Saving Energy on the Edge: In-Memory Caching for Multi-Tier Heterogeneous Networks

Jianwen Xu, Kaoru Ota, and Mianxiong Dong

To help deal with the data explosion problem by edge caching, the authors apply in-memory storage and processing to reduce energy consumption. They design two kinds of TTL in four cache replacement policies to cache data at the edge. They carry out a simulation experiment in a three-tier heterogeneous network structure using the RWP model and test the performance of in-memory caching and the traditional method.

Abstract

Recent years have witnessed billions of new manufactured sensors, equipments, and machines being connected to our almost omnipotent Internet. While enjoying the comfort and convenience brought by IoT, we also have to face tremendous energy consumption and carbon emissions that even cause climate deterioration. Extended from cloud computing, edge/fog computing and caching provide new thoughts on processing big data generated from distributed IoT devices. With the purpose of helping deal with the data explosion problem by edge caching, in this article we apply in-memory storage and processing to reduce energy consumption. We design two kinds of TTL in four cache replacement policies to cache data at the edge. We carry out a simulation experiment in a three-tier heterogeneous network structure using the RWP model and test the performance of in-memory caching and the traditional method. The analysis results manifest that our in-memory method is able to obtain better energy efficiency in edge caching, and has stable and low backhaul rate.

Introduction

From wearable devices to house furnishings, from vehicles to industrial facilities, nearly all that we ever regarded as things have been sharing the same comfort and convenience brought by the Internet. As a result, we are already living in a new Internet of Things (IoT) era. Since user/client volume is generally taken as a metric to measure the network scale, this time with the addition of devices and machines, our Internet may have to face network connections and data quantity in the near future several times those of the present. Moreover, big data generated by countless IoT devices is in high demand on real-time processing, which means we have to spend less time on much larger amounts of data with small single computations. Such condition calls for additional requirements in data transmission and processing, and brings huge energy consumption and carbon emissions leading to climate deterioration such as the greenhouse effect.

Together with the development of big data, cloud computing provides a new era of sharing computer processing resources. Nowadays we are able to rent and use computing resources with no need to purchase high-priced servers or data centers, just like borrowing books from a library. However, traditional cloud computing may not be suitable for handling distributed computational tasks in IoT scenarios. Considered as an extension of cloud computing, edge/fog computing attaches much importance to making real-time responses to massive numbers of users in which most requests require no complex calculation and can be processed by small devices at the edge of the network. While in the conventional method all computing tasks are allocated to a limited number of central servers, with edge computing we are able to rely on a mass of mobile devices, routers, and other dedicated edge devices to send immediate feedback to nearby requests. Therefore, as required contents have already been cached locally, edge computing can perform tasks without visiting remote data centers, and reduce time cost and energy consumption in procedures of transmission and propagation.

In computing, a cache is a temporary place for information or data storage, and caching refers to the process of storing and reading/writing in a cache. Just like the modern CPU cache structure and web cache technology, the cache can serve as an inherent trade-off to help balance the gap between different processing speeds and storage sizes. As a result, storage capacity provided by caching can improve computational performance by reducing extra latency and energy consumption.

Although devices in the edge computing scenario may not own multi-level CPUs or web cache software, caching can still occur in divided storage space with suitable strategies and methods. In conventional ways we can cache data or contents in a disk, which means there is no need to forward it upward when a request asks for anything already saved in a current edge device.

In-memory, which is broadly regarded as the RAM inside a computer, stands for high-speed operation capability compared to non-volatile memory (NVM) like a hard disk drive (HDD). Processing through in-memory can save a lot of time spent on I/O operations as well as energy consumption. Since the common storage volume of in-memory is much less than that of a disk, we need better space utilization and scheduling when designing an in-memory caching method. Additionally, in IoT scenarios we do not need a single edge device to cache too many contents since IoT data is frequently out of date. That is to say,

Digital Object Identifier: 10.1109/MCOM.2018.1700909

The authors are with Muroran Institute of Technology.

in-memory processing would be very appropriate for edge caching and improve energy efficiency of edge computing.

In this article, we focus on the solutions of edge caching based on in-memory processing from mathematical modeling, caching method designing to simulation and analysis. The main contributions are as follows:

- Design a three-tier heterogeneous network structure using Random Waypoint (RWP) model.
- Design two in-memory edge caching methods based on two kinds of time to live (TTL).
- Consider four cache replacement policies including first in first out (FIFO), least frequently used (LFU), least recently used (LRU), and random replacement (RR).
- Choose total energy consumption and backhaul rate as metrics to compare and analyze experimental results of in-memory edge caching and the conventional disk method under the same simulation setups.

