CCNHCaching: A High-speed Caching Throughput Simulator for Information-Centric Networks

Haifeng Li, Huachun Zhou, Wei Quan, Bohao Feng, Hongke Zhang

Institute of Electronic and Information Engineering, Beijing Jiaotong University, China
{haifengli, hchzhou, weiquan, bohaofeng, hkzhang}@bjtu.edu.cn

Abstract

Information-Centric Networks (ICN) have attracted great research interests in recent years. Due to the prominent feature of in-network caching, ICN can reduce network latency and improve network throughput. Many researches boost the efforts on the caching simulators. However, the throughput of caching is fundamental to wire-speed forwarding for ICN routers, and there is no corresponding ICN simulator for caching throughput evaluation. In this paper, we propose a caching throughput simulator named CCNHCaching. A reading/writing request queue mechanism is proposed to simulate sequential memory accesses. Combining this mechanism with open-source memory simulators, CCNHCaching can implement precise caching throughput and network performance simulation. Besides, we propose a requesting algorithm to generate realistic high-speed requesting traffic. We compare several state-of-the-art caching schemes and evaluate their caching throughput. The results show that CCNHCaching can provide better evaluation for caching schemes, hence, promoting the development of caching mechanisms.

Keywords: ICN, High-speed, Caching throughput, Simulation

1 Introduction

In recent decades, the Internet has evolved from an academic network into a global cyber. With the tremendous growth, enormous network applications such as 8K Video emerged to fulfill various needs. Rich and diverse needs gradually exposed the design problems of the Internet. Current Internet architecture is based on host-to-host communication model. It is inefficient to meet the demands such as information sharing and content distribution. To solve these problems, many researches start to integrate emerging arts into the design of future Internet [1-2]. A new clean-slate network architecture, named Information-Centric Networking (ICN) is proposed [3-4]. ICN uniquely names contents and equips routers with caching capability. Hence, it can cache contents along their propagation path; The receivers can fetch contents from the nearest caching routers rather than the original provider, which reduces the network traffic and access latencies and enhances user experience.

In-network caching has many advantages, and there are lots of caching optimization mechanisms proposed to improve them [5-7]. One of classic caching optimization mechanisms is the Least Recently Used (LRU) [8]. It keeps track of what was used when which is expensive if one wants to make sure the mechanism always discards the least recently used item. LRU has a good robustness for danymic content popularity, and a wide range of applicatios. Thomas *et al.* proposed an Object-oriented Packet Caching (OPC) to enlarge the caching capacity without additional storage resource [9]. Furthermore, Rossini *et al.* designed a novel two-layer cache scheme primarily with DRAM and SSD (DRAM_SSD) to achieve multi-terabyte caching capacity and multi-Gb/s data rate [10]. Besides, our previous works have contributed to the large-capacity and high-speed caching mechanism [11]. We proposed a hierarchical caching scheme composed of both SRAM and DRAM. It accelerates the caching throughput for ICN core routers. All of these works needed the development of ICN simulators.

To evaluate the above caching mechanisms, many ICN simulators have been proposed. ndnSim [12] is an ns-3 based simulator which provides caching simulation for typical cache replacement policies, such as LRU (Least Recently Used), FIFO (First In First Out) and LFU (Least Frequently Used). CCN-lite [13] is a lightweight implementation of NDN protocols. It supports simulator mode based on the OMNET++ framework and emulation mode which can run on multiple embedded devices. In emulation mode, CCN-lite can evaluate the caching throughput performance of caching schemes; while, due to the hardware constraint, the link rate would be usually low.

ICN lacks caching throughput simulator. Current ICN simulators only can provide the simulation for cooperative cache policies. They can evaluate their network performance, such as hit ratio, the reduction of network traffic and server workload. However, how to

*Corresponding Author: Haifeng Li; E-mail: haifengli@bjtu.edu.cn
DOI: 10.3966/160792642019052003005

evaluate the high-speed caching throughput performance is not investigated yet. The wire-speed forwarding is a fundamental issue in ICN routers, whose bottleneck is the throughput of caching schemes. Therefore, caching throughput simulation is important. In this paper, we propose an ICN simulator for high-speed caching throughput simulation, called CCNHCaching. It can simulate a caching scheme in multiple aspects, including network performance and throughput performance. Specially, to simulate the caching throughput, we elaborately design the Content Store (CS) simulation component in CCNHCaching, which is composed of a reading/writing request queue manager, caching schemes and memory simulators. The main contributions can be summarized as follows:

- We design and implement preliminarily an ICN simulator, called CCNHCaching [1], which can simulate caching schemes regarding network performance (e.g., hit ratio) and caching throughput. Especially, it supports the caching throughput simulation for high-speed (such as 100 Gb/s) caching schemes.
- To simulate the caching throughput exactly, a reading/writing request queue management is proposed. Besides, combining this mechanism with memory simulators, we implement multiple leading caching schemes.
- To generate realistic high-speed requesting traffic, we investigate the real request behaviors of the twenty thousand users in Beijing Jiaotong University thoroughly and design a high-speed requesting algorithm carefully.
- We compare the state-of-the-art caching schemes including LRU, OPC, DRAM_SSD, and HCaching, and evaluate the performance of CCNHCaching. The results demonstrate that CCNHCaching can provide more comprehensive and rigorous evaluation for caching schemes, hence, favoring the large-scale development of the caching schemes in ICN.

The remainder of this paper is organized as follows. Section 2 overviews related works. Section 3 describes the CCN system model simply. Section 4 presents the architecture of CCNHCaching, including the overall design, the caching throughput simulation, and the high-speed requesting algorithm. In Section 5, we compare several leading caching schemes and evaluate the performance of CCNHCaching. The conclusions and future works are given in Section 6.

2 Related Work

With the ICN development, there have been large amounts of emerging tools to evaluate proposed ICN mechanisms. We can categorize them into three groups: (1) ICN Simulators [12-14], (2) ICN Emulators [15-19] and (3) ICN Testbeds [20-21].

(1) ICN Simulators almost all are based on the discrete event simulation, allowing researchers to rapidly prototype and test their ICN proposals.

The ndnSim [12] is a simulator based on ns3 for the Named-Data Networking (NDN). By integrating the Named Data Networking Forwarding Daemon (NFD) and ndn-cxx library with ns-3, ndnSim provides a realistic NDN simulation behavior. The ndnSim implements the full-featured processing of NDN selectors based on the latest NDN packet format. However, ndnSim has limited support for caching simulation. Although there are several implementations of typical cache replacement policies (e.g., LRU (Least Recently Used), FIFO (First In First Out) and LFU (Least Frequently Used)), it does not concentrate on high-speed caching throughput simulation.

