Load Balancing for Stateful Forwarding by Mitigating Heavy Hitters: A Case for Multi-Threaded NDN Software Routers

JUNJI TAKEMASA1, (Member, IEEE), ATSUSHI TAGAMI1, (Member, IEEE), YUKI KOIZUMI2, (Member, IEEE), AND TORU HASEGAWA2, (Member, IEEE)

1KDDI Research, Inc., Fujimino 356-8502, Japan
2Graduate School of Information Science and Technology, Osaka University, Suita 565-0871, Japan

Corresponding author: Junji Takemasa (ju-takemasa@kddi-research.jp)

This work was supported by the Japan Society for the Promotion of Science (JSPS) KAKENHI under Grant 20H04176.

ABSTRACT Equal load balancing for dispatching incoming packets to multiple threads is a crucial requirement in the stateful forwarding of multi-threaded software routers to achieve high-speed forwarding and low packet loss simultaneously. However, equal load balancing is not trivial for Named Data Networking (NDN) routers because of their stateful forwarding. In other words, the consistency of flow states should be maintained so that multiple threads do not access the states simultaneously. Sharding, wherein packets of the same flow are dispatched to the same thread while keeping loads of threads equal, has been proposed; however, in this study, we reveal that heavy hitters like popular content packets cause load imbalance, which may eventually cause packet losses. This study proposes a load balancing mechanism for NDN routers by exploiting the fact that states of flows need not be rigorously maintained when content packets are returned from caches at intermediate routers.

INDEX TERMS Named data networking, information-centric networking, router architecture, sharding, mutual exclusion.

I. INTRODUCTION

Named Data Networking (NDN) [1] is a novel network architecture that provides useful in-network functionalities, such as caching [2] and stateful forwarding [3]. A high-speed NDN software router is expected to play a key role in early Internet-scale NDN deployment because of its flexibility, whereas a purpose-built hardware router is hard to change the network architecture [4]. High-speed forwarding has been a long-lasting research issue for such routers and several studies achieve forwarding at 40-80 Gbps by implementing NDN software router prototypes [5]–[7]. These research prototypes exploit the parallelism of multiple CPU cores provided by modern CPU devices and our research prototype completely exploits 22 CPU cores by pre-fetching forwarding data structures from slow memory devices, such as DRAM devices [8]. Because one thread is assumed to run on each CPU core, we, hereinafter, interchangeably use the words “CPU core” and “thread” interchangeably, and call an NDN software router an “NDN router”.

Despite their success, most studies do not consider how a Network Interface Card (NIC) dispatches incoming packets to threads. If such packets were equally dispatched to threads, multiple threads could completely exploit CPU cores without any packet loss. However, this equal load balance is difficult to achieve because a router should record and update a state of a flow, which is a sequence of Interest and Data packets of the same name. In NDN, an Interest and a Data packet are a request and a reply of a content object piece, respectively. Note that a content object, such as a video file, consists of pieces and each piece is transferred using a pair of Interest and Data packets of the same name.

Two dispatching schemes are candidates; however, none of them achieve equal load balance. The first scheme is round robin or random dispatching, whereby the NIC dispatches incoming packets to threads. If such packets were equally dispatched to threads, multiple threads could completely exploit CPU cores without any packet loss. However, this equal load balance is difficult to achieve because a router should record and update a state of a flow, which is a sequence of Interest and Data packets of the same name. In NDN, an Interest and a Data packet are a request and a reply of a content object piece, respectively. Note that a content object, such as a video file, consists of pieces and each piece is transferred using a pair of Interest and Data packets of the same name.

The associate editor coordinating the review of this manuscript and approving it for publication was Bhaskar Prasad Rimal.
same thread; thus, each thread exclusively reads and writes the flow states of the same names. Sharding has an inherent problem; that is, skewed popularity distributions of Data packets may cause load imbalance among threads. However, Saino et al. validate that the skewness of Data packet popularity distributions is much less than that of content object popularity distributions, assuming that a content object consists of many Data packets [10]. Therefore, sharding at the packet level does not cause load imbalance even if the skewness of content object popularity distributions is significant.

However, their assumption does not hold in the actual Internet because only a few leading packets of popular content objects are requested in many cases. In other words, it is not likely that all the pieces of a content object are requested. Several measurement studies show that the large skewness caused by such leading packets is observed in traffic traces of actual Internet Service Provider and Long Term Evolution backbone networks [11], [12]. The observations imply that packet-level sharding does not achieve equal load balance among threads.

A frontend cache of sharded systems [13], such as distributed cache systems and key-value store systems (KVSs), is a promising solution to mitigate these skewed distributions. The frontend cache is designed based on the observation that the root cause of such load imbalance is a small number of highly popular content objects (i.e., heavy hitters) [10], [13]. Here, shards are multiple servers that perform the same procedure as threads in a software router and servers in a distributed cache system/KVS. Fan et al. [13] proposed a frontend cache for KVSs and the key idea behind it is that a frontend cache, which is placed in front of multiple shards, handles heavy hitters (i.e., caches such popular content objects). This enables shards (i.e., the other servers) to only cache the unpopular content objects. Hence, equally dispatching such unpopular content objects to shards is easy to achieve. A simple sharding-based dispatching scheme is sufficient for an equal load balance to shards.

However, adopting the idea of the frontend cache [10], [13] is not trivial because of the difference between NDN software routers and the above-mentioned sharded systems. Although an NIC plays the same role as a frontend cache in other sharded systems, it cannot handle popular packets such as frontend cache. In other words, even if the NIC identifies highly popular packets, such packets should be handled by threads that work as shards.

In this study, we address this problem by leveraging a phenomenon specific to NDN functionality. The phenomenon is that flow states are not accessed by threads when incoming Interest packets hit Data packets in a cache. To the best of our knowledge, this is one of the first studies that shed light on this phenomenon. The proposed packet dispatching scheme is summarized below. An NIC handles differently popular and unpopular packets. The unpopular packets are dispatched according to a sharding-based dispatching scheme so that packets of the same name are dispatched to the same thread. In contrast, popular packets are dispatched to threads according to a random dispatching scheme rather than according to a sharding-based scheme. The random dispatching scheme with fine-grained locking reduces the probability that the flow states are locked.

This study makes the following contributions:

- We demonstrate that heavy hitters (i.e., highly popular Interest packets) are a root cause of load imbalance by precisely analyzing the time of handling packets at a modern CPU device. The analysis is based on the number of CPU cycles spent executing individual NDN functions. As the CPU cycle numbers are experimentally measured by running a well-designed NDN router on a modern CPU device, we believe that the analysis result is realistic.
- We propose a popularity-based packet dispatching scheme that spreads heavy hitters across threads while dispatching other packets according to sharding. The scheme addresses the load imbalance caused by highly popular Interest packets by leveraging the idea of the frontend cache. Additionally, we design a packet forwarding algorithm that implements the proposed packet dispatching scheme.
- We analytically estimate the performance of the proposed packet forwarding algorithm in terms of packet loss ratios and packet forwarding rates rather than measuring those of a prototype implementation. In the analysis, the proposed scheme is compared with a random dispatching scheme with mutual exclusion and a sharding-based scheme. The result validates that the proposed packet dispatching scheme achieves equal load balance. In other words, the scheme achieves both low packet loss ratios and high packet forwarding rates simultaneously.

The rest of this article is organized as follows. Section II explains the overview of the stateful forwarding of NDN. Section III describes the design rationale of the packet dispatching scheme by carefully analyzing the root cause of load imbalance at threads. Section IV designs a popularity-based packet dispatching scheme and integrates it to the NDN packet forwarding algorithm. Section V describes the model for analyzing the performance of the designed packet forwarding algorithm. Section VI analyzes the performance of the proposed packet forwarding algorithm and compares it with other algorithms. Section VII summarizes the related work, and Section VIII concludes this study.

II. NDN PACKET PROCESSING AND MUTUAL EXCLUSION

This section describes the NDN packet processing flow in terms of flow state updates and then the mutual exclusion mechanism of modern CPU architectures on which NDN routers run. Prior to the descriptions, to avoid confusion, Table 1 summarizes the terminology used throughout this study.
TABLE 1. Terminology.

| Term       | Description                                                                 |
|------------|-----------------------------------------------------------------------------|
| Name       | A unique identifier of a content object piece.                              |
| Interest packet | A request packet containing a content object piece. It has the name of a  |
|             | requesting content object piece.                                            |
| Data packet | A reply packet containing a requested content object piece. It has the name |
|             | of a content object piece.                                                  |
| Flow       | A sequence of Interest and Data packets of the same name.                   |
| Flow state | A PIT entry for a name. It records the incoming interfaces of unsatisfied  |
|             | Interest packets of the same name.                                          |
| Popularity | The average number of Interest packets of a name in time.                   |
| Popular name | A name of which popularity is greater than a predefined threshold.            |
| Popular packet | A packet of a popular name.                                                  |

A. NDN PACKET PROCESSING

Stateful forwarding of NDN is different from the stateless forwarding of conventional IP in terms of the write intensity of data structures. This subsection explains the NDN packet processing flow of an Interest and a Data packet from the perspective of data structure updates. An NDN router forwards Interest and Data packets with the following three data structures: a Forwarding Information Base (FIB), a Pending Interest Table (PIT) and a Content Store (CS). The FIB is a forwarding table for Interest packets. The PIT maintains the state of each unsatisfied Interest packet. Here, the term “unsatisfied” means that the Interest packet does not hit any Data packet in a cache called a CS. The PIT, which records incoming interfaces of unsatisfied Interest packets, is used as a forwarding table for replied Data packets. It records all the flow states of the unsatisfied Interest packets.

