the propeller position coordinates in the radar image denoted as (r_p, a_p), the propeller rotation frequency $\boldsymbol{F}(r_p, a_p, f)$ is as follows:

$$\boldsymbol{F}(r_p, a_p, f) = \frac{1}{L}\sum_{l=1}^{L} Image(r_p, a_p, l) e^{-j\frac{2\pi(l-1)}{L}f}, \qquad (8)$$

where f is frequency. The propeller rotation frequency should exceed a certain threshold for a drone to take off. A frequency gate is set for the FFT-calculated frequency spectrum to estimate the propeller rotation frequency. The propeller rotation frequency at takeoff is different for different drones due to differences in their specifications, such as drone weight and motor power. Therefore, the frequency gate depends on the drone model and should be adjusted appropriately for each drone. For example, in the case of the Phantom 3, the frequency gate was set to 150 Hz or higher because its takeoff requires a propeller rotation frequency of 150 Hz or higher. In this gate, the dominant frequency is due to propeller rotation and the peak frequency is sequentially stored in memory as a provisional estimation result of the propeller rotation frequency. Next, since the propeller rotation frequency varies with time due to disturbance, these provisional estimation results are evaluated using a histogram of 300 samples, and the frequency with the mode is used as the final estimation result of the drone's propeller rotation frequency. A small sample size is preferred for the histogram since a large number of samples may affect the distribution because of disturbances due to long observation time. Therefore, the sample size was set to empirically derived value of 300. Finally, the payload weight is estimated by comparing the estimated propeller rotation frequency with the reference dataset.

3. Experimental Setup and Results

3.1 Experimental Setup

We measured propeller rotation frequencies in two experimental measurement cases using an mmW FCM MIMO radar module. Case 1 involved four drones (Matrice 600, Phantom 3, Mavic Pro, and Mavic Mini) without payloads. Case 2 involved a Phantom 3 with several payload weights. Table 1 shows the specifications of the mmW FCM MIMO radar module. The MIMO radar, which is composed of a

3×4 ULA as shown in Fig. 7, presents a MIMO virtual array of 12 elements. Subarrays of 10 elements (= M) were selected from the MIMO virtual array and used for SSP to suppress the coherence of the echoes from each target. The application of KR product virtual array processing increased the number of virtual elements to 19 elements (= $2M - 1$), so the angular resolution was 6.0 degrees. The frequency bandwidth was 3.44 GHz, resulting in a range resolution of 4.4 cm. The number of slow-time samples was 256 (0.25 s), causing a frequency resolution of 4 Hz. The pulse reception interval was set to 0.97 ms, which was fast enough for observing the propeller rotation.

In case 1, we assessed four drones with different shapes, sizes, numbers of rotors, and propeller geometries, as shown in Table 2, and investigated whether the propeller rotation frequency of any drone could be estimated correctly. Each target was placed on a low-density styrofoam cylinder with its propeller rotating, as shown in Fig. 8(a). The antenna height was set to the height of the drone body. The distance between the radar and the target was adjusted for each drone so that the entire drone, including its propellers, would be covered by the antenna beam. Each drone was positioned so that one propeller was the closest to the radar to observe the echoes from the propeller in a manner that maximizes the signal-to-noise ratio.

In case 2, we tested the Phantom 3 with several payload weights W (= 0, 250, 500, 750, 1000 g) using the spring scale (Sect. 2.3.1) to investigate the effectiveness of the proposed payload weight estimation approach. The hovering target was suspended in the air using guide ropes to prevent it from flying outside the antenna beam, as shown in Fig. 8(b). The target was positioned so that the camera faced its front, as seen in Fig. 3. Since drones were expected to enter the radar coverage area at various flight altitudes, we evaluated the accuracy of the payload weight estimation method at different antenna elevation angles θ (=0°, 10°, 20°, and 30°).

Table 1 Specifications of mmW FCM MIMO radar module.

