Computer Communications 000 (2016) 1–11

Contents lists available at ScienceDirect

Computer Communications

journal homepage: www.elsevier.com/locate/comcom

A smart routing scheme for named data networks

Qing Li[b], Zongyi Zhao[a,*], Mingwei Xu[a], Yong Jiang[b], Yuan Yang[a]

[a] Department of Computer Science & Technology, Tsinghua University, China
[b] Graduate School at Shenzhen, Tsinghua University, China

ARTICLE INFO

Article history:
Received 4 September 2015
Revised 8 September 2016
Accepted 23 September 2016
Available online xxx

Keywords:
Named data network
Routing
Caching

ABSTRACT

As the popularity of content-delivery applications (e.g., YouTube) grows, the inefficiency of transmission on the Internet has emerged, since in the TCP/IP networks, routers are unaware of the passing contents and the same content might be transmitted through the same path multiple times. To solve the problem as well as other problems, a lot of Information Centric Networking (ICN) architectures, including Named Data Networking (NDN), are proposed. These architectures enable the routers to identify and cache contents.

In this paper, we design an efficient and feasible scheme of caching and routing for NDN, i.e., Selective cAching and Dynamic rOuting (SADO). First, we propose a selective caching method that balances caching load among routers and reduces the caching redundancy dramatically. Then, we provide a scalable method to maintain routing information for the in-router cached contents. After that, a probabilistic forwarding method for requests is proposed to minimize the expected cost of fetching contents. Finally, we evaluate the performance of SADO by comprehensive simulations. The simulation results show that SADO achieves a decrease of 34% in the cost of fetching a content and a decrease of 81% in the load carried by the source servers, and a cached content is utilized 0.8 times on average.

1. Introduction

The current Internet architecture, proposed more than 40 years ago, is facing a lot of challenges. For example, a content may be transmitted through the same path multiple times if more than one client in the same stub area request the content simultaneously as the routers are unaware of the passing contents, leading to transmission inefficiency of the content-delivery applications (e.g., YouTube, P2P, etc.). Another example is that a personal webpage will be inaccessible through its original address if the webpage is moved to a new website as its owner is employed by a different company or university, although the webpage still exists in the Internet. The root cause of the problems is that in IP networks we request a content through its location while actually we do not care where the content is located at all. To solve the challenges, a lot of Information Centric Networking (ICN) architectures [1–6] are proposed, where the clients request a content by its name instead of its location and the routers can cache contents and respond to requests.

Named Data Networking (NDN) [6,7] is one of the widely accepted ICN architectures. There are two basic types of packets in NDN, i.e., data packet and interest packet. When a client wants to request a content, it will create an interest packet, write the name of the content into the interest packet, and send the interest packet to its delegate router. On receiving an interest packet, the router will examine its *content store* to determine if the requested content is available. If the content is available, it will return the content to the requester as a data packet. Otherwise, it will examine its *Pending Interest Table (PIT)*. If a PIT entry corresponding to the interest packet has existed in the PIT, it will append the incoming interface of the interest packet to the interface list of the PIT entry. If such a PIT entry does not exist, the router will add the name and incoming interface of the interest packet as a new PIT entry into the PIT and forward the interest packet using the information from *Forwarding Information Base (FIB)*. If the matching FIB entry is not available or all the upstream links are congested, the router will send a NACK packet [8] to the downstream node from which the interest packet is from. When a data packet arrives at a router, the router will cache the data packet into its content store if the caching method requires to do that, and forward the data packet through all the interfaces in the interface list of the matching PIT entry. Thus the data packet always travels in reverse the same path as the interest packet requesting it.

However, NDN has several issues to be addressed before it could be deployed. Two of the issues are which data packets should be cached and how to construct the FIB of a router, i.e., the caching method and the routing method. The primitive caching

* Corresponding author.
E-mail address: zhaozong16@mails.tsinghua.edu.cn (Z. Zhao).

http://dx.doi.org/10.1016/j.comcom.2016.09.012
0140-3664/

method in NDN requires a router to cache every data packet passing by [6], which means that a router will have to cache 1.16 GB of data per second in a network with the bandwidth of 10 Gbps. This method is therefore infeasible as it requires an excessively large content store capacity in a router to allow a data packet to have enough residence time in the content store, while the cached data packets, including the packets for the popular contents, will be evicted frequently as the content store capacity of a router is limited. On the other hand, the high caching redundancy resulted in by this caching method will waste a lot of valuable caching capacity. In the extreme case, suppose that there are 10 routers in a transmission path, and the content store of every router could accommodate 1000 data packets. We further assume that there is a single server and a single client only in the network, which are located at the two ends of the path respectively. Since every router will cache every data packet passing by, the data packets cached by every router will be the same. As a result, only 1000 different data packets are cached on the path, while the path could accommodate up to 10,000 data packets in total, which will undermine the hit ratio severely. The researchers have observed the phenomenon early, so they proposed to cache the data packets at the hot spots in the network where they will be utilized with high probability [9], or cache the data packets with popular contents in higher priority [10]. The newly proposed caching methods work well in specific scenarios, but their drawbacks are obvious. For example, by caching the data packets in the hot spots, the routers located in the hot spots will be overloaded while the content stores of other routers will be underutilized.

As for the routing methods, previous works (e.g., NLSR [11]) on this topic usually don't consider the routing for the cached data packets. Instead, they maintain routing information for the files stored in source servers permanently and route interest packets to the source servers of requested data packets. So the cache hit occurs only when the requested data packet happens to be cached in the routers on the path from the requester to the server. The reason that the previous routing methods avoid routing interest packets to cached data packets explicitly is that it's expensive for every router to notify other routers of the cached data packets as that will introduce significantly high communication overhead. Moreover, since the cached data packets are evicted frequently and the residence time that a data packet stays in the content store of a router is short, the notified information may be outdated shortly because of packet eviction. However, if a cache hit occurs only on the path from the requester to the server, the function of caches in the routers will be really limited. In the above example, there are 10 routers on the path from the client to the source server and every router could accommodate 1000 data packets, then the path could accommodate at most 10,000 different data packets even if the caching method is highly efficient and caching redundancy is eliminated completely. As the amount of data packets in the Internet is so large that the cache hit ratio is expected to be very low even if the most popular 10,000 data packets are cached on the path. Thus we argue that routing to the cached data packets intentionally is necessary to fully utilize the content stores of routers in the network and improve the cache hit ratio, but heavy communication between routers should be avoided.

The common drawback of the prior works is that they consider the caching method and routing method separately. For example, they assume that the routing method is there and ask how we could improve the cache hit ratio through an efficient caching method. Alternatively, they may assume that the caching method is the local policy of a router and ask how we could route an interest to some source, which is usually the source server of the requested data packet. However, in this paper we argue that the caching method and routing method should be considered as a whole, in order to fully utilize the cached data packets. Intuitively, by considering the caching method and the routing method as a whole, the router could know where a data packet will be cached even if it does not cache the packet. Later when an interest packet requesting the data packet arrives, it will forward the interest packet towards the cached data packet, improving the cache hit rate and reducing the cost of fetching a data packet.

In this paper, we propose a combined scheme, i.e., *Selective cAching and Dynamic rOuting (SADO)*, to solve the caching and routing problem in NDN. The design of SADO is based on two principles. The first is that to guarantee a relatively high utilization ratio of the cached data packets, the frequent eviction of data packets should be avoided and a cached data packet should have a relatively long residence time to stay in the content store, which requires to reduce the caching redundancy of data packets. Thus we cache a data packet in a single router when it is transmitted from its provider (e.g., a router, the source server) to a client, instead of in every router on its transmission path as in [6]. We adopt a probabilistic method to determine the caching router of a data packet. As the content store capacity of different routers may be different, and a router with larger content store capacity could accommodate more data packets, the caching method should allow the router with larger content store capacity to have a greater chance of caching a given data packet. On the contrary, a router with larger traffic flowing through will serve more data packets in a unit of time. To avoid overwhelming its content store, in our caching method this router should have a lesser chance of caching a given data packet.

