IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received April 3, 2020, accepted April 24, 2020, date of publication April 28, 2020, date of current version May 13, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2991053

A Cache Invalidation Strategy Based on Publish/Subscribe for Named Data Networking

YUANZHI KAN[1,2], QUAN ZHENG[1,2,3], JIAN YANG[1,2], (Senior Member, IEEE), AND XIAOBIN TAN[1,2]

[1]Department of Automation, University of Science and Technology of China, Hefei 230022, China
[2]Laboratory for Future Networks, University of Science and Technology of China, Hefei 230022, China
[3]Institute of Advanced Technology, University of Science and Technology of China, Hefei 230088, China

Corresponding author: Quan Zheng (qzheng@ustc.edu.cn)

This work was supported by the CETC Joint Advanced Research Foundation under Grant 6141B08080101.

ABSTRACT Named Data Networking (NDN) aims to improve the efficiency of data delivery for the Internet. One of the typical characteristics of NDN is ubiquitous caching, that is to say, each network participant in NDN is capable of caching contents. This caching feature is beneficial for enhancing the data availability but also raises a problem of cache consistency. In this paper, we propose a novel strategy of cache invalidation, called PIOR (Proactive Invalidation with Optional Renewing), to provide strong consistency for NDN. PIOR is based on a lightweight publish/subscribe model, actively publishing the updated contents to the router nodes to guarantee the copy validity. We also conceive customized publish/subscribe rules to relieve the unbearable burden on the server imposed by the excessive publishing traffic. The advantage of PIOR lies in simple deployment and compatibility, since the invalidation process of PIOR is independent of the inherent process of NDN. We conduct extensive simulations over a real topology to evaluate the achievable performance of PIOR. The simulation results show that PIOR is able to achieve a high hit ratio and low server load at the low cost of network management.

INDEX TERMS Cache, consistency, invalidation, publish/subscribe, Named Data Networking.

I. INTRODUCTION

In the past decades, the major usage of the Internet has shifted from the information browsing to the content dissemination [1]. To achieve efficient content delivery, Information-Centric Networking (ICN), a modern Internet architecture, replaces the current host-centric communication model with the content-centric communication model [2]. As a promising approach to ICN, Named Data Networking (NDN) [3] provides an excellent foundation for data distribution systems. In NDN, communications are driven by consumers through interest packets and data packets. A consumer first sends a named interest packet for a desired content to the network. When a router receives this interest packet, the router first performs a Content Store (CS) lookup. If a match is found, it then is returned to the consumer. Otherwise, the router checks the Pending Interest Table (PIT). If a PIT entry with the same name is found, the router adds the incoming face of this interest packet to the in-record list of the matching PIT entry. Finally, if there is no matching entry in the PIT, the router creates a new PIT entry and then forwards this interest packet according to the Forwarding Information Base (FIB).

Throughout the process of data delivery in NDN, we can see that one of the fundamental features of NDN is in-network caching (CS). No matter the user terminals or the network infrastructures, all network participants in NDN have caches [4]. The routing node on the transmission path can cache the content passing by it, therefore the subsequent requests for the same content can be quickly responded by the nearest router. In-network caching, that has been widely employed by many computer systems, is a very useful technology to reduce the bandwidth usage over links, user-perceived delays and loads on the origin server [5].

However, due to the introduction of caching, how to guarantee the cache consistency has also attracted the attention of researchers. If, for example, a content is updated at the server, the copies of this content stored in caches may be inconsistent [6]. Hence, it is extremely important to deal with

The associate editor coordinating the review of this manuscript and approving it for publication was Jose Saldana.

the stale copies to ensure the copies received by users are valid.

Specifically, the solutions to cache consistency are classified into validation and invalidation [7]. Validation refers to an approach where the cache periodically checks the consistency of the cached copies with the original server. This solution can only provide a weak consistency because the content update could occur between two successive checks. Nevertheless, some scenarios, like real-time processing systems, require strong consistency that can be offered by invalidation.

Cache invalidation can be further divided into four basic schemes: (i) reactive invalidation: each time a request hits the cache, the cache will send an invalidation message to the server to verify the validity of the requested content [8]; (ii) proactive invalidation with removing: when a master content is updated at the server, the server will inform the caches to remove the stale copies of this content; (iii) proactive invalidation with renewing: when a master content is updated at the server, the server will push the latest content to the cache to replace the stale one; (iv) proactive invalidation with optional renewing: this scheme, as a mix of the second and the third ones, selects some contents to renew and the remains to remove. Popularity is often used as a criterion for selection in this scheme [9].

As copies of named contents are extensively distributed all over the in-network caches, cache consistency is a considerable challenge in NDN caching [6]. For reactive invalidation, each cache hit will yield a verification process, the number of which increases sharply with the scale-up of the network. Hence the application scenarios of this approach is limited in NDN. For proactive invalidation, the original server needs to maintain a list of caches that have obtained its contents [7]. Unfortunately, the information of this list, like address, cannot be provided inherently by NDN owing to the content-centric. Additionally, such a list must be maintained for every content [7], which brings significant overheads to the server. How to reduce these overheads also requires to be taken into account if the proactive invalidation is applied in NDN.

Another issue about proactive invalidation is how to be implemented. The communication model of NDN is a balanced model, which means one interest packet corresponds to one data packet [10] (single-interest single-data). The record of an interest packet that has been responded or has not been responded for a period of time will be removed from the PIT. However, the communication model of proactive invalidation is that one interest packet could correspond to multiple data packets (single-interest multiple-data). This contradiction here leads that the subsequent data packets in proactive invalidation could be dropped by the router due to the lack of the corresponding interest packet.

Naturally, an approach to the implementation of proactive invalidation is the publish/subscribe model in which a cache subscribes a time-sensitive content once and the server publishes the updated content multiple times. There have been several publish/subscribe models proposed for NDN in the literature [11]–[15]. Most of them are built on top of the NDN architecture to provide a publish/subscribe service fundamentally. Nevertheless, this kind of service is overqualified for the scenario of proactive invalidation with creating many new data structures and basic packets to the network. Besides, in the publish/subscribe model of proactive invalidation, the subscriber is the cache (or router) rather than the consumer (or user), and the published contents are only different versions of the same content rather than the hybrid of the different versions and contents in a separate publish/subscribe process. Thus, the traditional publish/subscribe model of NDN cannot satisfy the requirements of the proactive invalidation scenario.

In this paper, to solve the above issues, we develop a lightweight NDN publish/subscribe model, based on which we propose a cache invalidation strategy in the network layer of NDN, called PIOR (Proactive Invalidation with Optional Renewing). Several customized publish/subscribe rules are conceived in PIOR to reduce the overheads on publishing traffic and maintaining content status list. Only the caches that store the selected content copies can subscribe to the server, and only the selected contents can be published to the caches. The selected criteria are flexible and can be based on popularity, importance, etc.

The major contributions of our work are:

- We developed a lightweight publish/subscribe model for NDN. By adding control and track fields into the basic packets, this model is able to maintain the forwarding path without creating specialized PIT-like tables or semi-persistent interest packets.
- We propose a cache invalidation strategy, PIOR, based on the developed lightweight publish/subscribe model. In PIOR, the server actively publishes the updated contents to the router nodes to provide strong cache consistency.
- We conduct comprehensive simulation experiments via a real network topology and compare our PIOR strategy with several common cache consistency strategies including validation and invalidation. The experiment results show that PIOR can achieve a high hit ratio and low server load at the low cost of network management.

