Journal of Network and Computer Applications 216 (2023) 103651

Contents lists available at ScienceDirect

Journal of Network and Computer Applications

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

Efficient congestion control scheme based on caching strategy in NDN

Dapeng Qu [a], Jun Wu [a], Jiankun Zhang [a], Chengxi Gao [b,*], Haiying Shen [c], Keqin Li [d]

[a] Liaoning University, 66 Chongshan Middle Road, Huanggu District, Shenyang, 110036, Liaoning, China
[b] Shenzhen Institute of Advanced Technology, Chinese Academy of Sciences, 1068 Xueyuan Avenue, University Town, Nanshan District, Shenzhen, 518055, Guangdong, China
[c] University of Virginia, Charlottesville, VA 22904, USA
[d] State University of New York, New Paltz, NY 12561, USA

ARTICLE INFO

Keywords:
Named data networking
Caching strategy
Congestion control

ABSTRACT

As a pioneering network architecture, Named Data Networking (NDN) leverages the content-centric model and connectionless transmission mode to enhance network capacity. Despite the emergence of numerous congestion control protocols aimed at improving NDN transmission efficiency, most of these approaches implicitly assume that each interest packet elicits one single data packet, which significantly restricts network performance. To address this issue, we introduce C3NDN, a Congestion Control scheme that leverages caching strategy in NDN. C3NDN formulates a One-Interest-Multiple-Data model to improve network efficiency, and employs a probabilistic caching strategy to cache popular content at important nodes and optimize the in-network caching characteristic of NDN. Furthermore, C3NDN incorporates a congestion control algorithm based on the One-Interest-Multiple-Data model, which considers the bandwidth and delay information of the transmission path. Additionally, C3NDN implements a marking method that takes full advantage of node cache and reduces transmission time. Extensive experiments have been conducted to show the performance of C3NDN, and the results demonstrate that C3NDN can reduce the transmission time by up to 41.68% when compared with the other congestion control schemes.

1. Introduction

With the rapid development of Internet scale and its diversity, network application model is evolving from early resource sharing to current content requirement and distribution. However, current Internet infrastructure is mainly based on IP address which indicates end-to-end connection. Therefore, it is difficult for traditional end-to-end transmission mode to satisfy this change (Vasilakos et al., 2015). To solve this problem, Named Data Networking (NDN) is proposed. It changes the semantics of network service from delivering packets to a given destination to fetching content identified by a given name, thus helps the Internet evolving from the host-centric model to a content-oriented model (Zhang et al., 2014).

In NDN, a content requester sends out an interest packet carrying a name identifying its desired content. The interest packet is routed towards the content producer based on this name. The content provider (the content producer or some intermediate routers which cache the whole content) provides a corresponding data packet which carries the matching content, and the data packet is forwarded back to the content requester through the interest packet routing path reversely. However, if the requested content is too large to be encapsulated into a single data packet, the provider must divide the content into some fragments, and encapsulate each fragment into a data packet (Ren et al., 2016). If all data packets are returned at one time, it easily results in congestion and packet loss. If only one data packet is returned which is corresponding to the interest packet, then the requester needs to keep sending interest packets until receiving the full content, which easily causes high delay. Therefore, it is necessary to propose an efficient congestion control scheme which can transmit the all content fragments from a large content within a short time and without much packet loss, especially for big data and delay-sensitive applications (e.g., AR/VR) in NDN.

Besides, as an important characteristic of NDN, in-networking caching enables some intermediate nodes to cache the content to shorten the routing paths to the content requester and reduce packet delivery latency (Din et al., 2018). However, due to the space limit, the intermediate node can only cache part of the content fragments instead of the total content. Therefore, the intermediate nodes cannot fully respond to the received interest packet as a content provider in

* Corresponding author.
E-mail addresses: dapengqu@lnu.edu.cn (D. Qu), wujun1420@163.com (J. Wu), zhangjiankun33@163.com (J. Zhang), chengxi.gao@siat.ac.cn (C. Gao), hs6ms@virginia.edu (H. Shen), lik@newpaltz.edu (K. Li).

https://doi.org/10.1016/j.jnca.2023.103651
Received 8 June 2022; Received in revised form 9 February 2023; Accepted 19 April 2023
Available online 29 April 2023

traditional transmission mode in NDN. Thus, how to make use of the content fragments cached in intermediate nodes is an important issue for congestion control in NDN, since it can help mitigate congestion by utilizing the caching which holds some content fragments intermediate nodes and reducing the total traffic load on the content producer in the network.

Although there are some existing works which propose congestion control schemes and caching strategies for NDN, they fall short in reducing the transmission latency (Siddiqui et al., 2019). The main shortcomings manifest in the following ways.

- The majority of current congestion control schemes in NDN adopt One-Interest-One-Data model, in which one data packet is returned for one interest packet. When the content volume is large, the number of required interest packets is also large, which results in high bandwidth occupation and latency overhead in the network.
- Although some works have considered the possibility of using one interest packet to retrieve multiple data packets, they usually focus on the single content provider and lack the consideration for content fragments in in-network caching of intermediate nodes. As a result, only the nodes which hold the whole content are the content providers. This will increase the request latency and make congestion control schemes less effective.

To solve all the problems aforementioned, we propose C3NDN, a **C**ongestion **C**ontrol scheme based on **C**aching strategy in **NDN**. C3NDN adopts the One-Interest-Multiple-Data transmission mode to save network bandwidth and improve the transmission efficiency. In the One-Interest-Multiple-Data transmission mode, interest packets are also used to collect network information including bandwidth and delay, and then the content providers update their congestion window sizes to calculate the number of data packets to be returned for the received interest packet. Moreover, a **p**robabilistic **c**aching **s**trategy (PCS) and the corresponding **c**ache **u**pdating **s**trategy (CUS) which reduces content fragments redundancy and improves cache efficiency are proposed. For the congestion control scheme, a marking mechanism is also proposed to enable anyone which holds a content fragment to respond to the received interest packet, which distributes the transmission of a large content to different intermediate nodes and utilizes the caching of the near nodes to smooth the sending rate of data packets from some further nodes, then effectively relieves the congestion caused by one single content provider in traditional transmission mode in NDN.

The main contributions of this work are summarized as follows.

- We propose the One-Interest-Multiple-Data transmission mode to allow one interest packet to trigger multiple data packets, thus it only requires a few interest packets to get all fragments from the corresponding data packets to form a complete content in one content request, then perform congestion control on the content provider, therefore saving network bandwidth and improving content transmission efficiency.
- We propose a congestion control algorithm which collects the bottleneck link bandwidth and delay information to calculate the congestion window and sending rate of the content provider. We also propose a marking scheme, so that the cached content of each node can be transmitted in an orderly manner, making full use of the node cache and reducing transmission time.
- To further improve the transmission efficiency of C3NDN, we also propose a probabilistic caching strategy and the corresponding cache updating strategy. Each node calculates the caching probability based on the cache value of the content fragment, the popularity of the corresponding content, and the node importance. Moreover, the cache update strategy is executed based on the fragment retention value and redundancy when the space occupied by the cached content exceeds the rated threshold, thereby reducing the redundancy of the cached content and improving the cache efficiency.
- Finally, we conduct extensive experiments to evaluate the performance of C3NDN. Experimental results show that C3NDN has better transmission performance, including higher transmission rate and shorter transmission time. For example, C3NDN can reduce the transmission time by up to 41.68% when compared with the other congestion control schemes. Moreover, PCS also achieves a better performance, in terms of cache hit ratio, average hit counts, and cache replacement ratio.

The rest of this paper is organized as follows. Section 2 summarizes related works. Section 3 presents the system model and the design of C3NDN, Section 4 evaluates C3NDN's performance with extensive experiments under different scenarios. Finally, Section 5 concludes the paper.

2. Related work

In recent years, NDN has been popular in both industry and academic area, and there is a lot of work on NDN to improve network performance. Next, we summarize related works into two subjects, namely caching strategy and congestion control schemes.

2.1. Caching strategies in NDN

As a prominent role of NDN, in-network caching has been attracting plenty of attention. LCE (Leave Copy Everywhere) (Jacobson et al., 2009) is the original and default caching strategy which caches everything everywhere, namely each node caches each received data packet. It fully utilizes caching to improve the caching hit ratio and reduce the transmission latency. However, the limited cache space makes it infeasible to always get data from the cache in reality, because the node will update cache very frequently and unnecessarily to accommodate the new received data packets and the high cache redundancy of unpopular content cannot contribute to a high utilization. Meanwhile, these operations introduce a high overhead. LCP (Leave Copy Probability) (Arianfar et al., 2010) takes a probability calculation before caching the received content. It reduces cache redundancy and caching overhead, but the fixed probability setting does not solve the problem fundamentally, and the characteristic of content and node, such as content popularity and the node centrality, has not been considered. CPCCS (compound popular content caching strategy) (Naeem et al., 2019) divides content into two types, namely OPC (optimal popular content) and LPC (least popular content) based on the number of received data packets for each content file. It caches OPC in all intermediate nodes along the routing, and caches LPC only in the one-hop neighboring node. SDC (spatially dispersed caching) (Kamiyama and Murata, 2018) disperses content by assigning a binary ID to each router and limits cache targets at each router to content with names whose hash value coincides with the router ID. It utilizes the limited cache resources by avoiding duplicated caching of the same content among close routers. In Gui and Chen (2020), a cache placement strategy based on compound popularity (content popularity and node popularity) which enhances the reuse rate of the data packets is proposed. CCndnS (Content Caching strategy for NDN with Skip) (Rezazad and Tay, 2020) breaks a content file into small fragments and spreads them in the path between content requester and provider, so that the first fragment should be cached at the router close to the requester and the last one towards the content provider. In Alhowaidi et al. (2021), a software-defined, storage-aware routing mechanism that leverages NDN router cache-states, software defined networking and multipath forwarding strategies is proposed to improve the efficiency of very large data transfers. For the large data transfers, It presents a comprehensive analysis of NDN cache management and proposes a novel prefetching mechanism to improve data transfer performance. CaDaCa (Categorized Data for Caching) (Herouala et al., 2022) explores the role of data categorization in enhancing the cache mechanisms in NDN. The popular content

requests are categorized to enable in-depth knowledge about users' behavior. In Iqbal et al. (2022), a caching strategy is proposed to help source-driven forwarding. It caches diverse contents considering the gap between successive copies and content availability while admitting new contents. More caching strategies can be referred to several good surveys (Ioannou and Weber, 2016; Zhang et al., 2015a; Abdullahi et al., 2015).