The way in which these data structures are updated is described using the NDN packet processing flow illustrated in Figure 1. When an Interest packet arrives, the router looks up the CS. If the corresponding Data packet is found in the CS, it is sent back to the downstream router. Note that the router does not access the flow state of the name, i.e., the Xen-try in the PIT. Hereinafter, we interchangeably use the terms “flow state” and “PIT entry”. Otherwise, the router refers to the PIT. If the PIT entry is found, the router inserts the incoming interface into the entry. Otherwise, the router creates a new PIT entry in the PIT and writes the incoming interface at the PIT entry. It forwards the Interest packet to the upstream router according to the FIB. When a Data packet arrives, the router looks up the PIT entry to obtain the outgoing interface and then deletes it from the PIT. Subsequently, the router inserts the Data packet into the CS and evicts one of the Data packets from the CS according to a CS eviction algorithm.

B. MUTUAL EXCLUSION ACCORDING TO COMPARE-AND-SWAP INSTRUCTION

The challenge of multi-threading comes from write-intensive NDN packet processing. This subsection describes the mechanism of mutual exclusion on modern CPU devices and the overhead caused by the mutual exclusion for a PIT and a CS. The remainder of this subsection assumes fine-grained locking, where entries are individually locked. Hash table structures are used for the CS and PIT similar to most NDN software routers [5]–[7] as the probability of lock contention is low, as analyzed by David and Guerraoui [14].

Nevertheless, mutual exclusion on PC-based hardware platforms incurs serious overheads. In multi-core CPU platforms, the atomic compare-and-swap (CAS) instruction must be used to lock variables, such as flow states. However, the atomicity between a compare operation and a swap operation is ensured by the fact that the thread issuing the CAS instruction occupies the shared ring bus, which connects CPU caches and memory devices, in modern CPU devices [15]. This implies that the thread disturbs the other threads because their memory accesses are blocked.

The CAS instruction is issued every time a thread modifies a PIT entry. We experimentally investigate how the forwarding speed is degraded if the PIT entries are locked using the CAS instruction. The experiments were performed on an NDN software router that we developed in our previous study [9]. The results show that such frequent CAS instruction executions increase per-packet processing time by approximately 1.8 times larger in the case of 22 threads than that of the single thread.

III. DESIGN RATIONALE

A. PACKET DISPATCHING SCHEME

Inspired by the idea of frontend cache [13], we design a packet dispatching scheme. This scheme uses different packet dispatching rules depending on the popularity of names. Highly popular Interest packets are randomly dispatched to threads. This rule is called the random dispatching rule. Conversely, the other packets, specifically, unpopular Interest packets and Data packets, are dispatched to threads according to sharding. This rule is called the sharding-based dispatching rule.

The scheme solves the problems inherent to sharding and mutual exclusion owing to the random dispatching rule. First, as highly popular Interest packets are a root cause of load imbalance when the sharding-based dispatching rule is used, random dispatching of such Interest packets resolves the load imbalance. Second, although the sharding-based dispatching rule of Data packets requires mutual exclusion for highly
popular Data packets, CS hits of highly popular Interest packets reduce frequencies of mutual exclusion for such packets. This is because PIT entries are not accessed when Interest packets hit Data packets in a CS.

The analyses of the next subsections III-B and III-C validate the above claims quantitatively.

**B. ANALYSIS OF THREADS’ LOADS**

By precisely analyzing the loads of threads on realistic environments, we validate whether Interest packets with highly popular names are the root cause of load imbalance for NDN routers or not. The analysis here follows the model of the multi-threaded NDN router that we developed in our previous study [16]. The underlined number for each block in Fig. 1 indicates the measured CPU cycle number of each block. For example, block $B_1$ spends 340 CPU cycles for handling one packet on average. Assuming that all the threads have the same computing capacity in modern CPUs, the load of each thread is defined as the average number of CPU cycles consumed for handling packets in one second. This is expressed as a product of the arrival packet rate and the number of CPU cycles consumed in handling one packet. Given that $\lambda_n$ and $p_n$ are the arrival Interest packet rate and the CS hit rate at thread $n$, respectively, the arrival Data packet rate at thread $n$ is expressed as $\lambda_n(1-p_n)$. The CPU cycle number spent handling a Data packet $C_{Data}^{Data}$ includes the CPU cycles of blocks $B_3$ and $B_6$ in Fig. 1. An Interest packet in the cases of a CS miss and a CS hit incur different numbers of CPU cycles, $C^{I-miss}$ and $C^{I-hit}$, respectively. $C^{I-miss}$ includes the CPU cycles of blocks $B_1$, $B_3$, and $B_4$, and $C^{I-hit}$ includes those of blocks $B_1$ and $B_2$. Consequently, the load of thread $n$ is expressed as follows:

$$l_n = \lambda_n[(1-p_n)C^{I-miss} + p_nC^{I-hit}] + \lambda_n(1-p_n)C_{Data}^{Data}.$$  

(1)

Loads of individual threads are analyzed under a typical scenario where $10^7$ unique Data packets with different names are requested according to the Zipf distribution with the skewness $\alpha = 1.0$ [17]. The average request rate (i.e., that of Interest packets) is $10^7$ [packet/s] and 22 threads run on a router. Other detailed conditions are described in Section VI-A. The two observations are obtained from the analysis results.

First, load imbalance mainly comes from Interest packets rather than Data packets. Figure 2 shows the loads of threads when the NIC uses the sharding-based dispatching rule and when the router caches Data packets in a CS. In the x-axis of the figure, the threads are ordered based on the size of loads. The blank and hatched boxes indicate the loads of handling Interest and Data packets, respectively. The figure shows that the loads of handling Data packets are similar for individual threads. The reason is as follows. As Interest packets with highly popular names mostly hit Data packets in a CS, most of them are not forwarded to the next hop routers. Consequently, the number of highly popular Data packets is remarkably smaller than that of the corresponding Interest packets. This phenomenon reduces the differences between the numbers of Data packets that individual threads receive. Conversely, the loads of threads that receive highly popular Interest packets are much higher than those of the other threads.

Second, the root cause of such high loads is a small number of heavy hitter Interest packets. We analyze the number of arrival Interest packets in each name at thread 1, which is the most highly loaded thread. Interest packets with the most popular name account for 60.9% of the total number of arrival Interest packets at thread 1. The similar trend is observed for other highly loaded threads 2 and 3.

The observations indicate that the popularity-based packet dispatching scheme in Section III-A archives equal load balancing of threads. Thus, loads of threads are balanced only by randomly dispatching highly popular Interest packets.

**C. ANALYSIS OF FREQUENCIES OF LOCKING PIT ENTRIES**

This subsection demonstrates that the combination of the two principles of the popularity-based packet dispatching scheme reduces the frequency of locking CS entries and PIT entries by analyzing how CS and PIT entries are read and written by threads in two steps. The first step of the analysis demonstrates that CS entries need not be locked, and the second step demonstrates that the locking of PIT entries rarely occurs. This implies that even if highly popular Interest packets are randomly dispatched to threads according to the packet dispatching scheme, overheads caused by the locking of PIT entries are negligible.

For the preparation of the analysis, we demonstrate how blocks read and write entries of a CS and a PIT. Blocks $B_1$, $B_2$, $B_3$, $B_5$, and $B_6$ in Fig. 1 access the PIT and the CS, where access is read or write. Table 2 summarizes which data structure the block accesses and whether the access is read or write.

| Block | Data Structure | Access   |
|-------|----------------|----------|
| $B_1$ | CS             | Read     |
| $B_2$ | CS             | Read     |
| $B_3$ | PIT            | Write    |
| $B_5$ | PIT            | Write    |
| $B_6$ | CS             | Write    |

**TABLE 2. Access pattern of each block in Fig. 1.**

**FIGURE 2. Load of each thread using the sharding-based dispatching rule and CS.**
First, we describe how blocks access the CS. Blocks \( B_1 \) and \( B_2 \), which are executed upon receiving an Interest packet, perform the read access to the CS, whereas block \( B_6 \), which is executed upon receiving a Data packet, performs the write access to the CS. As demonstrated by Lim et al. [18], read accesses to an object in a cache of a key-value store system can be performed concurrently by multiple threads without mutual exclusion if the single thread exclusively performs the write access to the object. The above-mentioned patterns of CS accesses satisfy this condition. This finding indicates that Interest packets as read accesses to the CS can be dispatched to any thread without mutual exclusion for the CS, whereas Data packets as write accesses should be dispatched to the same thread according to the sharding-based dispatching rule.