This article is divided into six sections to cover all aspects of the research. We introduce the background and sort out the whole work flow. We present related works on edge caching and in-memory processing. We set up the mathematical model and elaborate the details of raised problem. We propose the designed in-memory edge caching methods using four replacement policies. We give simulation results and comparative analysis between in-memory edge caching and the conventional method. We summarize the previous work and draw conclusions.

Related Work

In this section, we present some related work about edge computing and edge caching, and then introduce some research on in-memory storage and processing.

Edge Computing and Edge Caching

Cloud services have long remained a part of people's lives, ever since cloud computing became known in 2005 and was quickly utilized in a wide variety of fields. Together with the current IoT boom, in 2020, the total amount of data created by any device will reach 600 ZB per year, while annual global data center IP traffic will only be 15.3 ZB at the end of this decade [1]. As a result, in the near future, we will no longer be able to put all computing tasks on the cloud and pin our hopes on continuously updating hardware levels, increasing the number of end equipments. We need edge devices to share the workload and solve the bottleneck in data transmission and processing [2].

Edge computing, before attracting wide attention and being extensively applied among research institutions, was already studied by a number of technology companies, such as the key players including Cisco Systems Inc., HP, and so on. Early in 2012, Bonomi *et al.*, from Cisco started by making clear the position of edge computing in the IoT era and proving that fog owns the characteristics of serving as platforms for IoT services from connected vehicles to smart grid to smart city. They defined the fundamental characteristics as low latency and location awareness, widespread geographical distribution, mobility, large numbers of devices, the predominant role of wireless connection, streaming and real-time applications, and heterogeneity [3]. Vaquero *et al.* from HP offered a comprehensive definition of the fog to include cloud, sensor networks, peer-to-peer networks, and so on. They also combine network functions virtualization (NFV) and software-defined networking (SDN) to achieve a new softwarization network management [4].

Many works on edge computing in recent years focus on interdisciplinary research and try to find the relation with other fields to help promote common development. Liu *et al*. designed a device-to-device video recovery system based on heterogeneous network for picocell edge users. In the paper they discussed the possibilities of achieving the video on demand (VoD) application to improve the current performance [5]. As a branch discipline, mobile edge computing pays more attention on wireless communication among smartphones, tablets, and other handheld devices. Sardellitti *et al.* considered a multiple-input multiple-output (MIMO) multicell system and design a whole set of joint optimization algorithms for mobile edge computing [6]. A research group from the European Telecommunications Standards Institute (ETSI) regarded mobile edge computing to an independent field of study and combined it with fifth generation (5G) mobile networks. They also analyze the market drivers and business value of mobile edge computing services [7].

In order to further utilize edge devices to balance the workload in the expanding network, caching on edge can make a contribution to reducing bandwidth usage, server load, and so on. Research on edge caching also goes in many different directions. Early in 2005, before cloud computing entered the public consciousness and was widely applied in production and living, Ramaswamy *et al*. proposed the idea of building cache clouds to deal with documents in edge networks. In the paper they designed a dynamic hashing scheme to improve document placement in cache clouds [8]. Gabry *et al*. put forward a maximum-distance separable (MDS) encoded caching scheme to achieve energy-efficient edge computing in heterogeneous networks [9]. In recent years, with fast development of wireless communication, caching on mobile/wireless edge has become a research hotspot. Liu *et al.* summarized the design aspects and challenges of wireless edge caching. They focused on two key features of content delivery traffic and compared the performance of caching at base stations and users [10].

In-Memory Storage and Processing

Compared to disk storage like HDD, flash memory, and faster solid-state drive (SSD), in-memory or main memory mainly refers to volatile RAM, which could spend the same amount of time while reading/writing data regardless of physical location. Limited by fault tolerance, consistent power supply, and high manufacturing cost from all kinds of electronic equipments and personal computers to large professional servers, we still cannot rely on main memory to store our data for a long time. However, the last decade has witnessed rapidly decreasing cost of main memory and growing demand for high-speed computing, which makes it possible for turning in-memory into the new disk.