The second is that we should maintain routing information for the cached data packets in a light-weight way while guaranteeing the accuracy of the routing information. Note that it's infeasible for a router to maintain routing information for every cached data packet in the network due to the significant communication overhead. Neither should the routers not maintain any routing information for the cached data packets at all and wait for the cache hit to happen on the path from the client to the source server, as this may result in the cache hit ratio to be low, which is stated before. A possible compromise is to allow a router to maintain routing information for the data packets passing by only, thus the communication overhead of maintaining routing information is avoided. Note that routers aim to reduce the cost of fetching data packets by maintaining routing information for the cached data packets. Since interest packets and data packets are always transmitted through a path which has relatively low cost, that a data packet is transmitted through a router implies that the router is located at a good path and it is easy to access the cached packet. By maintaining routing information for the data packets passing through, the router will maintain routing information for a large number of easy-to-access data packets around it, which helps to reduce the cost of fetching data packets.

We evaluate the performance of SADO by comprehensive simulations on tree topologies as well as a real network topology. The simulations show that SADO relieves the load borne by the source servers, reduces the cost of fetching a data packet, and improves the utilization ratio of cached data packets effectively.

In the paper we make the following contributions: First, we propose the idea of maintaining two separate routing tables, i.e., static routing table and dynamic routing table, for the data packets stored in source servers and cached in routers respectively, making the routing problem of NDN clearer. Second, we construct the dynamic routing table in a scalable way, where the accuracy of routing information is maintained without imposing extra load on the network. Third, we propose a caching method that balances the caching load among routers and reduces the caching redundancy effectively. Finally, we design a forwarding method that fully leverages the static routing information and dynamic routing information to minimize the cost of fetching a data packet probabilistically.

Please cite this article as: Q. Li et al., A smart routing scheme for named data networks, Computer Communications (2016), http://dx.doi.org/10.1016/j.comcom.2016.09.012

The remainder of the paper is organized as follows. At first we discuss the related work in Section 2. We introduce the framework of SADO in Section 3. In Section 4 we present the algorithm details of SADO, including the method of caching data packets and maintaining routing information for them as well as the method of forwarding interest packets. In Section 5, we evaluate the performance of SADO. We make a conclusion of this paper and introduce our future work in Section 6.

2. Related work

The topic of caching has been widely investigated even before NDN is proposed. In SAN (Storage Area Network), clients as well as arrays are equipped with cache memory. When a client requests a data object that is not in the client cache, the array cache or the array itself will supply the data object. When the request is completed, both the array cache and the client cache hold a copy of the data object. In fact, any data object in client cache exists in the array cache, if it is not evicted then, causing the waste of valuable cache capacity. Wong et al. [12] proposed DEMOTE to reduce the repetition between client cache and array cache. In DEMOTE, when a data object is transmitted from the array cache to the client cache, the copy of the data object in array cache is moved to the tail of the LRU (Least Recently Used) queue while another copy of the data object is inserted in the head of the LRU queue of the client cache, thus another insertion to the array cache may cause the data object to be evicted there. This behavior is similar to moving the data object from the array cache to the client cache. When a data object is evicted from the client cache, it is transmitted back to the array cache and inserted at the head of the LRU queue, rather than being discarded directly. The main principle of DEMOTE is to guarantee that only a single copy of a data object exists in the caches (including the array cache and client cache), making full use of the cache capacity. As DEMOTE achieves good performance in SAN, as shown in [12], it is not easy to adopt this method in NDN. In fact, it is necessary to maintain multiple copies of a data packet and allow clients to access the nearby copy in NDN, due to the large scale of the network. Moreover, a cache usually has a large number of down-stream caches in NDN, which is different from the assumption in [12] where an array cache corresponds to a single client cache. As a result, if a data packet is deleted from the upper-stream cache once it is cached in one of its down-stream caches, the requests for the data packet from other down-stream caches will be missed, introducing large penalty in performance.

LCD (Leave Copy Down) [13] is another caching method proposed to reduce the repetition between upper-stream caches and down-stream caches. LCD overcomes the drawback of DEMOTE so that it deals well with the scenario where an upper-stream cache corresponds to multiple down-stream caches. While the de facto caching method for the multi-level cache system is to cache a data object in every cache through which the data object travels, LCD caches only a copy of the data object in the immediate down-stream cache of the hit cache. LCD allows a popular data object to approach the clients as it is requested frequently, while the data objects that are less popular are limited in a few caches. This caching method can be easily modified and applied to NDN. However, a multi-level cache system is different from NDN, where each router has caching capability and acts as a cache. In the multi-level cache system, the caches are deployed near the source server, thus all the requests will arrive at the cache system finally and traverse through some of the caches, which means that every cache will service requests if its down-stream caches don't hold the requested data objects. So the target of LCD is to maximize the aggregate capacity of the cache system by reducing the repetition between caches. On the contrary, caches are everywhere in NDN, and a cached data packet may not be accessed again even if it is popular. Thus what is most desirable in NDN is to increase the probability that cached data packets are utilized, while reducing the repetition between caches is still an important consideration.

As there have been sufficient works about caching, researchers draw a lot of lessons from the existing solutions when designing the caching methods for NDN. The primary caching method in NDN is to allow a router to cache every data packet passing by [6]. As many papers stated [9,14,15], this blind caching method results in the cached data packets, including the packets with popular contents, to be evicted frequently. Moreover, the high caching redundancy wastes a lot of valuable in-network storage. Thus this caching method is inefficient in the sense of improving the cache hit ratio and the utilization of in-network storage. As the drawback of this caching method has long been observed, many advanced methods have been proposed.

As some contents may be more popular than others in real networks, making this kind of contents widely available in the network will provide convenience to the end users. With this in mind, WAVE [10] proposed to cache the popular contents preferentially. Specifically the data provider will set the *cache suggestion* bit of the data packet before sending it to the requester, where the data provider is the source server or a router that can supply the data packet. On receiving a data packet with the cache suggestion bit set, a router will cache the data packet and reset the cache suggestion bit to 0, thus down-stream routers will not cache the data packet again. So the more frequently a data packet is requested, the more copies of it will be cached in the network, and the closer to the clients the copies will be, thus the cost of fetching the popular contents will be reduced. Obviously, WAVE is a NDN-version of LCD, so it shares the drawbacks of LCD as discussed above, i.e., as it reduces the repetition between content stores of routers, the probability that a cached data packet is utilized may be low due to the large scale of NDN and the widespread distribution of caches in NDN.

One method to increase the probability that cached data packets are utilized is to cache the data packets in the "hot" routers where interest packets pass by frequently. Goh et al. [16,17] reported that the load borne by a node in shortest path based network is closely related to the betweenness of the node, where the betweenness of a node is defined as the number of shortest paths between any two nodes of the network that pass through the node. Everett and Borgatti [18] further found that the betweenness of a node in real networks is highly correlated with its betweenness in its ego network, where the ego network of a node consists of this node and all its neighboring nodes as well as all the links among them. As the ego network betweenness of a node could be computed locally, it is a good measure of importance of this node. Based on this observation, Chai et al. [9] proposed to cache a data packet in the router(s) with the highest ego network betweenness on its transmission path, to maximize the probability that the cached data packets are utilized. However, this method will make a small handful of routers with high ego betweenness overloaded, while the content stores of a lot of routers with relatively low ego betweenness are not fully utilized, resulting in the waste of valuable in-network storage.