Array shape		Unfiorm linear array
Number of elements	Tx	3
	Rx	4
Antenna spacing	Tx	8 mm
	Rx	2 mm
Beamwidth	Azimuth	±35 deg
	Elevation	±4 deg
Center frequency		78.72 GHz
Frequency bandwidth		3.44 GHz
Sweep time		57 μ s
Pulse repetition interval		0.97 ms
Number of sampling points in fast time (N)		240
Number of sampling points in slow time (L)		256

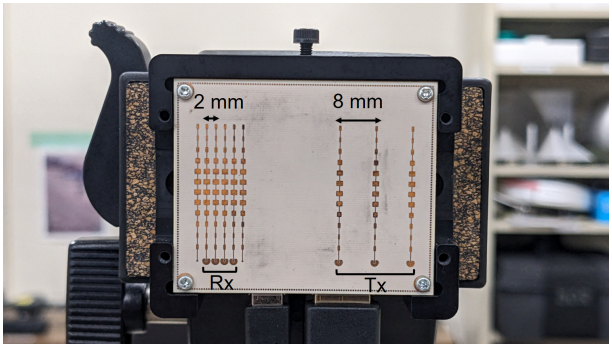
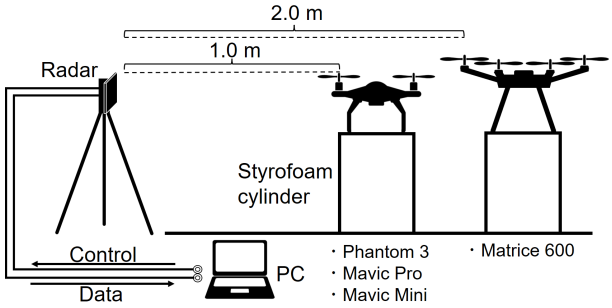
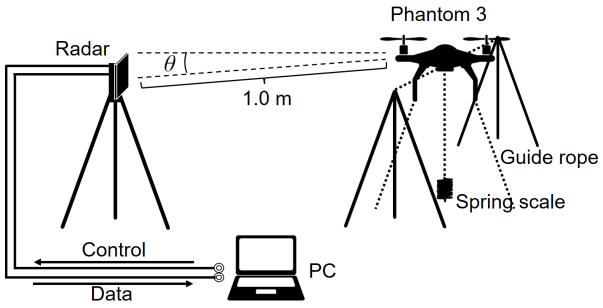


Fig. 7 MIMO radar.



(a) case 1

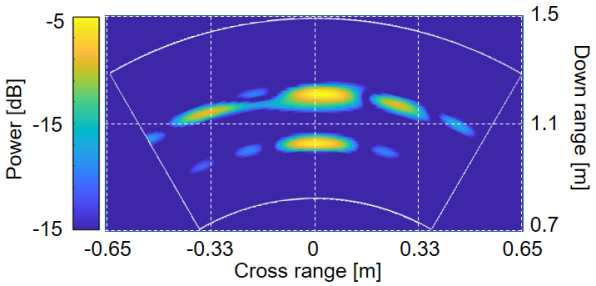


(b) case 2

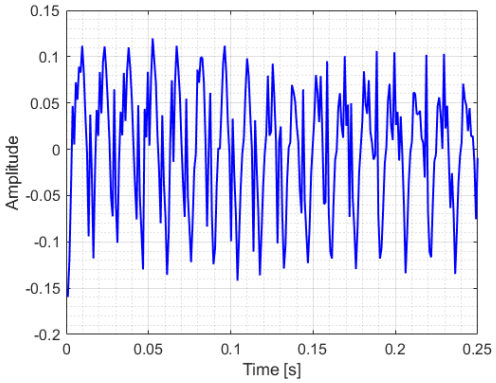
Fig. 8 Measurement environments.

Table 2 Tested drones.

	Matrice 600	Phantom 3	Mavic Pro	Mavic Mini
Width × Depth [mm]	1200 × 1200	400 × 400	380 × 350	200 × 180
Propeller diameter [mm]	535	235	205	120
Exterior				



(a) an example of measured 2D radar image



(b) fluctuations in signal strength due to propeller rotation

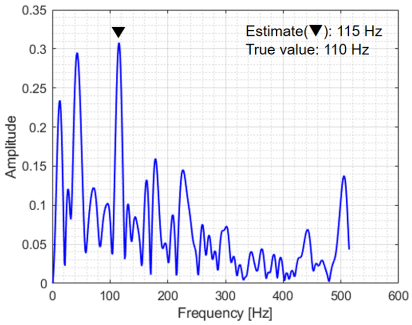
Fig. 9 Phantom 3 measurement results.