The rest of this paper is organized as follows. In the next section, we review the related work of cache invalidation and publish/subscribe in NDN. Section III describes the lightweight NDN publish/subscribe model and the PIOR algorithm. In section IV, we implement the PIOR algorithm and present simulation results. Finally, conclusions and future work are provided in Section V.

II. RELATED WORK

A. NDN PUBLISH/SUBSCRIBE MODELS

The research of the publish/subscribe is an active topic in NDN. One of the typical communication models in publish/subscribe is single-interest multiple-data [15], and several solutions to this model have been proposed.

Chen *et al.* [11] present a content oriented publish/subscribe system (COPSS) to provide efficient pub/sub-based content delivery for CCN. In COPSS a PIT-like table, Subscription Table (ST), is created to maintain the forwarding path of the data packet towards the subscribers. The record of the interest packet is stored in the ST to offer the outgoing face of the data packet for a long period of time. A compact version of COPSS for CCN-based Interest of Thing, COPSS-lite [12], also uses a PIT-like table to maintain above information.

Another solution to keep the outgoing face is using persistent interest packet. The Persistent Interests (PIs), proposed by Tsilopoulos and Xylomenos [13], is remained in the PIT until users explicitly unsubscribe or its lifetime expires. The Persistent Interest Packet (PIP), proposed by Nour *et al.* [14], is set with a persistence value, which is decremented with each data packet in the back-forwarding path. When the persistence value reaches 0, the record of the PIP is evicted from the PIT.

The Group-based Publisher-Subscriber (GbPS) architecture [15] uses both a dedicated table (Subscription Interest Table, SIT) and a semi-persistent interest packet (S-Interest) to offer the outgoing face of the data packet. And the publish/subscribe communication model is provided as an integrated part of NDN in GbPS.

It is noted that the above publish/subscribe models are built on top of the NDN architecture and aim to become fundamental communication services. Applying these models to the scenario of proactive invalidation is not straight forward, because the subscriber is shifted from the consumer to the cache. In addition, creating many new data structures for NDN to achieve the publish/subscribe model of proactive invalidation is a bit of overkill. Hence, a particular and lightweight publish/subscribe model is highly desired.

B. APPROACHES TO CACHE CONSISTENCY

It is acknowledged that caching generates massive copies of contents scattered throughout the network. If a content is updated at the server, the copies of this content stored elsewhere will become outdated. Without an special approach to renewing or removing the stale copies, these copies could be requested by unwitting users. As we mentioned above, there are two underlying approaches to guaranteeing cache consistency: validation and invalidation [7].

The cache validation can only provide weak consistency. The simplest validation scenario is the TTL-based (Time-To-Live) cache where each content is associated with a live time. The cached copy can be visited only when it leaves the server less than the live time. Moreover, according to the different settings of live time, the TTL-based can be divided into the fixed TTL [16] and the adaptive TTL [17].

Different from the cache validation, cache invalidation, usually classified into the reactive and the proactive, can provide strong consistency. However, there are extra overheads for invalidation on maintaining cache lists at the server and sending invalidation messages. Considering that the content copies are largely scattered across the in-network caches, these overheads could grow exponentially with the expansion of network scale [7]. Several invalidation scenarios for in-network caches also have been proposed such as Leases [18] for distributed file system, Piggyback server invalidation [19], IR-based cache invalidation [20], etc.

When providing cache consistency for NDN, the validation or invalidation strategy should take into consideration that NDN is content centric. Because the existing strategies of traditional in-network caches are based on TCP/IP network and the IP address is unavailable in NDN, these strategies cannot be applied directly. Version stamp or time stamp is commonly used in the design of cache consistency strategy for NDN.

Content Update Validation System (CUVS), proposed in [21], attaches a version number to each content name in ICN. A user can request for a specific version of content by adding the objective version number to the name of the interest packets. Nevertheless, the cost of distributing all contents' version numbers to all users and servers in advance is gigantic in CUSV. Feng *et al.* [22] propose a cost-effective Popularity-based Cache Consistency (PCC) mechanism, where two new types of packets, Query and Ack, are created to transfer controlling messages. The caches in PCC can spontaneously request for the updated version numbers from the server, which reduces the traffic of distributing the version numbers to the whole network. However, PCC will suffer from heavy overheads on transferring Query and Ack packets as the contents updating more frequently.

Both CUVS and PCC use version stamp to distinguish the different versions of the same content. Essentially, this approach is an extension of the TTL cache in which the version stamp that reaches the live time will be replaced by a new one. But in some scenarios, the version stamp could not be supported by the naming scheme of NDN and the consumers also might not know the latest version stamp when they just join the network.

By substituting version stamp with freshness in CCN, Quevedo *et al.* [23] propose the Consumer Driven Information Freshness Approach (CDIFA), in which the consumers are able to specify their desired content freshness in an optional field of the interest packet. When providing strong cache consistency, CDIFA is a Nocache-like strategy where all requests for the latest content will be forwarded to the server. This is because the freshness of the latest content is always 0 and the freshness of the content copies is always greater than 0. Therefore, the request response time in CDIFA increases.

As we can see, most of the current proposals for cache consistency in NDN either imposes significant overheads to network or increases request response time. Additionally, all these proposals are based on the version stamp [21], [22] or time stamp [23] (freshness is a kind of time stamp). Little work is based on the proactive invalidation. To the best of our knowledge, we are the first to propose a strategy of proactive invalidation for NDN, which can provide strong consistency with high performance and low cost.

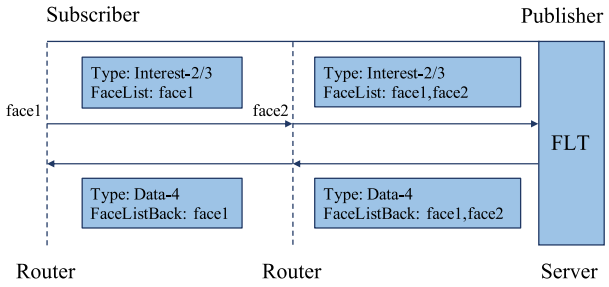


FIGURE 1. The lightweight publish/subscribe model.

III. PIOR: A CACHE INVALIDATION STRATEGY

In this section, we will introduce the full algorithm of the PIOR strategy. First, we present the lightweight publish/subscribe model and provide an overview of PIOR. Second, we give the data structures of the algorithm. Then we divide the whole algorithm into six parts to describe respectively. Finally, a thorough example is offered to show how the algorithm works.

A. THE LIGHTWEIGHT PUBLISH/SUBSCRIBE MODEL AND PIOR OVERVIEW

Fig. 1 shows the lightweight publish/subscribe model used by our PIOR strategy. The router (or cache) subscribes a time-sensitive content to the server by sending an interest-2 or interest-3 packet. At each router node along the forwarding path, the interest packet appends the current incoming face ID to its *FaceList* field. When receiving this interest packet, the server will record the *FaceList* field into the Face List Table (FLT for short) and then a subscribing process is done. After a period of time, there is a content updated in the server, which successively sends a data-4 packet according to the previously recorded *FaceList* field. When receiving this data packet, the router will update the outdated content cached before and then a publishing process is done.