2.2. Congestion control schemes in NDN

Congestion control is always an elementary issue of computer networks. Since NDN is receiver-driven, some researchers take congestion control schemes on the receiver (content requester) and adjust the sending rate of interest packets to avoid congestion (Muchtar et al., 2020). For example, ECP (Explicit Control Protocol) (Ren et al., 2015) detects the network congestion condition proactively, and sends explicit feedback to the content requester. Then the requester adjusts the sending rate of interest packets to control the sending rate of data packets from the content provider, thus to realize the congestion control. Because NDN adopts a receiver-driven hop-by-hop transport approach that facilitates in-network caching, traditional methods which keep a single round trip time (RTT) estimator for a multi-path flow are insufficient, because each routing path may experience different round trip times. Thus, CHoPCoP (Zhang et al., 2015b) utilizes explicit congestion control to deal with the multiple-source and multiple-path situation in NDN. In Lan et al. (2020), DRL-CCP (RL-based Congestion Control Protocol) which is based on deep reinforcement learning is proposed. It enables the content requester to automatically learn the optimal congestion control policy from historical congestion control experience to adjust of the interest packets sending window size. In Ye et al. (2020), HbHCM (Hop-byHop Congestion Measurement) and PAQM (Practical Active Queue Management) are proposed to detect congestion and generate explicit congestion notification at NDN nodes by monitoring the change of transmission delays. HbHCM measures the transmission delay in hop and PAQM converts the delay to notification signals to notify the content requesters. The in-network caching results in that the end-to-end flow control in current Internet cannot be applied to NDN. Thus, in Lee and Nakazato (2020), a diffusion-based flow control method for NDN is proposed. RevMax, a gateway-aware congestion control mechanism, is proposed for the purpose to overcome the drawbacks of the complexity and compatibility issues of the existing flow-based and hop-by-hop congestion control mechanisms (Li et al., 2020). The gateway offers a price for the content requester, and the requester adjusts the interest packet requesting rate according to the price. Then the optimal pricing policy for the gateway is formulated as a revenue maximization problem. In Song and Zhang (2021), the rate-based approach is analyzed which represents a more promising direction than existing window-based congestion control solutions. A BBR(Bottleneck Bandwidth and RTT(Round-trip propagation time)) - guided congestion control is proposed for bulk data fetching by a group of users in Hu et al. (2021). It applies RTT filtering and interest scheduling to improve BBR's efficiency in NDN. In Ye et al. (2021), a Network Utility Maximization (NUM) model is proposed to formulate multi-source and multipath transmission with in-network caches in NDN. Then a Delay-based Path-specified Congestion Control Protocol (DPCCP) is presented as a specific instance of the receiver-driven transmission solutions. DPCCP utilizes queuing delays to measure and control congestion levels of different bottlenecks. In Wu et al. (2022), a multi-path congestion control mechanism is proposed. It includes multi-path discovery and multi-path congestion control. In discovery process, a path tag is devised to uniquely mark each sub-path in the forwarding process and a tag-aware forwarding strategy is presented to discover and manage sub-paths. In congestion control process, some different metrics, such as packet loss, bandwidth, round trip time, and path centrality are integrated to assess paths, and the Upper Confidence Bound (UCB) algorithm is leveraged to select sub-paths to maximize network throughput. In Hashemi and Bohlooli (2021), an explicit feedback-based congestion control is proposed to manage content request sending rate. It employs a per-packet feedback computation to inform requester from the available resource of paths toward repositories and provides feedback to the forwarding mechanism of a router to adjust the sending rate of interest packets to each interface. In Yang et al. (2022), an Intelligent Edge-Aided Congestion Control scheme based on Deep Reinforcement Learning is proposed. It provides a proactive congestion detector which utilizes intermediate routers to transmit accurate congestion information along the path to content consumers through the data packets, and divides data packets into different congestion degrees by a lightweight clustering algorithm to obtain a reasonable transmission rate. Moreover, it distributes the estimated bandwidth resources to content consumers with different transmission needs to maintain fairness.

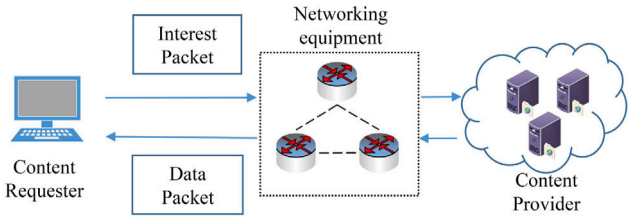


Fig. 1. System flow of NDN.

2.3. Summary of existing work

Although there are plenty of related works, nearly all caching strategies and congestion control schemes are designed separately, and congestion control schemes are taken without caching strategies, and then the in-network caching cannot be fully utilized. On the other hand, some caching strategies try to decrease content redundancy, which would influence the ability of intermediate node responding to interest packets. Different from existing works, our C3NDN and corresponding PCS can take caching for each content fragment from the viewpoints of content and nodes, and fully utilize the caching of content fragments to set the sending rate of data packets to avoid congestion.

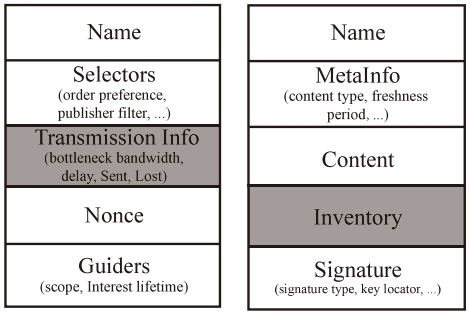
3. System design

In this section, we present C3NDN's design in detail. We first introduce the model of our system. Then we propose a caching strategy based on content and node property together with a cache updating strategy. Finally we propose a congestion control scheme based on the caching strategy.

3.1. System model

In this paper, NDN can be modeled as a connected graph $G = (V; E)$, where $V = \{v_i | 1 \le i \le N\}$ is the set of nodes, and $E = \{e_{i,j} | v_i, v_j \in V, 1 \le i; j \le N, i \ne j\}$ is the set of links. N is the total number of nodes in G, and $N = |V|$. At the same time, the content can be modeled as a set C, and a content file $c_k \in C$, where $1 \le k \le M$, $M = |C|$ is the total number of content files in NDN. In this paper, we assume that each node has the same cache size and each content fragment has the same size, but each content file has different sizes thus can be divided into different numbers of fragments.

To clarify NDN and be understood easily, a system flow of NDN is shown in Fig. 1. In NDN, the content requester send out an interest packet to the network, which mainly consists of the requested content name and other parameters. When the interest packet traverses through the network, if the intermediate nodes(like routers) contain partial content data, they will return the data packet which consists of the



(a) interest packet. (b) data packet.

Fig. 2. The structure of packets in NDN.

content name, data and other parameters, then send out the interest packet to content provider. Finally, the content provider returns the data packets with the remaining data which intermediate nodes fail to provide.

To support the proposed scheme in this paper, referring to the basic packet format in NDN, we design the structure of an interest packet and a data packet shown in Fig. 2 respectively. The field with shadow are the new ones. In the interest packet, the "Content Name" field indicates the identity of the content data. The "Selectors" field indicates the preferences like order preference, publisher filter and so on. The "Transmission info" field indicates the routing and content information in the whole transmission, such as available bandwidth and delay, along the routing path from the content requester to the content provider. "Sent" and "Lost" mean that the content information including the content fragments can be provided by the upstream nodes which are along the content requester to the current intermediate node, and the lost content fragments during the last transmission respectively. The "Nonce" field is used to detect looping interests. The "Guiders" field indicates the scope interest lifetime and so on. In the data packet, the "Content Name" field also indicates the identity of the content data. The "MetaInfo" field indicates the content type, freshness period and some other meta information. The "Content Data" includes the data of the requested content. The "Inventory" field means the content fragments sent by the content provider for the received interest packets. The "Signature" field indicates the signature type, key locator and so on. We present the details of our proposed methods in the following subsections.

To be understood easily, Fig. 3 shows the system workflow of C3NDN. Because C3NDN is designed for big data and delay-sensitive applications, for example AR/VR in NDN, the large content will be partitioned into some fragments, and each fragment can be encapsulated into a data packet. Moreover, as the basis of C3NDN, a probabilistic caching strategy and the corresponding cache updating strategy make some intermediate node cache some fragments. Thus, when a content requester sends an interest packet to request the content, if any intermediate nodes have cached some content fragments, they can respond with these fragments to the received interest packet, although they cannot provide the whole content. Meanwhile, the intermediate node forwards the interest packet to require the remaining content fragments for the whole content. When the content provider receives the interest packet, it will only return the other content fragments reversely along the routing path of the interest packet. The cached intermediate node can utilize the cache store (CS) to smooth the data packet flow to avoid congestion. Therefore, C3NDN can not only mitigate congestion, but also reduce content acquirement latency, because it transmits a part of content fragments from the content provider to the content requester, while the traditional method needs to transmit all content fragments from the content provider to the content requester.