Notice that in-network storage near the source server may be shared by more flows than that near the clients. ProbCache [15] suggests to cache data packets in the routers closer to the clients with higher probability, thus the in-network storage closer to the source server can be utilized by the clients near the source server. ProbCache determines the number of copies of a given data packet that should be cached on its transmission path based on the size of in-network storage of the path and the traffic flowing through this path. Moreover, ProbCache calculates another factor, CacheWeight, which increases and approaches 1 as the data packet

Please cite this article as: Q. Li et al., A smart routing scheme for named data networks, Computer Communications (2016), http://dx.doi.org/10.1016/j.comcom.2016.09.012

gets closer to the requester. The possibility that a data packet is cached in a router is the product of the number of copies and the value of CacheWeight. The weakness of ProbCache is that it's hard to estimate the traffic flowing through a path. ProbCache simply assumes that the traffic flowing through a router within 1 sec is equal to the content store capacity of the router. Notice that the content store capacity of a router is fixed, while the traffic flowing through this router is highly related to its location in the network and it varies from time to time. Thus the assumption of traffic flowing through a router in ProbCache is overly simplified. Another drawback of ProbCache lies in its design principle, i.e., preferring to caching data packets in the routers near clients, since the closer to clients a data packet is cached, the smaller the probability is that the data packet is utilized by other clients.

Previous works about routing in NDN usually avoid routing interest packets towards the data packets cached in routers. Instead, interest packets are always forwarded towards the source servers supplying the requested data packets. The routing framework sketched in [6] suggests that any routing method working well with IP networks should also work well with NDN since the NDN forwarding model is a superset of the IP model. For example, the names of data packets in NDN are hierarchical and can be aggregated, which is similar to the IP address, so the longest prefix matching is suitable for NDN as well. Moreover, as looping of interest packets is naturally prevented, an interest packet can be forwarded through multiple interfaces in NDN while a packet can be forwarded through a single interface only in IP networks, so the routing methods are expected to be more efficient than that in IP networks.

As [6] provides a solution framework for the routing problem in NDN, more work is needed to make the routing a reality. The first NDN routing protocol is OSPFN [19], which is an extension of OSPF. But this simple adaption of an IP routing protocol brings in many drawbacks, including the management of IP address space and the support for multi-hop forwarding [20]. So another link state routing protocol, NLSR [11], is proposed. In NLSR, a router will send "info" interest packet periodically to all of its neighboring routers. The neighboring routers will respond to this info packet to indicate that they are still alive. If there is no response for a few info packets, the neighboring router is considered down. There are two types of Link State Advertisements (LSAs), i.e., Adjacency LSA and Prefix LSA, to maintain the information for the links connecting this router and all the neighboring routers and the information for the prefixes registered to this router respectively. The LSAs are stored in a Link State Database (LSDB) and CCNx synchronization protocol, Sync [21], is used to synchronize the LSDBs between neighboring routers. Finally, for each router, NLSR removes all the neighboring links but one and then runs the Dijkstra algorithm to compute the FIB. This process is performed for each neighboring link and multiple next-hops for each destination node are produced.

While the forwarding plane in IP networks is stateless and packets are simply forwarded through the interface associated with the matching FIB entry, the forwarding plane in NDN is stateful (e.g., PIT records the names and incoming interfaces of interest packets) and more complicated forwarding strategy is needed. In [8] authors defined a new type of packet, i.e., NACK packet. When a router receives an interest packet from a down-stream router, if an interest packet with the same name and nonce has been received before and thus it is a duplicate interest packet, or the upper-stream router foresees that congestion will occur, or the upper-stream router has no available interface to forward the interest packet, it will return a NACK packet to the down-stream router. The authors further proposed a coloring scheme for the interfaces in FIB entries. The initial status of an interface is yellow. When data packets arrive through this interface in probing process, this interface is marked as green. When a pending interest packet on this interface times out, or a NACK packet with the code of "Duplicate" or "No Data" is received through this interface, this interface is marked as yellow. An interface is marked as red when it goes down. When transmitting interest packets, green interfaces are always preferred over the yellow interfaces and red interfaces are never selected.

Table 1
The structure of static routing table.

$Prefix_1$	$(face_1, dist_1)$, $(face_2, dist_2)$, ...
$Prefix_2$	$(face_3, dist_3)$, $(face_4, dist_4)$, ...

Table 2
The structure of dynamic routing table.

$Prefix_1$	$(face_1, dist_1, number_1)$, $(face_2, dist_2, number_2)$, ...
$Prefix_2$	$(face_3, dist_3, number_3)$, $(face_4, dist_4, number_4)$, ...

Since the routing methods and forwarding model presented above work well and are perpendicular to SADO, we won't discuss them deeper in this paper. Instead, observing that the existing caching methods and routing methods avoid performing routing around the cached data packets, we will try to consider the caching and routing as a whole and forward interest packets towards the cached data packets explicitly, which is heavily dependent on the existing routing methods and forwarding models.

3. Framework of SADO

Currently SADO focuses on the file distribution applications (e.g., YouTube) only, and we leave the research on more complicated applications as our future work. The file distribution applications provide the convenience that the size of the file to be transmitted is known a prior, which is important to SADO. As a starting point we will introduce the naming scheme of SADO at first. A data packet in NDN is named after the pattern of "< file-name> /< pkt-num> /< pkt-seq> ", where < file-name>, which identifies the file to be transmitted, could be in arbitrary form, < pkt-num> is the number of data packets the source file is segmented into, and < pkt-seq> is the sequence number of this data packet. For example, a data packet name of "ndn://youtube/transformer/1000/50" indicates that the data packet's source file is "ndn://youtube/transformer", the source file is segmented into 1000 data packets and this data packet has sequence number of 50. An interest packet has the same name as its corresponding data packet.

There are two routing tables, i.e., *static routing table* and *dynamic routing table*, in a router. As shown in Table 1, the static routing table is the same as that introduced in [6], i.e., each entry corresponds to a prefix and may contain multiple interfaces. There is a *distance*, which is the number of hops from this router to the destination server, associated with each interface. Maybe there are other tags such as color [8] associated with each interface to help to select the interface to forward interest packets, but this is out of the scope of this paper. The static routing table could be generated through the mature routing methods in NDN (e.g., NLSR [11]). The structure of the dynamic routing table is shown in Table 2. Each entry of the dynamic routing table corresponds to an extended prefix (or E-Prefix for short) which is the "< file-name> /< pkt-num> " part of the corresponding data packet name, thus a dynamic routing entry corresponds to a file instead of a single data packet. The information associated with the extended prefix consists of triples in the form of *(face, distance, number)*, where *face* indicates through which face the cached data packets can be reached, *distance* is the distance from current router to the caching

Please cite this article as: Q. Li et al., A smart routing scheme for named data networks, Computer Communications (2016), http://dx.doi.org/10.1016/j.comcom.2016.09.012

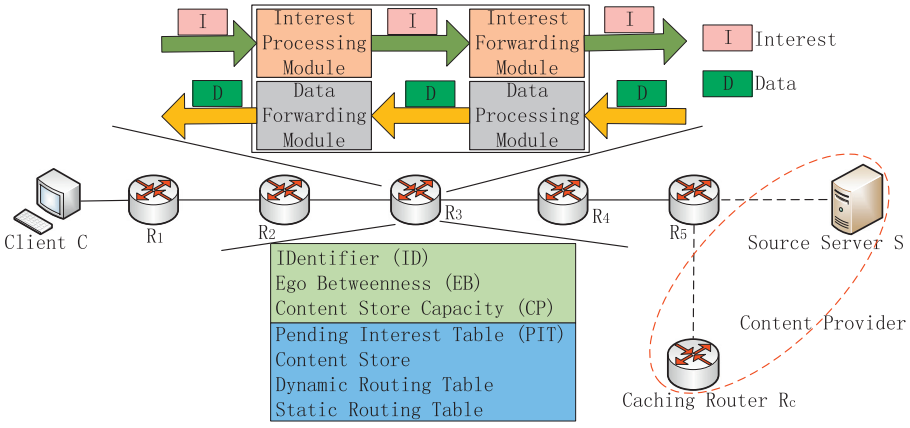

Fig. 1. Framework of SADO.

router (the router in which the data packets are cached), and *number* indicates how many data packets with this prefix are cached there.