By above processes, it is noted that the subscriber in the lightweight publish/subscribe model is the router and the published content is the latest version of the previously requested content. Besides, this model is packet-based which means the track of forwarding path is maintained in the packet. This design avoids creating a PIT-like table in the network layer. Thus, the implementation of the lightweight publish/subscribe model is much simpler than that of other existing publish/subscribe models.

Based on this model, PIOR is equipped with the subscribing and publishing processes, which work with other four processes (cleanup process, recording process, data coming process and interest coming process) to make up the whole algorithm. The relationship among the processes and the main algorithms of PIOR are illustrated in Fig. 2. The major characteristics of PIOR are summarized as follows.

- **Compatibility**: By tagging the contents that have requirements for validity, the PIOR process is independent of the original NDN process. The contents that have no requirements for validity can be requested normally.
- **Implementation simplicity**: PIOR requires no new types of packets. All controlling messages are transferred by the modified interest and data packets. The only new data structure added in PIOR is the FLT, relying on which the updated contents selected by some criteria can be actively published without changing the inherent forwarding mechanism of NDN.
- **Backtracking**: PIOR uses the *FaceList* field in the interest packet to track the upstream forwarding path, due to the lack of the notion ''address'' in NDN. When receiving an interest packet, the server records the information of *FaceList* in the FLT table, which provides the back-forwarding path to the updated contents.
- **Strong consistency**: With the entries in the FLT, the publishing process of PIOR updates the stale contents stored in the in-network caches. In addition, the cleanup process of PIOR ensures the tracking information of the contents stored in the caches is maintained in the FLT. In other words, if a content is no longer maintained in the FLT, the server will inform the caches to remove the copy of this content through the cleanup process. On the basis of these two processes, PIOR can guarantee the strong consistency in NDN.
- **High performance**: By publishing the updated contents, the requests for them can be responded on the in-network caches. Thus, PIOR can achieve a higher hit ratio and lower server load against other invalidation strategies, which has been demonstrated in our simulations.
- **Low cost**: The server in PIOR needs extra load to send controlling messages and to actively publish updated data packets. However, the subscribing mechanism can reduce the extra load by applying proactive invalidation only for the contents recorded in the FLT and the nodes forwarding these contents before. The simulation results also show that the proportion of the extra load in the total server load is pretty low in PIOR.

B. PACKET FORMATS AND FACE LIST TABLE

There are two basic NDN packet types, Interest and Data. In order to implement the algorithm of PIOR, we need add some new fields in original interest and data packets as shown in Fig. 3.

Fig. 3(a) is a modified interest packet. If a content requested by users has requirements for validity, the *InterestInvalidation* in the interest packet will be set to 1, otherwise to 0. *FaceList* is an array of incoming face IDs which keeps track of interest packets forwarded upstream toward the server.

Fig. 3(b) is a modified data packet. The function of *DataInvalidation* in the data packet is the same as that of *InterestInvalidation* in the interest packet to identify whether the content has requirements for validity. *FaceListBack* is also an

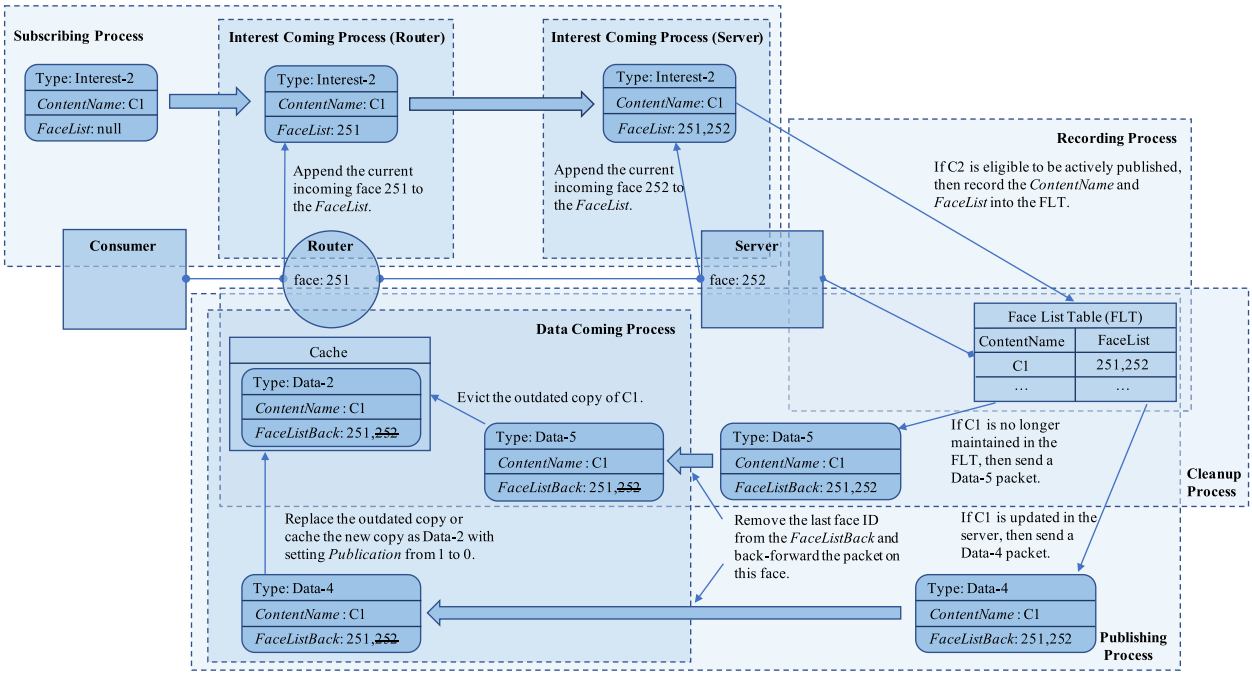


FIGURE 2. The relationship among the processes and the main algorithms of PIOR.

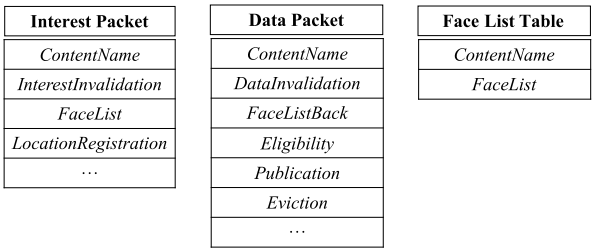


FIGURE 3. Packet formats and FLT (Face List Table).

TABLE 1. Interest packet types.

Type	InterestInvalidation	LocationRegistration
Interest-1	0	0
Interest-2	1	0
Interest-3	1	1

array of face IDs, on which the published data packets can be forwarded downstream to users. *Eligibility* is used to mark whether the requested content is eligible to be published (set to 1) or not (set to 0). *Publication* is used to identify whether the data packet is actively published (set to 1) or normally returned (set to 0). *Eviction* is used in the cleanup process. If a node receives a data packet with *Eviction*=0, the data of the same name cached in this node will be evicted.