Second, we describe how blocks access the PIT. Blocks \( B_3 \) and \( B_5 \), which are executed upon receiving an Interest and a Data packet, respectively, perform the write access to the PIT. Therefore, the PIT should be locked if Interest or Data packets of the same name are dispatched to different threads according to the first principle described in Section III-B. Here, an important observation is that both blocks \( B_3 \) and \( B_5 \) for the PIT are executed only when the Interest packet misses the CS at block \( B_1 \), as shown in Fig. 1. The router rarely executes blocks \( B_3 \) and \( B_5 \) because incoming Interest packets with highly popular names frequently hit the CS. This implies that the number of mutual exclusion is small even if highly popular Interest packets are randomly dispatched to threads.

The observations indicate that the first principle of the packet dispatching scheme in Section III-A decreases the probability that the PIT entries are locked. Section VI analytically validates that the probability is negligible.

IV. DESIGN
A. OVERVIEW
This section describes the design of a multi-threaded NDN router according to the packet dispatching scheme proposed in the previous section. The hardware platform is assumed to consist of an NIC, a CPU with multiple CPU cores (threads), and a memory device, as shown in Figure 3. The NIC receives a packet and then dispatches it to one of the threads. The thread handles the dispatched packet with the CS and the PIT stored on the memory device, as illustrated by the block diagram in Fig. 1. This section describes the design of the packet dispatching scheme at the NIC and the algorithms for handling PIT and CS entries.

FIGURE 3. Hardware platform of multi-threaded NDN router.

B. PACKET DISPATCHING SCHEME
The NIC dispatches an incoming packet according to the random dispatching rule or the sharding-based dispatching rule. As described in Section III-A, the NIC uses the random dispatching rule for highly popular Interest packets and the sharding-based dispatching rule for the other packets (i.e., unpopular Interest packets and Data packets). We define the CS hit rate of Interest packets of the same name as the popularity value of the name. Clearly, this is because a high CS hit rate implies that the Data packet of the name is frequently requested and calculating the CS hit rates for individual names is a negligibly light computation. Here, the thread identifiers are numbered from 1.

Figure 4 shows the packet processing flow at the NIC. The NIC first decides the dispatching rule to be used. If an incoming packet is a Data packet, then the NIC dispatches it to a thread according to the sharding-based dispatching rule. Conversely, if it is an Interest packet, the NIC decides the rule depending on the popularity of the name. If the CS hit rate of the name is higher than the predefined threshold (e.g., 0.8), then the name is regarded as being highly popular. We refer to the name as the highly popular name. The NIC dispatches packets of highly popular names according to the random dispatching rule. Otherwise, it dispatches the packet according to the sharding-based dispatching rule.

FIGURE 4. Packet processing flow at the NIC.

The NIC adds a bit that indicates whether a corresponding PIT entry should be locked or not, i.e., the \( L \) bit in Fig. 4 to a packet. The \( L \) bit is set for Interest and Data packets of highly popular names, as illustrated in Fig. 4. Note that modern commodity NICs have the capability to mark packets in their packet descriptor fields [19].

C. PIT ENTRY HANDLING
A PIT is used to record the flow state of the same name’s packet, as illustrated in Fig. 5. A PIT entry is prepared for each name and records the name, interfaces at which Interest packets arrive, and a lock variable used for mutual exclusion. The thread locks a PIT entry only when the \( L \) bit of an incoming packet is set to 1. Otherwise, it accesses the PIT entry without locking it.

We employ fine-grained pessimistic locking as the mutual exclusion mechanism. A lock variable of each PIT entry is a binary lock variable, which indicates whether the entry is being written (1) or not (0). When a thread receives a packet of which the \( L \) bit is set, it issues the CAS instruction for
FIFO queue. In the figure, the lookup table of thread #1 is
version variable, and a pointer to a Data packet saved in a
FIFO queue, as illustrated in Fig. 6a. The lookup table is
locking is performed.

structure of the CS, this subsection describes how optimistic
locking requires that a single writer writes a shared
variable, an exclusive CS is prepared for each thread, rather
than a shared CS for all the threads. After discussing the data
structure of the CS, this subsection describes how optimistic
locking is performed.

The data structure of a CS consists of a lookup table and
a FIFO queue, as illustrated in Fig. 6a. The lookup table is
indexed by names, and each entry has a name, an integer
version variable, and a pointer to a Data packet saved in a
FIFO queue. In the figure, the lookup table of thread #1 is
indexed by three names /a, /b, and /c, and two Data packets of
/b and /c are saved in the FIFO queue. The version variable is
used for optimistic locking. When the variable is an odd
value, the thread writes the CS entry to insert or evict a Data
packet. Conversely, if it is an even value, then the thread does
not access the CS entry.

A CS entry for an unpopular name is exclusively accessed
by a single thread as Interest and Data packets of unpopular
names are dispatched to the same thread according to the
sharding-based dispatching rule. All the packets of the same
name are handled sequentially by the thread, and thus, mutual
exclusion for the CS entry is not required.

Meanwhile, the same CS entry may be simultaneously
accessed when a highly popular Data packet is inserted into
the CS. The thread handling the Data packet, which we
call the thread of Data, may update two CS entries: the CS
entry of the inserted Data packet and that of the evicted Data
packet. As shown Fig. 6, thread #1 is about to insert Data
packet /a into the CS and evict Data packet /c from the CS.
If the thread of Data increments the both version variables
before the other threads handling Interest packets of the same
names, such as /a and /c, each of which we call the thread of
Interest, read the version variables, the threads sequentially
handle the Data and Interest packets. Therefore, no problem
occurs.

If the Interest packet reads the version variable, the two
threads of Data and Interest may access the same CS entry
simultaneously. The remainder of this subsection describes
that no problem occurs from the perspective of NDN protocol
execution in the following two cases. The first case is that
the thread of Interest accesses the CS entry to which a new
Data packet is about to be inserted, as illustrated in Fig. 6a.
Assume that thread #2 reads the version variable of name
/a before thread #1 increments the version variable. The
thread of Interest, thread #2, performs block B3 while the
thread of Data, thread #1, is inserting the Data packet into
the CS. Thread #2 forwards the Interest packet to the next
hop router before the Data packet is inserted into the CS.
However, this does not cause a failure in terms of the NDN
protocol. In other words, when the Data packet is replied to
the router, it forwards the Data packet to the sender of the
Interest packet.

The second case is that the thread of Interest accesses the
CS entry from which the Data packet is about to be evicted,
as illustrated in Fig. 6b. Here, thread #1 evicts Data packet /c
and thread #2 sends back the same Data packet. However,
before the thread of Interest sends back Data packet /c at
block B2, it notices that the Data packet is evicted during
block B1 by checking the version variable of /c. Specifically,
the thread of Interest reads the version of /c two times before
and after performing block B1. If the two values of the version
are different, Data packet /c has been evicted during block B1.
In this case, the thread of Interest discards the Data packet and
then retries from block B1. Otherwise, the thread of Interest
sends back the Data packet.

D. CS ENTRY HANDLING
We employ an exclusive First In First Out (FIFO) CS with
optimistic locking, as proposed by Lim et al. [18]. Because
optimistic locking requires that a single writer writes a shared
variable, an exclusive CS is prepared for each thread, rather
than a shared CS for all the threads. After discussing the data
structure of the CS, this subsection describes how optimistic
locking is performed.

The data structure of a CS consists of a lookup table and
(a) Executing block B3
(b) Executing block B2
FIGURE 6. Simultaneous Data and Interest packets.
algorithm is not ideally implemented as a result of hardware constraints.

1) APPROXIMATING NAME-BASED DISPATCHING IN MODERN NICs
A memory device of an NIC is assumed to record selected dispatching rules for all the names. However, modern memory devices of NIC devices [19] do not have a large capacity to store them. A feasible countermeasure is that names are hashed and dispatching rules are recorded for individual hash values. Modern NIC devices have up to 8k entries for classifying incoming packets and can add classifier values to the headers of packets that are dispatched to threads. We employ this classification functionality to use the names’ hash values as classifier values. However, adopting hash values rather than names arises concern when different names have the same hash value. Even if different dispatching rules are used by the names, the same dispatching rule should be applied to the packets of the names. A critical problem is when the names of highly popular and unpopular names have the same hash value. In such a case, packets of such names should pessimistically use the random dispatching rule so that mutual exclusion is performed for unpopular Interest packets. This negative effect is modeled and evaluated in Sections V and VI, respectively.

2) PIT INDEXED BY HASH VALUES
A PIT is usually indexed by hash values of names like in most NDN router implementations [5]–[7] because NDN uses variable-length names. To avoid redundant hash computations at an NIC and a thread, the NIC computes the hash values of the names that are used as indices of the PIT. This enables each PIT entry to be exclusively accessed by one thread. Here, the hash length supported by modern NICs [19] is an issue. However, their 32 bit length is long enough for a PIT at a thread to hold approximately $1 \times 10^6$ entries. Here, Carofiglio et al. [20] estimated $1 \times 10^6$ entries as the maximum number of PIT entries for backbone NDN routers.