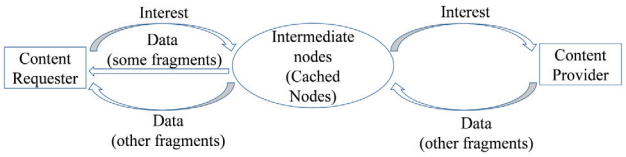


Fig. 3. Workflow of C3NDN.

3.2. Probabilistic caching strategy (PCS)

Since the content can be divided into multiple fragments, an effective caching strategy should consider two aspects, namely content fragment and node. The former contains the content popularity and the caching property of a content fragment, and the node property reflects the location of a node in NDN.

3.2.1. Content fragment property

Due to the distributed characteristic, each node has different viewpoints of content popularity. Thus, the popularity of content c_j in node v_i is calculated by Eq. (1).

$$po(v_i, c_j) = \frac{num(v_i, c_j)}{max(num(v_i, c_k)), \forall c_k \in C} \tag{1}$$

where $num(v_i, c_j)$ is the number of interest packets about content c_j received by node v_i, and $max(num(v_i, c_k)), \forall c_k \in C$ is the maximum number of interest packets about a content file received by node v_i.

Since a content file is divided into multiple fragments, the caching property of a content fragment can be decided according to the content fragment ratio and the content fragment caching ratio. The content fragment ratio measures how easy to cache the whole content for a single node, which indicates the content's easiness to be cached within a single node. Therefore, the larger number of content fragments, the less easy to cache the whole content for the single node, thus the less value of content fragment ratio. The content fragment caching ratio evaluates the percentage of content fragments cached by the node over the total number of the content fragments, which indicates a node's ability to cache a specific content. They are calculated as Eqs. (2) and (3):

$$R_f(c_j) = \begin{cases} 1, & Nfra(c_j) \le \delta^L_{Nfra} \\ \frac{\delta^L_{Nfra}}{\delta^U_{Nfra}}, & Nfra(c_j) > \delta^U_{Nfra} \\ \frac{\delta^L_{Nfra}}{Nfra(c_j)}, & otherwise \end{cases} \tag{2}$$

$$R_f(v_i, c_j) = \frac{Nfra(v_i, c_j)}{Nfra(c_j)} \tag{3}$$

where $R_f(c_j)$ denotes the content fragment ratio, and δ^L_{Nfra} and δ^U_{Nfra} are two thresholds denoting the lower and upper bounds of the number of the content fragments. Obviously, the larger the number of the fragments of a content file, the smaller content fragment ratio. Thus, when the number of the fragments $Nfra(c_j)$ of content c_j is small, it is easy for a node to get all fragments to form a content file, and then the node will be more willing to cache a content fragment. Here, $Nfra(c_j)$ is the number of fragments of a whole content c_j, and $Nfra(v_i, c_j)$ is the number of fragments of content c_j held by node v_i. $R_f(v_i, c_j)$ is the content fragment caching ratio denoting the ratio of the number of the content fragments cached in a node and the number of the content fragments. Obviously, the more content fragments are cached in a node, the larger the $R_f(v_i, c_j)$. Thus, the caching property of a fragment of content c_j in node v_i is calculated by Eq. (4):

$$cp(v_i, c_j) = max(R_f(c_j), R_f(v_i, c_j)). \tag{4}$$

Obviously, the larger the content fragment ratio, the larger the content fragment caching ratio, and the larger the caching property

of the content fragment. We use the maximum form to give equal importance to content fragment ratio and content fragment caching ratio. Specially, if a node caches a fragment from a content file with fewer fragments, or a node has cached most of the fragments from a content file, there will be a higher probability to cache this content fragment to be able to respond to later content requests.

3.2.2. Node property

The location of a node in NDN has important influence on network load. We consider degree centrality and eigenvector centrality because they reflect the number of neighbors and their importance respectively. Degree centrality is the simplest centrality measure in a graph, which means the ratio of the degree and the total number of nodes in the network. Thus, it can evaluate the importance of the node in the graph. Similarly, Eigenvector centrality is a more sophisticated view of centrality, which evaluates the node's ability to connect to other well-connected nodes. Since it is very hard to get an accurate network topology practically, we just compare the centralities of a node and its one-hop neighbors to denote the its relative centrality. The relative degree centrality and eigenvector centrality of a node are calculated by Eqs. (5) and (6):

$$R_{C_D}(v_i) = \frac{C_D(v_i)}{max(C_D(v_k))}, v_k \in (N(v_i) \cup v_i) \tag{5}$$

$$R_{C_E}(v_i) = \frac{C_E(v_i)}{max(C_E(v_k))}, v_k \in (N(v_i) \cup v_i) \tag{6}$$

where $C_D(v_i)$ is the degree centrality of node v_i, and can be calculated as the ratio of the number of its one-hop neighbor and the number of nodes in network. $max(C_D(v_k))$, $v_k \in (N(v_i) \cup v_i)$ denotes the maximum value of the degree centralities of node v_i and its all one-hop neighbors. The fraction form makes the node's relative degree centrality a pure decimal and achieves normalization. Similarly, the $C_E(v_i)$ is the eigenvector centrality of node vi, and can be calculated as the product of the eigenvector of the adjacent vector of the network and the corresponding eigenvalues. Thus, the node property of v_i can be calculated by Eq. (7):

$$d(v_i) = \gamma_D \times R_{C_D}(v_i) + \gamma_E \times R_{C_E}(v_i) \tag{7}$$

where γ_D, γ_E are two weight factors, $0 \le \gamma_D, \gamma_E \le 1$, and $\gamma_D+\gamma_E=1$. They determine the weight between degree centrality and eigenvector centrality in calculating node's property.

Because probabilistic caching strategy has been proven to be an effective and simple way (Naeem et al., 2022), when a node v_i receives a new content fragment about content c_j, we comprehensively consider the above three factors, and calculate caching probability as Eq. (8):

$$P(v_i, c_j) = d(v_i) \times po(v_i, c_j) \times cp(v_i, c_j) \tag{8}$$

Obviously, the more important the node is, the more popular the content is, the higher the caching property is, and the higher probability of the node caches the content fragment.

3.2.3. Algorithm pseudocode

Algorithm 1 shows the pseudo-code of PCS. When an intermediate node receives a data packet, it will calculate the caching property of this content fragment according to Eq. (4), the popularity of the content according to Eq. (1), the node property according to Eq. (7), the caching probability based on Eq. (8), and finally take probabilistic caching.

3.3. Cache updating strategy (CUS)

Because there is limited space for each node's caching, the node needs to update its caching timely to accommodate new and more popular contents.

Algorithm 1 Pseudocode of PCS

Input: received data packet, CS (cache store);
Output: CS^*;

```
Get the content fragment from the received data packet;
Calculate caching property of this content fragment in current node (Eq. (4));
Calculate the popularity of the content in current node (Eq. (1));
Calculate node property of current node (Eq. (7));
Calculate caching probability P (Eq. (8));
Generate a random number ran_num;
if P > ran_num then
    Cache the content fragment;
end if
```

3.3.1. Content fragment caching property

To choose the appropriate content fragment in cache updating strategy, each node maintains two properties for caching each content fragment, namely fragment self caching value and fragment neighbor caching redundancy. The former means the value of reserving a content fragment in its own cache. Based on the above analysis in the last subsection, it is calculated by Eq. (9):

$$f_c(v_i, c_j) = \omega_p \times po(v_i, c_j) + \omega_r \times R_f(v_i, c_j) \tag{9}$$

where ω_p, ω_r are two weight factors, $0 \le \omega_p, \omega_r \le 1$, and $\omega_p+\omega_r = 1$. They determine the weight between content popularity and content fragment caching ratio in calculating content fragment caching value.

The fragment neighbor caching redundancy means the number of the same content fragments cached in one-hop neighbors. Obviously, if the same content fragments are cached in two neighboring nodes, which will not significantly improve the performance in terms of hitting hop count. Moreover, if the fragment neighbor caching redundancy is large, namely there are plenty of same content fragments cached in neighboring nodes, it will reduce the diversity of caching and increase the content requiring latency. To reduce the huge overhead of cache updating and exchanging information, we define two thresholds, namely δ_{CS}^L and δ_{CS}^U to determine the cache updating strategy, and $0 < \delta_{CS}^L < \delta_{CS}^U < 100\%$. Only when the ratio of node caching occupation is larger than the upper threshold δ_{CS}^U, the cache updating strategy is taken.

3.3.2. Algorithm pseudocode

Algorithm 2 shows the pseudo-code of CUS. When the ratio of node's caching occupation δ_{CS} exceeds the upper threshold δ_{CS}^U, it will take the cache updating strategy, namely it first updates the neighbor fragment caching redundancy by exchanging Hello packets with neighbors, and then removes the content fragments with the maximum caching redundancy until δ_{CS} is lower than the lower threshold δ_{CS}^L. If two or more cached content fragments with the same fragment neighbor caching redundancy, the one with smaller fragment self caching value will be removed.

3.4. Congestion control scheme based on caching strategy (C3NDN)

Next, we introduce our congestion control scheme based on PCS and CUS, namely C3NDN.