CCN-lite [13] is a lightweight yet functionally interoperable implementation of the CCNx and NDN protocols. It supports a simulation mode using OMNET++ simulation framework and an emulation mode based on Linux, Android, Arduino *et al.* platforms. CCN-lite offers a clean packet scheduler to support at chunk-level, packet-level, and fragment-level. Packet fragmentation supports running the CCNx protocol natively over Ethernet. However, CCNx-lite focuses on providing a lean alternative for educational purposes and a tiny CCNx core (1000-2000 lines of C language) for embedded devices. It is not optimized for high performance with sophisticated data structures.

ccnSim [14] is a scalable chunk-level simulator for Content-Centric Network (CCN). It is a C++ package built on the top of Omnet++ framework. To efficiently simulate large-scale CCN mechanisms, Chiocchetti et al. develop and optimize ccnSim in the respects of memory- -occupancy and CPU-time. However, ccnSim focuses on dealing with different caching cooperative algorithms or policies and cannot evaluate the throughput of caching schemes.

(2) ICN Emulators are another way to evaluate ICN architecture, which can verify ICN schemes with the real operating system and the whole protocol stack.

CCN-Joker [15] is a lightweight Java-based CCN emulator. It is designed to run on top of limited resource wireless devices, and study CCN performance in mobile ad hoc network or vehicular ad hoc network. CCN-Joker is a basic implementation of CCN, and it primarily focuses the CCN application on wireless networks.

CCNx [16] is the emulation implementation for ICN, which provides the libraries and components to demonstrate the basic CCNx protocols. Since CCN proposes evolutions in the traditional Internet, CCNx mainly focuses on the interoperability of new protocols layers. Alternative similar emulation implementation is NDNx [17], which is a fork of CCNx. CCNx and

[1] The source code can be available at https://github.com/iplab-code/ccnhcaching.

NDNx concern heavily about security issues, end-node and router design, while less attention is paid to caching operations.

Mini-CCNx [18] is another ICN emulator, which extends the Mininet-HiFi [19] with CCNx implementation. Due to the lightweight Linux Container (LXC) techniques, Mini-CCNx can run CCN emulation with a large-scale topology. However, Mini-CCNx is based on NDNx which an outdated model of NDN is. Besides, Mini-CCNx mainly pays attention to emulating the emulation hardware without regarding the throughput evaluation of caching schemes.

(3) ICN Testbeds are real hardware infrastructures that can provide real test scenarios for ICN. The Global Environment for Network Innovations (GENI) [20] is a novel suite of infrastructure which is designed to support network experiments. EmuStack [21] is a testbed based on OpenStack. It emulates various network architectures, such as ICN and SINET [22-23], with overlay networking technology. Network and computing virtualization make EmuStack flexible and programmable to emulate experimental protocols. However, due to limitation of physical hardware resource, they all cannot emulate high-speed (such as 100 Gb/s) caching throughput due to the limited hardware.

The caching throughput in ICN is crucial to the wire-speed forwarding in ICN routers. Unfortunately, there is a lack of ICN simulators which specially focus on the high-speed caching throughput simulation. Although these ICN emulators/testbeds including CCN-Joker, CCNx/NDNx, Mini-CCNx and EmuStack can implement the emulation of caching throughput, they are expensive and seriously constrained by the limited hardware resource. Therefore, in this paper, we propose a caching throughput simulator for the high-speed caching schemes in ICN, named CCNHCaching, to further promote the ICN caching mechanisms development.

3 CCN System Model

CCNHCaching is proposed to simulate caching schemes for CCN, a specific architecture design under the broad ICN umbrella. We outline the CCN system model in this section.

In CCN, communications are driven by the content receivers. There are two types of packets: *interest packet* and *data packet*. The content is named uniquely and requested by receiver with interest packet. The nodes with the requested content respond interest packets with data packets. On-path routers forward and cache the data packets. There are three primary components in routers: *Forwarding Information Base (FIB), Content Store (CS)* and *Pending Interest Table (PIT)*. The FIB is a table that maintains the relationships between prefixes of content names and interfaces in CCN routers. The CS stores the requested contents after forwarding them so that it can respond the following requests quickly. The PIT keeps track of the interest packets that have been requested recently but not responded yet. The interest packets recorded in the PIT are dropped and not forwarded.

The packet flow in this model is presented in Figure 1. Upon reception of an interest packet on an interface *I*, a lookup operation is issued to the CS, checking for whether the requested content is cached in the CS (the first step in Figure 1). If the content is available, the CCN router reads the content from the CS and sends it back on *I* (the eighth step in Figure 1). Otherwise, the CCN router checks in the PIT whether the requested content has been already forwarded upward (the second step in Figure 1); if an entry is found in the PIT, the CCN router updates the entry to track that the interface *I* is waiting for the requested content. If the entry is not found, a new entry is added and the interest packet is forwarded to at least one interfaces based on longest prefix match on the content name prefix in the FIB (the third and fourth steps in Figure 1). When a data packet is received after a Round-Trip Time (RTT) (the fifth step in Figure 1), the CCN router checks for the pending requests in the PIT. If the entry is found, the data packet is stored in the CS and forwarded toward all the requesting interfaces as listed in the PIT (the seventh and eighth steps in Figure 1); otherwise, the data packet is discarded (the sixth step in Figure 1).

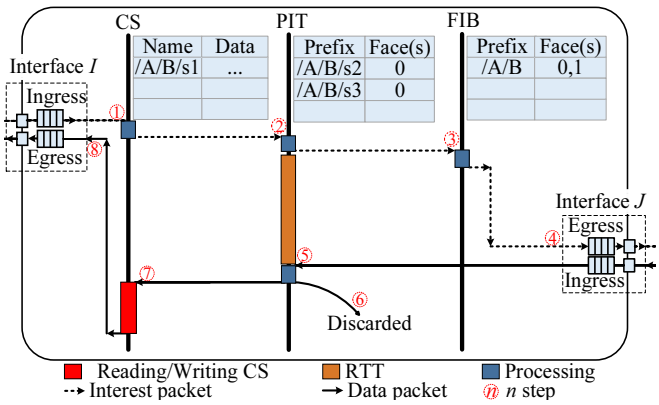


Figure 1. Packet flow in the CCN system model

As shown in Figure 1, the delays in the packet flow are composed of three types: reading/writing CS delay, Round-Trip Time (RTT) and processing delay. For the caching throughput simulation, we disregard the packet processing delays, focusing on the reading/writing CS latency, which is the main bottleneck. The reading/writing CS latency directly determines the caching throughput which is equal to the transmitted data size divided by the latency.