3.2 Experimental Results and Discussion

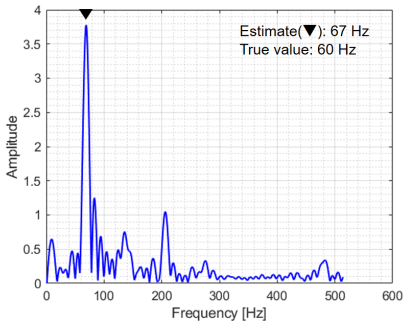
3.2.1 Case 1

Figure 9 shows the Phantom 3 measurement results. Several strong echoes are seen in Fig. 9(a). The strong peak at (0, 1.2 m) is an echo from the body, and the peaks at (0, 1.0 m), (−0.30, 1.1 m), and (0.25, 1.2 m) are the echoes from the propellers. Since the rear propeller was obscured by the body, no echo from the rear propeller is observed. Fig. 9(b) shows the waveform of the signal intensity fluctuation due to a propeller. This waveform was generated through the time-series sampling of the signal intensity of the (0, 1.0 m) pixel, which corresponds to a propeller in the 2D radar image. The DC component of the waveform was removed. The waveform amplitude fluctuates due to changes in the radar cross section during propeller rotation. The fluctuation period is related to the propeller rotation speed, and similar periodic fluctuations are observed in the other tested drones.

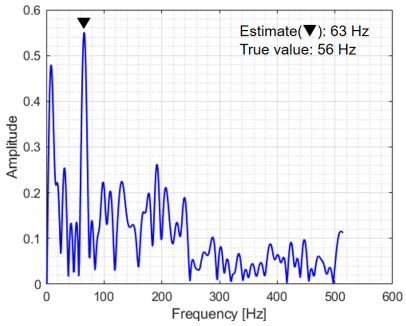
Figure 10 shows the frequency spectrum of the time waveform of each drone. The frequency corresponding to the maximum value in the frequency spectrum is denoted as ▼ in the figure, which is the estimated propeller rotation frequency. The true value of the propeller rotation frequency was measured using a digital tachometer. Figures 10(a), (c), and (d) show strong peaks in the low-frequency component (under 50 Hz). These peaks may have been caused by the vibration of the drone arms due to propeller rotation; each drone was placed on the styrofoam cylinder, so the drone body could not have caused vibration. Arm vibration is a unique characteristic of drones that have separate bodies and arms, such as the Matrice 600, Mavic Pro, and Mavic Mini. These peaks can be removed through filter processing using a high-pass filter or by setting a frequency gate. Figure 11



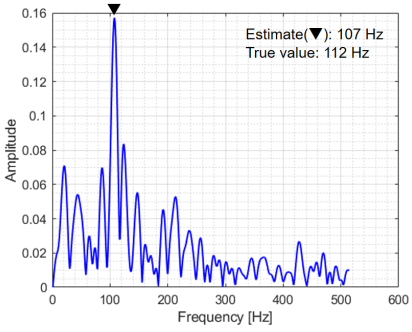
(a) Matrice 600



(b) Phantom 3



(c) Mavic Pro



(d) Mavic Mini

Fig. 10 Examples of propeller rotation frequency spectra.

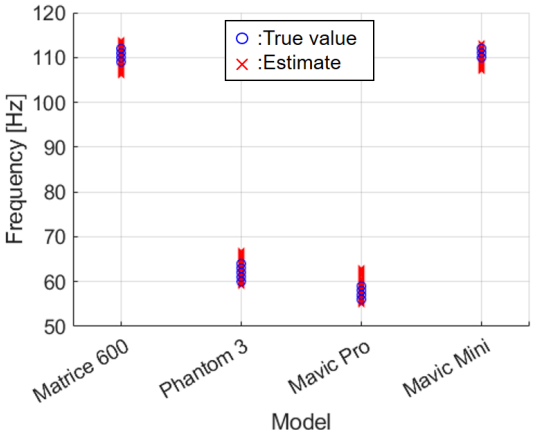


Fig. 11 Estimation results of the propeller rotation frequency for each drone.

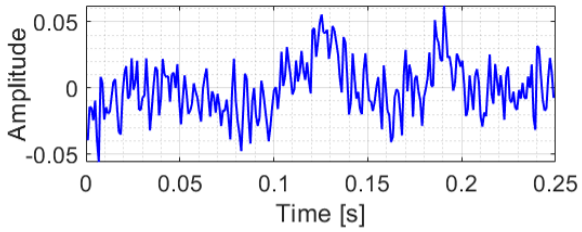


Fig. 12 An example of the signal intensity waveform in Case 2.