In SADO when an interest packet is transmitted from the client to the content provider (the source server or a caching router) it will select a single router to be the caching router of the response data packet, and when the response data packet is transmitted back to the client, it will be cached in this caching router and all the other routers on the transmission path will maintain routing information for the cached copy. To achieve this, we extend the headers of data packet and interest packet to record some essential information which will be explained in Section 4. As Fig. 1 shows, when an interest packet arrives at a router, it will first be processed by the *interest processing module*. This module determines whether current router should be selected to be the new caching router of the response data packet, based on the caching information in the interest packet and this router's properties. If current router is selected, this module will update the caching information in the interest packet to record the decision. Then the interest packet is passed to the *interest forwarding module*. This module queries the static routing table as well as the dynamic routing table and selects a face based on the returned information to forward the interest packet. At the beginning stage, the dynamic routing table is empty, thus the module will forward the interest packet based on the static routing information solely, which routes the interest packet to the source server. Later, as dynamic routing information is collected, this module will leverage the static routing information and dynamic routing information to select the best face to forward the interest packet. In the case where dynamic routing information is present, the interest packet may be forwarded to a caching router instead of the source server. For example, in Fig. 1, if the interest forwarding module of R_5 finds that the face to the caching router (R_c) is a better choice than the face to the source server (S), it will forward the interest packet to R_c.

When the interest packet finally arrives at the content provider, the provider will copy the caching information from the interest packet to the response data packet and send the data packet back to the client. When the response data packet arrives at a router, it is first processed by the *data processing module*. If the caching information in the data packet indicates that current router is the caching router selected previously, this module will cache the data packet. Otherwise, the module will be able to infer where the data packet will be cached (or has been cached) and calculate the distance from current router to the caching router, from the caching information. Thus the module will update the dynamic routing table accordingly. Then the data packet is passed to the *data forwarding module*, where the data packet is forwarded using the information in the matching PIT entry. Since the function of data forwarding module is largely the same as that in the original NDN, we won't discuss it deeper.

Notice that a router maintains dynamic routing information for the data packets passing through it only. If some data packets are cached on the nearby routers but have not been transmitted through this router, this router won't be able to utilize the cached copies. As this may result in inefficiency to some extent, it's a reasonable comprise since utilizing all the packets cached on nearby routers may require flooding, which will introduce a lot of load to the network. Moreover, even if this router cannot utilize the packets cached on nearby routers, it's likely that the upstream routers are able to utilize them.

To complete the routing task, there are three challenges to address in SADO: 1) How do the interest processing modules on the transmission path collaborate with each other to select the caching router of the response data packet, thus the caching load among the routers is balanced? 2) How to record the caching information so that the caching router can cache the response data packet while the other routers can maintain routing information for the cached copy? 3) What is the "best face" to forward an interest packet and how to leverage the dynamic routing information and static routing information to select the best face?

4. Algorithm details

This section explains the functions of interest processing module, data processing module and interest forwarding module in Fig. 1. When an interest packet is transmitted from the client to the data provider, it will collect relevant information about the transmission path, which is completed by the interest processing modules of the routers; when the response data packet is transmitted from the data provider to the client, the routers will cache the data packet or maintain dynamic routing information for the cached copy of the data packet, which is completed by the data processing module of the routers. The data forwarding module will leverage the dynamic routing information as well as the static routing information to forward the interest packet.

4.1. Collect information

The method of determining the caching router in SADO integrates the ideas proposed in [9] and [22]. As shown in Fig. 1, a router has three properties, i.e., *identifier (ID), ego betweenness (EB)* and *content store capacity (CP)*. According to [18], the ego network betweenness centrality (or ego betweenness in short) of a router in real, random networks is highly correlated with the between-

Please cite this article as: Q. Li et al., A smart routing scheme for named data networks, Computer Communications (2016), http://dx.doi.org/10.1016/j.comcom.2016.09.012

Table 3
The extra fields in interest packet header and data packet header.

Field	Interest packet	Data packet
URD	Distance from client to current router	Distance from client to current router
ARD	Distance from client to temporary caching router	Distance from client to ultimate caching router
WT	Weight of the temporary caching router	–
HV	Hash value of router weight and packet prefix	–

ness centrality of the router in the whole network, where the betweenness centrality of a router is defined as the number of shortest paths between any two routers passing through this router.[16] and [17] further reported that the load borne by a node in a shortest path based transport network is closely related to the betweenness centrality of the node. So the ego betweenness of a router is a good measure of the traffic flowing through this router. Note that the ego betweenness of a router is related to the connections among this router and its neighbors only, so it is static and can be computed locally. To balance the caching load among routers, we define the *weight* of a router as the ratio of its content store capacity to its ego betweenness, and a router with larger weight should have a greater chance of caching a given data packet.

As shown in Table 3, we add four extra fields into the interest packet header, i.e., *URD* (cURrent Router Distance), *ARD* (cAching Router Distance), *WT* (WeighT) and *HV* (Hash Value); We add another two fields *URD* and *ARD* into the data packet header. The meaning of the fields will be clarified later in the paper. We have a predefined hash function h which takes two strings as its inputs and outputs a uniformly random number between 0 and 1.

Suppose interest packet *IN* is produced by client *C* and responded by data provider *P*. Note the data provider can be the source server of the requested data packets as well as a router that could supply the requested data packet. The path *IN* travels is "*C*-R_0-R_1 – ⋯-R_{n-1}-*P*", where R_0, R_1,…, R_{n-1} are routers. The values of URD, ARD, WT and HV are initialized to 0 when *IN* is generated. When *IN* arrives at any router R_i, it is first processed by the interest processing module of R_i as follows:

1. Increase the URD of *IN* by one, so URD counts the number of hops the interest packet has traveled from its sender. If this router has the requested data packet in its content store, end the process.
2. Extract *IN*'s values of HV and WT and assign them to v_1 and ω_1 respectively, i.e., $v_1 = IN.HV, \omega_1 = IN.WT$.
3. Hash the ID of this router and the extended prefix of *IN* to a random number v_2, and assign the weight of this router to ω_2, i.e., $v_2 = h(R_i.ID, IN.prefix), \omega_2 = \frac{R_i.CP}{R_i.EB}$.
4. Compare ω_1 and ω_2 and assign the values of α_1 and α_2 as follows:
   if $\omega_1 < \omega_2$,
   $$\alpha_1 = \frac{2\omega_1}{\omega_1 + \omega_2}, \alpha_2 = 1$$
   else,
   $$\alpha_1 = 1, \alpha_2 = \frac{2\omega_2}{\omega_1 + \omega_2}$$
5. Compare $\alpha_1 \cdot v_1$ and $\alpha_2 \cdot v_2$. If $\alpha_2 \cdot v_2 \geq \alpha_1 \cdot v_1$, select the current router as the new temporary caching router, and update WT, HV and ARD of *IN* by $IN.WT \leftarrow \omega_2$, $IN.HV \leftarrow v_2$, $IN.ARD \leftarrow IN.URD$. So WT records the weight of the temporary caching router, HV records the hash value computed from the caching router's identifier and the interest packet's extended prefix, and ARD records the distance in hops from the client to the temporary caching router.

When the interest packet finally arrives at the data provider, the URD and ARD of *IN* record the distances from the client to the data provider and the ultimate caching router respectively. WT is the weight of the ultimate caching router, and HV is the hash value of the interest packet's extended prefix and the ID of the ultimate caching router. We take the router ID as one input of the hash function, thus we will obtain different hash values in different routers. Since we take the interest packet prefix instead of its whole name as the other input of the hash function, all the interest packets corresponding to the same file and passing through the same path will select the same caching router. This observation will benefit the maintenance of dynamic routing table in the second stage.

To simplify the description, let ω_0, ω_1, …, ω_{n-1} denote the weights of R_0, R_1, …, R_{n-1} respectively. In the following we will prove that the probabilities of R_0, R_1, …, R_{n-1} to cache the response data packet are in the ratio of $\omega_0 : \omega_1 : \cdots : \omega_{n-1}$, based on which we will further prove that the number of data packets cached in a router is proportional to the content store capacity of this router in the whole network scale.