In the near future, we are no longer able to put all computing tasks on the cloud and pin our hopes on continuously updating hardware levels, increasing the number of end equipments. We need edge devices to share the workload and solve the bottleneck in data transmission and processing.

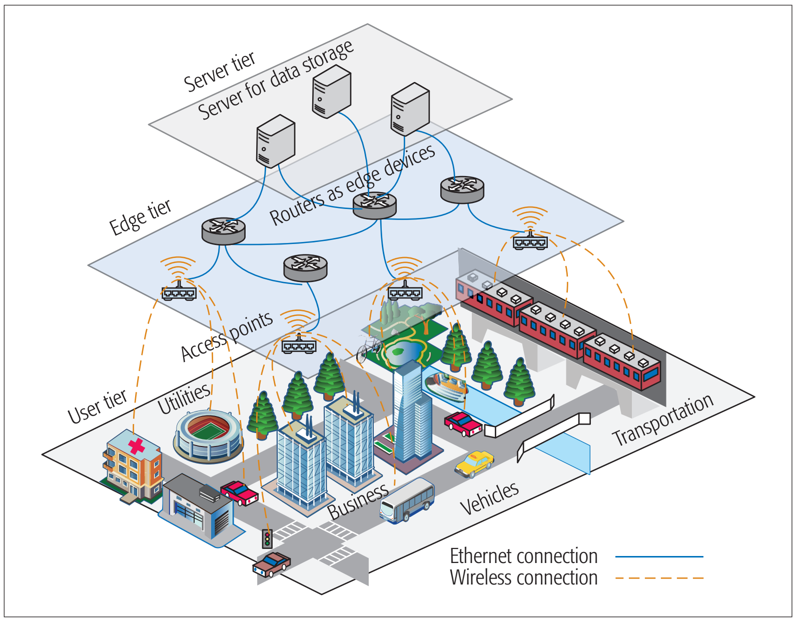

Figure 1. A three-tier heterogeneous network structure.

Related works on in-memory storage and processing involve different levels from application domain analysis and technical breakthroughs to business development prospects. Zhang *et al.* introduced the recent years' development in in-memory big data management and processing. In the paper they classified and summarized all existing commercial and academic management systems for in-memory operations [11]. Beneventi *et al.* applied in-memory processing tools to help do machine learning in high-performance computing (HPC) infrastructure models [12].

Problem Formulation

In this section, we design a three-tier heterogeneous network structure as the experimental scenario to simulate in-memory edge caching for big data. As shown in Fig. 1, from top to bottom a three-tier network model can be described as follows:

- Server tier: In this tier multiple servers play the role of cloud data centers; each file is only stored in one of the servers.
- Edge tier: In this tier we use routers as both forwarders and edge devices that can cache files in packets passing by.
- User tier: This tier is made of user nodes that keep moving randomly and requesting files originally stored in servers or cached in routers.

System Outline

In our designed network structure, user nodes move in the random waypoint (RWP) model, which is one of the most popular mobility models applied in mobile ad hoc networks (MANETs) [13]. In Fig. 1, user nodes (u_1, u_2, ..., u_n) in the user tier send packets to request files at random time intervals and move to their next positions under normal distribution before sending again. Each file, with unfixed size, is randomly saved in one of the servers (s_1, s_2, ..., s_n) in the server tier. Then, in the edge tier, routers (r_1, r_2, ..., r_n) as edge devices will check if the needed files are already cached in storage before forwarding to neighbor r or upward to s. Since information and contents easily become outdated, once the original files in the server tier are modified or deleted, all cached copies become useless. That is to say, edge caching needs to consider TTL to make sure that most requests are satisfied with unexpired files [14]. Transmissions between the user tier and the edge tier are wireless broadcasting, those inside the edge tier are wired broadcasting, and those between the edge tier and the server tier are wired point-to-point.

Normally, RAM is associated with volatile types of memory whose data storage would be lost when power is off. That is to say, in-memory may not support long-term storage, which can be suitable for scenarios of edge caching. As a result, we consider both TTL for cached data and consistency of in-memory storage in designing caching methods.