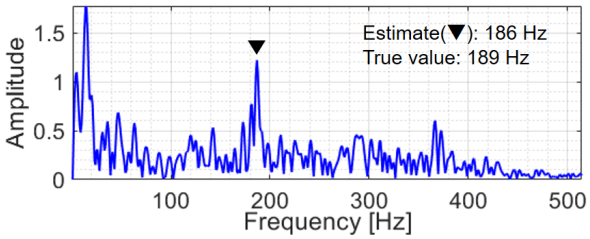


Fig. 13 An example for the estimation result of the propeller rotation frequency in Case 2.

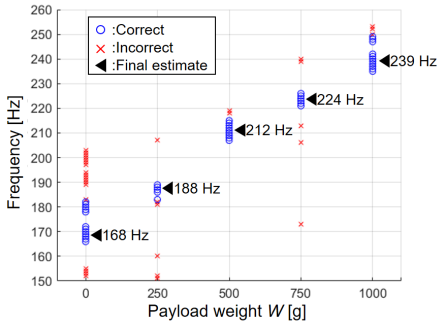
shows the estimated and measured rotation frequencies for each drone. Measurements were obtained for 556 slow-time samples. Subsequently, a total of 300 estimates of the propeller rotation frequency were obtained by performing FFT on the measured data while shifting the FFT window length of 256 samples by one sample at a time. From Fig. 11, in Case 1, where there are almost no fluctuations other than that caused by the propeller, the propeller rotation frequency can be estimated with an error of less than a few hertz for all tested drones. The main factor that causes the estimated value to vary more than the true value is the estimation error caused by the FFT.

3.2.2 Case 2

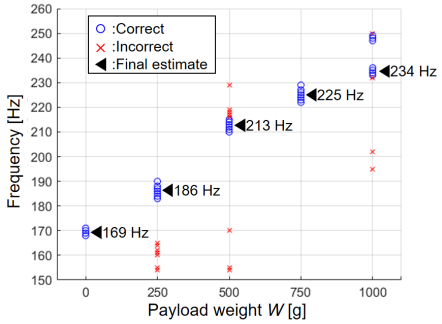
Measurements were obtained for 556 slow-time samples. Further, a total of 300 propeller rotation frequency estimates were obtained by applying FFT on the measured data while shifting the FFT window length of 256 samples by one sample at a time. Figure 12 shows an example of the signal intensity waveform in Case 2, in which an increase is observed in the irregular fluctuation components compared to that exhibited by the waveform of Case 1 shown in Fig. 9(b). This irregularity is attributed to the shaking and vibration of the drone's body during hovering. The frequency spectrum of the waveform in Fig. 12 is shown in Fig. 13, along with its corresponding estimated propeller rotation frequency (▼). In addition to the peak representing the propeller rotation frequency (Fig. 10(b)), the spectrum has a large peak in the low-frequency region, attributed to the shaking and vibration of the drone body. However, the propeller rotation frequency can be estimated by performing a peak search after passing the frequency spectrum through a frequency gate, similar to Case 1. Figure 14 shows the provisional estimates of the propeller rotation frequency at each payload weight. The blue circles (○) and red crosses (×) in the graphs denote the correct and incorrect estimates, respectively, compared with the reference dataset. Figure 14 indicates that the estimates increase with the payload weight, as shown in Fig. 4. In addition, the correct estimates (blue circles) at each payload weight vary due to temporal changes in the propeller rotation frequency caused by drone attitude control. The incorrect estimates (red crosses) are insufficient or excessive frequencies for maintaining the drone's hovering state. These misestimates may have been caused by random disturbances, such as body sway due to attitude control or body vibration due to propeller rotation.