When *IN* arrives at R_0, since its initial value of HV is 0, R_0 is selected to be the caching router temporarily. Then *IN* is forwarded to R_1.

Consider R_0 and R_1. The hash values of their IDs and *IN*'s prefix are v_0 and v_1 respectively. Notice that v_0 and v_1 are independent of each other and distributed in the interval [0, 1] uniformly. If $\omega_0 > \omega_1$, we have $\alpha_0 = 1$ and $\alpha_1 = \frac{2\omega_1}{\omega_0+\omega_1}$. So R_1 is selected to be the caching router in a probability of

$$p\{\alpha_1 v_1 \geq \alpha_0 v_0\} = p\left\{v_0 \leq \frac{2\omega_1}{\omega_1 + \omega_0} \cdot v_1\right\} = \frac{\omega_1}{\omega_1 + \omega_0}.$$

R_0 is selected to be the caching router in a probability of

$$1 - p\{\alpha_1 v_1 \geq \alpha_0 v_0\} = \frac{\omega_0}{\omega_1 + \omega_0}.$$

Thus the probabilities that R_0 and R_1 are selected to be the caching router are in the ratio of ω_0: ω_1. The case where $\omega_0 \leq \omega_1$ can be proved in the similar way.

Similarly, we can prove that for any two routers R_i and R_j, their probabilities to be selected to be the caching router are in the ratio of ω_i: ω_j, thus the probabilities of R_0, R_1,…, R_{n-1} to cache the response data packet are in the ratio of $\omega_0 : \omega_1 : \cdots : \omega_{n-1}$.

As the ego betweenness of a router is highly correlated with the betweenness centrality of the router in the whole network scale, and the load borne by a router is closely related to its betweenness centrality [16–18], we define a simplified model where the traffic flowing through a router is proportional to the ego betweenness of the router. Consider two arbitrary routers R_i and R_j in this model. Their ego betweennesses are EB_i and EB_j, and their content store capacities are CP_i and CP_j respectively. Since their probabilities to cache a given data packet are in the ratio of $\frac{CP_i}{EB_i} : \frac{CP_j}{EB_j}$, we can assume that their probabilities to cache a data packet are $\lambda\frac{CP_i}{EB_i}$ and $\lambda\frac{CP_j}{EB_j}$ respectively, where λ is a constant. We assume that the traffic in packets flowing through R_i and R_j are μEB_i and μEB_j respectively, where μ is another constant. So the numbers of data packets that will be cached in R_i and R_j are in the ratio of

$$\lambda\frac{CP_i}{EB_i} \cdot \mu EB_i : \lambda\frac{CP_j}{EB_j} \cdot \mu EB_j = CP_i : CP_j.$$

Thus the number of data packets cached in a router is proportional to the content store capacity of this router, and the caching load is balanced among the routers. While the proof above adopts many rough assumptions, it is reasonable on the whole.

After being processed by the interest processing module, if the current router cannot supply the requested data packet, this interest packet will be passed to the interest forwarding module. The interest forwarding module will select the best face to forward the interest packet. The method of selecting the forwarding face will be introduced in Section 4.3.

4.2. Cache data packet & maintain routing information

When the interest packet arrives at the data provider at last, its URD records the number of hops from the client to the provider, which is the length of the transmission path, and its ARD records the number of hops from the client to the ultimate caching router. The provider sets the URD and ARD of the response data packet to the corresponding values in the interest packet, and sends the data packet back to the client. When the data packet arrives at router R_i, it is first processed by the data processing module of R_i. The module decreases the URD value of this data packet by 1 at first, thus URD records the distance from the client to the current router. Note that ARD records the distance from the client to the caching router. Then the module compares the values of URD and ARD. If $URD = ARD$, current router is the caching router selected previously, so the data packet is cached. Otherwise, if $URD > ARD$, the data packet will be cached in later routers. The distances from current router to the caching routers are $URD - ARD$, and the caching routers can be reached through the faces through which the data packet is forwarded, i.e., the faces in the PIT entry corresponding to this data packet. In contrast, if $URD < ARD$, the data packet must have been cached in a previous router. The distance from the current router to the caching router is $ARD - URD$, and the caching router can be reached through the incoming face of the data packet.

In the case where $URD \neq ARD$, the module will maintain dynamic routing information for the data packet. Since clients usually request a file instead of a single data packet and the interest packets corresponding to the same file and passing through the same path will select the same caching router, the data packets corresponding to the same file tend to be cached in the same router. As in Table 2, we maintain a dynamic routing entry for a file instead of a single data packet, thus the size of the dynamic routing table is reduced dramatically while its accuracy is maintained to a large extent. A dynamic routing item consists of an extended prefix and multiple triples in the form of *(face, distance, number)*. Now suppose that the module has obtained the extended prefix of the data packet and a set of faces through which the caching routers can be reached as well as the distance from current router to the caching routers.

If an entry corresponding to the prefix doesn't exist in the dynamic routing table, the module will simply add an entry for the prefix, and a triple with the number as 1 will be added to the item for each face in the face set. If an entry has existed, the module will update the entry as follows. For each face in the face set, if a triple corresponding to the face doesn't exist in the entry, such a triple will be created with the number as 1. Otherwise, the distance of the triple is updated to the weighted average value of the original distance and the newly obtained distance with the weights as the original number and 1 respectively and then the number in the triple is increased by 1. For example, if the original triple is $(face_1, dist_1, num_1)$, and now a new data packet with the same extended prefix could be reached through $face_1$ and the distance between current router to the caching router is $dist_2$, we will update the triple as follows:

$$dist_1 \leftarrow \frac{dist_1 \times num_1 + dist_2}{num_1 + 1}$$
$$num_1 \leftarrow num_1 + 1$$

After being processed by the data processing module, the data packet is passed to the data forwarding module, where the data packet is forwarded using the information in the matching PIT entry.

Since a cached data packet will be evicted due to the limited content store capacity of routers, the dynamic routing information needs to be updated to fit the caching status of the in-network storage. In SADO every router will maintain an estimate T, which is the moving average duration that a data packet is kept in the content store of this router and is updated every time a data packet is evicted. SADO conservatively assume that every data packet for which the router maintains dynamic routing information will be evicted in T sec. So when a router adds the dynamic routing information about a data packet into the dynamic routing table, it will start a timer which will expire in T sec and remove the routing information about the packet from the dynamic routing table when the timer expires.

4.3. Forward interest packets

Since the static routing table and dynamic routing table may provide multiple faces to forward an interest packet, the interest forwarding module must decide which one to use. The interest forwarding module queries the static routing table at first. As there may be multiple available faces in the matching entry (e.g., multiple green faces as described in [8]), only the best face as well as the distance associated with the face will be returned. Suppose this query returns the face f_1 to the server and the distance d_1 from current router to the server. If it forwards the interest packet through f_1, the cost of fetching the target data packet is $2d_1$. The cost here refers to the total distance traveled by the interest packet and its response data packet. Then the module queries its dynamic routing table. The dynamic routing table may provide several choices to forward the interest packet. Suppose one choice indicates that the distance from current router to the corresponding caching router is d_2, and there are m data packets with matching prefix cached there. Assume the name of the interest packet indicates that the source file is segmented into n data packets. Now consider that the module forwards the interest packet through the face provided by this choice. Since a dynamic routing entry corresponds to a file instead of a single data packet, this choice cannot guarantee that the requested data packet exists in the caching router. If the requested data packet exists in the caching router, the cost of fetching the data packet is $2d_2$. This happens in a probability of $\frac{m}{n}$. In the case where the requested data packet doesn't exist there, this router will have to request the data packet from the source server again. In this case the cost of fetching the requested data packet is $2d_1 + 2d_2$, and this happens in a probability of $\frac{n-m}{n}$. So the expected cost of fetching the requested data packet is

$$\frac{m}{n} \cdot 2d_2 + \frac{n-m}{n} \cdot (2d_1 + 2d_2).$$

Using this approach, the interest forwarding module can compute the expected cost for every available face provided by the static routing table as well as the dynamic routing table. As in [8], the router will select the face with the lowest cost to forward the interest packet and start a re-try timer at the same time. If the re-try timer expires before the requested data packet is fetched back, the router will stop trying alternative faces and send a NACK packet to the downstream router from which the router received the interest packet. Otherwise, if the router receives a NACK packet

Please cite this article as: Q. Li et al., A smart routing scheme for named data networks, Computer Communications (2016), http://dx.doi.org/10.1016/j.comcom.2016.09.012

from the upstream node before the re-try timer expires, it will try the faces with larger cost until the requested data packet is returned back or the re-try timer expires.