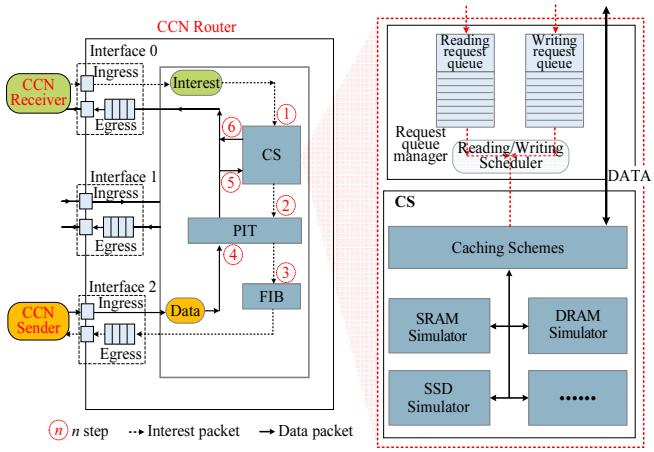
4 Simulator Architecture

In this section, we discuss the design details in CCNHCaching. First, we introduce the design architecture to explain what the difference of

CCNHCaching is. Second, we discuss in detail the implementation of high-speed caching throughput simulation, presenting why and how CCNHCaching can realize the caching throughput simulation. Third, a high-speed requesting algorithm is proposed to generate the high-speed realistic simulation traffic.

4.1 Design Overview

Figure 2(a) illustrates the overall structure of CCNHCaching. In essence, it is a C++ module built on the top of the *ns-3* which is a discrete-event network simulator framework towards network research. CCNHCaching keeps track of some events in an ns3 event queue (NSEQ), which are scheduled to execute at a specified simulation time. The job of the simulator is to execute the events in sequential time order. Once the completion of an event occurs, the simulator will move to the next event or will exit if there are no more events in the event queue. Note that the framework implements the full-duplex network interfaces without ingress queues as shown in Figure 2(a). Therefore, it only indirectly controls the receiving rate by controlling the end-to-end transmitting rate with egress queue.



(a) overall structure (b) CS component

Figure 2. Implementation of CCNHCaching

There are three types of nodes in CCNHCaching, namely, CCN routers, CCN receivers and CCN senders. The CCN router is mainly composed of three components: *Content Store (CS), Pending Interest Table (PIT) and Forwarding Information Base (FIB)*. The CCN receiver and CCN sender consist of PIT and FIB components. These components play the same roles in the CCN system model described in Section 3. The details of the three nodes are described as follows.

The CCN receiver represents users to request contents according to the given traffic trace data. The traffic trace data can be created by traffic generator or formatted with the data set collected from actual network traffic. The CCN receiver is linked to the access router, which only has a neighbor router. One access router can connect amounts of CCN receivers where the requesting algorithm runs to send interest packets at a customizable rate. The requesting algorithm is key to generate high-speed link throughput, which will be further discussed in Subsection 4.3.

The CCN router is the crucial node to simulate the throughput of caching schemes. As shown in Figure 2, the CCN router aggregates interest packets with the PIT and forwards the unrequested interest packets based on the FIB (the second and third steps in Figure 2). Upon the arrival of data packets, the CCN router forwards them by the PIT and stores them into the CS (the fourth and fifth steps in Figure 2). Hence, when the cached data packets are requested again, the CCN router can respond directly with the content that has been cached in the CS (the first and sixth steps in Figure 2).

The CCN sender is an aggregation of content providers. It is responsible for responding interest packets with the data packets. In CCNHCaching, content providers are not CCN nodes and do not contain the whole CCN stack. In fact, all of the content providers reside in a CCN sender, serving all content requests without regard to whether the requested contents exist.

In CCNHCaching, there are three different cache placement policies, including edge caching, universal caching and caching based on betweenness centrality. In the edge caching, caches are placed only at the access CCN routers in the network. In the universal caching, all CCN routers are equipped with the CS and allowed to cache data packets. In the betweenness centrality caching, all CCN routers can cache data packets, but data packets are stored at the on-path routers with the highest betweenness centrality [24].

In CCNHCaching, the most important feature is supporting the high-speed caching throughput simulation. As shown in Figure 2(b), we design a request queue simulation mechanism to manage reading/writing requests for the CS. We combine this mechanism with various memories simulators, such as DRAM simulator and SSD simulator, to provide the exact access time for different reading/writing operations. Besides, between the request queue manager and memories simulators, the caching schemes can be implemented. Currently, there are four leading caching schemes which have been implemented in CCNHCaching, including LRU, OPC, DRAM_SSD, and HCaching.

4.2 Caching Throughput Simulation

In CCNHCaching, the structure of the CS is elaborately designed to support the throughput simulation for caching schemes. As shown in Figure 2(b), at a high level, this key structure is composed of three parts: (1) request queue manager, (2) caching schemes and (3) memory simulators.

(1) Request queue manager is composed of three

parts: *reading request queue*, *writing request queue* and *reading/writing scheduler*. The reading request queue maintains the information about users requesting the content in the CS. The writing request queue tracks the requests for storing the forwarded data packets into CS. The reading/writing scheduler is responsible for how to schedule these reading/writing requests to access the CS.

The process flow of request queue manager is as follow. Once the request queue manager receives a reading request or writing request, it enqueues them to the reading request queue or writing request queue, respectively. Assume we use the single port memory chips in the CS, which cannot support simultaneous read and write operations. We use a polling mechanism to schedule the reading and writing request queues in reading/writing scheduler. Therefore, the CS serves the reading requests and writing requests alternately. When a reading request arrives in the CS, the CS calculates its access latency based on the cache schemes and memory simulators. The CS adds the latency to the current simulation time, getting the specified simulation time t_e. It uses t_e as the executing time of the reading request end event and puts the request end event into the event queue (NSEQ) in *ns-3*, waiting t_e for being executed. After the reading request end event is executed, the request queue manager enables the writing request to hit the CS. If the writing request queue is empty, the request queue manager immediately allows the reading request queue to access the CS again; otherwise, it needs to wait for executing the writing request end event and then enable the reading request.