3.4.1. Congestion control scheme

The basic dataflow of C3NDN is as follows. When an intermediate node receives an interest packet, it first checks its CS to see if there is matched content. If yes, a data packet can be sent back to the incoming interface of the interest packet; otherwise, it checks whether the name of the interest packet is in PIT (Pending Interest Table) already. If yes, it adds the incoming interface of the interest packet to the existing PIT entry; otherwise, it builds a new entry for this interest packet in

Algorithm 2 Pseudocode of CUS

Input: CS^{*};
Output: CS^{**};

```
if δ_CS > δ^U_CS then
    Update caching redundancy of each cached content fragment by exchanging Hello packets with neighbors;
    while δ_CS > δ^L_CS do
        Remove the content fragment with the maximum neighbor caching redundancy;
        if two or more cached content fragments with the same neighbor caching redundancy then
            Remove the content fragment with smaller fragment self caching value;
        end if
    end while
end if
```

PIT, calculates the available bandwidth for the future data flow, and then calculates and updates the "bandwidth" and "delay" fields in the interest packet header. Finally, it forwards the updated interest packet based on FIB (Forwarding Information Base). The available bandwidth B_d and delay D_y are calculated by Eqs. (10) and (11):

$$B_d = \frac{B - \sum_{l=1}^{m} b_l}{n - m + 1}, (0 \le m < n) \tag{10}$$

$$D_y = D_y + d_y \tag{11}$$

where B is the total link capacity, n is the total number of flows in the link, m is the number of flows whose occupied bandwidth is smaller than (B/n) among these n flows, b_l is the occupied bandwidth of the lth flow among these m flows. d_y is the latency from the previous node to the current node, and D_y is the latency from the content requester to the current node. Then we can update the bottleneck bandwidth B_h for this data flow according to Eq. (12):

$$B_h = \begin{cases} B_d, & B_h = 0 \\ min(B_h, B_d), & B_h \neq 0 \end{cases} \tag{12}$$

and update B_h information in the packet header.

If the current bottleneck bandwidth is 0, namely there is not any recording about the Bandwidth of the routing path, B_h is the current available bandwidth B_d, otherwise, it is the minimum of current bottleneck bandwidth B_h and the available bandwidth B_d.

In fact, the actual bandwidth and delay values provided by a routing path are heavily influenced by the amount of traffic load it transferred. How to accurately obtain them is really hard, complex and beyond the scope of this paper. In fact, these information can be obtained through methods like those methods in Javadtalab et al. (2015), Paul et al. (2016) or others.

When the interest packet reaches the content provider, it will first calculate the congestion window based on the "bandwidth" and "delay" fields in the interest packet header. Then, it generates data packets with the cached content, and enables the "Inventory" field to record the content fragments which can be sent based on the congestion window. The congestion window $cwnd$ is calculated according to Eq. (13):

$$cwnd = \frac{B_h \times D_y}{size} \tag{13}$$

where $size$ is the size of the data packet. Then the content provider returns the data packets using the bandwidth as B_h, and the data packet will carry the information about the packet sending rate and id of the content fragment.

When an intermediate node receives a data packet, it first takes the PCS as in the above subsection, and then checks and updates the PIT entry. If the data packet is the last packet for the data flow of the required content, it deletes the entry after it forwards the data packet to the outgoing interface based on the PIT entry; otherwise, it prolongs the lifetime of this entry. When the content requester receives the data packet, it counts the received data packets based on the "Inventory" field. If there are still some content fragments to receive, the requester generates and sends new interest packets to request them. If there are some data packets missing, namely some data packets cannot be received before timeout, the requester generates and sends new interest packets whose "Lost" field identifies these lost content fragments. This process will continue until all content fragments are received by the requester.

Algorithm 3 Pseudocode of processing a received interest packet

Input: *received interest packet* ;
Output: *Forwarding interest packet / Sending data packet* ;

```
Check CS and compare it with the "Sent" field in packet header;
if there are all required content fragments in CS then
    Calculate cwnd (Eq. (13));
    Generate data packets with the content fragments;
    Send these data packets based on the cwnd;
    Drop the interest packet;
else
    Build a new entry for the interest packet in PIT;
    Calculate available bandwidth (Eq. (10));
    Calculate bottleneck bandwidth (Eq. (12));
    if there is any required content fragment in CS then
        Calculate cwnd (Eq. (13));
        Generate data packets with the cached content fragments;
        Send these data packets based on the cwnd;
    end if
    Update the "bandwidth", "delay" and "Sent" fields in interest packet header;
    Forward the interest packet based on FIB;
end if
```

3.4.2. Marking mechanism

To make full use of in-network caching in NDN, we further propose a marking mechanism to enable the intermediate node which caches some content fragments to be a content provider which provides data packets with these fragments for the received interest packet. Specifically, when an intermediate node caches some content fragments for the received interest packet, it will first compare the cached fragments and the "Sent" field in interest packet header, since the "Sent" field means that the content fragments can be provided by the upstream nodes to the content requester. It can directly provide some data packets with these fragments which are cached and not in the "Sent" field. If there are still some missing fragments, that is, some content fragments cannot be provided by the nodes along the routing path from the content requester to the current node, then it will calculate the bandwidth and delay as in the above subsection, modify the corresponding fields in interest packet header, and forward the interest packet based on FIB. If the content required by the interest packet can be totally provided by intermediate nodes and its previous nodes, namely the "Sent" field already has all the content fragments, the interest packet will be dropped.

On the other hand, when the data packet reaches an intermediate node which still has some fragments to be sent to the content requester, the node will cache the content fragment in this data packet, and then send all cached content fragments to this requester orderly. With the marking mechanism, that is, the intermediate node marking the content fragments which has been cached in the current node, it fully utilizes the in-networking caching, reduces the number of content fragments requested to the subsequent nodes, and saves bandwidth and avoids potential congestion.

Algorithm 4 Pseudocode of processing a received data packet

Input: *received data packet* ;
Output: *Sending interest packet / Forwarding data packet / ∅* ;

```
if the current node is the content requester then
    Check the "Inventory" field in the data packet header;
    if there are still some content fragments to be received or some fragments missing then
        Generate and Send an interest packet for these content fragments;
    end if
else
    if there are some content fragments in transmission then
        Cache the data packet;
        Send all cached content fragments to the requester orderly;
    end if
    Update corresponding entry in PIT;
    Forward the data packet to the incoming interface of the entry in PIT;
end if
```

3.4.3. Algorithm pseudocode

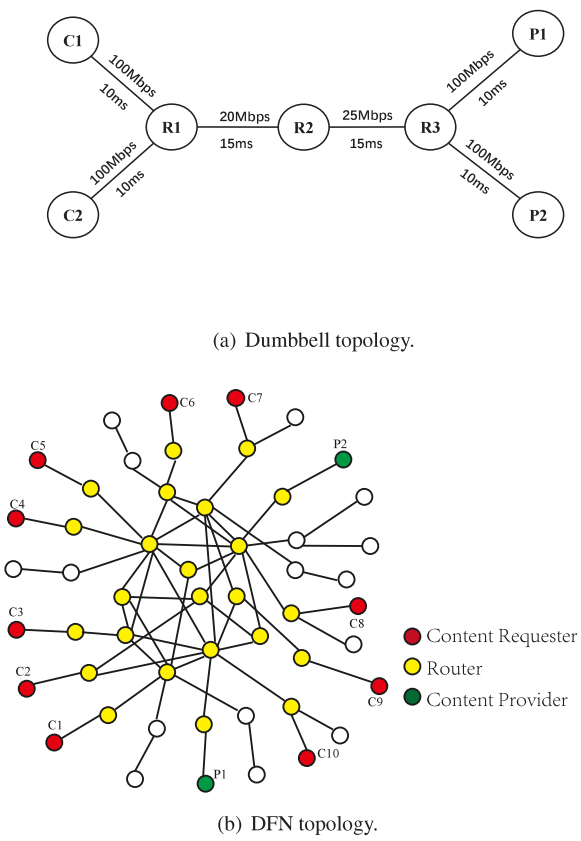
Since C3NDN involves the interactions between interest packets and data packets, we will provide two algorithms about the process of receiving them respectively. Algorithm 3 shows the pseudo-codes of processing a received interest packet. When a node receives an interest packet, it will check if it has all required content fragment. If so, we calculate cwnd, and return the corresponding data packets using the speed based on cwnd, and drop the interest packet. Otherwise, it will update PIT, and calculate available bandwidth and bottleneck bandwidth. If the node contains some required content fragment, it will return corresponding data packets using the speed according to the updated cwnd. Finally, it updates related fields in the interest packet header and forwards the interest packet based on FIB. And algorithm 4 shows the pseudo-codes of processing a received data packet. When the content requester receives a data packet, it will check the missing fragment and sends out the interest packet. If other nodes receive the data packets and there are some content fragments in transmission, they will cache the data packet and send all cached content fragments to the requester in order. Then, they will update PIT and forward the data packet.

4. Performance evaluation

In this section, we use the open-source ndnSIM (Mastorakis et al., 2017) to evaluate the performance of the proposed C3NDN. ndnSIM implements the NDN protocol stack in the NS3 network simulator.

4.1. Experimental settings

We use two different topologies in our experiments as illustrated in Fig. 4. One is a dumbbell topology which is widely applied in congestion control research in NDN. There are two content requesters, namely C1 and C2 and two content providers, namely P1 and P2. The other is a popular and realistic topology German Research Network (DFN) topology which is widely applied in NDN research (Qu et al., 2022). In DFN, there are ten content requesters, namely C1, C2, ..., C10, two content providers, namely P1 and P2, and 24 routers. Moreover, there are 20 contents, namely c1, c2, ..., c20, and all the odd contents are in P1, and all the even contents are in P2. The range of the number of fragments of a content is 1–50, which follows normal distribution with mean as 25 and standard deviation as 12. The value will be round up to an integer to make sure that the number of fragments is an integer. The content requests follow zipf-like distribution. Other parameters are



(a) Dumbbell topology.