By setting above new fields to different values, we obtain various types of interest packets and data packets which are listed in Table 1 and 2 respectively.

TABLE 2. Data packet types.

Type	DataInvalidation	Eligibility	Publication	Eviction
Data-1	0	0	0	0
Data-2	1	1	0	0
Data-3	1	0	0	0
Data-4	1	1	1	0
Data-5	1	1	1	1

Fig. 3(c) shows a new data structure, the FLT (Face List Table), which is created in the server and has three main functions: (i) storing the *ContentName* of the content which needs to be published actively, (ii) storing the track of interest packets forwarded upstream and aggregating the overlapping *FaceList*, and (iii) providing forwarding paths for the actively pushed data packets.

C. SUBSCRIBING PROCESS

In the subscribing process (Algorithm 1), the routers on the forwarding path of the interest packet subscribe a content with validity requirements to the server. As long as the content is updated and exists in the FLT, the server will push the latest content to these routers.

The user first requests for a content with validity by sending an interest packet of which the *InterestInvalidation* is set to 1. At each node on the interest packet forwarding path, if the cache is hit, the data packet will be returned immediately. Otherwise, the interest packet will be forwarded to the next node after appending the requesting face at this node to the *FaceList*. When the interest packet reaches the server,

Algorithm 1 Subscribing Process

initialize (a user sends an *Interest-2* packet)

```
1: while the current node is not the server do
2:   if hit the cache then
3:     return the Data-2 packet
4:   else
5:     if the previous node is not the user then
6:       append the requesting face to the FaceList of the
         interest packet
7:       forward the interest packet
8:     end if
9:   end if
10: end while
11: if the requested content is eligible to be actively pub-
    lished then
12:   activate the recording process
13:   prepare a Data-2 packet
14:   return the Data-2 packet
15: else
16:   prepare a Data-3 packet
17:   return the Data-3 packet
18: end if
```

Algorithm 2 Publishing Process

initialize (a content is updated at the server)

```
1: if the content name is in the Face List Table then
2:   prepare a Data-4 packet
3:   get the FaceList corresponding to this content in the
     Face List Table
4:   get the last face ID in the FaceList as the current
     forwarding face
5:   remove the last face ID from the FaceList and write the
     remainder in the FaceListBack of the data packet
6:   send the Data-4 packet on the face obtained in line 4
7: end if
8: while the length of FaceListBack is not 0 do
9:   if Publication=1 then
10:    if there is a content with the name in the CS then
11:      remove the stale data from the CS
12:    end if
13:    cache the new data's copy as Data-2 by setting
       Publication=0 of the copy
14:    get the last face ID in the FaceListBack as the current
       forwarding face
15:    remove the last face ID from the FaceListBack of
       the data packet
16:    forward the Data-4 packet on the face obtained in
       line 14
17:  else
18:    forward the Data-1 or Data-2 or Data-3 packet
       normally
19:  end if
20: end while
```

if the requested content is eligible to be actively pushed, the *ContentName* and *FaceList* will be inserted into the FLT and then a data packet with *DataInvalidation*=1, *Eligibility*=1, *Publication*=0, *Eviction*=0 and *FaceListBack*=null will be returned. Otherwise, the server only returns a data packet with setting above *Eligibility* to 0.

D. PUBLISHING PROCESS

Once a content with validity requirements is updated at the server, publishing process (Algorithm 2) will be activated. If the updated content is in the FLT (supposing the entry is *(C1, [251, 251, 252])*), the server will prepare a new data packet with *DataInvalidation*=1, *Eligibility*=1, *Publication*=1 and *Eviction*=0. Afterwards the last face (*252*) of the *FaceList* in the FLT is used as the current forwarding face. Removing the last face from the *FaceList*, we obtain a new face list (*[251, 251]*) which will be written into the *FaceListBack* of the data packet. Finally the updated data packet will be sent on face *252*.

At each node on the data packet back-forwarding path, if there is a content with the same name cached in the CS, the stale content will be removed. Then a new copy of this content will be cached after setting *Publication* to 0. Afterwards, similar to the forwarding process described above, we remove the last face from the *FaceListBack* and use it as the current forwarding face to send the data packet.

E. CLEANUP PROCESS

To avoid the FLT being too enormous to maintain, the FLT must be kept at a reasonable size. For examples, associate each entry with an expiration time, fix the size of the FLT or remove the content's eligibility for residing in the FLT. Evicting the entry from the FLT will activate the cleanup process (Algorithm 3). It can ensure that the nodes on the path of the *FaceList* do not cache the content which is no longer actively published by the server.

F. RECORDING PROCESS

In the recording process (Algorithm 4), the entries of contents that are eligible to be actively published are inserted into the FLT as (*ContentName*, *FaceList*). Moreover, We also merge the entries, the *FaceList* of which coincides at the tail, to reduce the load of the server actively publishing.

G. DATA COMING PROCESS

As mentioned above, we add four new fields in the data packet. Hence the node needs to handel five different types of the data packets (Table 2) separately in the data coming process (Algorithm 5):

- **Data-1** is the data packet without validity requirements and will be forwarded normally.
- **Data-2** is sent normally from the server and is eligible to be actively published. The router will cache it and forward it normally. In particular, if the Data-2 is

Algorithm 3 Cleanup Process

initialize (the FLT no longer maintains a content entry)

```
1: prepare a Data-5 packet with an empty data segment
2: get the FaceList corresponding to this content in the FLT
3: get the last face ID in the FaceList as the current forwarding face
4: remove the last face ID from the FaceList and write the remainder in the FaceListBack of the data packet
5: send the Data-5 packet on the face obtained in line 3
6: while the length of FaceListBack is not 0 do
7:    if Eviction=1 then
8:       if there is a content with the same name in the CS then
9:          remove the stale data from the CS
10:      end if
11:   end if
12:   get the last face ID in the FaceListBack as the current forwarding face
13:   remove the last face ID from the FaceListBack of the empty data packet
14:   forward the empty Data-5 packet on the face obtained in line 12
15: end while
```

Algorithm 4 Recording Process

initialize (an *interest-2* or *interest-3* packet arrives at the server and the requested content is eligible to be actively published)

```
1: if there is a content with the same name in the FLT then
2:    if the FaceList of the interest packet coincides with that of the FLT at the tail then
3:       insert the longer one in the FLT
4:       remove the original entry
5:    else
6:       insert the ContentName and FaceList in the FLT
7:    end if
8: else
9:    insert the ContentName and FaceList in the FLT
10: end if
```

Algorithm 5 Data Coming Process

initialize (a router receives a data packet)

```
1: if DataInvalidation=0 then
2:    forward the Data-1 packet normally
3: else
4:    if Publication=0 then
5:       if Eligibility=1 then
6:          if there is a content with the same name in the CS then
7:             request for this content by sending an Interest-3 packet
8:             remove the stale data from the CS
9:          end if
10:         cache the new data's copy
11:         forward the Data-2 packet normally
12:      else
13:         forward the Data-3 packet normally
14:      end if
15:   else
16:      if Eviction=0 then
17:         if there is a content with the same name in the CS then
18:            remove the stale data from the CS
19:         end if
20:         cache the new data's copy as Data-2 by setting Publication=0 of the copy
21:         forward the Data-4 packet according to the FaceListBack
22:      else
23:         if there is a content with the same name in the CS then
24:            remove the stale data from the CS
25:         end if
26:         forward the Data-5 packet according to the FaceListBack
27:      end if
28:   end if
29: end if
```

not cached in the current router before, this router will request for this content by sending an interest packet with *InterestInvalidation*=1, *LocationRegistration*=1 (*Interest-3*). Unlike the subscribing process, the routers on the forwarding path of *Interest-3* will not respond to it. When the interest packet reaches the server, the *FaceList* of it will be inserted into the FLT (see details in Section *Recording Process*). In other words, the location of the requesting router is registered in the server. The purpose of this operation is for the server to remember the routers where the content copies with the validity requirements are located. Therefore, the updated content can be pushed (see details in Section *Publishing Process*) and the copies of the content which is no longer maintained in the FLT can be evicted from the cache (see details in Section *Cleanup Process*) in time.