The request queue manager is the key to simulate the throughput of the CS exactly. This mechanism makes the access latency have the accumulative impacts on the subsequent access requests. However, when the speed of the CS processing requests is less than the speed of the requests arrival, these accumulative impacts would make the reading/writing queues exceeded. Therefore, the size of the request queue should be configured according to the ICN congestion control algorithm and the requesting algorithm, which are further discussed in Subsection 4.3.

(2) Caching scheme is responsible for organizing memory structure and managing the cached content, to improve the CS performance. Currently, we implement four leading caching schemes in CCNHCaching, which include LRU, OPC, DRAM_SSD, and HCaching.

LRU [8] is the Least Recently Used caching scheme. It uses Static Random-Access Memory (SRAM) as index memory, and Dynamic Random-Access Memory (DRAM) as the primary one. The LRU algorithm runs at the SRAM, to index and replace the content cached in DRAM. It keeps track of what was used and when, and always discards the least recently used item. LRU algorithm is theoretically realizable. However, it is not cheap. To fully implement LRU algorithm, it is usually necessary to maintain a linked list of all data packets cached in DRAM, which would consume large amounts of SRAM resources.

OPC [9]0 is an Object-oriented Packet Caching scheme. It utilizes SRAM as the index memory, and DRAM as the primary one. Currently, SRAM is expensive and size-limited, which restricts the caching performance for the commodity routers seriously. Therefore, Thomas *et al.* proposed the OPC to overcome this SRAM bottleneck. OPC combines the object-level indexing in SRAM with the packet-level storage in DRAM. It can increase the usable DRAM capacity for the commodity routers without additional SRAM resources. Besides, it introduces several simple yet effective algorithms for the content lookup, insertion, and eviction operations. These algorithms can address some caching problems such as the looped replacement and the large object poisoning. However, there is some side effect of these algorithms. For example, when the cached content size is large, they would result in the sharp decline on the OPC's caching throughput.

DRAM_SSD [10] is a two-layer caching scheme which utilizes DRAM as cache memory and Solid-State Drives (SSD) as primary one. The index memories in DRAM_SSD are composed of SRAM and DRAM. SRAM holds the index for the data packets cached in DRAM, and a part of DRAM maintains the index for the data packets stored in SSD. By the novel two-layer caching structure, DRAM_SSD succeeds in moving SSD bottleneck from access time to the external data rate. Besides, with a triggering prefetching mechanism, DRAM_SSD utilizes the high-speed external data rate in SSD fully. Therefore, it can achieve multi-Terabyte caches to sustain content streaming at multi-Gb/s speed. DRAM_SSD can be well qualified as the edge ICN routers. However, due to the writing characters and lifetime of the general SSD chips, it probably is not fast and endurable enough to serve the core ICN routers.

HCaching [11] is a hierarchical caching scheme which uses Static Random-Access Memory (SRAM) as cache memory and DRAM as primary one. HCaching adopts a two-layer structure, which maps a content chunk into a single row of DRAM and employs a little SRAM as a cache of a DRAM to accelerate DRAM. This design leverages the merits of both SRAM and DRAM and make an optimal trade-off between them. Besides, HCaching proposes a prefetching strategy is proposed to reduce access latency to maximize throughput. When a chunk request is received, HCaching can proactively fetch chunk-size packets from DRAM into SRAM in batches. Furthermore, an improved A^2 buffering algorithm [25] is adapted to index the cached content-chunks in DRAM efficiently. With all these combinatorial peculiarities, HCaching has great potential to reduce excessive utilization rate of SRAM

and improve total caching efficiency.

(3) Memory simulators are responsible for providing the exact access latency for various memory technologies. Typically, the caching schemes are composed of multiple types of memories. For example, the DRAM_SSD caching scheme mainly consists of two types of memories architectures: DRAM and SSD. These memory architectures are complex, and it is hard to evaluate their performance. Therefore, the memory simulators are utilized to provide a fast and early performance estimation for the customized caching systems. Fortunately, there have been lots of memory simulators in the memory community, such as SRAM simulator, DRAM simulator, and SSD simulator.

SRAM simulator is responsible for providing the access time of SRAM. One cell of SRAM is composed of six transistors. The all-transistor structure makes its state very stable, and the access dealy is short, and the access timing of different access modes remains unchanged. The SRAM has a simple interface, so the SRAM simulator is simple to implement.

DRAM simulator is responsible for providing the access time of DRAM. DRAM uses the two states of the capacitor (charged or discharged) to represent information, However, capacitor has a leakage effect. If no periodic refresh (supplementary charge) is performed, its state will change and the information will fade quickly. This feature leads to DRAM chip needing a very complex design. The access latency is determined by its request modes. To simulate the access latency, Wang et al. [26] implemented DRAMsim. It can simulate various DRAM technologies, including SDRAM, DDR, DDR2, DRDRAM, and FB-DIMM. DRAMsim provides access delay according to different parameters and access modes. Besides, Kim et al. [27] designed a fast, accurate, scalable DRAM simulator, Ramulator. Ramulator provides many out-of-the-boxes supports for DRAM technology standards, including DDR3/4, LPDDR3/4, GDDR5, WIO1/2, and HBM.

SSD simulator is responsible for providing the access time of SSD. SSD is based on flash technology. There are two types of flash: NAND and NOR. NAND flash has higher storage density and lower power consumption than NOR flash. Therefore NAND-based SSD is used widely. NAND-based SSDs support three operations: reading, writing, and erasing. With different operations, the granularity and delay are different. The basic storage unit of SSD is page, and its size is 2KB-16KB. 128 or 256 pages form a block, hence each block size is 256KB-4MB. The minimum unit of reading/writing operations in SSD is page. However the minimum unit for erasing operation is block. If you need to rewrite a page, you first need to erase the entire corresponding block where the page resides. Generally, the latency of writing operations is 4-5 times longer than that of reading operations. The delay of erasing operation is much greater than that of writing operations. Therefore, as data storing SSD increases, its performance becomes worse and worse. Additionally, the writing cycle of SSD is between 3K and 100K, which is very limited for a network cache system that frequently writes data. These SSD features will affect the design of the caching mechanism. To accurately assess its impact on the performance of the cache mechanism, SSD emulators such as Flashsim [28] need to be used.

4.3 High-speed Requesting Algorithm

To evaluate the high-speed throughput for caching schemes, a requesting algorithm is needed to generate the high-speed realistic network traffic. We first investigated the users' request characteristics at the export gateway in Beijing Jiaotong University. There are twenty thousand students and teachers, and the export link peak is close to 5 Gb/s. We summarize the characteristics of the real user requests as follows:

(1) The user requests have a great redundancy. With the twenty seconds sampling period, the redundancy of the requests is up to forty percent.