The histogram of provisional estimates was evaluated to determine the final estimate of the propeller rotation frequency, thus avoiding the abovementioned misestimates. When the propeller rotates at a rotation frequency closer to the frequency boundary in the reference dataset, the estimation accuracy of the propeller rotation frequency would be affected by the bin size of the histogram. In this study, the bin size was set to 1 Hz to align with the measurement resolution of the digital tachometer. For example, Fig. 15 shows the histogram of the provisional estimates at an elevation angle $\theta = 20°$ and a payload weight $W = 250$ g. Most of the provisional estimates are at approximately 197 Hz, which is within the frequency range of the reference dataset at $W = 250$ g. However, approximately 30% of the estimates are outside the frequency range, leading to payload weight misestimation. Therefore, 197 Hz, which has the highest occurrence probability, is the propeller rotation frequency in our experiment. Final estimates presented in Fig. 14 correspond to the propeller rotation frequency determined using the mode in their histograms.

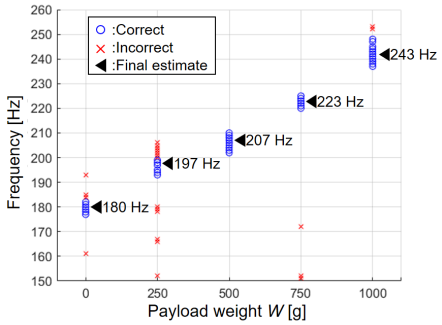
Each payload weight is classified by comparing the propeller rotation frequency determined from the histogram with the corresponding value in the reference dataset in Fig. 4. Table 3 shows the payload weight classification results at each antenna elevation angle. Each column (row) in the table represents the instances of the estimated (actual) payload weights. "Other" means that the payload weight could not be



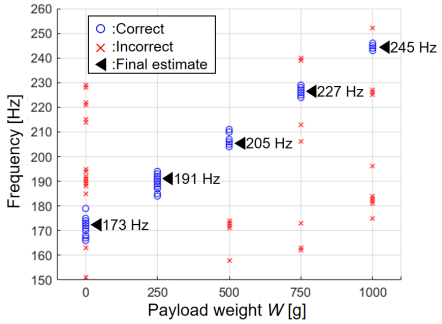
(a) $\theta = 0°$



(b) $\theta = 10°$



(c) $\theta = 20°$



(d) $\theta = 30°$

Fig. 14 Provisional estimates of propeller rotation frequency vs. payload weight.

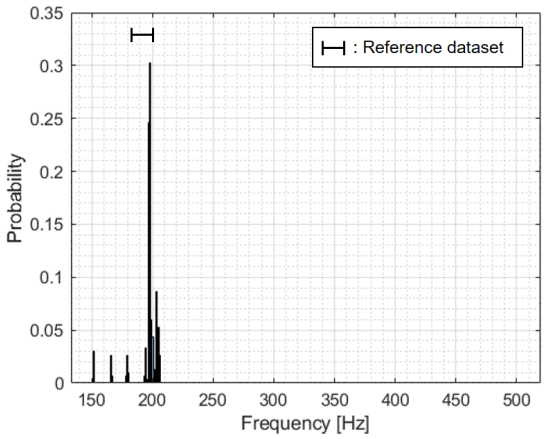


Fig. 15 Histogram of provisional estimates ($\theta = 20°$, $W = 250$ g).

Table 3 Payload weight classification results.

(a) $\theta = 0°$

Payload weight W [g] \ Estimated payload weight [g]	0	250	500	750	1000	Other
0	95	5	0	0	0	0
250	0	100	0	0	0	0
500	0	0	100	0	0	0
750	0	0	0	100	0	0
1000	0	0	0	21	79	0

(b) $\theta = 10°$

Payload weight W [g] \ Estimated payload weight [g]	0	250	500	750	1000	Other
0	100	0	0	0	0	0
250	0	78	0	0	0	22
500	0	0	99	1	0	0
750	0	0	0	100	0	0
1000	0	0	0	0	100	0

(c) $\theta = 20°$

Payload weight W [g] \ Estimated payload weight [g]	0	250	500	750	1000	Other
0	77	23	0	0	0	0
250	0	100	0	0	0	0
500	0	0	100	0	0	0
750	0	0	0	100	0	0
1000	0	0	0	0	100	0

(d) $\theta = 30°$

Payload weight W [g] \ Estimated payload weight [g]	0	250	500	750	1000	Other
0	100	0	0	0	0	0
250	0	100	0	0	0	0
500	0	28	72	0	0	0
750	0	0	0	100	0	0
1000	0	0	0	0	100	0

estimated because the estimated propeller rotation frequency was outside the range of the reference dataset. To evaluate the accuracy of the payload classification, we performed 100 classification runs by taking a 54-second (55600 samples in slow-time) measurement and dividing the measured data into 100 segments (556 samples in slow-time per segment). Each cell in the table represents the probability of 100 classification runs corresponding to each measurement of the actual