- **Data-3** is sent normally from the server and is not eligible to be actively published. The router will not cache it but will forward it normally.
- **Data-4** is published from the server. The router will cache it and forward it according to the *FaceListBack* (see details in section *Publishing Process*).
- **Data-5** is a type of data packet with an empty data segment actually. It is sent by the server to inform the router to remove its copies with the same name from the cache. The router forwards it also according to the *FaceListBack*.

Algorithm 6 Interest Coming Process (Router)

initialize (a router receives an interest packet)

```
if InterestInvalidation=0 then
    if there is a same name of the interest packet existing in the PIT then
        update the incoming faces list of the PIT entry accordingly
    else
        create a new PIT entry
        forward the Interest-1 packet
    end if
else
    if LocationRegisteration=0 then
        if there is a same name of the interest packet existing in the PIT then
            update the incoming faces list of the PIT entry accordingly
        else
            create a new PIT entry
            append the incoming face to the FaceList
            forward the Interest-2 packet
        end if
    else
        append the incoming face to the FaceList
        forward the Interest-3 packet
    end if
end if
```

Algorithm 7 Interest Coming Process (Server)

initialize (a server receives an interest packet)

```
if InterestInvalidation=0 then
    return a Data-1 packet
else
    if LocationRegisteration=0 then
        if the requested content is eligible to be actively published then
            activate the recording process
            return a Data-2 packet
        else
            return a Data-3 packet
        end if
    end if
else
    activate the recording process
end if
```

H. INTEREST COMING PROCESS

The interest packets are classified into three types (Table 1) in our algorithm:

- **Interest-1** is the interest packet without validity requirements and will be forwarded normally.
- **Interest-2** is initially sent by the user. Every time the interest packet arrives at a node, the incoming face of this node will be appended to the *FaceList* (see details in section *Subscribing Process*).
- **Interest-3** is initially sent by the router of which the location will be registered at the server. The interest packet also appends the incoming face of the current router to the *FaceList* on the forwarding path. Finally, the server will not respond to this interest packet, but will insert the *FaceList* of it in the FLT.

Due to the aggregation effect of the PIT, when an interest packet of *Interest-1* or *Interest-2* with a name that has been seen previously is received by the router, the PIT entry will be updated accordingly and the interest packet will not be forwarded. However, the interest packet of *Interest-3* does not yield the aggregation of the PIT and is forwarded after appending the incoming face to the *FaceList*. The details of the interest coming process for the router and server are presented in Algorithm 6 and 7 respectively.

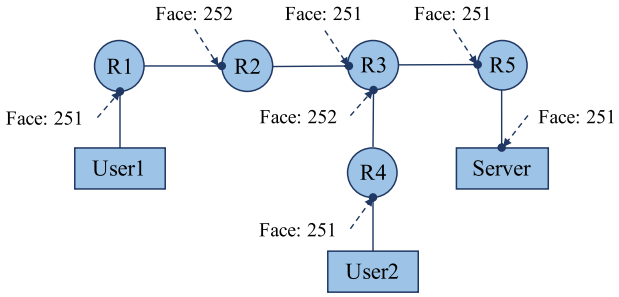


FIGURE 4. Example topology.

I. EXAMPLE

In order to illustrate our algorithm intuitively, we combine the six processes described above together through an example. Fig. 4 shows the example topology where the incoming face of each node is marked.

Supposing that User 1 sends an *Interest-2* packet to request for content C1 that is eligible to be actively published. When this packet reaches the server, the *FaceList* of it is [251, 252, 251, 251, 251] which will be recorded in the FLT with the name of the content as an entry (C1, [251, 252, 251, 251, 251]) (denoted as Entry-1). The server then sends a *Data-2* packet which will be cached on the each node of the back-forwarding path.

Once content C1 is updated at the server, a *Data-4* packet is published with initial *FaceListBack* [251, 252, 251, 251] on Face 251. Receiving this packet, Router 5 caches its copy, the *Publication* of which is set to 0 in the CS. Then this packet continue to be forwarded on Face 251 with the modified *FaceListBack* [251, 252, 251]. The same action will be repeated on Router 3, 2, 1 in turn until the data packet reaches User 1.

Shortly afterwards, User 2 also requests for content C1 by sending an *Interest-2* packet. According to the topology, this packet is responded on Router 3 which successively returns a

Data-2 packet. Because Router 4 did not cache C1 before, when receiving the *Data-2* packet, Router 4 will send an *Interest-3* packet to the server to register its location in the FLT. The *FaceList* of this interest packet is [252, 251, 251] which does not coincide with the *FaceList* of Entry-1 when arriving at the server. Thus a new FLT entry (C1, [252, 251, 251]) (denoted as Entry-2) is created.

When content C1 is updated at the server later, two *Data-4* packets will be published to Router 1 and 2 in accordance with the *FaceList* of Entry-1 and Entry-2 respectively. Similarly, the data packets of the above publishing process are replaced with *Data-5* packets in the cleanup process when content C1 is no longer maintained in the server.

IV. EXPERIMENTAL EVALUATION

To evaluate the performance of PIOR, we conduct enormous performance simulations using ndnSIM 2.1 [24]–[26], which is a widely used simulation platform for NDN. We also compare PIOR against other common invalidation strategies in terms of hit ratio, server load and proportion of extra load. Particularly, the last metric, extra load, represents the load that is actively published by the server in PIOR and Reactive Invalidation.

A. EXPERIMENTAL SETUP

Abilene network [27] is a high-performance backbone network, consisting of 11 nodes and 14 links. Fig. 5 shows the core topology of Abilene which is used in our simulations to make the simulation results close enough to reality.

A content server is installed on node 0. Each of nodes 0∼10 is connected to 5 users (55 users in total), and each user requests for 5000 contents with mean rate 4 req/s. The distribution of request probability follows the Zipf-Mandelbrot law [17] with α ranging from 0.2 to 1. The cache size of each node is the same and ranges from 50 to 150. The default cache replacement strategy is Least Recently Used (LRU) that is commonly applied in most of the cache systems [5]. The content update period is uniformly distributed over $[0, 2u]$, where u is ranging from 5s to 50s. The size of a data packet is 1024 bytes and the size of an interest packet (or signaling, e.g., *Data-5*) is 64 bytes.