3) COUNTING CS HITS
Threads are assumed to count CS hit rates for hash values rather than names. By employing a state-of-the-art counting technique [21], the computation time for counting the rate is negligible and as a result, it is much shorter than that of NDN packet processing.

4) IMPROVING CS HIT RATES
Implementations according to a design of this study achieve high CS hit rates as well as high-speed forwarding. We employ a frequency-based CS admission algorithm, such as [22], [23] with a simple FIFO CS eviction one. As demonstrated by Takemasa et al. [22] and Einziger et al. [23], such CS admission algorithms provide CS hit rates that are comparable to sophisticated CS eviction algorithms [24], [25]. However, to improve CS hit rates, an exclusive frequency-based CS admission can be performed with a FIFO CS eviction on each CS.

On the contrary, an exclusive CS for each thread rather than a shared CS for all the threads may be an issue for high CS hit rates. However, Saino et al. [10] revealed that the differences in hit rates of the two types of CSs are negligible. Therefore, we consider that exclusive CSs do not degrade CS hit rates.

V. ANALYSIS MODEL
A. OVERVIEW
We create a queuing model for an NDN router to evaluate the popularity-based packet dispatching scheme in terms of network performance. The queuing model consists of multiple threads, each of which is independently running on the router. The thread performs the NDN packet processing shown in Fig. 1 for the Interest and Data packets as the service. We calculate the packet forwarding rates and packet loss ratios of threads according to the queuing model.

Specifically, we create the queuing model of the popularity-based packet dispatching scheme and those of the three schemes as references for comparison. We call the four schemes popularity-based, sharding, mutual exclusion and ideal schemes.

- The popularity-based scheme is defined in Section IV. Hereinafter, the predefined threshold of CS hit rates for deciding highly popular Interest packets is called $\theta$.
- The sharding scheme dispatches incoming Interest packets according to the sharding-based dispatching rule. In other words, all packets of the same name are assigned to the same thread, and thus mutual exclusion is never performed in the PIT.
- The mutual exclusion scheme dispatches incoming Interest packets randomly to the threads. In other words, mutual exclusion is performed in the PIT every time an Interest packet or a Data packet arrives.
- The ideal scheme dispatches incoming Interest packets randomly to threads, but mutual exclusion is assumed to never be performed in the PIT. The scheme is an oracle that knows everything.

Note that all the schemes use an exclusive CS for each thread and that the PIT is shared by all the threads. Because the same framework is used by the models, the subsection of the sharding scheme describes the framework itself.

B. SHARDING SCHEME
1) FRAMEWORK: QUEUING MODEL
The router has $N$ threads and each thread $1 \leq n \leq N$, $n \in \mathbb{N}$ has its own Rx queue of packets with the size $Q_n$. Because each thread operates independently, we model thread $n$ as an M/M/1/K queuing system. Here, for simplicity, we assume that Interest and Data packets arrive at the router according to the Poisson process. Given that $\lambda_n$ and $\mu_n$ are the arrival rate of packets and the service rate of handling packets at thread $n$, respectively, we can derive the packet loss ratio at thread $n$, $q_n$, from a well-known blocking probability of the M/M/1/K
queuing model as follows:

\[ q_n = \frac{(1 - \lambda_n / \mu_n)(\lambda_n / \mu_n)^{\gamma_n}}{1 - (\lambda_n / \mu_n)^{\gamma_n + 1}}. \]

The average packet loss ratio of an entire NDN router is calculated as \( \sum_{n=1}^{N} \lambda_n q_n / \sum_{n=1}^{N} \lambda_n \). With \( q_n \), the packet forwarding rate of thread \( n \), \( \gamma_n \), is expressed as \( \gamma_n = \lambda_n (1 - q_n) \). The total packet forwarding rate of an entire NDN router is calculated as \( \sum_{n=1}^{N} \gamma_n \). The remainder of this subsection describes how to derive \( \lambda_n \) and \( \mu_n \).

2) ARRIVAL PACKET RATE

The arrival rate of packets at thread \( n \) is calculated as the sum of that of Interest packets, \( \lambda_n \), and that of Data packets, \( \gamma_n \), i.e., as \( \lambda_n = \lambda_n^I + \lambda_n^D \). \( \lambda_n^I \) is calculated as \( \lambda_n^I = \sum_{c \in G} \lambda^I(c) I_n(c) \), where \( \lambda^I(c) \) and \( G \) are the arrival Interest packet rate with name \( c \) at the router and a set of all the names, respectively. \( I_n(c) \) is the ratio of the arrival Interest packet rate with name \( c \) at thread \( n \) to the total arrival rate at the router \( \lambda_n \). Note that \( I_n(c) \) indicates the packet dispatching scheme of Interest packets with name \( c \). Interest packets with name \( c \) arrive at thread \( n \) only if name \( c \) is dispatched to thread \( n \) according to the sharding-based dispatching rule. Hence, \( I_n(c) \) is expressed as follows:

\[ I_n(c) = \begin{cases} 1 & \text{if } c \text{ is dispatched to } n \\ 0 & \text{else.} \end{cases} \]

Note that we determine to which thread name \( c \) is dispatched by simply computing the random number ranging from 1 to \( N \). Data packets with name \( c \) arrive at thread \( n \) if Interest packets with name \( c \) miss the CS, and if name \( c \) is dispatched to thread \( n \) according to the sharding-based dispatching rule. Given that \( D_n \) is the ratio of the arrival Data packet rate with name \( c \) at thread \( n \) to the total arrival rate with name \( c \) at the router, \( \lambda_n^D \) is calculated as \( \sum_{c \in G} \lambda^D(c)(1 - p_n(c)) D_n(c) \), where \( p_n(c) \) is the CS hit rate of Interest packets with name \( c \) at thread \( n \). Note that \( D_n(c) \) indicates the packet dispatching scheme of Data packets with name \( c \). Because the Interest and the Data packets with name \( c \) are dispatched to the same thread, \( D_n(c) \) is expressed as follows:

\[ D_n(c) = I_n(c). \]

The CS hit rate \( p_n(c) \) is derived by applying a single FIFO CS model [26] to a CS of thread \( n \) because each thread independently runs a FIFO eviction algorithm on its CS. \( p_n(c) \) is equal to \( p_n^{IN}(c) \), the probability that the Data packet with name \( c \) is in the CS of thread \( n \). \( p_n^{IN}(c) \) is expressed as follows:

\[ p_n^{IN}(c) = \frac{\lambda(c) D_n(c) \tau_n}{1 + \lambda(c) D_n(c) \tau_n}, \]

where \( \tau_n \) is the mean CS eviction time for all Data packets at thread \( n \), as proved by Martina et al. [26]. \( \tau_n \) is derived as the unique root of \( S_n = \sum_{c \in G} p_n^{IN}(c) \), where \( S_n \) is the CS size of thread \( n \). The average CS hit rate of Interest packets at thread \( n \), \( p_n \), is calculated as the average of the CS hit rate of Interest packets with name \( c \) arriving at thread \( n \). Because the arrival Interest packet rate with name \( c \) at thread \( n \) is \( \lambda(c) I_n(c) \), \( p_n \) is expressed as follows:

\[ p_n = \frac{\sum_{c \in G} \lambda(c) I_n(c) p_n(c)}{\sum_{c \in G} \lambda(c) I_n(c)}. \]

3) SERVICE RATE

The service rate at thread \( n \), \( \mu_n \), is calculated as the number of packets that thread \( n \) can handle in one second. Given that thread \( n \) has a CPU processing capacity of \( F \) cycles/s, which corresponds to the operating frequency of one CPU core, \( \mu_n \) is calculated as \( \mu_n = F / C_n \), where \( C_n \) is the average number of CPU cycles that thread \( n \) spends for handling one NDN packet. \( C_n \) is derived as a quotient of the total number of CPU cycles spent for processing packets in one second \( \phi_n \) and the arrival rate of packets \( \lambda_n \) at thread \( n \). \( C_n \) is expressed as \( C_n = \phi_n / \lambda_n \).