5. Performance evaluation

In this section we will verify the advantages of SADO over another three caching methods through comprehensive simulations. The three methods adopt the static routing method which routes a given interest packet to the source server of the requested data packet. The first caching method is that proposed in [6], which requires a router to cache each data packet passing by. We denote the caching method by CCN. As many researchers have pointed out, this caching method is of low efficiency. We evaluate its performance in this section and take it as a baseline for other more advanced caching methods. The other two caching methods we compare against are EgoBetw [9], which caches a data packet in the router with the largest ego betweenness in its transmission path, and WAVE [10], which caches data packets with popular contents preferentially.

5.1. Simulation settings

We have developed a discrete event-driven platform for the simulations. The source code of the simulation platform could be downloaded from [23]. In the simulations, each file consists of 100 data packets and the size of each data packet is 1500 bytes. There are a large number of clients in the network and each client is connected to a single router. When a client is started, it randomly selects a file and requests all the data packets corresponding to the file. After current file is completed, the client selects another file and requests the data packets corresponding to the file again. The probability that a file is selected follows the Zipf-like distribution [24], i.e., the ith most popular file is selected in a probability of $\frac{C}{i^{\alpha}}$ where $C = (\sum \frac{1}{i^{\alpha}})^{-1}$. As the value of α used in [10] is 0.85 and that used in [9] is 1.0, in our simulations we set the value of α to 0.85. The content stores take LRU as the eviction algorithm. Notice that the routing tables in SADO are different from the FIBs of routers in TCP/IP networks and they may need longer processing delay. In our future work we will be devoted to improving the routing tables to make them meet the requirements of real deployment. At this stage, to verify the effectiveness of SADO and avoid considering the processing delay of routers, we measure the total distance traveled by an interest packet and its response data packet instead of the delay experienced by the clients directly. To avoid congestion of the network, each client has a sending window of 10, i.e., a client has to wait until some of its outstanding interest packets are consumed and the sending window moves forward before it could send out another interest packet.

In our simulations, we use three metrics, i.e., average distance ratio, server load ratio and average utilization ratio of cached data packets, to evaluate the performance of the routing schemes. The average distance ratio is defined as

$$\text{average distance ratio} = \frac{\sum_{i=1}^{K} \frac{d_i}{D_i}}{K}$$

where d_i is the total distance traveled by an interest packet and its response data packet we measured, and D_i is the total distance the interest packet and its response data packet would travel if the response data packet is fetched from the server directly. K is the number of (interest packet, data packet) pairs in the sample.

The server load ratio is defined as

$$\text{server load ratio} = \frac{n}{N},$$

where n is the number of data packets supplied by the servers and N is the number of data packets received by the clients, including the data packets supplied by servers as well as routers.

The average utilization ratio of cached data packets is defined as

$$\text{average utilization ratio} = \frac{m}{M},$$

where m is the total number of cache hits occurring in all routers, and M is the number of data packets cached by all the routers (including those that have been evicted), thus the average utilization ratio of cached data packets indicates how many times a cached data packet is utilized on average.

Since the interest packets and data packets corresponding to the same server are always transmitted along a tree of which the server is the root, the performance of the schemes on a tree topology is expected to be consistent with their performance on a real network topology. We will test the performance of the schemes on the tree topologies at first and then on a real network topology extracted from CAIDA [25]. For the simulations for tree topologies, the impact caused by the height of the tree as well as the number of files in the network will be examined. Basically we assume that the capacity of content store is same for all the routers, and then we will verify the performance of SADO in balancing the number of data packets cached by routers by setting the content store capacities of different routers to be different. Moreover, we will examine the performance of the caching methods when number of clients attached to each router changes.

5.2. Simulations on binary tree topologies

In the simulations on tree topologies, there is a single server which is located at the root of the tree, while the clients are located at the leaves and the routers at the internal nodes. We first examine the performance of the caching methods on a binary tree of 8 levels. The files supplied by the server is categorized into 10 types, where each type has a unique prefix. To examine the impact caused by the number of files in the network on the performance of the caching methods, we increase the number of files in each type from 100 to 200, thus the total number of files in the network is increased from 1000 to 2000. The content store capacity of each router is such that it could accommodate up to 10 files, i.e., 1000 data packets.

Fig. 2 shows that the performance of SADO is better than that of other caching methods, that's because SADO routes interest packets towards cached data packets intelligently while in other caching methods cache hit occurs only when the requested data packet is cached on the path from the client to the source server. Especially, SADO achieves the best performance compared with other schemes when there are 100 files associated with each prefix. At this point the average distance ratios of SADO and CCN are 0.57 and 0.68, their server load ratios are 0.18 and 0.51, and their utilization ratios are 1.18 and 0.16 respectively. Since the total size of in-network storage is fixed, the probability that the requested data packet is cached in the content stores is decreased as the number of files increases, degenerating the performance of the four caching methods. Another finding is that the performance of EgoBetw is the same as that of CCN. That's because each router has the same value of ego betweenness in the binary tree topology and the response data packet will be cached in every router in its transmission path, which is the same as that of CCN.

Fig. 3 presents the distribution of distance ratios of the caching methods when there are 140 files in each type. The figure shows that in SADO cache hit occurs uniformly in the network. In contrast, the cache hit tends to occur in the routers near the clients in the other three caching methods. For example, only 28.4% of the distance ratios are within the interval of [0.0, 0.95] while 20.7% of

Please cite this article as: Q. Li et al., A smart routing scheme for named data networks, Computer Communications (2016), http://dx.doi.org/10.1016/j.comcom.2016.09.012

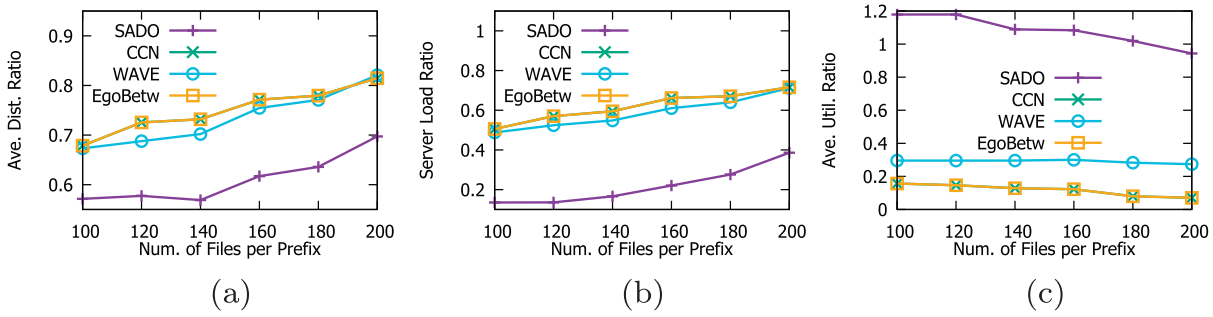

Fig. 2. (a) Average distance ratio, (b) server load ratio and (c) average utilization ratio of cached data packets for various numbers of files associated with each prefix.