Performance Metrics

In the face of mass data generated by billions of IoT devices, we always prefer less energy consumption in data transmission and processing. For this reason, edge caching aims to reduce repeated data transmission from original servers, which means that unnecessary energy consumption on packet delivery between the edge tier and the server tier can be saved. Moreover, caching itself also consumes extra energy while keeping RAM or disk memory running. To determine and sum up the overall cost of the entire simulation on the three-tier network architecture, we may consider three parts. The part to maintain all devices in the three tiers is not expressed in the equation since it is a fixed cost and can only be reduced by shutting down some devices [9]. In summary, we use two equations to present the calculation of total energy consumption,

$$E_{total} = E_{cache}(t) + E_{trans}$$

$$E_{cache}(t) = \omega \sum_{i=1}^{n} \left(s_i^{cache} t_i \right)$$

in which $E_{cache}(t)$ and E_{trans} stand for caching energy cost and transmission energy cost, respectively. $E_{cache}(t)$ relates to running time and can be calculated by energy consumption per byte ω. n represents the times when the current caching size of all r in the edge tier is changed. Thus, we sum up the n products of variational caching sizes s_i^{cache} and their duration t_i.

$$\begin{aligned} E_{trans} &= E_{send} + E_{recv} \\ &= \sum_{j=1}^{x} \left(m_{send} s_i^{trans} + b_{send} + m_{recv} s_i^{trans} + b_{recv} \right) \end{aligned}$$

Comparatively, E_{trans} has no relation to time and only depends on size of data being transmitted, s_i^{trans}. Here we separately calculate the energy consumption while devices in the three tiers are sending and receiving packets. Moreover, as a heterogeneous network model, communications between the user tier and the edge tier and among r inside the edge tier are regarded as wireless, while those from the edge tier to the server tier and backward are Ethernet transmissions. m and b are linear coefficients obtained from experimental results [15].

To figure out the total energy cost as close as possible to the practical situation, we take some more details into consideration: first, different bit rates of wireless and Ethernet transmissions, and second, the wave propagation speed. We choose speed of light and thick coax as the communication media for wireless and wired, respectively.

Besides energy consumption, we also pay attention to how edge caching helps reduce workload on end servers. We add a backhaul rate as another metric to test what role in-memory edge caching may take in completing the task of fetching files from servers across tiers.

In-Memory Edge Caching Method

In this section, we propose two caching methods based on different TTL designs: TTL of requested times (TRT) and caching time (TCT).

TRT counts how many times a cached file being requested by *u* in the user tier. When the needed file found at any *s* in the server tier is sent back, each *r* in the full path may check if it already has a copy of the file. If not, *r* will cache the file in its in-memory or disk. Later, once an *r* receives a request and finds the needed file in cached data, it may send back a copy and count the times the cached file has been requested. If number of requests reaches the maximum value set, the cached file will be dropped. Accordingly, in the calculation equation of $E_{cache}(t)$ in the last section, s_i^{cache} is changed and s_{i+1}^{cache} is needed.

Rather than counting requested times, the other caching method, TCT, keeps a timer for each cached file. Caching and routing follow the same rules as the first method; similarly, when any timer reaches the maximum value, the cached file will be dropped.

In addition, both methods consider the volume of in-memory/disk storage, that is, if the next file to be cached exceeds the memory volume of *r*, before caching the new data, we have to free up some space by popping out cached data. As a result, to decide which one to drop when the volume of disk or in-memory is full, we apply four common cache replacement policies based on different ideas on how to improve the performance of edge caching. Another point to note is that the drop behaviors in cache replacement policies have no relation to TTL designs since both are needed to guarantee the usability of cached data, that is, still alive and within the capacity of the current *r*.

The FIFO policy regards volume disk/in-memory as a FIFO queue and drops the head of the queue that gets pushed in the earliest when the queue is full. Second, the LFU policy does not consider the order of cached files and chooses the cached file with the least requested times currently. Third, similar to LFU, LRU also focuses on the cached files that are not very popular but chooses the least recently used one to drop. Last, the RR policy serves as a contrast to compare the performance of TTL designs with the other policies.

Simulation and Analysis

In this section, we carry out experimental simulations to compare the performance of edge caching in in-memory and conventional disk memory under two TTL designs.

The simulation scenario is a 10 km^2 open area, and we set 2000 user nodes in the user tier with random initial positions. Each user node randomly moves to the next position under normal distribution after sending a request packet to fetch one of the 200 files originally stored in one of the five servers in the server tier. We use 100 routers in the edge tier as edge devices to cache the files locally. Four cache replacement policies are applied in designing edge caching methods.