(b) DFN topology.

Fig. 4. The network topology.

set as shown in Table 1, and their values are determined based on the best experimental results from multiple experiments.

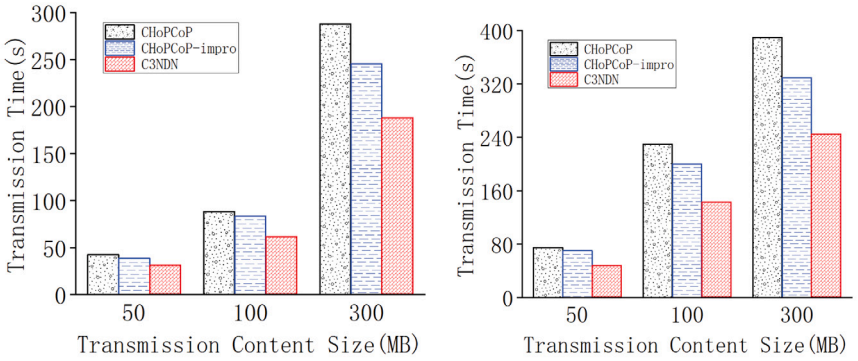
For the caching strategy, recall that SDC (Kamiyama and Murata, 2018) is a closely related work. It disperses contents by assigning a binary ID to each router and limits the cache targets at each router to content with names whose hash value equals to the router ID. LCE (Jacobson et al., 2009) is the default caching strategy that caches each received content everywhere in NDN. LCP (Arianfar et al., 2010) is a simple and effective caching strategy and caches each received content. Therefore, we choose LCE, LCP and SDC for performance comparison with our proposed PCS. While for congestion control strategy, recall that CHoPCoP (Zhang et al., 2015b) which utilizes explicit congestion control for the multiple-source multiple-path situation. The content requesters and intermediate routers use AIMD (Additive Increase Multiplicative Decrease) and RED (Random Early Detection) respectively. Moreover, considering the One-Interest-One-Data model of CHoPCoP, we improve it by making content providers change the number of interest packets to control congestion and name the improved version of CHoPCoP as CHoPCoP-impro.

To demonstrate the performance comprehensively, we use the following metrics to evaluate and compare the performance of different schemes in terms of congestion control and caching.

- *Transmission Time* (*TMT*). The period from the time of the content requester sending the first interest packet to the time of receiving the last data packet.
- *Transmission Rate* (*TMR*). The receiving rate of data packets at the corresponding content requester.
- *Number of Packet Loss* (*NPL*). The number of data packets dropped by the intermediate routers when there is a congestion.
- *Queue Length* (*QEL*). The average lengths of the queues in all involved routers.

Table 1
Parameter settings.

Parameter	Symbol	Value
Bandwidth of each link	$bandwidth$	100 Mbps
Delay of each link	$delay$	10 ms
Default content size	-	100 KB
Cache capacity of each node	CS	50 data packets
Data packet size	-	1124 Bytes
Interest packet sending rate	-	150 packets/s
Bounds of content fragments	$\delta^L_{s\_ch}$, $\delta^U_{s\_ch}$	10, 40
Bounds of caching replacement	δ^L_{CS}, δ^U_{CS}	60%, 95%



(a) Dumbbell topology. (b) DFN topology.

Fig. 5. *TMT* comparison of different congestion control schemes.

- *Cache Hit Ratio* (*CHR*). The ratio of the number of received interest packets from the content requesters that hit in current caches and the total number of received interest packets from the content requesters.
- *Average Hit Count* (*AHC*). The average of the hop counts from the content requester to the content provider which responds to the received interest packets (including content provider and intermediate routers).
- *Cache Redundancy Degree* (*CRD*). The average of the number of involved routers caching the same content fragment in the whole network.
- *Cache Replacement Ratio* (*CRR*). The ratio of the number of the content fragments being replaced and the total number of content fragments in the cache.

The former four metrics, namely *TMT*, *TMR*, *NPL* and *QEL* measure the overall performance of the proposed C3NDN. Obviously, *TMT* is the most important metric to evaluate C3NDN. The latter four metrics, naming *CHR*, *AHC*, *CRD* and *CRR* measure the ability of the proposed PCS and corresponding CUS.

We run each experiment for ten times and report the average.

4.2. Experimental results

We first demonstrate the *TMT* of our C3NDN, and then comprehensively evaluate C3NDN based on PCS and CUS.

4.2.1. TMT evaluation and analysis

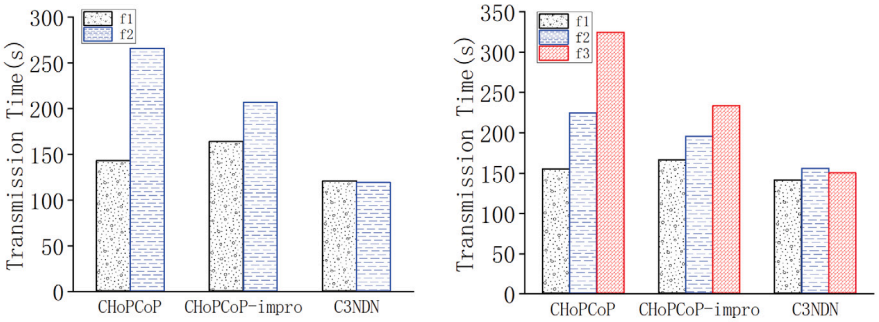
To comprehensively demonstrate the *TMT* of C3NDN under different conditions, we first set that there is no content cached in the intermediate node, and test *TMT*. As shown in Fig. 5, C3NDN consumes the least transmission time for different content sizes. Moreover, its advantage over CHoPCoP and CHoPCoP-impro will increase with the increment of the required content sizes. The reduced *TMT* reaches to 31.6% and 24.5% over the other two different schemes in dumbbell topology without other flows respectively, and reaches to 35.3% and 27.2% over the other schemes in DFN with other flows respectively.

Second, we further compare *TMT* when multiple flows share the bottleneck bandwidth. There are two data flows, namely f1(P1-C1) and f2(P2-C2) in Dumbbell, and three flows, namely f1(P1-C1), f2(P2-C2) and f3(P3-C3) in DFN. Flow f1 starts at the beginning, then f2 and f3 (only in DFN) successively start in order every other 5 s. As shown in Fig. 6, these competing flows in C3NDN have very similar *TMT*, and differences among *TMT*s of different flows in CHoPCoP-impro is in the middle, and that in CHoPCoP is the largest. The reason is that the early generated flow usually occupies the more bandwidth, and thus the new flow consumes more *TMT* to transmit the required content. C3NDN considers the bottleneck bandwidth and shares it among different competing flows.

Lastly, we compare *TMT* with/without some cached required content fragments. We put 1/4 required content in R2 in Dumbbell and R4 in DFN respectively. As shown in Fig. 7, C3NDN and CHoPCoP-impro take advantages of the cached content fragments and achieve the reduced transmission delay. C3NDN reduces about 19.7% and 21.6% respectively, and CHoPCoP-impro reduces about 10.3% and 13.8% respectively. While CHoPCoP does not utilize the cached content fragments, and miscalculates the queue length, and begins the slow start phase frequently, and *TMT* increases about 32.9% and 37.5% respectively.

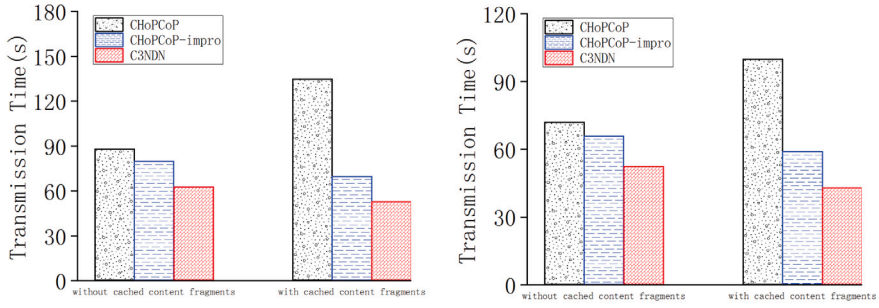
4.2.2. Evaluation of congestion control

To demonstrate our proposed congestion control scheme comprehensively, we first take C3NDN without any caching strategy, and then test it based on PCS. Fig. 8 demonstrates the *QEL* and *TMR* of three different congestion control schemes with the transmission time. It is obvious that C3NDN has a much smaller *QEL* than CHoPCoP and CHoPCoP-impro. Because the sending windows of CHoPCoP and CHoPCoP-impro increase additively and decrease multiplicatively, and the intermediate nodes adjust queue length to control congestion. The continuous increase of congestion window would contribute to a longer queue in intermediate nodes, which would give feedback to decrease the congestion window. The two situations happen alternately, which causes a continuous oscillation of *QEL*. CHoPCoP sends the interest packets whose number equals to the congestion window with one



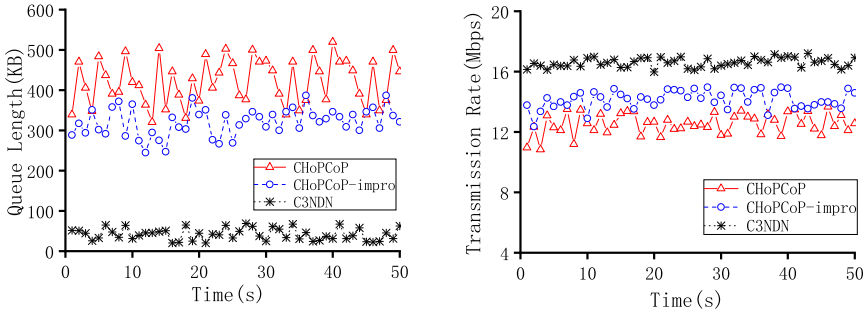
(a) Dumbbell topology. (b) DFN topology.