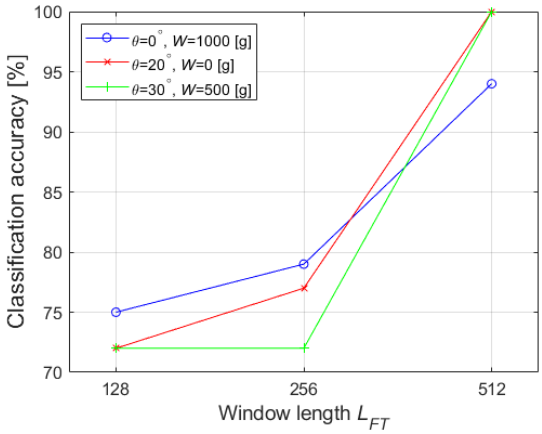


Fig. 16 Comparison of classification accuracy for different FFT window lengths.

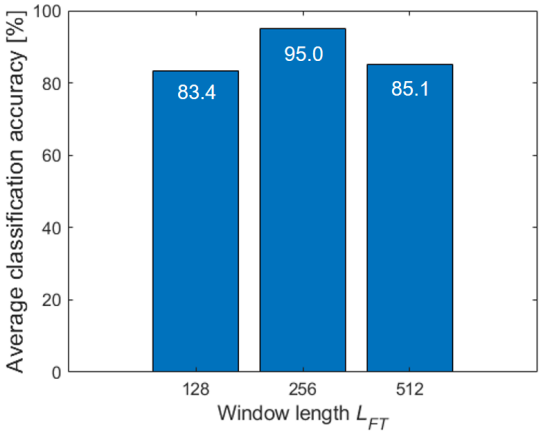


Fig. 17 Average classification accuracy for different FFT window lengths.

payload weight W. The blue cells in the table represent the probability of correct classification, which is defined as the classification accuracy. The average classification accuracy of all blue cells, is more than 94.4% at each elevation angle. The results show that the proposed method can accurately classify most of the payload weights, and there is almost no difference in the average classification accuracy between elevation angles.

In Table 3(b), 22% are classified as "Other" at the actual payload weight W = 250 g. This is probably because the case of W = 250 g caused more body shaking and vibration than other cases, thereby affecting the original signal intensity fluctuations of the propeller. Table 3(a), (c), and (d) show misclassifications where the payload is classified as lighter or heavier than its actual weight. Misclassifications occurred irregularly for any weight at any elevation angle, indicating the absence of a consistent error trend that depends on the elevation angle. The main reasons of these misclassifications are sudden random body shaking and frequency estimation errors caused by the FFT. Further, we discuss the FFT estimation error in detail.

We investigated the effect of the FFT window length on estimation accuracy. Figure 16 shows the classification accuracy for different FFT window lengths (L_{FT}), where ∘, ×, and + indicate the classification accuracy for $\theta = 0°$ and W = 1000 g, for $\theta = 20°$ and W = 0 g, and for $\theta = 30°$ and W = 500 g, respectively. Figure 16 confirms that the classification accuracy improves with longer window lengths because the frequency resolution increases with the window length. Figure 17 shows the average classification accuracies at all elevation angles and payload weights. The figure indicates that the average classification accuracy degrades in the case of LFT = 512 despite the improved frequency resolution compared with that of LFT = 256. With longer window lengths, the effects of drone body shaking and vibration are more likely to show in the signal intensity waveform, which is used to estimate the propeller rotation frequency. Since the frequency components due to these disturbances became the mode in the histogram, the average classification accuracy declined. Therefore, a trade-off exists between the effect of disturbances and the frequency resolution, and setting a window length that considers the effect of disturbances is important for the proposed method.