Bit rate of transmission	
Wireless (802.11ad)	6.8 Gb/s
Ethernet	10 Gb/s
Wave propagation speed of transmission	
Wireless (air)	*c* (speed of light)
Ethernet (thick coax)	0.77 *c*
Device settings of edge tier	
MTR of disk (SSD)	2500 MB/s
MTR of in-memory (DDR3)	6400 MB/s
Disk volume	256 MB
In-memory volume	25 MB
Energy consumption coefficients	
	10^{-8} J/MB 10^{-6} J
Broadcast send	$2.1 \times$ size + 272
Point-to-point send	$0.48 \times$ size + 431
Broadcast receive	$0.26 \times$ size + 50
Point-to-point receive	$0.12 \times$ size + 316
Power consumption of caching	8×10^{-3} W/MB

Table 1. Experiment setups.

Normally RAM is associated with volatile types of memory whose data storage would be lost when power is off. That is to say, in-memory may not support long time storage which can be suitable for scenarios of edge caching. As a result, we consider both TTL for cached data and consistency of in-memory storage in designing caching methods.

As shown in Table 1, the setups of the experiment include bit rate and wave propagation speed of packet transmission, the edge tier's device settings, and the energy consumption coefficients of both transmission and caching. As a result, to calculate the total energy consumption E_{total}, we have to count the number of packets and sum up their sizes, then compute the time cost from transmission and reading/writing from disk/in-memory. We carry out 10 rounds of simulations in two designed edge caching methods and four cache replacement policies. We repeat the part of each method and policy 10 times and get the averaged results. The simulation environment is MATLAB R2017b.

Figure 2 shows the simulation result of total energy consumption of the TRT method in four cache replacement policies. Blue, red, green, and yellow represent the policies of FIFO, LFU, LRU, and RR. The solid lines and dotted lines stand for caching in in-memory storage and disk storage, respectively. From the patterns of eight broken lines, we can know that the overall energy consumption of traditional disk caching is larger than in-memory caching. In our 10 rounds of simulation, the trends of disk and in-memory methods are also different. Energy consumption of disk in TRT varies like a checkmark symbol that first falls down a little and then, after a smooth transition period. increases rapidly to a high value. Our in-memory method in TRT shows a similar trend in the first half from round 1 to 5, but becomes steady after that, which means energy consumption on edge caching may finally enter a stable state. The differences

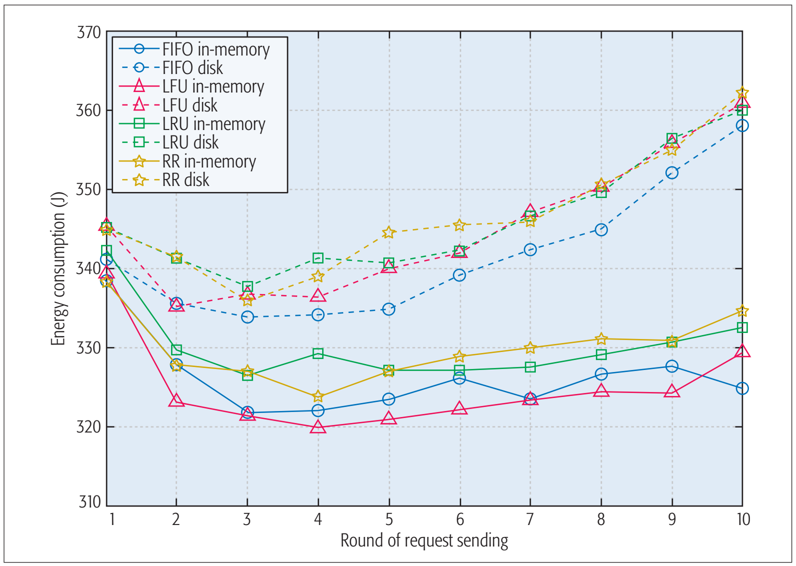

Figure 2. Total energy consumption of edge caching in TTL of requested times (TRT).