Fig. 6. *TMT* comparison of different congestion control schemes among competing flows.



(a) Dumbbell topology. (b) DFN topology.

Fig. 7. *TMT* comparison of different congestion control schemes with/without cached required content fragments.



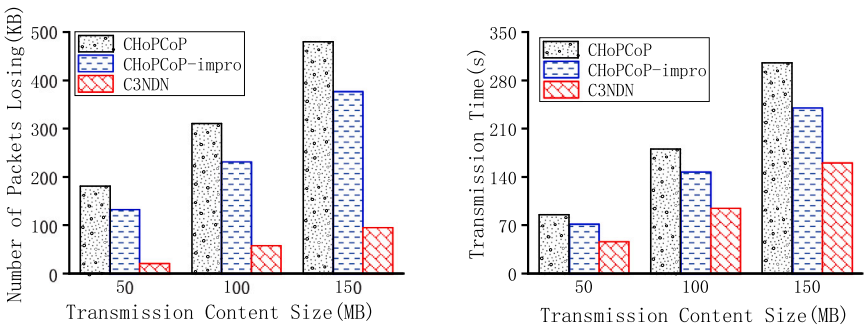
(a) Queue Length (*QEL*). (b) Transmission Rate (*TMR*).

Fig. 8. Performance comparison of different congestion control schemes with transmission time.

time, and plenty of data packets would contribute to the longest *QEL*. CHoPCoP-impro adjusts interest packet sending rate based on the routing path condition to avoid the data packets from piling up to a certain extent, and gets a shorter *QEL*. While C3NDN calculates the number of data packets based on the information of routing paths, and controls the interest packet sending rate under the available bandwidth of the whole routing paths, thus achieving the shortest *QEL*. Meanwhile, the shorter *QEL* contributes to the better *TMR*, thus C3NDN achieves the highest *TMR*, CHoPCoP-impro achieves the second *TMR*, and CHoPCoP has the worst *TMR*, as shown in Fig. 8b.

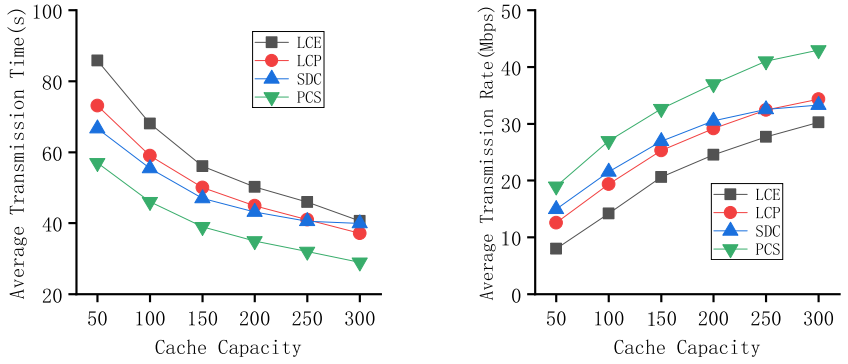
Fig. 9a demonstrates the *NPL* of three different congestion control schemes with different content sizes. C3NDN has the least packet loss. Moreover, with the increase of the transmission content sizes, its advantages over the other two strategies become larger. When the transmission content size is 150MB, C3NDN outperforms CHoPCoP and CHoPCoP-impro by 76.3% and 65.2%, respectively. As analyzed in the last paragraph, C3NDN has the shortest and stablest *QEL* which contributes to the smallest *NPL*. Fig. 9b demonstrates the *TMT* of three different congestion control schemes with different transmission content sizes. Because C3NDN has the least packet loss, it consumes the least *TMT*. Moreover, with the increase of the transmission content sizes, C3NDN's performance in reducing transmission time becomes more superior. CHoPCoP sends plenty of interest packets one time, and CHoPCoP-impro sends interest packets based on the routing path condition, thus, CHoPCoP-impro gets the second performance in *NPL* and *TMT*, and CHoPCoP has the least one.

We further evaluate the performance of C3NDN with different cache capacities. As shown in Fig. 10, with the increase of cache capacity, PCS achieves the best performance, namely the highest *TMR* and the least *TMT*, SDC gets the second best performance, and LCP and LCE have the third and worst respectively. With the increase of cache capacity, the performance improvements of SDC over LCP and LCE become less. When the cache capacity is 250, SDC and LCP achieve similar performance. When the cache capacity increases further, the performance of SDC is less than that of LCP. Because SDC bounds content fragments and the hash of routers ID, namely each router only



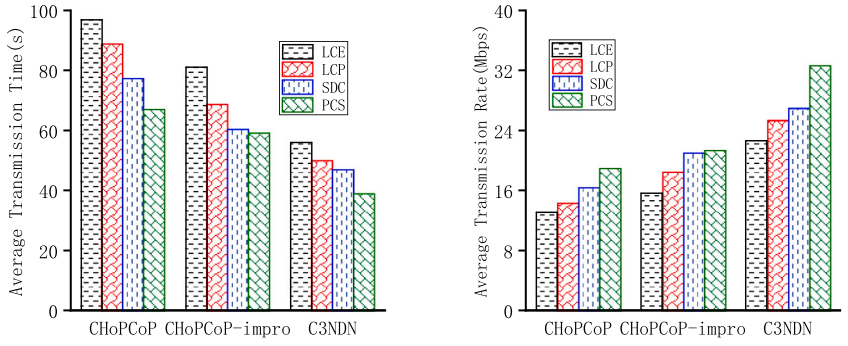
(a) Number of Packet Losing (*NPL*). (b) Transmission Time (*TMT*).

Fig. 9. Performance comparison of different congestion control schemes with different transmission content sizes.



(a) Transmission Time (*TMT*). (b) Transmission rate (*TMR*).

Fig. 10. Performance comparison of different caching strategies with C3NDN under different cache capacities.



(a) Transmission Time (*TMT*). (b) Transmission rate (*TMR*).

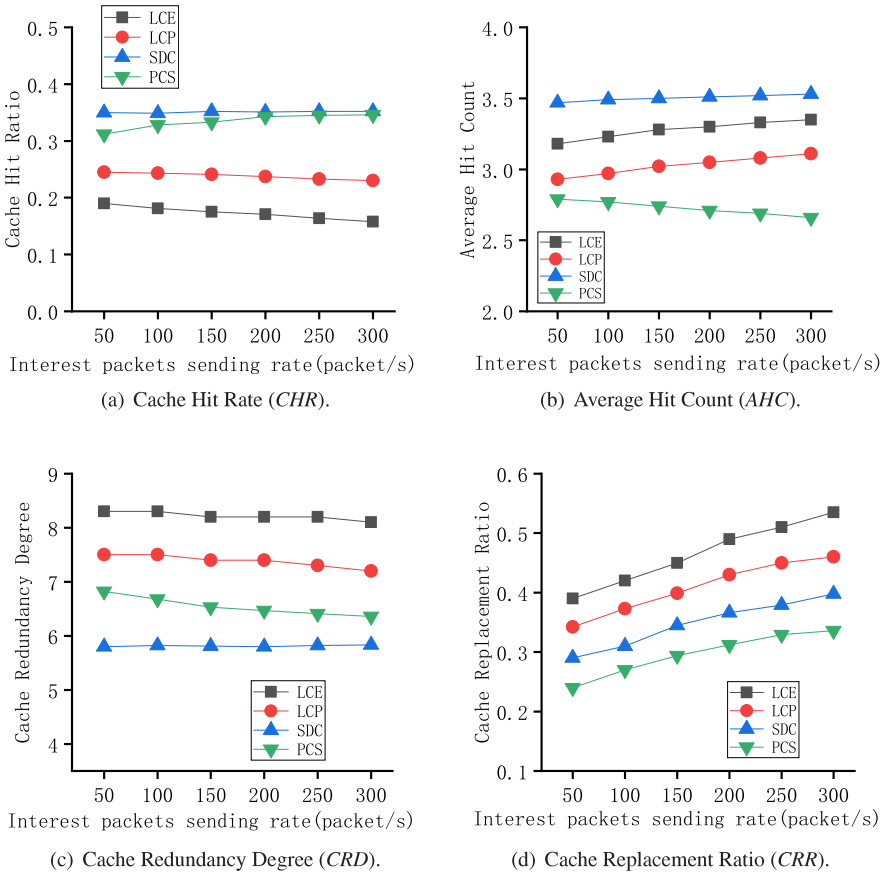
Fig. 11. Performance comparison of different congestion control schemes with different caching strategies.

caches those matched content fragments, and the large cache capacity cannot bring further performance improvement.

Next, we evaluate the performance of the congestion control schemes with different caching strategies. As shown in Fig. 11, C3NDN achieves the highest *TMR* and finishes the whole transmission with lower *TMT* than CHoPCoP and ChoPCoP-impro, no matter what kind of caching strategy is combined. When there are some cached content fragments in the transmission path, the cache marking can mark the cached content fragments in intermediate nodes, make them transmit orderly, and achieve a higher transmission rate. While CHoPCoP misjudges congestion for caching and enters slow start frequently, it has a reduced transmission rate. When there are some cached content fragments in intermediate nodes, ChoPCoP-impro does not cache received data packets. Then, the queue length of data packets in intermediate nodes may exceed the threshold, and thus it also gets a reduced transmission rate. Obviously, ChoPCoP-impro is better than CHoPCoP, since the former uses the One-Interest-Multiple-Data model. When the congestion control scheme is chosen, PCS achieves the best performance, SDC achieves the second performance, and LCP and LCE have the third and worst performance.