(2) The user requests overlap each other. The phenomenon is presented in Figure 3, where the circles stand for different users' requests and their diameters denote the life-time spans of these users' requests. In the real network traffic, the user requests blend each other, rather than follow one after another.

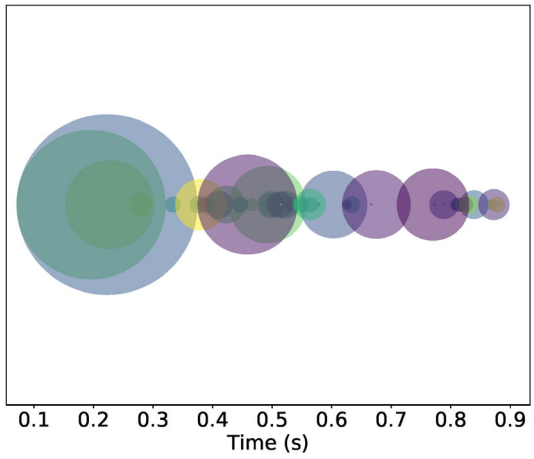


Figure 3. Requesting time sequence of realistic network flows (circles stand for different network flows, and their diameters denote the life-time spans of these network flows)

Based on the above observation, we design a high-speed requesting algorithm to simulate the realistic users' requests. The request process is outlined in Figure 4. We first generate the "traffic trace data" with GlobeTraff traffic generator [29] or collected real network traffic. The "request array" is employed to send the interest packets. It loads n contents to request from the "traffic trace data", and then send the interest packet of them alternately, with the rate v. The values of both n and v can be set according to user

requirement. When request packets for a content is sent completely, the requesting algorithm pushes a new content from the "traffic trace data", until all the contents in the "traffic trace data" are requested. The general flow of the requesting algorithm is as follows: First, "n" content is pushed into the "request array" from "traffic trace data", and then interest packets of n content are sent alternately with rate v; if all requests of a content are sent, the content is removed from "request array" and a new content is loaded from "traffic trace data".

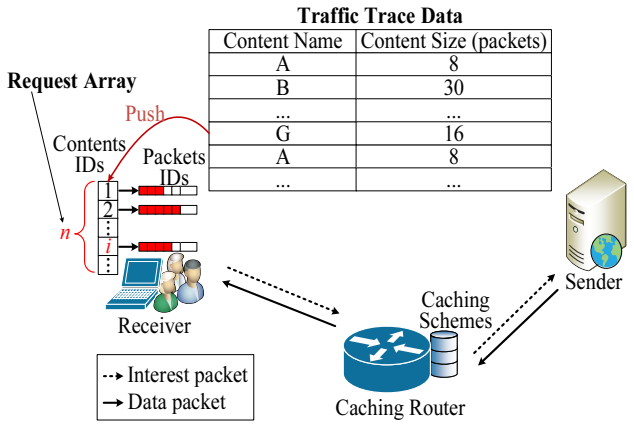


Figure 4. Simple topology to represent the way to send users requests

The detail of the requesting algorithm is described in Algorithm 1. Let i denotes the index number of the i-th content in the "request array". The algorithm first initializes its value to zero and then fills the "request array" with n contents from the "traffic trace data" (lines 1 to 8). After that, it checks whether the "request array" is empty; if it is, it means that all contents are requested, and the algorithm ends (lines 9 to 11); otherwise, it assembles the interest packet p_i of the i-th content (a content consists of multiple packets) in the "request array" (lines 14, 15). The function also checks whether p_i is the last interest packet in the i-th content. If it is, the function removes the i-th content from the "request array" (lines 16 to 18). After that, the algorithm sends the interest packet with rate v (line 20), and the value of i is increased by one (line 21). When the value of the increased i is equal to n, we reset it to zero (line 22 to 24). Finally, it continues to request the data packet for the next content in the "request array", until both "traffic trace data" and "request array" are empty.

In the preceding algorithm, we can adjust the value of n to control the number of parallel flow, and the value of v to control the rate of requesting traffic flexibly. Besides, to prevent packet loss in network, a congestion control mechanism with an acknowledgment would be needed. However, for simplicity, we can set the size of the reading/writing request queues as infinity, hence, eliminating the impacts of various congestion control algorithms.

Algorithm 1: The high-speed requesting algorithm.

```
   Input  : i
   Output: None.
1  i ← 0
2  send Interest Packet(i):
3  while the size of requesting array is less than n do
4      if trace data is empty then
5          break
6      end
7      push a content into requesting array
8  end
9  if requesting array is empty then
10     return
11 end
12 p ← none
13 while p is none do
14     get the ith content from requesting array
15     p ← a interest packet of the ith content
16     if interest packets about the ith content all were sent then
17         delete the ith content from requesting array
18     end
19 end
20 send p with the rate v
21 i ← i + 1
22 if i is equal to n then
23     i ← 0
24 end
25 send Interest Packet(i)
```

5 Evaluation

In this section, we test the functionality and evaluate the performance of CCNHCaching in a server. This server is an identical Dell™ PowerEdge™ R720 2U rack server with one 2.4GHz Intel Xeon E5-2609 processor, 10M of L3 cache per core, 32 GB RAM, and Broadcom 5720 Quad Port 1GbE BASE-T. Without loss of generality, we demonstrate the throughput simulation with three nodes: a receiver, a caching router, and a sender, as shown in Figure 4. The link delays are set as 5 ms. Besides, to achieve a realistic traffic mix with variable content size and popularities, we use GlobeTraff [29] to create the traffic trace data. The generating parameters are similar to those in [9].

5.1 Redundancy

Link throughput has an influence on traffic redundancy, which affects the network performance such as hit ratio. For a given traffic trace data, the redundancy is related to the caching throughput since CCN has the aggregating functionality. The PIT keeps track of the interest packets that have been requested recently but not responded yet. The coming interest packets recorded in the PIT will be dropped and not forwarded by CCN routers. This leads to that the coming time interval of the same requests would be affected by the link throughput.

Figure 5 presents the relationship between

redundancy and link throughput. Note that, to eliminate the caching throughput's impact on link throughput, we disable the caching capability. Besides, the link bandwidth is set as 100 Gb/s, and the requesting algorithm is configured to achieve different link throughputs which range from 10 Mb/s to 100Gb/s. When the link throughput is 10 Mb/s, the traffic redundancy is close to the trace data redundancy, namely, fifty-eight percent. With the increasing link throughput, the network traffic redundancy falls off. This is because the high-speed link throughput shortens the sending time interval of the redundant requests, and thus more requests are aggregated in CCN nodes. Therefore, the link throughput decreases the traffic redundancy, influencing the network performance of caching schemes such as hit ratio. It is greatly important to support the caching throughput simulation if we want to evaluate a caching scheme comprehensively and correctly.