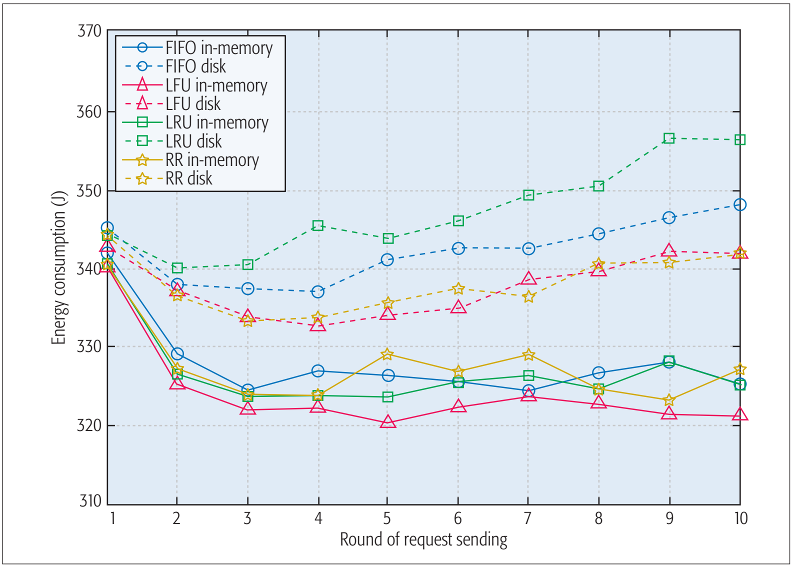

Figure 3. Total energy consumption of edge caching in TTL of caching time (TCT).

in the trends are in line with our expectation since disk, which owns larger storage volumes and lower MTR, calls for more unit energy consumption and longer operation time. The positive correlation of the two factors lead to the continuing growth of total consumption.

Different cache replacement policies also have respective functions on the results of total energy cost. From the four lines of disk caching in the top half of the figure we can see three of them, LFU, LRU, and RR, fluctuate until coinciding in the end. Only the blue line of FIFO policy shows a slight advantage in total energy consumption, which means a simpler rule may be more suitable for disk caching by TRT. Then, from the other four colorful lines of in-memory caching, although there are still some changes in energy cost values, we may get the overall order of the policies as RR, LRU, FIFO, and LFU. RR does suffer from random choice of dropping cached files, which costs more energy in total. The special case of round 4 when LRU exceeds RR may be explained by some occasional error when RR just drops the right files that are not requested much later. LFU seems to have the optimal performance in in-memory caching by TRT because of the most suitable dropping choice, which focuses on reserving popular files probably from the server directly connected to the current router with only one hop and dropping the most unpopular one, which may come from a remote server after several hops of forwarding. The green line applying LRU also pays attention to the popularity of cached files but directly considers the caching time and drops the one not being requested for the longest interval. However, compared to LFU, the practice of LRU may face exceptions like some popular files being dropped due to timing when other files are just being cached or requested.

The experimental result of the other TTL design is shown in Fig. 3. In comparison with TRT, the trend of the disk's four colorful lines of different cache replacement policies is relatively smoother; gaps between each pair of policies are visible. However, the order of lines in different policies is somewhat abnormal. The green line applying LRU shows the highest total energy consumption, and RR policy even achieves high performance in value. From our point of view, since disks have far larger storage volume than in-memory, although more files are cached live for enough time, not many of them finally wait for a request or even get a remote request, which may save no time over fetching the original file from servers. As a result, replacement policies trying to delay the time for dropping popular files may in turn cause more energy cost.

The patterns of in-memory by TCT are similar to TRT. RR policy fluctuates more violently. LRU seems to improve its performance compared to TRT and even shows a certain degree of convergence with FIFO. From our point of view, the replacement policy here shares the same method with TCT in dropping cached files by keeping track of their timestamps. The collective effect may lead to some promotion in efficiency.

To make clear how TRT and TCT methods may help reallocate the whole workload and achieve energy-efficient caching, we add the experimental results of backhaul rates. As the values of results in different cache replacement policies are similar to each other, here we choose FIFO as an example. We change the set maximum requested times and caching time of each single file being cached in routers as edge devices and get two 3D surface graphs (values of maximum caching time are corresponding values in the simulation).

Under the same coordinate axis, Fig. 4 shows the variation of backhaul rates by rounds of packet sending and maximum requested times/caching time. First, in Fig. 4a, when the number of request times is small, regardless of which round of packet sending, more 70 percent of requests still have to reach the server tier to get needed files. The other side, the numerical range of the same maximum times in 10 rounds, does not change very much, which means that although they stay high in value with a small number of request times, the backhaul rate of the TRT method can rapidly reach steady state as the number increases. Finally, more than 70 percent of requests can be satisfied by in-memory edge caching.