\( \phi_n \) is calculated as a product of the number of CPU cycles that thread \( n \) spends for handling one packet and the arrival packet rate at thread \( n \). However, as shown in Fig. 1, there are multiple packet processing flows for the same arrival packet, depending on its type and whether the packet hits the CS or not, and CPU cycle numbers are different for the flows. Therefore, for each flow, we calculate the number of CPU cycles spent at the flow in one second as a product of the number of CPU cycles spent for handling one packet at the flow and the arrival rate of packets for which the flow is executed. Next, we calculate \( \phi_n \) as the sum of the CPU cycle numbers of all the flows. As shown in Fig. 1, there are three packet processing flows: An Interest packet processing flow in the case of a CS miss, that in the case of a CS hit, and a Data packet processing flow. Note that we refer to these three processing flows as I-miss, I-hit, and Data flows. The number of CPU cycles spent at the I-miss flow in one second \( \phi_n^{I-miss} \) is calculated as the product of the number of CPU cycles spent handling one Interest packet at the I-miss flow \( \lambda_n^{I-miss} \) and the arrival rate of Interest packets that miss the CS \( \lambda_n^{I-miss}(1 - p_n) \). The number of CPU cycles spent at the I-hit flow \( \phi_n^{I-hit} \) and that spent at the Data flow \( \phi_n^{Data} \) are calculated similar to \( \phi_n^{I-miss} \). Specifically, \( \phi_n^{I-miss}, \phi_n^{I-hit}, \text{ and } \phi_n^{Data} \) are expressed as follows:

\[ \phi_n^{I-miss} = \lambda_n^{I-miss}(1 - p_n) C_n^{I-miss}, \]
\[ \phi_n^{I-hit} = \lambda_n^{I-hit} p_n C_n^{I-hit}, \]
\[ \phi_n^{Data} = \lambda_n C_n^{Data}, \]

where \( C_n^{I-miss}, C_n^{I-hit}, \text{ and } C_n^{Data} \) denote the number of CPU cycles spent for processing one packet at the I-miss, I-hit, and Data flows, respectively. From Fig. 1, \( C_n^{I-miss}, C_n^{I-hit}, \text{ and } C_n^{Data} \) are expressed as follows:

\[ C_n^{I-miss} = C_1 B_1 + C_2 B_2 + C_3 B_3, \]
\[ C_n^{I-hit} = C_1 B_1 + C_2 B_2, \]
\[ C_n^{Data} = C_5 B_5 + C_6 B_6, \]

where \( C_i B_j \) denotes the number of CPU cycles spent for block \( B_j \) in Fig. 1.
C. MUTUAL EXCLUSION SCHEME

The packet loss ratio and the packet forwarding rate are derived similarly to the derivation demonstrated in Section V-B1. However, \( \lambda_n \) and \( \mu_n \) are derived differently from the derivations demonstrated in Sections V-B2 and V-B3, respectively. The remainder of this subsection describes how to derive \( \lambda_n \) and \( \mu_n \).

1) ARRIVAL PACKET RATE

\( \lambda_n^I \) and \( \lambda_n^D \) are calculated similarly to the derivation demonstrated in Section V-B2. However, \( I_n(c) \) of Eq. (2) and \( D_n(c) \) of Eq. (3) are derived differently as follows. Because incoming Interest packets are spread to all the \( N \) threads uniformly, \( I_n(c) \) of Eq. (2) is rewritten as follows:

\[
I_n(c) = 1/N.
\]

As incoming Data packets with name \( c \) are dispatched to the thread determined by the sharding-based dispatching rule, \( D_n(c) \) of Eq.(3) is rewritten as follows:

\[
D_n(c) = \begin{cases} 
1 & \text{if } c \text{ is dispatched to } n \\
0 & \text{else}.
\end{cases}
\]

Note that we determine a thread to which the Data packet with name \( c \) is dispatched by simply computing the random number ranging from 1 to \( N \).

The probability that the Data packet with name \( c \) is in a CS of thread \( n \), \( p_n^D(c) \), is derived similarly to Eq. (4) because the mutual exclusion scheme also employs exclusive CSs similarly to the sharding scheme. However, the CS hit rate of Interest packets with name \( c \) at thread \( n \), \( p_n(c) \), is not equal to \( p_n^D(c) \) similar to \( p_n(c) \) of the sharding scheme. This is because thread \( n \), to which an Interest packet with name \( c \) is dispatched randomly, may be different from thread \( m \) to which the Data packet with name \( c \) is dispatched according to the sharding-based dispatching rule. In this case, thread \( n \) checks the CS of thread \( m \), as described in Section IV-D. Therefore, \( p_n(c) \) is expressed as \( p_n(c) = p_n^N(c) \), where thread \( m \) satisfies the condition \( D_m(c) = 1 \).

2) SERVICE RATE

\( \mu_n \) and \( \phi_n \) are derived similarly to the derivation demonstrated in Section V-B3. However, \( C^{I-miss} \), \( C^{I-hit} \), and \( C^{Data} \) are derived differently from Eq.(7) because the CAS instruction for the mutual exclusion of the PIT adds redundant CPU cycles to those for handling one packet.

The CAS instruction is executed once for locking a PIT entry every time a block of PIT processing is performed in a packet processing flow. Precisely, because the I-miss flow performs PIT processing at block \( B_3 \) of Fig. 1, it spends the number of CPU cycles spent for executing the CAS instruction, \( C^{CAS}(N) \), at block \( B_3 \) in addition to that spent for the I-miss flow, which is calculated as \( C^{I-miss} \) of Eq.(7). \( C^{CAS}(N) \) is the number of CPU cycles spent for executing the CAS instruction when \( N \) threads are running. Similarly, the Data flow performs PIT processing at block \( B_5 \), thereby spending \( C^{CAS}(N) \) at block \( B_5 \) in addition to \( C^{Data} \) of Eq.(7). The I-hit flow does not perform PIT processing at blocks \( B_1 \) and \( B_2 \), thereby spending CPU cycles similar to the \( C^{I-hit} \) of Eq.(7): \( C^{I-miss} \), \( C^{I-hit} \), and \( C^{Data} \) are expressed as follows:

\[
\begin{align*}
C^{I-miss} &= C^{B_1} + C^{B_3} + C^{B_5} + C^{CAS}(N), \\
C^{I-hit} &= C^{B_1} + C^{B_2}, \\
C^{Data} &= C^{B_5} + C^{B_6} + C^{CAS}(N).
\end{align*}
\]

(10)

\( C^{CAS}(N) \) is derived from the number of running threads \( N \) and the number of CPU cycles spent for executing the CAS instruction in the case of single thread execution, \( C^{CAS} \), as explained below.

In modern CPUs [15], when a thread executes the CAS instruction, it occupies the shared ring bus of the CPU device through which CPU caches and the DRAM device are accessed. While a thread is executing the CAS instruction, the other \( N - 1 \) threads wait until the CAS instruction of the thread is completed. Here, we assume that the time of CAS instruction execution is proportional to the number of running threads. With this assumption, \( C^{CAS}(N) \) is derived as \( N C^{CAS} \), where \( C^{CAS} \) is the number of CPU cycles spent for executing the CAS instruction when no other threads are running.

We validate the assumption by measuring the number of CPU cycles spent for executing the CAS instruction under the condition that multiple threads run concurrently on a modern CPU device. Figure 7 shows the number of CPU cycles spent for executing the CAS instruction on a thread when the number of running threads is changed from 1 to 20 on an Intel Xeon E5 2699 v4 CPU [27]. In the measurements, each thread executes the CAS instruction repetitively to fetch and write a cache line, which is chosen uniformly from all \( 10^5 \) cache lines, and performs a hash computation with 831 CPU cycles. The hash computation emulates the 864 cycles for handling one NDN packet on average, as measured in Section VI-A. To measure the number of CPU cycles in the ideal case where the data fetch latency of the cache line is minimized as much as possible, each thread prefetches the cache line to its L1D CPU cache before executing the CAS instruction. Figure 7 also shows the number of the CPU cycles estimated according to \( C^{CAS}(N) \). The figure reveals that the \( C^{CAS}(N) \) well approximates the stopping time due to the CAS instruction.

D. IDEAL SCHEME

The ideal scheme is similar to the mutual exclusion scheme, except that no mutual exclusion occurs. Specifically, \( C^{CAS} \)
is assumed to be zero in the calculation of $C_{\text{CAS}}(N)$ of the mutual exclusion scheme. This scheme is used as a reference for evaluating how much the other three schemes degrade the performance.

### E. POPULARITY-BASED SCHEME

The packet loss ratio and the packet forwarding rate are also derived similarly to those of the sharding scheme demonstrated in Section V-B1. However, $\lambda_n$ and $\mu_n$ are different as a result of the use of both dispatching rules and the hash collisions of names. In other words, if a hash value of an unpopular name collides with one of the hash values of highly popular names, the random dispatching rule, rather than the sharding-based rule, is used by the unpopular name.

1) ARRIVAL PACKET RATE

$\lambda_n^I$ and $\lambda_n^D$ are derived similarly to the derivations demonstrated in Sections V-B2 and V-B3, respectively. However, $I_n(c)$ and $D_n(c)$ are derived differently from those of Eqs.(2) and (3), as described above.