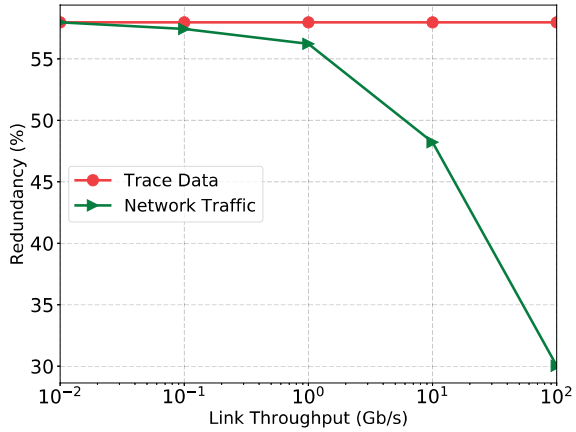
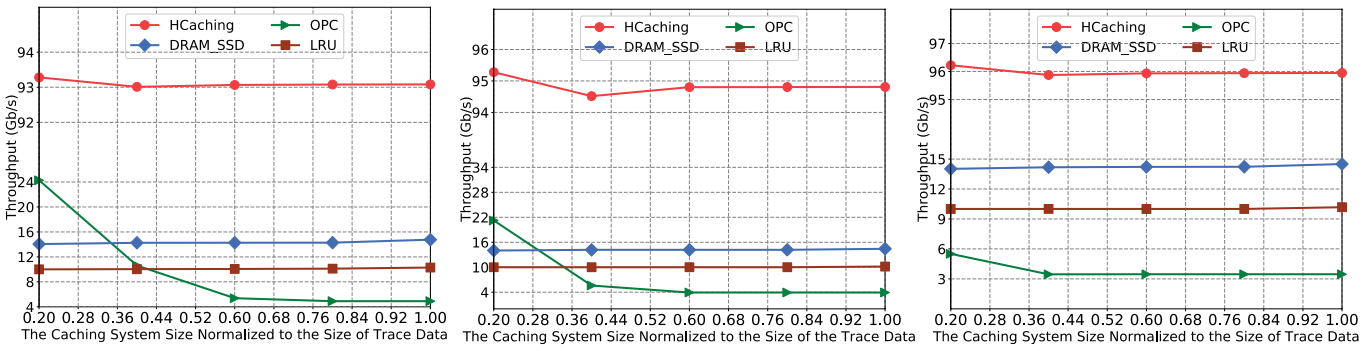


Figure 5. The relationship between redundancy and link throughput

5.2 Caching Throughput Comparisons

In CCNHCaching, the most important feature is supporting the caching throughput simulation. We replicate four state-of-the-art caching schemes in CCNHCaching, including LRU, OPC, DRAM_SSD, and HCaching. We now compare them regarding the caching throughput, demonstrating the importance of the caching throughput simulation. First, we used GlobeTraff to generate three types of trace data, which have different average content sizes: 9KB, 13KB, and 19KB. Of course, because of functional tests only, these average content sizes are not optimally selected. Besides, the value of v in the requesting algorithm is configured to achieve the 100 Gb/s network traffic.

Figure 6(a), Figure 6(b) and Figure 6(c) show the throughput performance of each caching schemes when the average content is 9KB, 13KB, and 19KB respectively. For HCaching, it has good support for the caching throughput which is stable and up to 100 Gb/s. For DRAM_SSD and LRU, the throughputs are more than 10 Gb/s. Besides, with caching capacity increasing, the throughput slowly increases. For OPC, the throughputs are less than 10 Gb/s. Meanwhile, as the capacity of cache system increases, its throughput decreases drastically; as the average size of the content increases, its throughput also drastically decreases. The specific reasons for exploring these phenomena are beyond the scope of this chapter, and explained in our another paper [11]. Obviously, CCNHCaching can support the high-speed caching throughput simulation well. With this support, researchers can evaluate the advantages or disavantages of caching mechanisms more comprehensively, hence, designing them better.



(a) 9KB (b) 13KB (c) 19KB

Figure 6. Comparision of the caching throughput in the caching schemes, when the average sizes of contents are

5.3 Simulator Performance

In this subsection, we compare the performance which consists of memory occupancy and the execution time, with the different number of requests.

Figure 7 depicts the memory occupancy versus the requests number. The size of the caching schemes is configured as the sum of the independent contents in the trace data. The caching schemes consume a different number of memories. This is because different caching schemes have individual cache data structures, which require the different number of

memories. Besides, when the number of requests is 10^7, the largest memory occupancy is about 3 GB for DRAM_SSD, and the lowest one is about 500 MB for None (disabling caching capability). With the increasing number of requests, these values of memory occupancies rise up slowly. Therefore, the memory occupancy in CCNHCaching always keeps on a low level, and CCNHCaching has a good performance regarding memory occupancy.

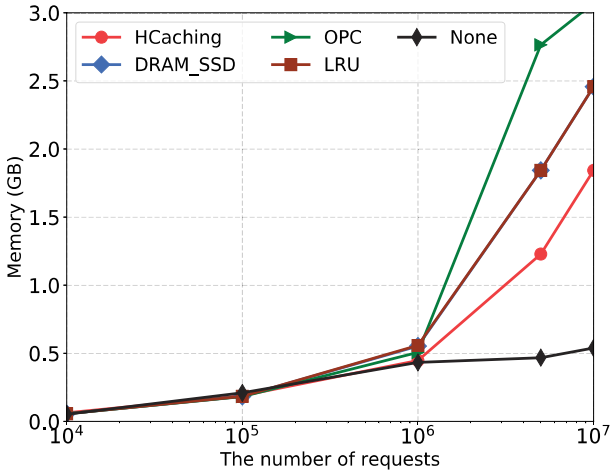


Figure 7. Memory occupancy versus the number of requests

Figure 8 presents the execution time in CCNHCaching. We can decompose the total execution time as the sum of *bootstrap* time t_B, and the simulating time t_{sim} which is spent in running the CCN node dynamically. The t_B is composed of the times such as filling the catalog, allocating data structures and building the simulation network. Its value is nearly a constant, and just with little relevance to the number of requests. The t_{sim} is the CPU time which is required to run various processing programs such as the indexing algorithms and memory simulators. Its value is associated with the time complexity of these programs. Therefore, except the number of requests, the execution times are determined by the time complexity of the caching schemes primarily. For example, HCaching uses the Bloom Filter to index the cached content, which is complexity, thus the execution time of HCaching is slightly longest in all the caching schemes. However, the longest execution time is just more than 1 hours for 10^7 requests, which is acceptable completely.