4. Conclusions

In this paper, we propose a method for classifying the payload weight of a drone by estimating the propeller rotation frequency from radar images obtained using an mmW FCM MIMO radar. The proposed method necessitates a pre-prepared reference dataset that can relate the payload weight to the propeller rotation frequency for each drone model. Two experimental measurement cases were conducted to investigate the effectiveness of our proposal. In case 1, we tested four drones to determine whether the propeller rotation frequency of any drone could be correctly estimated. The experimental results showed that the propeller rotation frequencies of all drones could be estimated. In case 2, measurement experiments were conducted on a hovering drone with five different payloads in a stable position to evaluate the accuracy of payload weight classification. Results revealed that the proposed method could classify the payloads in a 250 g step with an average accuracy of more than 94.4%. However, as the FFT window length for estimating the propeller rotation frequency increased, the classification accuracy decreased due to the increased influence of disturbances. Therefore, an appropriate window length should be set for accurate classification.

We plan to investigate the possibility of classification of payloads in moving drones at far range in the future. Moreover, we aim to implement algorithms that are robust to disturbances, such as body shaking and vibration.

Acknowledgments

This work was supported by JSPS KAKENHI Grant Number 21K04102.

References

[1] Research Briefs, 38 Ways Drones Will Impact Society: From FightingWar To ForecastingWeather, UAVs Change Everything, Retrieved Dec. 17, 2019, from https://www.cbinsights.com/research/drone-impact-society-uav/, accessed Feb. 15. 2023.

[2] J.P. Yaacoub, H. Noura, O. Salman, and A. Chehab, "Security analysis of drones systems: Attacks, limitations, and recommendations," Internet of Things, vol.11, p.100218, Sept. 2020. DOI: 10.1016/j.iot.2020.100218

[3] E. Vattapparamban, I. Guvenc, A.I. Yurekli, K. Akkaya, and S. Uluagac, "Drones for smart cities: Issues in cybersecurity, privacy, and public safety," Proc. 2016 International Wireless Communications and Mobile Computing Conference, Paphos, Cyprus, pp.216–221, Sept. 2016. DOI: 10.1109/IWCMC.2016.7577060

[4] A. Coluccia, G. Parisi, and A. Fascista, "Detection and classification of multirotor drones in radar sensor networks: A review," Sensors, vol.20, no.15, p.4172, July 2020. DOI: 10.3390/s20154172

[5] J.J.M. de Wit, R.I.A. Harmanny, and G. Premel-Cabic, "Micro-Doppler analysis of small UAVs," Proc. 9th European Radar Conference (EuRAD), Amsterdam, Netherlands, pp.210–213, Oct. 2012.

[6] P. Molchanov, K. Egiazarian, J. Astola, R.I. Harmanny, and J.J.M. de Wit, "Classification of small UAVs and birds by micro-Doppler signatures," Proc. 10th European Radar Conference (EuRAD), Nuremberg, Germany, vol.6, no.3-4, pp.172–175, Oct. 2013. DOI: 10.1017/S1759078714000282

[7] S. Rahman and D.A. Robertson, "Millimeter-wave micro-Doppler measurements of small UAVs," Proc. SPIE Defense + Security, Radar Sensor Technology XXI, Anaheim, CA, United States, vol.10188, pp.307–315, May 2017. DOI: 10.1117/12.2261942

[8] S. Rahman and D.A. Robertson, "Multiple drone classification using millimeter-wave CW radar micro-Doppler data," Proc. SPIE Defense + Commercial Sensing, Radar Sensor Technology XXIV, vol.11408, pp.50–57, April 2020. DOI: 10.1117/12.2558435

[9] M. Kurosaki, K. Ogawa, R. Nakamura, and H. Hadama, "Experimental study on multiple drone detection using a millimeter-wave fast chirp MIMO radar," Proc. 2023 IEEE Topical Conference on Wireless Sensors and Sensor Networks (WisNet), Las Vegas, NV, USA, pp.16–19, Jan. 2023. DOI: 10.1109/WiSNeT56959.2023.10046220

[10] D. Dhulashia, N. Peters, C. Horne, P. Beasley, and M. Ritchie, "Multi-frequency radar micro-Doppler based classification of micro-drone payload weight," Front. Signal Process., vol.1, p.781777, Dec. 2021. DOI: 10.3389/frsip.2021.781777

[11] M. Ritchie, F. Fioranelli, H. Borrion, and H. Griffiths, "Multistatic micro-Doppler radar feature extraction for classification of unloaded/loaded micro-drones," IET Radar, Sonar & Navigation, vol.11, no.1, pp.116–124, Jan. 2017. DOI: 10.1049/iet-rsn.2016.0063