$I_n(c)$ is defined for the four cases, depending on the following differences. First, if the CS hit rate of name $c$ is larger than $\theta$, Interest packets with name $c$ are dispatched according to the random dispatching rule. Therefore, $I_n(c)$ is $1/N$ because the thread is randomly selected among the $N$ threads. Second, if the CS hit rate of name $c$ is less than or equal to $\theta$, and if the hash value of name $c$ collides with one of the hash values of highly popular names, $I_n(c)$ is also $1/N$ for the same reason. Otherwise, Interest packets with name $c$ are dispatched according to the sharding-based dispatching rule. Therefore, $I_n(c)$ is either 1 or 0 depending on whether all the packets are dispatched to thread $n$ or not. $I_n(c)$ of Eq.(2) is rewritten as follows:

$$I_n(c) = \begin{cases} 
1/N & \text{if } \theta < p_n(c) \\
1/N & \text{elif hash of } c \text{ collides} \\
& \text{a hash of highly popular names} \\
1 & \text{elif } c \text{ is dispatched to } n \text{ (sharding)} \\
0 & \text{else (sharding).} 
\end{cases} \tag{11}$$

Note that, before deriving $I_n(c)$ and $D_n(c)$ for each name, the hash collisions of names are pre-computed in the following way. Assuming a uniform random hash function with hash length $H$, the hash values of all the names are calculated. Next, a set of hash values, each of which includes at least one name whose CS hit rate is larger than $\theta$, are collected. Finally, unpopular names whose hash values are included in the set are extracted.

Although the Data packet with name $c$ is always dispatched according to the sharding-based dispatching rule, $D_n(c)$ is derived differently from Eq.(3) as a result of the use of both the random and the sharding-based dispatching rules for Interest packets. If name $c$ employs the random dispatching rule for Interest packets, $D_n(c)$ is calculated as that of Eq.(9), similar to the mutual exclusion scheme. Otherwise, $D_n(c)$ is calculated as that of Eq.(3), similar to the sharding scheme. Note that the condition that name $c$ employs the random dispatching rule for Interest packets is determined similarly to the derivation of $I_n(c)$ of Eq.(11), as explained in the second paragraph of Section V-E1.

Note that the CS hit rate $p_n(c)$ is derived similarly to that in the case of the mutual exclusion scheme. In other words, $p_n(c)$ is expressed as $p_n(c) = p_m^N(c)$, where thread $m$ satisfies the condition $D_m(c) = 1$.

2) SERVICE RATE

$\mu_n$ is derived similarly to the derivation demonstrated in Section V-B3. However, $\phi_n^{l-\text{miss}}, \phi_n^{l-\text{hit}},$ and $\phi_n^{\text{Data}}$ are derived differently, as explained below. Because each name employs a different dispatching rule, i.e., the sharding-based dispatching rule or the random one, for Interest packets, mutual exclusion in the PIT is required for some names. However, it is not required for the others. This means that the number of CPU cycles spent handling one packet is different for each name. Given $C^l-\text{miss}(c), C^l-\text{hit}(c),$ and $C^{\text{Data}}(c)$ as $C^l-\text{miss}, C^l-\text{hit},$ and $C^{\text{Data}}$ for name $c$, $\phi_n^{l-\text{miss}}, \phi_n^{l-\text{hit}},$ and $\phi_n^{\text{Data}}$ of Eq.(6) are rewritten as follows:

$$\phi_n^{l-\text{miss}} = \sum_{c \in G} \lambda(c) I_n(1 - p_n(c)) C^l-\text{miss}(c),$$

$$\phi_n^{l-\text{hit}} = \sum_{c \in G} \lambda(c) I_n p_n(c) C^l-\text{hit}(c),$$

$$\phi_n^{\text{Data}} = \sum_{c \in G} \lambda(c) (1 - p_n(c)) D_n C^{\text{Data}}(c). \tag{12}$$

If name $c$ employs the random dispatching rule for Interest packets, packets with name $c$ require mutual exclusion in a PIT. In this case, $C^l-\text{miss}(c), C^l-\text{hit}(c),$ and $C^{\text{Data}}(c)$ are calculated as $C^l-\text{miss}, C^l-\text{hit},$ and $C^{\text{Data}}$ of Eq.(10), respectively. Otherwise, $C^l-\text{miss}(c), C^l-\text{hit}(c),$ and $C^{\text{Data}}(c)$ are calculated as $C^l-\text{miss}, C^l-\text{hit},$ and $C^{\text{Data}}$ of Eq.(7) because packets with name $c$ do not require such mutual exclusion. Note that the condition that name $c$ employs the random dispatching rule for Interest packets is determined similarly to the derivation of $I_n(c)$ of Eq.(11), as explained in the second paragraph of Section V-E1.

### VI. ANALYSIS RESULTS

This section evaluates the four schemes according to the analysis models: the popularity-based, sharding, mutual exclusion and ideal schemes. Hereinafter, we refer NDN routers with these four schemes as proposal, sharding, mutex, and ideal routers, respectively. We make the evaluation according to the following steps: First, we analyze the packet forwarding rates and packet loss ratios of all four routers. Second, we analyze the load of each thread and the mutual exclusion overhead to compare the proposal router with the sharding and mutex routers. Finally, we validate that the proposal router works well for various workloads and hardware settings.
A. SCENARIO SETTINGS

In this study, we consider a typical skewed workload. There are $1 \times 10^7$ unique Data packets with different names, and they are requested according to the Zipf distribution with the skewness $\alpha = 1.0$ [17]. As a hardware setting, an NDN router has a Xeon E5 2699 v4 CPU device with 2.2 GHz × 22 cores [27], and one thread runs on each CPU core. The total CS size of all CSs is $1 \times 10^4$ Data packets, which correspond to 0.1% of the total number of Data packets. The CS size is divided into each thread at the same size. In this case, the total CS hit rate of the ideal router is approximately 0.367. Each thread can hold up to 128 packets in its Rx queue. We assume that the data rate of NIC devices is not a bottleneck. The proposal router dispatches Interest packets according to the random dispatching rule for names whose CS hit rate is larger than 0.8. In other words, $\theta$ is 0.8.

The number of CPU cycles spent for each block $C_f$ in Fig. 1 is measured on an NDN router prototype built on top of the Data Plane Development Kit driver [28]. The prototype implements a CS, a PIT, and a FIB as a hash table [5] and a FIFO CS eviction algorithm. The CS and PIT sizes are $10^3$ and $10^5$, respectively, and the FIB holds $10^7$ different name prefixes. A Consumer sends Interest packets according to the Zipf distribution with the skewness $\alpha = 1.0$ for $10^3$ 1KB Data packets stored on a Producer. The NDN router prototype handles received packets with a single thread. For the hardware of the prototype, we use a computer using one Xeon E5 2699 v4 CPU [27], sixteen DDR4 64 GB DRAM devices, and two Intel XL710 40 Gbps NICs [19]. The number of CPU cycles is measured using the Read Time Stamp Counter instruction. The measured number for each block is summarized in Fig. 1. Note that the number of CPU cycles spent transmitting an Interest packet and that spent transmitting a Data packet are included in $B_d$ and $B_g$, respectively.

We use a name, rather than its hash value in the remainder of the section, except for Section VI-E3. In other words, we analyze the negative effects resulting from hash collisions in Section VI-E3.

B. PACKET FORWARDING RATES AND PACKET LOSS RATIOS

Here, we compare the packet forwarding rates and packet loss ratios of the four routers. Figure 8a shows the packet forwarding rates of the proposal, sharding, mutex, and ideal routers when the arrival Interest packet rate at the router is changed from $1.0 \times 10^6$ to $4.5 \times 10^7$ packet/s. The packet forwarding rate of the mutex router is smaller than the ideal rate of the ideal router by approximately 45.4%, while those of the other two routers are high even when the arrival Interest packet rate is low. Conversely, the proposal router achieves packet loss ratios similar to those of the ideal router.

Finally, we evaluate the packet forwarding rate under the condition that the packet loss ratio is equal to 0.1%. This is because the packet loss ratio in the Internet service is required to be 0.1% according to the International Telecommunication Union-Telecommunication Y.1541 [29]. Figure 8c shows the packet forwarding rates under the above condition. The packet forwarding rates of the proposal, sharding, and mutex routers are lower than the ideal rate of the ideal router by 2.9%, 31.6%, and 45.4%, respectively. The results demonstrate that the proposal router practically achieves high packet forwarding rates while simultaneously satisfying low packet loss ratios compared to the mutex and sharding routers.

Note that, in the remainder of this section, we use the condition where the packet loss ratio is 0.1%.

C. EQUAL LOAD BALANCING AND MUTUAL EXCLUSION OVERHEADS

This subsection validates that the popularity-based packet dispatching scheme achieves equal load balancing and small mutual exclusion overheads.

1) EQUAL LOAD BALANCING

Figure 9 shows the loads of all the threads for the proposal, sharding, and ideal routers when the arrival Interest packet rate at the router is $2.4 \times 10^7$ packet/s. The sharding router is selected as a reference as the objective of the popularity-based packet dispatching scheme is to resolve the load imbalance of the sharding scheme, as described in Section III-B. Note that the value of the arrival Interest packet rate is selected so that the packet loss ratio of the sharding router becomes 0.1%.
In the figure, the threads are numbered in the order of the load of a thread. Here, the number of CPU cycles that a thread spends for handling packets in one second is considered as the load.