6 Conclusion and Future Work

This work proposes and evaluates an ICN simulator, called CCNHCaching, which supports high-speed caching throughput simulation. Through presenting the basic CCN system model, we introduce the overall design of CCNHCaching. The reading/writing request queue manager is designed carefully to simulate the

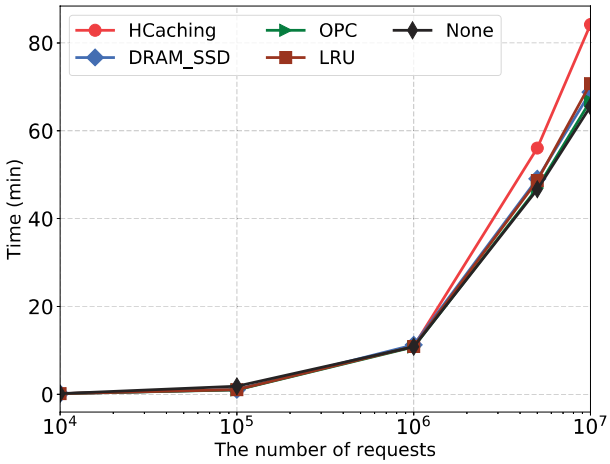


Figure 8. Execution time versus the number of requests

sequential the memory operations like hardware. By combining the queue mechanism with memory simulators which provides the exact access time for the memories, we implement the evaluation of the throughput for ICN caching schemes. Besides, to provide realistic high-speed traffic for evaluating this throughput, we propose a realistic high-speed requesting algorithm. Multiple leading caching schemes are implemented in CCNHCaching and evaluated on throughput performance. The results demonstrate that CCNHCaching can provide more details of these caching schemes and can favor the designs of ICN caching schemes better. Finally, we evaluate the CCNHCaching performance regarding the memory occupancy and the execution time. In the future work, we will replicate more ICN high-speed caching schemes and memory simulators to CCNHCaching. Meanwhile, the preliminary code implementation will be rewritten and optimized to make CCNHCaching conform to the ns-3 coding style, hence, introducing it to the official release of ns-3.

Acknowledgments

This research is supported by NSAF under Grant No. U1530118, the National Natural Science Foundation of China (NSFC) under Grant No. 61602030, the National Key R&D Program under Grant No. 2016YFE0122900, the Fundamental Research Funds for the Central Universities under Grant No. 2017YJS032.

References

[1] H. Zhang, W. Quan, H.-C. Chao, C. Qiao, Smart Identifier Network: A Collaborative Architecture for the Future Internet, *IEEE Network*, Vol. 30, No. 3, pp. 46-51, May, 2016.

[2] N. Cheng, F. Lyu, J. Chen, W. Xu, H. Zhou, S. Zhang, M. Xue, X. S. Shen, Big Data Driven Vehicular Networks, *IEEE Network*, Vol. pp, No. 99, pp. 1-7, April, 2018.

[3] G. Xylomenos, G. N. Ververidis, V. A. Siris, N. Fotiou, G. Tsilopoulos, X. Vasilakos, K. V. Katsaros, G. C. Polyzos, A Survey of Information-centric Networking Research, *IEEE Communications Surveys & Tutorials*, Vol. 16, No. 2, pp. 1024-1049, July, 2014.

[4] W. Quan, C. Xu, J. Guan, H. Zhang, L. A. Grieco, Scalable Name Lookup with Adaptive Prefix Bloom Filter for Named Data Networking, *IEEE Communications Letters*, Vol. 18, No.1, pp. 102-105, January, 2014.

[5] W. Quan, C. Xu, A. V. Vasilakos, J. Guan, H. Zhang, L. A. Grieco, Tb2f: Tree-bitmap and Bloom-filter for a Scalable and Efficient Name Lookup in Content-centric Networking, *IFIP Networking*, Trondheim, Norway, 2014, pp. 1-9.

[6] B. Feng, H. Zhou, G. Li, H. Zhang, H. C. Chao, Least Popularly Used: A Cache Replacement Policy for Information-centric Networking, *Journal of Internet Technology*, Vol. 17, No.1, pp. 1-10, January, 2016.

[7] Y. Zeng, M. Jin, J. Li, An Approach for Robust In-network Caching in Information-centric Networks, *Journal of Internet Technology*, Vol. 17, No. 3, pp. 503-513, January, 2016.

[8] R. P. Jelenković, A. Radovanović, Least-recently-used Caching with Dependent Requests, *Theoretical Computer Science*, Vol. 326, No. 1-3, pp. 293-327, October, 2004.

[9] Y. Thomas, G. Xylomenos, C. Tsilopoulos, G. C. Polyzos, Object-oriented Packet Caching for Icn, *Proc. 2nd Int. Conf. Information-Centric Netw. ICN '15,* San Francisco, American, 2015, pp. 89-98.

[10] G. Rossini, D. Rossi, M. Garetto, E. Leonardi, Multi-terabyte and Multi-gbps Information Centric Routers, *Proc. IEEE INFOCOM*, Toronto, Canada, 2014, pp. 181-189.

[11] H. Li, H. Zhou, W. Quan, B. Feng, H. Zhang, S. Yu, Hcaching: High-speed Caching for Information-centric Networking, *GLOBECOM 2017*, Singapore, 2017, pp. 1-6.

[12] A. Afanasyev, I. Moiseenko, L. Zhang, Ndnsim: ndn Simulator for ns-3, *Technical Report NDN-0005*, October, 2012.

[13] C. Scherb, M. Sifalakis, *Lightweight Implementation of the Content-centric Networking Protocol*, https://github.com/cn-uofbasel/ccn-lite, 2017.

[14] R. Chiocchetti, D. Rossi, G. Rossini, Ccnsim: An Highly Scalable CCN Simulator, *IEEE International Conference on Communications*, Budapest, Hungary, 2013, pp. 2309-2314.

[15] I. Cianci, L. Grieco, G. Boggia, Ccn-java Opensource Kit Emulator for Wireless Ad Hoc Networks, *ACM Proceedings of the 7th International Conference on Future Internet Technologies*, New York, American, 2012, pp. 7-12.