The strategies to be compared with PIOR are: (i) Reactive Invalidation: Each time a request hits the cache, the cache will send an invalidation message to the server, which successively either returns a latest full content if the requested content is outdated or a validation signaling if the content is deemed up to date. (ii) Freshness: Freshness is a TTL-like strategy where each data packet is associated with a live time. This strategy is natively supported in NDN. (iii) Nocache: Nocache is the most direct way to ensure the contents obtained by the users are valid. The performance of Nocache is regarded as a baseline for the performance of other strategies in our experiments.

In addition, to simplify the complexity of implementation, we select the top N popular contents to be published actively in PIOR, and N is equal to the cache size in different scenarios.

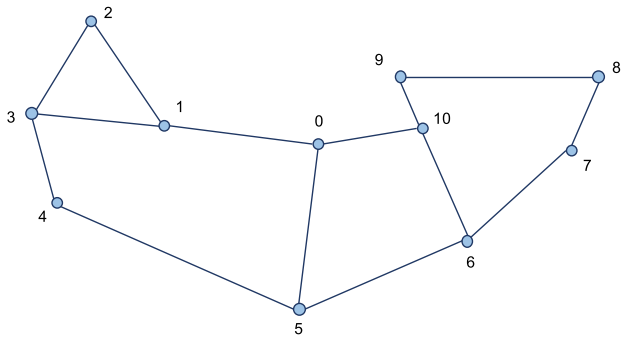


FIGURE 5. Abilene network topology.

B. HIT RATIO

The hit ratio here refers to the average hit ratio of the whole network. Given that all requests are forwarded directly to the server in Nocache, there is no metric of hit ratio in this strategy. Thus we only consider PIOR, Reactive Invalidation and Freshness in this section.

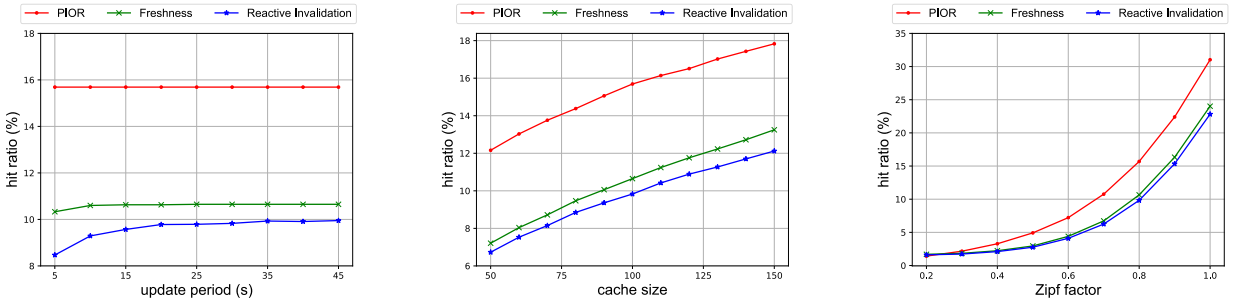
Fig. 6(a) shows the impact of different update period on hit ratio. It is noticed that the curve of PIOR is almost a straight line. That's because (i) once popular contents are updated, the server will publish them to caches immediately, and (ii) popular contents contribute more to the average hit ratio. For the same reason, on average, the hit ratio of PIOR is 64.52% and 49.68% higher than that of Reactive Invalidation and Freshness respectively.

Fig. 6(b) shows the impact of different cache size on hit ratio. With the increase of cache size, more contents are stored in caches, leading to the growth of all three curves. From Fig. 6(b), we can see that PIOR still maintains a large advantage on hit ratio, 59.75% and 47.97% higher than Reactive Invalidation and Freshness respectively on average.

Fig. 6(c) shows the impact of different Zipf factors on hit ratio. For large Zipf factors, PIOR still performs the best among all the strategies. However, when the Zipf factor is small, the curve of the content popularity flattens, resulting that popular contents cannot have a significant effect on the average hit ratio. Therefore, the advantage of PIOR is not obvious on hit ratio for small Zipf factors. Also from Fig. 6(c), we notice that the hit ratios of all three strategies are pretty low in the case of a small Zipf factor, because more contents have opportunities to be requested by users, making the contents in caches be replaced more frequently.

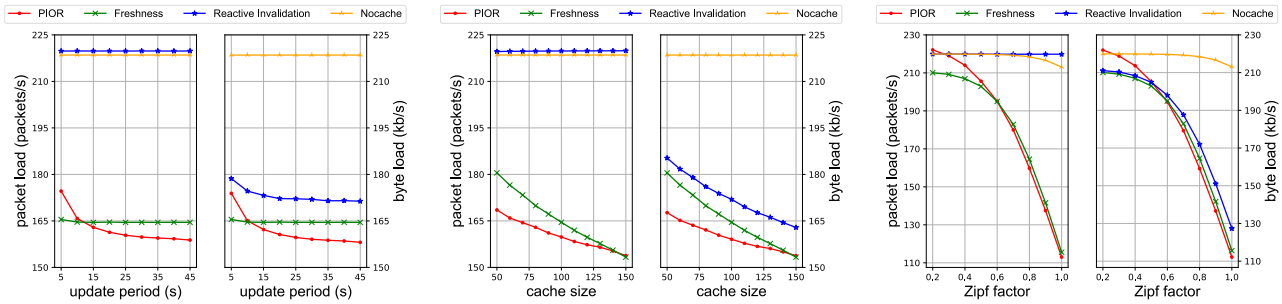
C. SERVER LOAD

Considering that the server in PIOR or Reactive Invalidation sends not only data packets but also signaling and the latter is much smaller than the former, we provide two indicators to measure the server load, one is the number of packets sent per second (denoted as packet load, packets/s), the other is the number of kilobytes transferred per second (denoted as byte load, kb/s).



(a) Fix cache size to 100 and Zipf factor to 0.8. (b) Fix update period to 30s and Zipf factor to 0.8. (c) Fix update period to 30s and cache size to 100.

FIGURE 6. Hit ratio vs update period / cache size / Zipf factor.



(a) Fix cache size to 100 and Zipf factor to 0.8. (b) Fix update period to 30s and Zipf factor to 0.8. (c) Fix update period to 30s and cache size to 100.

FIGURE 7. Server load vs update period / cache size / Zipf factor.

Most obviously in Fig. 7, the packet load curves are almost identical with the corresponding byte load curves, except for the curves of Reactive Invalidation. This is because that there is a large amount of signaling transferred between the nodes and the server for controlling communications in Reactive Invalidation, and the size of signaling is quite small. Hence the number of packets is pretty greater than the number of kilobytes numerically (given that one data packet is one kilobyte) in this strategy. Due to space limitations, we only consider the byte load in the following of this section.