4.2.3. Evaluation of caching strategy

The performance comparison of different caching strategies with different interest packets sending rates is shown in Figs. 12. Obviously, our proposed PCS achieves the best performance, in terms of *AHC* and *CRR*. It achieves the second best performance and performs worse than SDC in terms of *CHR* and *CRD*. Because we cache a content fragment from the viewpoint of both node and content, namely the content fragment with higher popularity will be cached in the node with larger importance. Each intermediate router caches the received



(a) Cache Hit Rate (*CHR*). (b) Average Hit Count (*AHC*).

(c) Cache Redundancy Degree (*CRD*). (d) Cache Replacement Ratio (*CRR*).

Fig. 12. Performance comparison of different caching strategies with different interest packets sending rates.

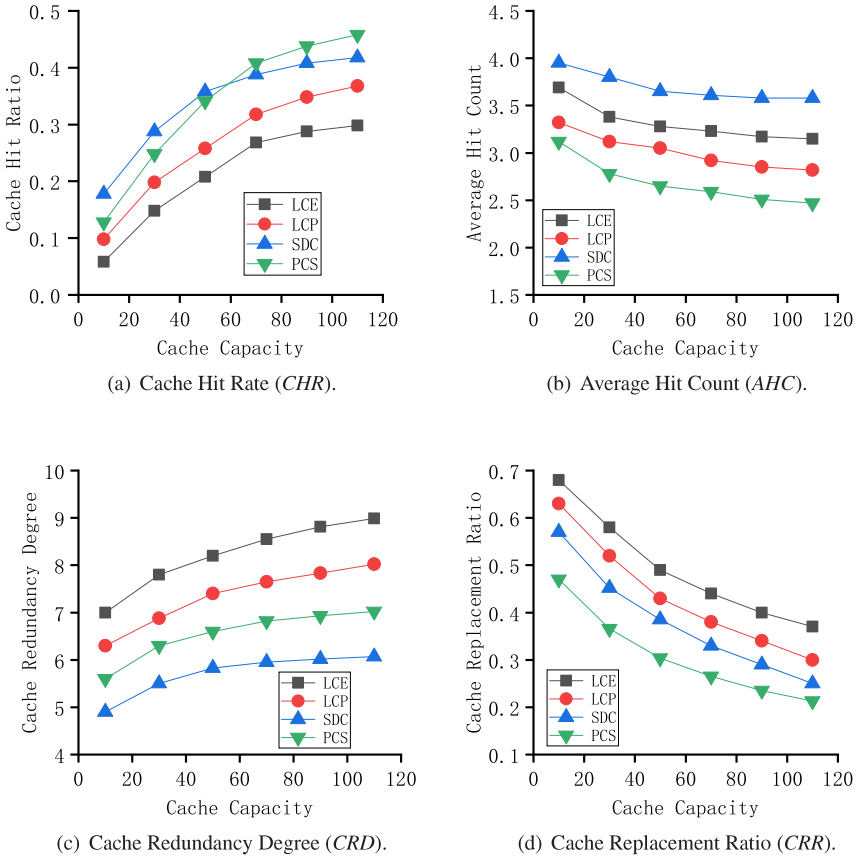
content fragment independently, while the intermediate routers in SDC take caching strategy based on the comprehensive consideration of all routers. Moreover, we actively replace content fragments with higher redundancy degree and lower popularity when too much cache store is occupied. Thus, we can obtain the required content from some intermediate routers with the lowest hops and the most stable caching state. The independent caching strategy, instead of the cooperative caching in SDC, makes our proposed strategy have a higher *CRD*. Furthermore, with the increase of the interest packets sending rate, the corresponding data packets will also increase, CUS will be taken more frequently, since we replace the content fragments with high redundancy degree. That is, if a content fragment has more duplicated caching, it will have a higher probability of being taken out. Thus, our proposed PCS and CUS improve the diversity of cached content, and achieve better *CHR* and *CRD*. For example, when the interest packets sending rate is 50 packets per second, PCS gets 0.312 at *CHR*, which is lower than 0.350 of *SDC*, and when the sending rate increases to 300 packets per second, it becomes 0.347, which is very similar to 0.352 of *SDC*.

As two classic and simple caching strategies, LCE and LCP cache received content everywhere or with a fixed probability, but do not consider the special conditions of nodes and content. Thus, they get worse performances than SDC and our proposed PCS in the *CHR*, *CRD* and *CRR*. They are better than SDC only in *AHC*, because SDC tries to cache more content and there is less same content along the routing paths. Thus, SDC needs the highest *AHC* to get a required content. While LCE caches each received content and the limited cache space makes it take caching and replacing operation very frequently. Hence, it performs worse than LCP in all four metrics.

The performance comparison of different caching strategies with different cache capacities is shown in Figs. 13. The results are very similar to those in Figs. 12; our proposed PCS achieves the best performance in *AHC* and *CRR*, and the second best performance in *CRD*. In *CHR*, when the cache space is small, namely 10, 30, and 50, it achieves the second best performance which is lower than SDC, while with the increase of the cache capacity, PCS becomes the best one. That is because SDC bounds content fragments and the hash of routers ID, namely each router only caches those matched content fragments, and the large cache capacity will not bring an obvious increment of *CHR*. When the cache capacity is small, its *CHR* will increase rapidly, but when it exceeds 60, the *CHR* will be steady. However, other three caching strategies benefit from the increase of cache capacity, and all have a sharp increase. SDC has a lower *CRD* than PCS with an independent caching, because SDC uses cooperative caching. Moreover, LCP and LCE still have the third and the worst performance in all four metrics, respectively.

5. Conclusion

In this paper, we propose C3NDN, an efficient congestion control scheme based on caching for NDN, which consists of the caching strategy PCS, the cache updating strategy CUS, and the congestion control scheme on top of them. C3NDN proposed the One-Interest-Multiple-Data model for NDN, in which one interest packet can trigger multiple data packets, thus saving network bandwidth and improving transmission efficiency. Then, PCS caches received data packets with probability to make popular content cached in important nodes, CUS takes content fragments with less popular and high redundancy out, and C3NDN adjusts the sending rate of data packets by considering the bandwidth and delay information of the transmission path to avoid congestion. Extensive experimental results show that C3NDN can



(a) Cache Hit Rate (*CHR*). (b) Average Hit Count (*AHC*).

(c) Cache Redundancy Degree (*CRD*). (d) Cache Replacement Ratio (*CRR*).

Fig. 13. Performance comparison of different caching strategies with different cache capacities.

achieve better performance than other schemes in terms of reducing transmission time, increasing transmission speed, reducing packet loss and so on, and the deep dive experiments also validate the effectiveness of our congestion control and caching schemes.

CRediT authorship contribution statement

Dapeng Qu: Conceptualization, Writing – original draft, Writing – review & editing, Visualization, Supervision, Project administration, Funding acquisition. **Jun Wu:** Methodology, Software, Validation. **Jiankun Zhang:** Formal analysis, Investigation, Resources, Data curation. **Chengxi Gao:** Writing – original draft, Writing – review & editing, Visualization, Funding acquisition. **Haiying Shen:** Writing – review & editing. **Keqin Li:** Writing – review & editing.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Dapeng Qu reports financial support was provided by National Key Research and Development Program of China. Dapeng Qu reports financial support was provided by National Natural Science Foundation of China. Dapeng Qu reports financial support was provided by Natural Science Basic Research Fund of Liaoning Provincial Education Department. Dapeng Qu reports financial support was provided by Postgraduate Education Reform Project of Liaoning Province. Chengxi Gao reports financial support was provided by the Basic Research Program of Shenzhen. Haiying Shen reports financial support was provided by U.S. NSF grants. Haiying Shen reports financial support was provided by Microsoft Research Faculty Fellowship.

Data availability

No data was used for the research described in the article.

Acknowledgments

This work is supported by National Key Research and Development Program of China under Grant No. 2022YFB4500800, the National Natural Science Foundation of China under Grant No. 62032013, Natural Science Basic Research Fund of Liaoning Provincial Education Department under Grant LJC202002, Postgraduate Education Reform Project of Liaoning Province under Grant LNYJG2022012, the Basic Research Program of Shenzhen under Grant JCYJ20220531100804009, U.S. NSF grants under Grants NSF-2206522 and NSF-1822965, and Microsoft Research Faculty Fellowship under Grant No. 8300751.

References

Abdullahi, I., Arif, A.S.M., Hassan, S., 2015. Survey on caching approaches in information centric networking. J. Netw. Comput. Appl. 56 (1), 48—59.

Alhowaidi, M., Nadig, D., Hu, B., Ramamurthy, B., Bockelman, B., 2021. Cache management for large data transfers and multipath forwarding strategies in Named Data Networking. Comput. Netw. 9 (199), 515–528.

Arianfar, S., Nikander, P., Ott, J., 2010. On content-centric router design and implications. In: Proc. of 2nd Workshop Re-Architect. pp. 1–5.

Din, I.U., Hassan, S., Khan, M.K., Guizani, M., Ghazali, O., Habbal, A., 2018. Caching in information-centric networking: strategies, challenges, and future research directions. IEEE Commun. Surv. Tutor. 20 (2), 1–34.