[16] CCNx, http://www.ccnx.org/, April, 2018.

[17] NDNx, https://github.com/named-data/ndnx, April, 2018.

[18] C. M. S. Cabral, C. E. Rothenberg, Mini-ccnx: Fast Prototyping for Named Data Networking, *ACM SIGCOMM workshop on Information-Centric Networking*, Hong Kong, China, 2013, pp. 33-34.

[19] N. Handigol, B. Heller, B. Lantz, N. Mckeown, Reproducible Network Experiments Using Container-based Emulation, *Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies*, Nice, France, 2012, pp. 253-264.

[20] C. Elliott, GENI-global Environment for Network Innovations, *IEEE 33rd Conference on Local Computer Networks*, Hyatt Regency Montreal Montreal, Canada, 2008, pp. 8.

[21] H. Li, H. Zhou, H. Zhang, B. Feng, W. Shi, Emustack: An Openstack-based Dtn Network Emulation Platform (extended version), *Mobile Information Systems*, Vol. 2016, No. 2016, pp. 1-11, October, 2016.

[22] W. Quan,Y. Liu, H. Zhang, S. Yu, Enhancing Crowd Collaborations for Software Defined Vehicular Networks, *IEEE Communications Magazine*, Vol. 55, No.8, pp. 80-86, August, 2017.

[23] W. Quan, K. Wang, Y. Liu, N. Cheng, H. Zhang, X. S. Shen, Software-Defined Collaborative Offloading for Heterogeneous Vehicular Networks, *Wireless Communications and Mobile Computing*, Vol. 2018, Article ID 3810350, pp 1-9, April, 2018.

[24] W. Chai, D. He, L. Psaras, G. Pavlou, Cache Less for more in Information-centric Networks (extended version), *Computer Communications*, Vol. 36, No. 7, pp. 758-770, February, 2013.

[25] G. Zalateu, Aging Bloom Filter with Two Active Buffers for Dynamic Sets, *IEEE Trans. Knowl. Data Eng.*, Vol. 15, No. 12, pp. 1630-1632, January, 2010.

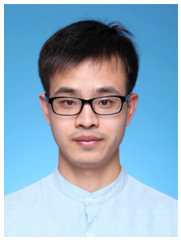
[26] D. Wang, B. Ganesh, N. Tuaycharoen, K. Baynes, A. Jaleel, B. Jacob, Dramsim: A Memory System Simulator, *ACM SIGARCH Computer Architecture News*, Vol. 33, No. 4, pp. 100-107, September, 2005.

[27] Y. Kim, W. Yang, O. Mutlu, Ramulator: A Fast and Extensible Dram Simulator, *IEEE Computer Architecture Letters*, Vol. 15, No. 1, pp. 45-49, June, 2016.

[28] Y. Kim and B. Tauras, A. Gupta, B. Urgaonkar, Flashsim: A Simulator for Nand Flash-based Solid-state Drives, *IEEE First International Conference on Advances in System Simulation*, Porto, Portugal, 2009, pp. 125-131.

[29] K. Katsaros, G. Xylomenos, G. Polyzos, Globetraff: A Traffic Workload Generator for the Performance Evaluation of Future Internet Architectures, *IEEE 5th International Conference New Technologies, Mobility and Security (NTMS)*, Istanbul, Turkey, 2012, pp. 1-5.

Biographies



Haifeng Li received the B.E. degree in communication and information systems from the Beijing Jiaotong University, Beijing, China, in 2013. He is currently working toward his Ph.D. degree at the School of Electronic and Information Engineering, Beijing Jiaotong University. He has participated in several national research programs of China such as "973 Program" and "863 Program". His research interests include the future Internet, Delay Tolerant Networking, High-Performance Computing and Cloud Computing.

Huachun Zhou received his B.S. degree from People's Police Officer University of China in 1986. He received his M.S. and Ph.D. degrees from Beijing Jiaotong University in 1989 and 2009 respectively. His research interests include mobility management, mobile and secure computing, routing protocols and network management technologies. His recently research projects include Research on Models and Algorithms of Information-Centric Mobile Internet, supported by National Natural Science Foundation of China, and Research on the Future Space-Ground Integrated Network, supported by National High Technology of China.

Wei Quan received his Ph.D. degree in Communication and Information System from Beijing University of Posts and Telecommunications (BUPT), Beijing, China in 2014. During 2014-2016, he worked as a post doctor at National Engineering Lab for Next Generation Internet Technologies (NGIT), Beijing Jiaotong University (BJTU), Beijing, China. He currently works as a Lecturer at School of Electronic and Information Engineering, BJTU. He has published more than 20 papers in prestigious international journals and conferences including IEEE Wireless Communications Magazine, IEEE Network Magazine, IEEE Communications Letters, IFIP Networking, IEEE WCNC, etc. and serves as technical reviewers for some important international journals and conferences. His research interests include key technologies for the future Internet, 5G network architecture, vehicular networks and Internet of energy. He is a Member of IEEE, ACM, and a Senior Member of CAAI (Chinese Association of Artificial Intelligence).

Bohao Feng received the B.E. degree in communication and information systems from the Beijing Jiaotong University, Beijing, China, in 2011. He is currently pursuing the Ph.D. degree at the School of Electronic and Information Engineering, Beijing Jiaotong University. He has participated in several national research programs of China such as "973 Program" and "863 Program". His research interests are in the wide areas of network technologies including future network architectures, network caching, mobile networks and satellite networks.

Hongke Zhang received the M.S. and Ph.D. degrees in electrical and communication systems from the University of Electronic Science and Technology of China (formerly known as Chengdu Institute of Radio Engineering) in 1988 and 1992, respectively. From September 1992 to June 1994, he was a Postdoctoral Research Associate at Beijing Jiaotong University (formerly known as Northern Jiaotong University). In July 1994, he joined Beijing Jiaotong University, where he is a Professor. He currently directs a National Engineering Lab on Next Generation Internet in China. He is a Senior member of IEEE and has published more than 100 research papers in the areas of communications, computer networks, and information theory. He is the author of eight books written in Chinese and the holder of more than 30 patents. Dr. Zhang received the Zhan Tianyou Science and Technology Improvement Award in 2001, the Mao Yisheng Science and Technology Improvement Award in 2003, the First-Class Science and Technology Improvement Award of the Beijing government in 2005, and other various awards. He is now the Chief Scientist of a National Basic Research Program ("973 Program").