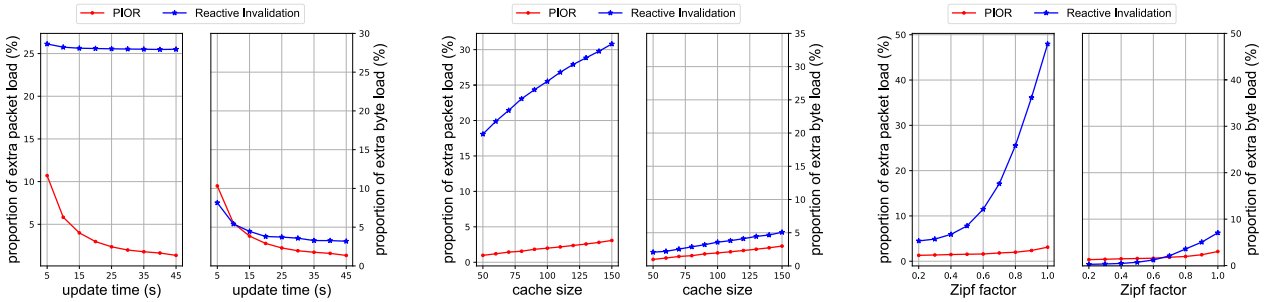
The curves of PIOR, Reactive Invalidation and Freshness decrease first and then flatten in Fig. 7(a). Since the server in PIOR publishes the contents at a higher frequency when content update period is small, the byte load of PIOR is 5.11% higher than that of Freshness, but 2.66% lower than that of Reactive Invalidation for update period of 5 seconds. However, as the update period increasing, the publishing frequency reduces, making the byte load of PIOR lower than that of Reactive Invalidation and Freshness. From Fig. 7(a), it is also noticed that the change range of the byte load for PIOR is larger than that for other strategies. In other words, the byte load of PIOR is more sensitive to changes in update period.

For the same reason mentioned in the analysis of Fig. 6(b), with the growth of the cache size, more requests are responded at caches, resulting the decline of the byte load as shown in Fig. 7(b). We also note that PIOR has a great advantage on byte load for small cache size. The maximum advantage over the second best strategy, Freshness, is around 7.11%. While for large cache size, the byte loads of PIOR and Freshness are almost the same but still 5.6% lower than that of Reactive Invalidation.

As shown in Fig. 7(a) and Fig. 7(b), due to the server responding to all requests directly in Nocache, the content update period and cache size have no effect on the byte load, the curves of which are straight lines.

From Fig. 7(c), it is seen that when the Zipf factor is small, PIOR has no advantages on server load in comparison to Reactive Invalidation and Freshness with the maximum disadvantage of 5.62%, and even has the same performance as Nocache. However, with the increasing of the Zipf factor, PIOR performs better than other strategies gradually.

Associating Fig. 7(c) with Fig. 6(c), now we can conclude that, PIOR does not outperform other strategies in terms of hit ratio and byte load when the Zipf factor is less than (or equal to) 0.2. Nevertheless, when the Zipf factor equals 0.3, the byte load of PIOR is 4.03% higher than that of Reactive Invalidation and Freshness on average, but the hit ratio of PIOR can achieve up to 17.39% improvement. When the Zipf factor equals 0.4, the above two data are 2.53% and c respectively. In other words, PIOR can obtain a great improvement of hit ratio with a low cost of byte load for not too small Zipf



(a) Fix cache size to 100 and Zipf factor to 0.8.

(b) Fix update period to 30s and Zipf factor to 0.8.

(c) Fix update period to 30s and cache size to 100.

FIGURE 8. Proportion of extra load vs update period / cache size / Zipf factor.

factor (larger than 0.2). Furthermore, when the Zipf factor is larger than 0.5, the popularity of hot contents become higher, yielding greater benefits of actively publishing these popular contents and reducing the cache replacement rate. Thus in this case, PIOR outperforms other strategies in terms of both hit ratio and byte load.

D. PROPORTION OF EXTRA LOAD

In this section, we introduce the extra load to measure the cost which is incurred by the server to guarantee the cache consistency. For PIOR, this cost contains the publishing load (*Data-4*) and the cleanup load (*Data-5*). For Reactive Invalidation, this cost represents the feedback load which is generated after the server receiving validation messages. For other strategies, there is no extra load.

Additionally, to better reflect the impact of extra load on the server, we use the proportion of extra load in total load as a substitute for the quantity of extra load. In general, the higher the proportion, the greater the cost for the server to maintain the cache consistency. Similar to the server load, we also separate the extra load into the extra packet load and the extra byte load.

From Fig. 8, it is apparent to see that the proportion of extra packet load for PIOR is much lower than that for Reactive Invalidation, and the average advantages are 86.79%, 92.06% and 72.95% in terms of update period, cache size and Zipf factor respectively. For Reactive Invalidation, only when the cache is hit, will the node send validation messages to the server, and a larger cache size as well as Zipf factor can improve the hit ratio. Thus, the proportion of extra packet load for Reactive Invalidation increases noticeably with the growth of the cache size and Zipf factor. On the contrary, for PIOR, only when the content is updated or stale, will the server publish the latest content or cleanup signaling. Thus, the proportion of extra packet load grows slowly as the cache size and Zipf factor increasing, while the proportion reduces noticeably as the update period increasing.

Fig. 8 also shows that the difference between the proportion of extra packet load and the proportion of extra byte load for PIOR is very small, while this difference for Reactive Invalidation is prominent. This indicates that the major packets in the extra packet load of PIOR is the actively published data packet, and the major packets in the extra packet load of Reactive Invalidation are the signaling.

For the proportion of extra byte load, we can see from Fig. 8 that PIOR keeps an edge over Reactive Invalidation in most conditions, except for small update period and small Zipf factors. However, the proportion of extra byte load for PIOR is low enough (3.28%, 1.96% and 1.81% on average in terms of update period, cache size and Zipf factor respectively), which has indicated that the cost of the server guaranteeing the cache consistency in PIOR is little.

V. CONCLUSION

In this work, we propose a proactive cache invalidation strategy, PIOR, to provide strong consistency for NDN based on a lightweight publish/subscribe model. The updated contents will be actively published to the nodes where the requests for these contents have been forwarded before. The FLT table and *Eligibility* field are used in PIOR to limit the flow size of the actively publishing, avoiding the server suffering from a heavy overhead of extra load. By tagging the contents which have requirements for validity, we also separate the invalidation process of PIOR from the inherent process of NDN, simplifying the PIOR deployment on NDN and making PIOR fully compatible with the original NDN mechanism. Finally, we evaluate the performance of PIOR in terms of hit ratio, server load and extra load against Freshness, Reactive Invalidation and Nocache. The simulation results demonstrate that PIOR outperforms other invalidation strategies and has a low overhead for the server.

Indeed, for PIOR, actively publishing popular (or important) contents can improve the hit ratio, but on the other hand it also increases the extra load of the server. How to balance the hit ratio and extra load by controlling the number of published contents and the size of the FLT table is the key to further enhancing the performance of PIOR. We will keep on investigating these issues in the future.

REFERENCES

[1] S. Borst, V. Gupta, and A. Walid, "Distributed caching algorithms for content distribution networks," in *Proc. IEEE INFOCOM*, Mar. 2010, pp. 1–9.