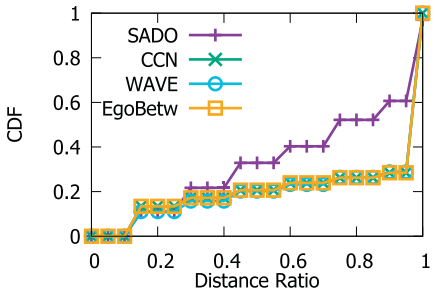

Fig. 3. The Cumulative Distribution Function (CDF) of the distance ratios.

the distance ratios are within the interval of [0.0, 0.45] for CCN (and for EgoBetw as well), which means about 73% of the cache hits occur in the range that is 3 hops away from the client. On the contrary, 54.3% of the cache hits occurs in the range that is 3 hops away from the clients and 45.7% of the cache hits occur in the range that is 4 through 6 hops away from the clients for SADO.

We further examine the performance of the routing schemes on binary trees of various heights. In this group of simulations, the height of the binary tree increases from 6 levels to 11 levels. There are 100 types of files in the network and each type corresponds to 100 files. The content store of each router could accommodate 10 files. Fig. 4 shows that SADO outperforms the other three caching methods for every scenario. As the height of the tree increases, the number of routers as well as the size of in-network storage increases exponentially. As a result, the average distance ratio and server load ratio of the caching methods grows better. However, since the CCN, EgoBetw and WAVE tend to utilize the routers nearby and cache hit occurs only in the path from the client to the server in the caching methods, they fail to fully leverage the newly added in-network storage, while SADO routes interest packets towards cached data packets intentionally. As a result, the performance of SADO increases more obviously than the other caching methods. While the average distance ratio and server load ratio of the caching methods grows better, the average utilization ratio of cached data packets of the caching methods grows worse naturally as the height of the tree increases, since more unpopular data packets are cached.

Moreover, we test the performance of the four caching methods when changing the content store capacity of each router. We find that when the content store capacity of routers increases, the performance of CCN, EgoBetw and EgoBetw is improved dramatically at first, and then improved slowly once the content store capacity exceeds a threshold, which is caused by the heavy-tail distribution of the popularity of the data packets.

5.3. Simulations on real network topology

In the group of simulations we test the performance of the caching methods on a real network topology which is extracted from CAIDA [25] and is from a single AS. There are 196 routers in the network and the diameter of the network is 6. The content store of each router could accommodate 10 files. We choose 100 routers as the client delegates and another 5 routers as the server delegates. There are 15 clients connected to each client delegate and 1 source server connected to each server delegate. We increase the number of files a server can supply from 200 to 700.

While SADO outperforms the other three caching methods constantly, it achieves the best performance when each server can supply 500 files, as shown in Fig. 5. Since the dynamic routing information is not absolutely correct and the forwarding decision is made based on probability, an interest packet in SADO may be forwarded towards a router where the requested data packet does not exist, introducing extra cost in fetching data packets. Hence the average distance ratio in SADO is only slightly better than that in the other three caching methods. However, SADO achieves a really good performance in releasing the load borne by the source servers and increasing the utilization ratio of cached data packets. For example, the load borne by the source servers in SADO is 33% of that

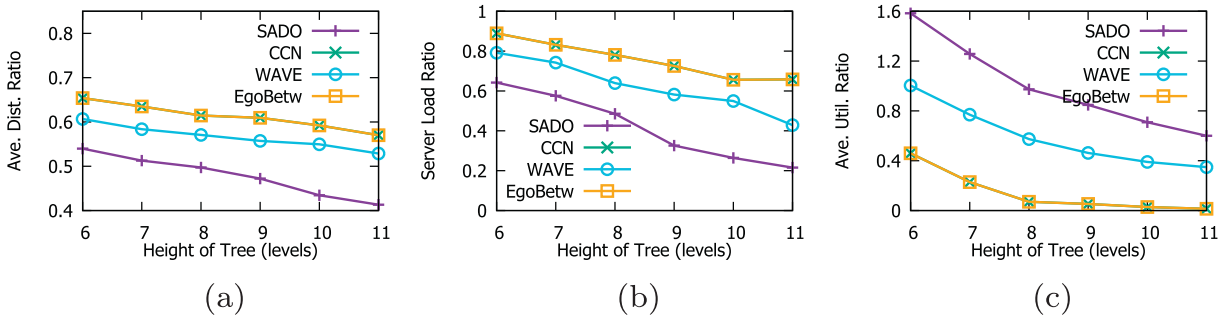

Fig. 4. (a) Average distance ratio, (b) server load ratio and (c) average utilization ratio of cached data packets when the height of the topology tree increases from 6 levels to 11 levels.

Please cite this article as: Q. Li et al., A smart routing scheme for named data networks, Computer Communications (2016), http://dx.doi.org/10.1016/j.comcom.2016.09.012

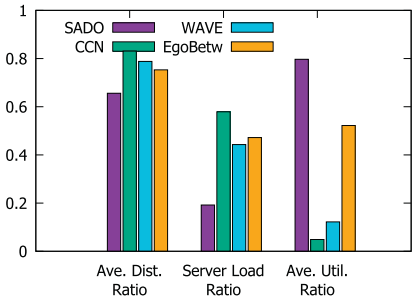

Fig. 5. The performance of the four caching methods on a real network topology.

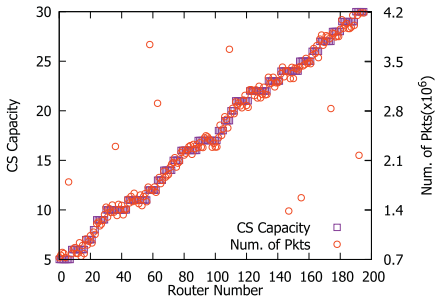

Fig. 7. The distribution of content store (CS) capacity and number of data packets cached of each router.

in CCN, and the average utilization ratio of cached data packets in SADO is 6.5 times over that in WAVE and 1.5 times over that in EgoBetw. The smallest difference in average distance ratio between CCN and SADO in the simulations is 0.04 and occurs when there are 400 files in the network, but at this point their difference in server load ratio is as large as 0.21 and their difference in utilization ratio is as large as 0.65, which shows the great advantages of SADO in reducing the load on servers and improving the utilization ratio of the cached data packets.

To examine the influence on performance caused by the number of clients attached to each client delegate, we run the above simulation again. In this simulation, each server could supply 500 files and we increase the number of clients attached to each client delegate from 5 to 30. As shown in Fig. 6, the number of clients attached to each client delegate does not have an obvious influence on the performance of the caching methods. We believe that it is because that all the clients select the files following the same rule (i.e., the zipf rule with the parameter of 0.85). As a result, more clients requesting files is equivalent to a client requesting files with greater rate.

As a router with larger content store capacity will cache a data packet passing by with higher possibility in SADO, we will verify the performance of SADO in balancing the number of data packets cached in routers in the following simulation. The topology of network is the same as that in last simulation and each source server could supply 500 files. The content store capacities of the routers are randomly distributed in the range of [5, 30] files. The link bandwidth between any two routers is 1 Gbps. We run the simulation for 10 sec and record the number of data packets cached by each router. We run the simulation for six rounds and take the average number of cached data packets in each router. The content store capacity as well as the number of cached data packets in each router is shown in Fig. 7, which demonstrates that SADO could balance the number of data packets cached in each router very well.

6. Conclusion and future work

In the paper we propose a scheme, i.e., SADO, to cache data packets and route interest packets, and verify its effectiveness through simulations. The main feature of SADO is that it considers the caching and routing as a whole and routes interest packets towards the packets cached on routers intentionally. We argue that it is necessary for utilizing the in-network storage efficiently since the in-network storage in a single transmission path is extremely limited compared with the large amount of contents in the whole network, and the improvement in the cache hit ratio will be poor even if the caching method is ideally efficient without routing around the cached data packets intentionally. As the previous works usually avoid considering routing for the cached data packets since the packets are evicted frequently, our work is a good starting point for the dynamic routing in NDN.