Then in Fig. 4a, the main difference of the pattern is the variation when we continue to send packets. That is, no matter how long the caching time is set, the server tier still has to take charge of most of the workload in the first rounds. However, with more and more files being fetched originally from end servers and then cached in in-memory of passed routers, TCT can also reach a comparatively ideal low backhaul rate, although higher in value than TRT with some fluctuations.

Conclusion

In this article, we focus on how to improve the energy efficiency of edge caching using in-memory storage and processing. Here we build a three-tier heterogeneous network structure and propose two edge caching methods using different TTL designs and cache replacement policies. We use total energy consumption and backhaul rate as the two metrics to test the performance of in-memory caching and compare with the conventional method based on disk storage. The simulation results show that in-memory storage and processing can help save more energy in edge caching and share a considerable percentage of the workload.

Acknowledgments

This work is partially supported by JSPS KAKENHI Grant Numbers JP16K00117 and JP15K15976, and the KDDI Foundation. Mianxiong Dong is the corresponding author.

References

[1] Cisco Public White Paper, "Cisco Global Cloud Index: Forecast and Methodology, 2015-2020"; www.cisco.com/c/dam/en/us/solutions/collateral/service-provider/global-cloud-index-gci/white-paper-c11-738085.pdf; accessed Sept. 2017.

[2] W. Shi *et al.*, "Edge Computing: Vision and Challenges," *IEEE Internet of Things J.*, vol. 3, no. 5, Oct. 2016, pp. 637–46.

[3] F. Bonomi *et al.*, "Fog Computing and Its Role in the Internet of Things," *Proc. 1st Ed. MCC Wksp. on Mobile Cloud Computing*, no. 4, 2012, pp. 13–16.

[4] L. M. Vaquero and L.Rodero-Merino, "Finding Your Way in the Fog: Towards a Comprehensive Definition of Fog Computing," *ACM SIGCOMM Comp. Commun. Review*, vol. 44, no. 5, Oct. 2014, pp. 27–32.

[5] Z. Liu *et al.*, "Device-to-Device Assisted Video Frame Recovery for Picocell Edge Users in Heterogeneous Networks," *Proc. IEEE ICC 2016*, May. 2016, pp. 1–6.

[6] S. Sardellitti, G. Scutari, and S. Barbarossa, "Joint Optimization of Radio and Computational Resources for Multicell Mobile-Edge Computing," *IEEE Trans. Signal and Info. Processing over Networks*, vol. 1, no. 2, June 2015, pp. 89–103.

[7] Y. C. Hu *et al.*, "Mobile Edge Computing: A Key Technology towards 5G," ETSI White Paper, vol. 11, no. 11, 2015, pp. 1–16.

[8] L. Ramaswamy, L. Liu, and A. Iyengar, "Cache Clouds: Cooperative Caching of Dynamic Documents in Edge Networks," *Proc. 25th IEEE Int'l. Conf. Distributed Computing Systems*, June 2005, pp. 229–38.

[9] F. Gabry, V. Bioglio, and I. Land, "On Energy-Efficient Edge Caching in Heterogeneous Networks," *IEEE JSAC*, vol. 34, no. 12, Dec. 2016, pp. 3288–98.

[10] D. Liu *et al.*, "Caching at the Wireless Edge: Design Aspects, Challenges, and Future Directions," *IEEE Commun. Mag.*, vol. 54, no. 9, Sept. 2016, pp. 22–28.

[11] H. Zhang *et al.*, "In-Memory Big Data Management and Processing: A Survey," *IEEE Trans. Knowledge and Data Engineering*, vol. 27, no. 7, July 2015, pp. 1920–48.

[12] F. Beneventi *et al.*, "Continuous Learning of HPC Infrastructure Models Using Big Data Analytics and In-Memory Processing Tools," *Design, Automation Test in Europe Conf. Exhibition*, Mar. 2017, pp. 1038–43.

[13] C. Bettstetter, G. Resta, and P. Santi, "The Node Distribution of the Random Waypoint Mobility Model for Wireless Ad Hoc Networks," *IEEE Trans. Mobile Computing*, vol. 2, no. 3, June 2003, pp. 257–69.