The figure shows that the popularity-based packet dispatching scheme distributes the loads of highly loaded threads, such as threads #1 to #6 of the sharding router, to the other threads. Hence, the loads of all the threads become similar. Evidently, the difference between the proposal and sharding routers in terms of load equality affects the packet loss ratios. Many packets are dispatched to the highly loaded threads of the sharding router, and several packets are lost at the threads. Figure 10 shows the packet loss ratios of all the threads of the three routers. It is evident that such packet losses do not occur at the proposal router.

2) MUTUAL EXCLUSION OVERHEADS
A combination of the popularity-based packet dispatching scheme and optimistic locking reduces the frequency of CAS instruction executions. The frequencies of the proposal and mutex routers are 0.017 and 0.712, respectively. Note that such frequency is calculated as the average number of CAS instructions executed in the processing of one packet. Clearly, this reduction contributes to the improvement of the packet forwarding rate, as shown in Fig. 8a.

D. POPULARITY SKEWNESS
We compare the proposal and sharding routers in terms of load balance for two skewness values of popularity distributions of workloads. First, we create workloads by changing the popularity skewness $\alpha$ of the Zipf distribution because the skewness affects the load balance among threads [10]. We select 0.8 and 1.2 as the popularity skewness $\alpha$ according to measurement studies of content traffic [17], [30], [31]. Second, we compare the two types of routers that work as intermediate and edge routers while assuming that the routers work as edge routers thus far.

1) DIFFERENT SKEWNESS
Figures 11a and 11b show the packet forwarding rates for the cases of $\alpha = 1.2$ and $\alpha = 0.8$, respectively. Note that they represent the packet forwarding rates in the case that the packet loss ratio is 0.1%. The figures show that the proposal router achieves a higher packet forwarding rate than the sharding router in the case of $\alpha = 1.2$ by resolving the load imbalance of the sharding router. Meanwhile, the sharding router achieves a good load balance in the case of $\alpha = 0.8$, and thus, there is no room for improvement of the popularity-based packet dispatching scheme.

2) INTERMEDIATE ROUTER
The subsection evaluates the packet forwarding rates of an intermediate router in a line topology. Figure 12a shows the packet forwarding rates of the intermediate router. In Figure 12a, the packet forwarding rates of the proposal and sharding routers are similar and clear differences are not observed. This is because the popularity skewness is mitigated by CS hits of popular Interest packets at the edge router. Figure 12b shows the arrival Interest packet rates of the top-100 popular names at the edge and intermediate routers. Clearly, popularity skewness is mitigated at the intermediate router.
These results show the following two implications. First, spreading Interest packets with a few popular names is useful for edge routers where CS hits occur for popular names. Second, the sharding-based dispatching rule is sufficient for intermediate routers to achieve equal load balance.

E. VARIOUS HARDWARE SETTINGS

The evaluations up to this point use the same hardware setting; thus, this subsection evaluates packet forwarding rates under different hardware settings in terms of the number of threads, CS size, and hash table length at the NIC.

1) NUMBER OF THREADS

We change the number of threads from 4 to 56, assuming a state-of-the-art high-end CPU device with 56 CPU cores [32], while 22 threads are used for the evaluations thus far. Figure 13 shows the packet forwarding rates of the four routers. Note that arrival Interest packet rates are selected for all the settings so that the packet loss ratios become 0.1%.

![Figure 13. Packet forwarding rates in the case of 0.1% packet loss ratio.](image)

The proposal and ideal routers provide proportional packet forwarding rates to the number of threads, whereas the sharding and mutex routers do not do so. The reasons why the sharding and mutex routers do not work well are summarized below. For the sharding router, the larger the number of threads, the more significant the load imbalance among threads becomes. A similar phenomenon of load imbalance caused by the number of threads is reported in an existing study [10]. For the mutex router, the frequency of CAS instruction executions increases as the number of threads increases.

2) CS SIZE

The packet forwarding rates are evaluated for various CS sizes as CS hit rates may affect the performance of the popularity-based packet dispatching scheme. According to the caching studies [31], we adopt $10^3$, $10^4$, and $10^5$ packets as CS sizes, which correspond to 0.01%, 0.1%, and 1.0% of $10^7$ unique Data packets. Note that the average CS hit rates for CSs of $10^3$, $10^4$, and $10^5$ packets are 0.077, 0.367, and 0.687, respectively, in the case of the ideal router.

Figure 14 shows the packet forwarding rates of the four routers. The figure provides the following two observations:

The first observation is that the packet forwarding rate of the proposal router is less than that of the ideal router by approximately 10.0% when the CS size is $10^3$, which corresponds to 0.01% of the total number of Data packets. Such a small CS size results in low CS hit rates for popular names, and the low CS hit rates prevent the proposal router from spreading packets with popular names to threads. However, we believe that such a small CS size is not actually used in NDN networks.

The second observation is that a large CS size improves the packet forwarding rates of the proposal, ideal, and mutex routers. This is because the high CS hit rate resulting from the large CS size reduces the average packet processing time, thereby reducing the packet loss ratios of all the threads. In other words, reducing the packet loss ratio improves the packet forwarding rate. Note that the packet forwarding rate of the sharding router does not increase because the packet loss ratio of the highly loaded thread is not reduced even if the CS size increases. Specifically, the large CS size does not increase the CS hit rates of a few highly loaded threads, which cause most of the packet losses, whereas it increases the CS hit rates of the other lowly loaded threads.

3) NEGATIVE EFFECT OF HASH TABLE LENGTH

A PIT is indexed by names and the rule is set at each name in the evaluations up to this point. In other words, the hash values of names are not used for deciding the dispatching rules of names. However, modern NICs do not have sufficient memory to accommodate a large number of pairs of names and rules. As discussed in Section IV-E1, the PIT is indexed by the hash values of names. According to the maximum hash table length of commodity NICs [19], we change the hash table length of the NIC from 512 to 8192 entries.

Figure 15 shows the packet forwarding rates of the proposal router for various hash table lengths. The packet forwarding rates where names are used, rather than hash values, are shown as the references in the figure. The hash-based dispatching degrades the packet forwarding rate by 30.0% compared to the name-based dispatching when the hash table length is 512. However, the packet forwarding rate of the hash-based dispatching is similar to that of the name-based dispatching when the hash table length
modeled the packet forwarding rates and packet loss ratios. However, their focus is on the performance of a single-threaded router [22], [33], [34] or that of a multi-threaded router that assumes perfect load balance through sharding [2], [5]–[7]. In contrast, we break the assumption of a perfect load balance using sharding. Our study is one of the first studies that shed light on the performance tradeoff between sharding and mutual exclusion on multi-threaded NDN routers. Load imbalance due to sharding has been theoretically analyzed by Saino et al. [10]. They demonstrated the efficiency of a frontend cache [13], where a small frontend cache stores responses popular content objects, and chunking, where a popular content object is split into multiple chunks, for load balancing. Although the frontend cache is applicable to multi-threaded NDN routers, inter-thread communication between a frontend cache and a backend cache occurs. This overhead is not acceptable for fast NDN packet processing; thus, we did not utilize this approach. Although chunking assumes that chunks of the same content object are requested uniformly, this assumption does not hold in realistic Internet traffic [11], [12].

VIII. CONCLUSION

This study addresses load balancing while avoiding mutual exclusion on multi-threaded NDN routers. The key technique is a popularity-based packet dispatching scheme that spreads Interest packets with highly popular names to all threads while dispatching other packets through sharding. High loads resulting from Interest packets with highly popular names, which are root causes of load imbalance, are shared by all threads without mutual exclusion. We analytically modeled the packet forwarding rates and packet loss ratios of multi-threaded NDN router designs. The analytical performance study demonstrates that the proposed dispatching scheme achieves high packet forwarding rates and low packet loss ratios.