[12] J.S. Patel, C. Al-Ameri, F. Fioranelli, and D. Anderson, "Multi-time frequency analysis and classification of a micro-drone carrying payloads using multistatic radar," The Journal of Engineering, vol.2019, no.20, pp.7047–7051, Oct. 2019. DOI: 10.1049/joe.2019.0551

[13] S. Rahman, D.A. Robertson, and M.A. Govoni, "Radar signatures of drones equipped with heavy payloads and dynamic payloads generating inertial forces," IEEE Access, vol.8, pp.220542–220556, Dec. 2020. DOI: 10.1109/ACCESS.2020.3042798

[14] O.A. Ibrahim, S. Sciancalepore, and R.D. Pietro, "Noise2Weight: On detecting payload weight from drones acoustic emissions," Future Generation Computer Systems, vol.134, pp.319–333, Sept. 2022. DOI: 10.1016/j.future.2022.03.041

[15] K. Ogawa, M. Kurosaki, R. Nakamura, and H. Hadama, "2D imaging of a drone using a millimeter-wave fast chirp MIMO radar based on Khatri-Rao product virtual array processing," Proc. 2023 IEEE Topical Conference on Wireless Sensors and Sensor Networks (WisNet), Las Vegas, NV, USA, pp.1–4, Jan. 2023. DOI: 10.1109/WiSNeT56959.2023.10046224

[16] J. Li and P. Stoica, MIMO Radar Signal Processing, Wiley-IEEE Press, 2008.

[17] W.K. Ma, T.H. Hsieh, and C.Y. Chi, "DOA estimation of quasi-stationary signals via Khatri-Rao subspace," Proc. 2009 IEEE International Conference on Acoustics, Speech and Signal Processing, Taipei, Taiwan, pp.2165–2168, April 2009. DOI: 10.1109/ICASSP.2009.4960046

[18] W.K. Ma, T.H. Hsieh, and C.Y. Chi, "DOA estimation of quasi-stationary signals with less sensors than sources and unknown spatial noise covariance: A Khatri-Rao subspace approach," IEEE Trans. Signal Process., vol.58, no.4, pp.2168–2180, April 2010. DOI: 10.1109/TSP.2009.2034935

[19] H. Yamada, N. Ozawa, Y. Yamaguchi, K. Hirano, and H. Ito, "Angular resolution improvement of ocean surface current radar based on the Khatri-Rao product array processing," IEICE Trans. Commun., vol.E96-B, no.10, pp.2469–2474, Oct. 2013. DOI: 10.1587/transcom.E96.B.2469

[20] S.U. Pillai and B.H. Kwon, "Forward/backward spatial smoothing techniques for coherent signal identification," IEEE Trans. Acoust., Speech, Signal Process., vol.37, no.1, pp.8–15, Jan. 1989. DOI: 10.1109/29.17496

[21] S. Shirai, H. Yamada, and Y. Yamaguchi, "A novel DOA estimation error reduction preprocessing scheme of correlated waves for Khatri-Rao product extended-array," IEICE Trans. Commun., vol.E96-B, no.10, pp.2475–2482, Oct. 2013. DOI: 10.1587/transcom.E96.B.2475

Kenshi Ogawa received the B.E. and M.E. degrees in Information Engineering from The University of Kitakyushu, Japan, in 2016 and 2018, respectively. Since 2022, he has been a student at the Graduate School of Science and Engineering, National Defense Academy of Japan. He is currently focusing on microwave/millimeter radio propagation and radar systems, and working for his D.E degree. He is a student member of the IEICE and IEEE.

Masashi Kurosaki received the B.E. and M.E. degrees from the Graduate School of Science and Engineering, National Defense Academy of Japan, in 2017 and 2023, respectively. Until 2023, he was working on microwave/millimeter radio propagation and radar systems.

Ryohei Nakamura received B.E., M.E., and D.E. degrees in information engineering from The University of Kitakyushu, Japan, in 2009, 2011, and 2014, respectively. In 2014, he was a research associate in the Department of Communication Engineering, National Defense Academy of Japan, and has been an Associate Professor in the same department since 2020. His major research interests include wireless communications, microwave/millimeter radio propagation, radar sensor systems, and network systems. He is a member of the IEICE and IEEE.