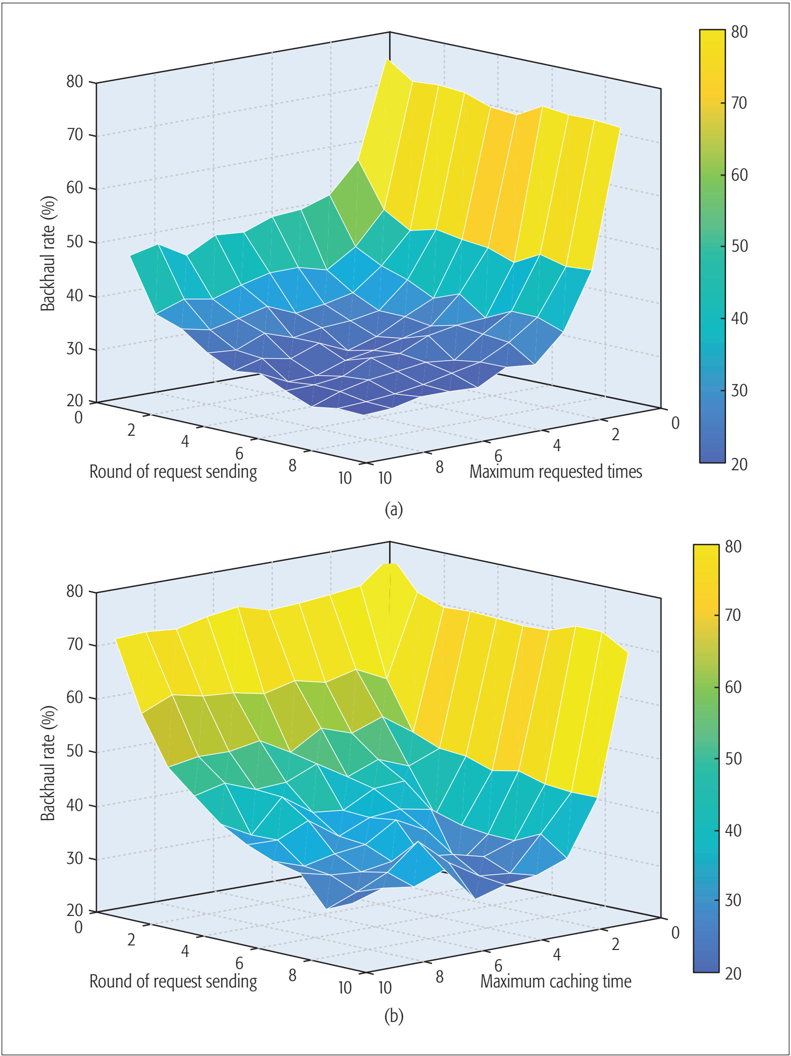

Figure 4. Backhaul rate of in-memory edge caching in two TTL designs: a) TTL of requested times; b) TTL of caching time (TCT).

[14] Z. Zhou *et al.*, "Energy-Efficient Matching for Resource Allocation in D2D Enabled Cellular Networks," *IEEE Trans. Vehic. Tech.*, vol. 66, no. 6, June 2017, pp. 5256–68.

[15] L. M. Feeney and M. Nilsson, "Investigating the Energy Consumption of a Wireless Network Interface in an Ad Hoc Networking Environment," *Proc. IEEE INFOCOM 2001*, vol. 3, 2001, pp. 1348–1557.

Biographies

Jianwen Xu received his B.Eng. degree in electronic and information engineering from Dalian University of Technology, China, in 2014, and his M.Eng degree in information and communication engineering from Shanghai Jiaotong University, China, in 2017. He is currently pursuing a Ph.D. degree in electrical engineering at Muroran Institute of Technology, Japan. His main fields of research interest include distributed systems and the Internet of Things.

Kaoru Ota received her M.S. degree in computer science from Oklahoma State University in 2008, and her B.S. and Ph.D. degrees in computer science and engineering from the University of Aizu in 2006 and 2012, respectively. She is currently an assistant professor with the Department of Information and Electronic Engineering, Muroran Institute of Technology. She serves as an Editor for *IEEE Communications Letters*.

Mianxiong Dong received his B.S., M.S., and Ph.D. in computer science and engineering from the University of Aizu. He is currently an associate professor in the Department of Information and Electronic Engineering at Muroran Institute of Technology. He serves as an Editor for *IEEE Communications Surveys & Tutorials*, *IEEE Network*, *IEEE Wireless Communications Letters*, *IEEE Cloud Computing*, and *IEEE Access*.