As the content store is not permanent storage and cached data packets will be evicted frequently, it's necessary to update the dynamic routing information. In Section 4.2 we maintain a moving average duration T that a cached data packet is kept in local content store and assume that every data packet for which this router maintains routing information will be evicted in T sec. As the method works well in our simulations, we found that it is memory consuming. For example, suppose the average residence time of data packets is T = 5 sec, the link bandwidth is 10 Gbps, each timer need 5 bytes and the average size of a data packet is 1500 bytes, then the router will need up to 20 MB for the timers. In our future work we will devote to developing an more efficient method to update the dynamic routing table. Moreover, the interest forwarding module needs to determine a best face to forward an interest packet based on the information from the static routing table and the dynamic routing table. This will cause nonnegligible delay anyway. In the future we will improve our method of processing an interest packet and make SADO meet the requirements of real deployment.

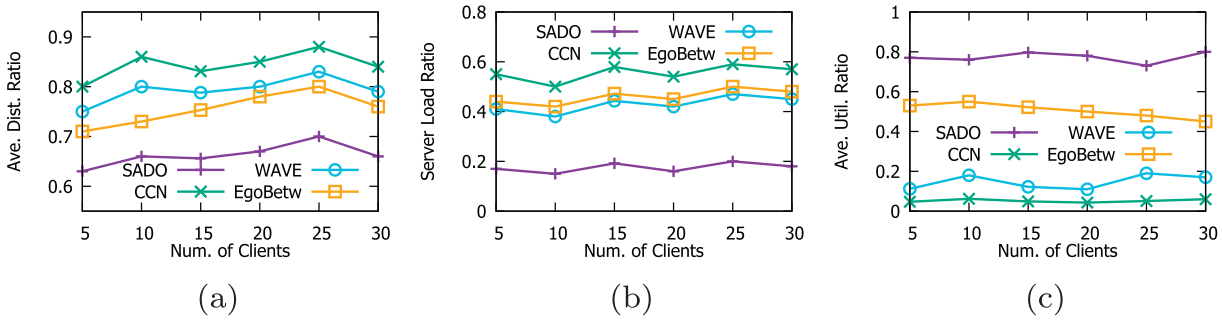

Fig. 6. (a) Average distance ratio, (b) server load ratio and (c) average utilization ratio of cached data packets for various number of clients attached to each client delegate.

Please cite this article as: Q. Li et al., A smart routing scheme for named data networks, Computer Communications (2016), http://dx.doi.org/10.1016/j.comcom.2016.09.012

Acknowledgements

This work is supported by the National Natural Science Foundation of China under grant No. 61402255 and No. 61502268, the R&D Program of Shenzhen under grant No. ZDSYS20140509172959989, No. JCYJ20150630170146830 and No. Shenfagai (2015)986, and the China Postdoctoral Science Foundation.

References

[1] T. Koponen, M. Chawla, B.-G. Chun, A. Ermolinskiy, K.H. Kim, S. Shenker, I. Stoica, A Data-oriented (and Beyond) Network Architecture, in: Proceedings of the 2007 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, in: SIGCOMM '07, Kyoto, Japan, 2007.
[2] M. D'Ambrosio, C. Dannewitz, H. Karl, V. Vercellone, MDHT: A Hierarchical Name Resolution Service for Information-centric Networks, in: Proceedings of the ACM SIGCOMM Workshop on Information-centric Networking, in: ICN '11, Toronto, Canada, 2011.
[3] PSIRP/PURSUIT, (http://www.fp7-pursuit.eu/PursuitWeb/).
[4] NetInf/SAIL, (http://www.sail-project.eu/).
[5] D. Raychaudhuri, MobilityFirst Vision & Technical Approach Summary, MobilityFirst External Advisory Board Meeting, 2011.
[6] V. Jacobson, D.K. Smetters, J.D. Thornton, M.F. Plass, N.H. Briggs, R.L. Braynard, Networking Named Content, in: Proceedings of the 5th International Conference on Emerging Networking Experiments and Technologies, in: CoNEXT '09, Rome, Italy, 2009.
[7] Named-Data Networking, (http://named-data.net/).
[8] C. Yi, A. Afanasyev, I. Moiseenko, L. Wang, B. Zhang, L. Zhang, A case for stateful forwarding plane, Comput. Commun. 36 (7) (2013) 779–791.
[9] W.K. Chai, D. He, I. Psaras, G. Pavlou, Cache "Less for More" in Information-centric Networks, in: Proceedings of the 11th International IFIP TC 6 Conference on Networking - Volume Part I, in: IFIP'12, Prague, Czech Republic, 2012.
[10] K. Cho, M. Lee, K. Park, T.T. Kwon, Y. Choi, S. Pack, WAVE: Popularity-based and collaborative in-network caching for content-oriented networks, in: Computer Communications Workshops (INFOCOM WKSHPS), 2012 IEEE Conference on, 2012.
[11] A.K.M.M. Hoque, S.O. Amin, A. Alyyan, B. Zhang, L. Zhang, L. Wang, NLSR: Named-data Link State Routing Protocol, in: Proceedings of the 3rd ACM SIGCOMM Workshop on Information-centric Networking, in: ICN '13, Hong Kong, China, 2013.
[12] T.M. Wong, J. Wilkes, My Cache or Yours? Making Storage More Exclusive, ATEC, 2002.
[13] N. Laoutaris, S. Syntila, I. Stavrakakis, Meta algorithms for hierarchical Web caches, IPCCC, 2004.
[14] J.M. Wang, J. Zhang, B. Bensaou, Intra-AS Cooperative Caching for Content-centric Networks, in: Proceedings of the 3rd ACM SIGCOMM Workshop on Information-centric Networking, in: ICN '13, Hong Kong, China, 2013.
[15] I. Psaras, W.K. Chai, G. Pavlou, Probabilistic In-network Caching for Information-centric Networks, in: Proceedings of the Second Edition of the ICN Workshop on Information-centric Networking, in: ICN '12, Helsinki, Finland, 2012.
[16] K.-I. Goh, B. Kahng, D. Kim, Universal behavior of load distribution in scale-Free networks, Phys. Rev. Lett. 87 (2001) 278701.
[17] K. Goh, J. Noh, B. Kahng, D. Kim, Load distribution in weighted complex networks, Phys. Rev. E, Stat., Nonlin.,Soft Matter Phys. 72 (2005) 017102.
[18] M. Everett, S.P. Borgatti, Ego network betweenness, Social Netw. 27 (1) (2005) 31–38.
[19] L. Wang, A.K.M.M. Hoque, C. Yi, A. Alyyan, B. Zhang, OSPFN: An OSPF Based Routing Protocol for Named Data Networking, Technical Report, 2012.
[20] L. Zhang, A. Afanasyev, J. Burke, V. Jacobson, k. claffy, P. Crowley, C. Papadopoulos, L. Wang, B. Zhang, Named data networking, SIGCOMM Comput. Commun. Rev. 44 (3) (2014) 66–73, doi:10.1145/2656877.2656887.
[21] PARC. CCNx Open Source Platform, (http://www.ccnx.org).
[22] D.G. Thaler, C.V. Ravishankar, Using name-based mappings to increase hit rates, IEEE/ACM Transactions on Networking (TON) 6 (1) (1998) 1–14.
[23] SADO Simulator, (https://github.com/xinshengzzy/Simulator.git).
[24] L. Breslau, P. Cao, L. Fan, G. Phillips, S. Shenker, Web caching and Zipf-like distributions: evidence and implications, in: Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE, in: INFOCOM '99, Columbia, USA, 1999.
[25] The CAIDA UCSD Internet Topology Data Kit, (http://www.caida.org/data/internet-topology-data-kit).

Please cite this article as: Q. Li et al., A smart routing scheme for named data networks, Computer Communications (2016), http://dx.doi.org/10.1016/j.comcom.2016.09.012