Gui, Y., Chen, Y., 2020. A cache placement strategy based on compound popularity in named data networking. IEEE Access 8 (1), 196002–196012.

Hashemi, S.N.S., Bohlooli, A., 2021. 3CP: Coordinated congestion control protocol for named-data networking. IEEE Trans. Netw. Serv. Manag. 3 (18), 3918–3932.

Herouala, A.T., Ziani, B., Kerrache, C.A., el Karim Tahari, A., Lagraa, N., Mastorakis, S., 2022. CaDaCa: a new caching strategy in NDN using data categorization. Multimedia Syst. 4 (19), 4932–4947.
Hu, Y., Serban, C., Wang, L., Afanasyev, A., Zhang, L., 2021. BBR-Inspired Congestion Control for Data Fetching over NDN. In: Proc. of IEEE Military Communications Conference. MILCOM, pp. 141–143.
Ioannou, A., Weber, S., 2016. A survey of caching policies and forwarding mechanisms in information-centric networking. IEEE Commun. Surv. Tutor. 18 (4), 2847–2886.
Iqbal, S.M.A., Asaduzzaman, Hoque, M.M., 2022. A source-driven probabilistic forwarding and caching strategy in NDN and SDN-based NDN. Int. J. Commun. Syst. 6 (35), 1–33.
Jacobson, V., Smetters, D.K., Thornton, J.D., Plass, M.F., Briggs, N.H., Braynard, R.L., 2009. Networking named content. In: Proc. of 5th International Conference on Emerging Networking Experiments and Technologies (CoNEXT). pp. 1–12.
Javadtalab, A., Semsarzadeh, M., Khanchi, A., Shirmohammadi, S., Yassine, A., 2015. Continuous one-way detection of available bandwidth changes for video streaming over best-effort networks. IEEE Trans. Instrum. Meas. 1 (64), 190–203.
Kamiyama, N., Murata, M., 2018. Spatially-dispersed caching in information-centric networking. In: Proc. of ICC. pp. 1–6.
Lan, D., Tan, X., Lv, J., Jin, Y., Yang, J., 2020. A deep reinforcement learning based congestion control mechanism for NDN. In: Proc. of IEEE International Conference on Communications. ICC, pp. 1–6.
Lee, C., Nakazato, H., 2020. Congestion control using diffusion method in Named Data Networking. In: Proc. of 45th IEEE Conference on Local Computer Networks. LCN, pp. 333–336.
Li, W., Wang, S., Xu, Y., Lu, S., 2020. Charging on the route: an online pricing gateway congestion control for ICNs. IEEE Trans. Netw. Serv. Manag. 17 (1), 196002–196012.
Mastorakis, S., Afanasyev, A., Zhang, L., 2017. On the evolution of ndnSIM: an open-source simulator for NDN experimentation. ACM Comput. Commun. Rev. 47 (3), 19–33.
Muchtar, F., Abdullah, A.H., Al-Adhaileh, M., Zamli, K.Z., 2020. Energy conservation strategies in Named Data Networking based MANET using congestion control: a review. J. Netw. Comput. Appl. 15 (152), 1–52.
Naeem, M.A., Nor, S.A., Hassan, S., Kim, B.-S., 2019. Compound popular content caching strategy in named data networking. Electronics 8 (7), 771–779.
Naeem, M.A., Ullah, R., Meng, Y., Ali, R., Lodhi, B.A., 2022. Caching content on the network layer: a performance analysis of caching schemes in ICN-based internet of things. IEEE Internet Things J. 9 (9), 6477–6495.
Paul, A.K., Tachibana, A., Hasegawa, T., 2016. NEXT-FIT: available bandwidth measurement over 4G/LTE networks – a curve-fitting approach. In: Proc. of 30th International Conference on Advanced Information Networking and Applications. AINA, pp. 25–32.
Qu, D., Lv, G., Qu, S., Shen, H., Yang, Y., Heng, Z., 2022. An effective and lightweight countermeasure scheme to multiple network attacks in NDN. IEEE/ACM Trans. Netw. 2 (30), 515–528.
Ren, Y., Li, J., Shi, S., Li, L., Chang, X., 2015. An interest control protocol for named data networking based on explicit feedback. In: Proc. of 11th ACM/IEEE Symposium on Architectures for Networking and Communications Systems. ANCS, pp. 190–200.
Ren, Y., Li, J., Shi, S., Li, L., Wang, G., Zhang, B., 2016. Congestion control in named data networking – a survey. Comput. Commun. 86 (1), 1–11.
Rezazad, M., Tay, Y., 2020. Decoupling NDN caches via ccndns: design, analysis, and application. Comput. Commun. 151 (1), 338–354.
Siddiqui, S., Waqas, A., Khan, A., Zareen, F., Iqbal, M.N., 2019. Congestion controlling mechanisms in content centric networking and named data networking – a survey. In: Proc. of 2nd International Conference on Computing, Mathematics and Engineering Technologies (ICoMET). pp. 1–7.
Song, S., Zhang, L., 2021. Exploring rate-based congestion control in NDN. In: Proc. of 8th ACM Conference on Information-Centric Networking. ICN, pp. 141–143.
Vasilakos, A., Li, Z., Simon, G., You, W., 2015. Information centric network: research challenges and opportunities. IEEE Commun. Mag. 52 (1), 1–10.
Wu, F., Yang, W., Muhua Sun, J.R., Lyu, F., 2022. Multi-path selection and congestion control for NDN: An online learning approach. IEEE Trans. Netw. Serv. Manag. 4 (19), 1977–1989.
Yang, J., Chen, Y., Xue, K., Han, J., Li, J., Wei, D.S.L., Sun, Q., Lu, J., 2022. IEACC: An intelligent edge-aided congestion control scheme for named data networking with deep reinforcement learning. IEEE Trans. Netw. Serv. Manag. 4 (19), 4932–4947.
Ye, Y., Lee, B., Flynn, R., Xu, J., Fang, G., Qiao, Y., 2021. Delay-based network utility maximization modelling for congestion control in Named Data Networking. IEEE/ACM Trans. Netw. 5 (29), 2184–2197.
Ye, Y., Lee, B., Qiao, Y., 2020. Hop-by-hop congestion measurement and practical active queue management in NDN. In: Proc. of IEEE Global Communications Conference. GLOBECOM, pp. 1–6.
Zhang, L., Afanasyev, A., Burke, J., Jacobson, V., Claffy, k., Crowley, P., Papadopoulos, C., Wang, L., Zhang, B., 2014. Named data networking. ACM SIGCOMM Comput. Commun. Rev. 44 (3), 66–73.
Zhang, M., Luo, H., Zhang, H., 2015a. A survey of caching mechanisms in information-centric networking. IEEE Commun. Surv. Tutor. 17 (3), 1473—1499.
Zhang, F., Zhang, Y., Reznik, A., Liu, H., Qi, C., Xu, C., 2015b. Providing explicit congestion control and multi-homing support for content-centric networking transport. Comput. Commun. 69 (1), 69–78.

Dapeng Qu received the B.S. degree from the Department of Mathematics, Central South University, in 2003, the M.S. degree from the Department of Information Science and Technology, Central South University, in 2006, and the Ph.D. degree from the Department of Information Science and Technology, Northeastern University, in 2012. He is currently an Associate Professor and an Advisor for master candidates with the College of Information, Liaoning University, China. He has published over 50 academic articles in several journals and conference proceedings. His research interests include future Internet and computer networks.

Jun Wu is currently pursuing the M.E. degree from the College of Information, Liaoning University, China. Her research interests include future Internet and social networks.

Jiankun Zhang received the M.S. degree in computer science and technology from Liaoning University, China, in 2021. His research interests include congestion control and future Internet.

Chengxi Gao (Member, IEEE) is an assistant professor at Shenzhen Institute of Advanced Technology (SIAT), Chinese Academy of Sciences (CAS). Before joining CAS, he was a research associate in City University of Hong Kong. He received his Ph.D. degree from the Department of Computer Science, City University of Hong Kong, and his B.S. and M.S. degrees from the Department of Computer Science, Northeastern University, China. His research interests include data center networking and distributed machine learning system.

Haiying Shen (Senior Member, IEEE) received the B.S. degree in computer science and engineering from Tongji University, China, in 2000, and the M.S. and Ph.D. degrees in computer engineering from Wayne State University in 2004 and 2006, respectively. She is currently an Associate Professor with the Department of Computer Science, University of Virginia. Her research interests include distributed computer systems, cloud and edge computing, machine learning, big data, and cyber–physical systems. She is a Microsoft Faculty Fellow of 2010 and a Senior Member of IEEE.

Keqin Li (Fellow, IEEE) is a SUNY Distinguished Professor of computer science with the State University of New York. He is also a National Distinguished Professor with Hunan University, China. His current research interests include cloud computing, fog computing and mobile edge computing, energy-efficient computing and communication, embedded systems and cyber–physical systems, heterogeneous computing systems, big data computing, high-performance computing, CPU–GPU hybrid and cooperative computing, computer architectures and systems, computer networking, machine learning, intelligent and soft computing. He has authored or coauthored over 840 journal articles, book chapters, and refereed conference papers, and has received several best paper awards. He is currently an associate editor of the ACM Computing Surveys and the CCF Transactions on High Performance Computing. He has served on the editorial boards of the IEEE Transactions on Parallel and Distributed Systems, the IEEE Transactions on Computers, the IEEE Transactions on Cloud Computing, the IEEE Transactions on Services Computing, and the IEEE Transactions on Sustainable Computing. He is an IEEE Fellow.