[2] G. Xylomenos, C. N. Ververidis, V. A. Siris, N. Fotiou, C. Tsilopoulos, X. Vasilakos, K. V. Katsaros, and G. C. Polyzos, "A survey of information-centric networking research," *IEEE Commun. Surveys Tuts.*, vol. 16, no. 2, pp. 1024–1049, 2nd Quart., 2014.
[3] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard, "Networking named content," in *Proc. 5th Int. Conf. Emerg. Netw. Exp. Technol.*, 2009, pp. 1–12.
[4] B. Ahlgren, C. Dannewitz, C. Imbrenda, D. Kutscher, and B. Ohlman, "A survey of information-centric networking," *IEEE Commun. Mag.*, vol. 50, no. 7, pp. 26–36, Jul. 2012.
[5] S. Podlipnig and L. Böszörmenyi, "A survey of Web cache replacement strategies," *ACM Comput. Surv.*, vol. 35, no. 4, pp. 374–398, Dec. 2003.
[6] I. U. Din, S. Hassan, M. K. Khan, M. Guizani, O. Ghazali, and A. Habbal, "Caching in information-centric networking: Strategies, challenges, and future research directions," *IEEE Commun. Surveys Tuts.*, vol. 20, no. 2, pp. 1443–1474, 2nd Quart., 2018.
[7] M. Rabinovich and O. Spatscheck, *Web Caching and Replication*, vol. 67. Boston, MA, USA: Addison-Wesley, 2002.
[8] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, "Hypertext transfer protocol–http/1.1," RFC Editor, Redmond, WA, USA, Tech. Rep. RFC2616, 1999.
[9] A. Iyengar, E. Nahum, A. Shaikh, and R. Tewari, "Enhancing Web performance," in *IFIP World Computer Congress*. Cham, Switzerland: Springer, 2002, pp. 95–126.
[10] L. Zhang, A. Afanasyev, J. Burke, V. Jacobson, K. C. Claffy, P. Crowley, C. Papadopoulos, L. Wang, and B. Zhang, "Named data networking," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 3, pp. 66–73, Jul. 2014.
[11] J. Chen, M. Arumaithurai, L. Jiao, X. Fu, and K. K. Ramakrishnan, "COPSS: An efficient content oriented publish/subscribe system," in *Proc. ACM/IEEE 7th Symp. Archit. Netw. Commun. Syst.*, Oct. 2011, pp. 99–110.
[12] H. Wang, S. Adhatarao, M. Arumaithurai, and X. Fu, "COPSS-lite: Lightweight ICN based Pub/Sub for IoT environments," 2017, *arXiv:1706.03695*. [Online]. Available: http://arxiv.org/abs/1706.03695
[13] C. Tsilopoulos and G. Xylomenos, "Supporting diverse traffic types in information centric networks," in *Proc. ACM SIGCOMM Workshop Inf.-Centric Netw. (ICN)*, 2011, pp. 13–18.
[14] B. Nour, K. Sharif, F. Li, and H. Moungla, "A distributed ICN-based IoT network architecture: An ambient assisted living application case study," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Dec. 2017, pp. 1–6.
[15] B. Nour, K. Sharif, F. Li, S. Yang, H. Moungla, and Y. Wang, "ICN publisher-subscriber models: Challenges and group-based communication," *IEEE Netw.*, vol. 33, no. 6, pp. 156–163, Nov. 2019.
[16] J. Jung, A. W. Berger, and H. Balakrishnan, "Modeling TTL-based Internet caches," in *Proc. IEEE 22nd Annu. Joint Conf. IEEE Comput. Commun. Soc. (INFOCOM)*, vol. 1, Jun. 2003, pp. 417–426.
[17] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker, "Web caching and zipf-like distributions: Evidence and implications," in *Proc. IEEE INFOCOM*, Jun. 1999, vol. 1, no. 1, pp. 126–134.
[18] C. Gray and D. Cheriton, *Leases: An Efficient Fault-Tolerant Mechanism for Distributed File Cache Consistency*, vol. 23, no. 5. New York, NY, USA: ACM, 1989.
[19] B. Krishnamurthy and C. E. Wills, "Piggyback server invalidation for proxy cache coherency," *Comput. Netw. ISDN Syst.*, vol. 30, nos. 1–7, pp. 185–193, Apr. 1998.
[20] G. Cao, "A scalable low-latency cache invalidation strategy for mobile environments," *IEEE Trans. Knowl. Data Eng.*, vol. 15, no. 5, pp. 1251–1265, Sep. 2003.
[21] Z. Feng, M. Xu, Y. Wang, and Q. Li, "An architecture for cache consistency support in information centric networking," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Dec. 2013, pp. 2126–2131.
[22] B. Feng, H. Zhou, H. Zhang, J. Jiang, and S. Yu, "A popularity-based cache consistency mechanism for information-centric networking," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Dec. 2016, pp. 1–6.
[23] J. Quevedo, D. Corujo, and R. Aguiar, "Consumer driven information freshness approach for content centric networking," in *Proc. IEEE Conf. Comput. Commun. Workshops (INFOCOM WKSHPS)*, Apr. 2014, pp. 482–487.
[24] A. Afanasyev, I. Moiseenko, and L. Zhang, "ndnSIM: NDN simulator for NS-3," Univ. California, Los Angeles, Los Angeles, CA, USA, Tech. Rep. NDN-0005, Oct. 2012. [Online]. Available: http://named-data.net/techreports.html
[25] S. Mastorakis, A. Afanasyev, I. Moiseenko, and L. Zhang, "ndnSIM 2: An updated NDN simulator for NS-3, Revision 2," Univ. California, Los Angeles, Los Angeles, CA, USA, Tech. Rep. NDN-0028, Nov. 2016.
[26] S. Mastorakis, A. Afanasyev, and L. Zhang, "On the evolution of ndnSIM: An open-source simulator for NDN experimentation," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 47, no. 3, pp. 19–33, Sep. 2017.
[27] *The Abilene Network Topology*. Accessed: Jan. 10, 2020. [Online]. Available: http://abilene.internet2.edu

YUANZHI KAN received the B.S. degree from the Department of Automation, University of Science and Technology of China (USTC), Hefei, in 2015, where he is currently pursuing the M.S. degree with the Department of Automation. His current main research direction is caching and modeling of NDN.

QUAN ZHENG received the B.S. degree in production process automation from the Dalian University of Technology, Dalian, China, in 1992, and the M.S. degree in automatic control theory and application and the Ph.D. degree in computer software and theory from the University of Science and Technology of China(USTC), Hefei, China, in 1995 and 2003, respectively.

He is currently an Associate Professor with the Department of Automation and the Deputy Director of the Laboratory for Future Networks. His research interests include video semantic retrieval, media content distribution, video quality detection, and future networks.

JIAN YANG (Senior Member, IEEE) received the B.S. and Ph.D. degrees from the University of Science and Technology of China (USTC), Hefei, China, in 2001 and 2006, respectively.

From 2006 to 2008, he was a Postdoctoral Scholar with the Department of Electronic Engineering and Information Science, USTC. Since 2008, he has been an Associate Professor with the Department of Automation, USTC. He is currently a Professor with the School of Information Science and Technology, USTC. His research interests include future networks, distributed system design, modeling and optimization, and multimedia over wired and wireless, and stochastic optimization. He was a recipient of the Lu Jia-Xi Young Talent Award from the Chinese Academy of Sciences, in 2009.

XIAOBIN TAN received the B.S and Ph.D. degrees from the University of Science and Technology of China (USTC), Hefei, China, in 1996 and 2003, respectively.

He is currently an Associate Professor with the School of Information Science and Technology, USTC. His research interests include network performance optimization and information security.