REFERENCES

[1] L. Zhang, “Named data networking,” SIGCOMM Comput. Commun. Rev., vol. 44, no. 3, pp. 66–73, Jul. 2014.
[2] R. B. Mansilha, L. Saino, M. P. Barcellos, M. Gallo, E. Leonardi, D. Perino, and D. Rossi, “Hierarchical content stores in high-speed ICN routers: Emulation and prototype implementation,” in Proc. 2nd Int. Conf. Information-Centric Netw., 2015, pp. 59–68.
[3] C. Yi, A. Afanasyev, J. Moiseenko, L. Wang, B. Zhang, and L. Zhang, “A case for stateful forwarding plane,” Comput. Commun., vol. 36, no. 7, pp. 779–791, Apr. 2013.
[4] J. McCauley, Y. Harchol, A. Panda, B. Raghavan, and S. Shenker, “Enabling a permanent revolution in Internet architecture,” in Proc. ACM SIGCOMM, Aug. 2019, pp. 1–14.
[5] W. So, A. Narayanan, and D. Oran, “Named data networking on a router: Fast and DoS-resistant forwarding with hash tables,” in Proc. Archit. Netw. Commun. Syst., Oct. 2013, pp. 215–226.
[6] D. Kirchner, R. Ferdous, R. L. Cigno, L. Maccari, M. Gallo, D. Perino, and L. Saino, “Augustus: A CCN router for programmable networks,” in Proc. Conf. 3rd ACM Conf. Inf.-Centric Netw., 2016, pp. 31–39.
[7] J. Takemasa, Y. Koizumi, and T. Hasegawa, “Toward an ideal NDN router on a commercial off-the-shelf computer,” in Proc. 4th ACM Conf. Inf.-Centric Netw., Sep. 2017, pp. 43–53.
[8] J. Takemasa, Y. Koizumi, and T. Hasegawa, “Data prefetch for fast NDN software routers based on hash table-based forwarding tables,” Comput. Netw., vol. 173, May 2020, Art. no. 107188.
[9] J. Takemasa, Y. Koizumi, and T. Hasegawa, “Analysis of mutual exclusion overhead of NDN packet forwarding on multi-core software routers,” in Proc. 5th ACM Conf. Inf.-Centric Netw., Sep. 2018, pp. 1–2.
[10] L. Saino, I. Psaras, and G. Pavlou, “Understanding sharded caching systems,” in Proc. 35th Annu. IEEE Int. Conf. Comput. Commun., Apr. 2016, pp. 1–9.
[11] A. Anand, C. Muthukrishnan, A. Akella, and R. Ramjee, “Redundancy in network traffic: Findings and implications,” in Proc. 11th Int. joint Conf. Meas. Model. Comput. Syst., 2009, pp. 37–48.
[12] S. Woo, E. Jeong, S. Park, J. Lee, S. Ihm, and K. Park, “Comparison of caching strategies in modern cellular backhaul networks,” in Proc. 11th Annu. Int. Conf. Mobile Syst., Appl., 2013, pp. 319–332.
[13] B. Fan, H. Lim, D. G. Andersen, and M. Kaminsky, “Small cache, big effect: Provable load balancing for randomly partitioned cluster services,” in Proc. 2nd ACM Symp. Cloud Comput., 2011, pp. 1–12.
[14] T. David and R. Guerraoui, “Concurrent search data structures can be blocking and practically wait-free,” in Proc. 28th ACM Symp. Parallelism Algorithms Archit., 2016, pp. 337–348.
[15] Intel 64 and IA-32 Architectures Optimization Reference Manual. Accessed: Sep. 5, 2019. [Online]. Available: http://www.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-optimization-manual.pdf
[16] K. Ohsumi, J. Takemasa, Y. Koizumi, T. Hasegawa, and I. Psaras, “Power consumption model of NDN-based multicore software router based on detailed protocol analysis,” IEEE J. Sel. Areas Commun., vol. 34, no. 5, pp. 1631–1644, May 2016.
[17] S. K. Fayazbaksh, Y. Lin, A. Tootoonchian, A. Ghodsi, T. Koponen, B. Maggs, K. C. Ng, V. Sekar, and S. Shenker, “Less pain, most of the gain: Incrementally deployable ICN,” in Proc. ACM SIGCOMM Conf., 2013, pp. 147–158.
[18] H. Kim, D. Han, D. G. Andersen, and M. Kaminsky, “MICA: A holistic approach to fast in-memory key-value storage,” in Proc. USENIX NSDI, Apr. 2014, pp. 429–444.
[19] Intel X710/XXV710/XL710 datasheet. (2020). [Online]. Available: https://www.intel.com/content/dam/www/public/us/en/documents/datasheets/xl710-10-40-controller-datasheet.pdf
[20] G. Carofiglio, M. Gallo, L. Muscariello, and D. Perino, “Pending interest table sizing in named data networking,” in Proc. 2nd Int. Conf. Inf.-Centric Netw., 2015, pp. 49–58.
[21] Z. Liu, R. Ben-Basat, G. Einziger, Y. Kassner, V. Braverman, R. Friedman, and V. Sekar, “Nitosketch: Robust and general sketch-based monitoring in software switches,” in Proc. ACM Special Interest Group Data Commun., Aug. 2019, pp. 334–350.

[22] J. Takemasa, K. Taniguchi, Y. Koizumi, and T. Hasegawa, “Identifying highly popular content improves forwarding speed of CDN software router,” in Proc. IEEE Globecom Workshops, Dec. 2016, pp. 1–6.

[23] G. Einziger, R. Friedman, and B. Manes, “TinyLFU: A highly efficient cache admission policy,” ACM Trans. Storage, vol. 13, no. 4, p. 35, 2017.

[24] N. Megiddo and D. S. Modha, “ARC: A self-tuning, low overhead replacement cache,” in Proc. USENIX FAST, Mar. 2003, pp. 115–130.

[25] M. Arlitt, L. Cherkasova, J. Dilley, R. Friedrich, and T. Jin, “Evaluating content management techniques for Web proxy caches,” ACM SIGMETRICS Perform. Eval. Rev., vol. 27, no. 4, pp. 3–11, Mar. 2000.

[26] V. Martina, M. Garetto, and E. Leonardi, “A unified approach to the performance analysis of caching systems,” in Proc. IEEE Conf. Comput. Commun., Apr. 2014, pp. 2040–2048.

[27] (2018). Intel processor E5-2699 v4. [Online]. Available: https://ark.intel.com/content/www/us/en/ark/products/91317/intel-xeon-p%20processor-e5-2699-v4-55m-cache-2-60-ghz.html

[28] (2017). Data Plane Development Kit (DPDK). [Online]. Available: http://www.intel.com/content/www/us/en/communications/data-plane-development-kit.html

[29] Network Performance Objectives for IP-Based Services, ITU-T, Y.1541, Dec. 2011.

[30] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker, “Web caching and zipf-like distributions: Evidence and implications,” in Proc. IEEE INFOCOM. Conf. Comput. Commun., Mar. 1999, pp. 126–134.

[31] Y. Sun, S. K. Fayaz, Y. Guo, V. Sekar, Y. Jin, M. A. Kaafar, and S. Uhlig, “Trace-driven analysis of ICN caching algorithms on Video-on-Demand workloads,” in Proc. 10th ACM Int. Conf. Emerg. Netw. Exp. Technol., 2014, pp. 363–376.

[32] (2018). Intel Platinum 9282 Processor. [Online]. Available: https://ark.intel.com/content/www/us/en/ark/products/194146/intel-xeon-p%20platinum-9282-processor-77m-cache-2-60-ghz.html

[33] T. Song, H. Yuan, P. Crowley, and B. Zhang, “Scalable name-based packet forwarding: From millions to billions,” in Proc. 2nd Int. Conf. Inf.-Centric Netw., 2015, pp. 19–28.

[34] H. Yuan and P. Crowley, “Scalable pending interest table design: From principles to practice,” in Proc. IEEE Conf. Comput. Commun., Apr. 2014, pp. 2049–2057.

JUNJI TAKEMASA: (Member, IEEE) received the master’s and Ph.D. degrees in information science from Osaka University, Japan, in 2016 and 2019, respectively. He is currently a Research Engineer with the Future Communication Architecture Laboratory, KDDI Research and Development Laboratories, Inc. His research interests include information centric networking, high-speed network systems, and green networking. He is a member of IEICE and IPSJ.

ATSUSHI TAGAMI: (Member, IEEE) received the M.E. and Ph.D. degrees in computer science from Kyushu University, Japan, in 1997 and 2010, respectively. He joined KDDI Corporation, in 1997, where he has been engaged in research and development on performance measurement of communication networks and overlay networking. He is currently a Senior Manager with the Future Communication System Laboratory, KDDI Research, Inc.

YUKI KOIZUMI: (Member, IEEE) received the master’s and Ph.D. degrees in information science from Osaka University, Japan, in 2006 and 2009, respectively. He is currently an Associate Professor with the Graduate School of Information Science and Technology, Osaka University. His research interests include information centric networking and mobile networking. He is a member of ACM and IEICE.

TORU HASEGAWA: (Member, IEEE) received the B.E., M.E., and Dr. Informatics degrees in information engineering from Kyoto University, Japan, in 1982, 1984, and 2000, respectively. After receiving the master’s degree, he worked as a Research Engineer at the KDDI Research and Development Laboratories (former KDD Research and Development Laboratories) for 29 years and moved to Osaka University. He is currently a Professor with the Graduate School of Information Science and Technology, Osaka University. He has published over 100 articles in peer-reviewed journals and international conference proceedings, including MobiCom, ICNP, the IEEE/ACM TRANSACTIONS ON NETWORKING, and Computer Communications. His current research interests include the future Internet, information centric networking, mobile computing, and so on. He is a Fellow of IPSJ and IEICE. He received the Meritorious Award on Radio of ARIB, in 2003; the Best Tutorial Paper Award, in 2014, from IEICE; and the Best Paper Award, in 2015, from IEICE. He has served on the program or organization committees of several networking conferences, such as ICNP, P2P, ICN, CloudNet, ICC, and Globecom. He served as a TPC Co-Chair of Testcom/Fates 2008, ICNP 2010, P2P 2011, and the Global Internet Symposium 2014.

* * *