Optimization Methods for Packet Caching in Named Data Networking (Postprint)

Authors: Zhi Jiang, Li Jun, Wu Haibo

Date: 2018-05-24T00:00:00+00:00

Abstract

In traditional network caching systems, packet-level caching is difficult to implement. The emergence of Information-Centric Networking has alleviated this challenge. Nevertheless, packet-level caching still faces serious scalability issues. By analyzing several problems that currently limit the implementation of packet-level caching, this paper proposes a grouped packet caching optimization method. This method reduces high-speed memory usage by establishing indexes based on group prefixes rather than individual packet prefixes, while group-level popularity is also leveraged to optimize caching decisions. Numerous evaluation metrics are defined, and extensive experiments are conducted to evaluate the performance of this scheme. Experimental results show that, compared to previous packet-level caching schemes, this method can substantially reduce high-speed memory usage and achieves significant improvements in server load reduction rate, average hop count reduction rate, and average cache hit ratio.

Full Text

Preamble

Grouped-Packet Caching Optimization Approach for NDN

Zhi Jiang[1],[2], Li Jun[1], Wu Haibo[1]
(1. Computer Network Information Center, Chinese Academy of Sciences, Beijing 100190, China;
2. University of Chinese Academy of Sciences, Beijing 100190, China)

Abstract: Packet-level caching is difficult to implement in traditional caching systems. The emergence of Information-Centric Networking (ICN) has alleviated this problem. However, packet-level caching still faces severe scalability issues. By analyzing the current problems that limit the implementation of packet-level caching, this paper proposes a grouped-packet caching optimization

approach. This method reduces high-speed memory usage by creating indexes based on group prefixes rather than individual packet prefixes, while group-level popularity is also used to optimize caching decisions. We define comprehensive evaluation metrics and assess the proposed scheme' s performance through extensive experiments. Experimental results demonstrate that compared with previous packet-level caching schemes, this approach significantly reduces high-speed memory consumption while achieving substantial improvements in server load reduction ratio, average hop reduction ratio, and average cache hit ratio.

Keywords: packet caching; named data networking (NDN); in-network caching; caching decision

0 Introduction

Modern Internet applications have shifted from end-to-end communication models to content distribution and retrieval. According to Cisco's Visual Networking Index report, content retrieval applications (such as Internet video, web data, and file sharing) are projected to account for over 85% of total network traffic by 2021. Network caching represents a crucial technology for alleviating this traffic pressure. Caching devices can temporarily store content objects and respond to subsequent user requests. Research indicates that finer-grained caching (such as packet-level caching) achieves better performance than caching entire content objects, as packet-level caching enables more granular caching decisions that can further improve network performance, particularly when different portions of an object exhibit varying popularity. This capability significantly reduces network bandwidth consumption and content server load while improving user experience through reduced request latency.

However, packet-level caching remains difficult to implement in traditional TCP/IP networks because it requires high-overhead techniques such as Deep Packet Inspection (DPI), creating severe performance bottlenecks and scalability issues. While the emergence of Information-Centric Networking (ICN) architectures has mitigated some problems facing packet-level caching, significant challenges persist: limited high-speed memory (e.g., SRAM) capacity, line-rate forwarding requirements, and packet-level popularity statistics.

To implement packet-level caching at lower cost, this paper introduces a grouped-packet caching (GPC) approach based on the principle of spatial locality. In GPC, we use group prefixes instead of individual packet prefixes to create index entries, thereby reducing high-speed memory consumption. We also employ a Bloom Filter to filter invalid cache queries and accelerate request forwarding. Additionally, we utilize group-level popularity to optimize caching decisions. The main contributions of this paper are summarized as follows:

a) We analyze the problems limiting packet-level caching implementation and introduce the concept of grouped packets based on spatial locality to alle-

viate scalability issues in packet-level caching.

b) We propose a packet-level caching mechanism called GPC that significantly accelerates forwarding of uncached packets while substantially reducing high-speed memory usage.

c) We define comprehensive evaluation metrics to assess GPC' s effectiveness. Experimental results demonstrate significant reductions in high-speed memory consumption and superior caching performance compared to several mainstream packet-level caching schemes.

---

1 Related Work

In traditional TCP/IP networks, IP packets are identified by five-tuple headers while their payloads remain invisible to network devices. Implementing packet-level network caching requires specific methods (such as suppressing replicated data, delta coding, or deep packet inspection) to eliminate data redundancy. Due to high computational overhead, packet-level caching is difficult to achieve in traditional TCP/IP networks, which typically store complete data objects rather than individual packets.

The emergence of Information-Centric Networking (ICN) has made packet-level caching feasible. ICN architectures divide objects into named data chunks, which users retrieve by sending a series of requests containing chunk names. These named chunks can be cached in any ICN-enabled device (such as routers), which can then respond to requests for cached chunks. This capability has attracted increased research attention to packet-level caching schemes.

In prior work, researchers improved overall caching effectiveness by determining content caching locations but did not consider content popularity' s impact on caching performance—a factor that cannot be ignored. Recently, scholars have widely recognized that content popularity plays a crucial role in improving caching performance. Consequently, some caching strategies consider both caching location and content popularity to enhance efficiency. However, these studies lack analysis of the relationship between object popularity and packet popularity, while also overlooking the overhead of packet-level popularity statistics, which creates serious scalability issues.

Other researchers have articulated the advantages of chunk caching and proposed utility quantification methods. Evaluation results show that simple utility-based in-network caching algorithms and low-complexity uniform chunking are sufficient to maximize chunk caching benefits. Thomas et al. proposed an object-oriented packet caching mechanism (OPC) where cache index entries are established for entire objects rather than individual packets (with an object divided into multiple packets). This approach significantly saves high-speed memory usage. To enable packet-level caching, OPC requires the caching system to continuously store the first n blocks of an object without gaps, marked in the cache

index. While OPC addresses two common packet caching problems—cyclic replacement and large-object pollution—it does not consider popularity differences within objects, leaving room for performance improvement. Moreover, storing continuous n data blocks lacks flexibility (chunked caching must be stored sequentially), resulting in rigid caching decisions.

Building on these existing works, we propose a novel optimization approach for packet-level caching.

---

2 Problem Analysis

2.1 Limited Storage Resources

In network caching devices, content data is typically stored in low-speed memory such as DRAM or SSD, while query indexes reside in high-speed memory like SRAM. In traditional caching, one index entry corresponds to an entire content object, whereas in packet-level caching systems, one index entry corresponds to an individual packet. Since a content object usually comprises hundreds of chunks, the number of required index entries increases dramatically, consuming substantial high-speed memory for index tables. The cost of high-speed memory thus limits packet-level caching development.

2.2 Invalid Cache Lookups

In ICN architectures, the caching system is no longer a single device but a complex network of numerous devices. Although the aggregate capacity of the caching network can be large, individual cache nodes face severe capacity constraints due to line-rate forwarding requirements. Consequently, single cache nodes have relatively low cache hit rates—most user requests cannot be satisfied by a single caching device. These missed requests should be forwarded directly to the next hop. Nevertheless, routers always query the index table in external memory, creating performance bottlenecks.

2.3 The Challenge of Packet-Level Popularity Statistics

To improve network caching system efficiency, rapidly pushing high-popularity content objects to the network edge is essential. Traditional network caching nodes typically store entire objects rather than packets. In reality, different portions of a content object—particularly video files—may exhibit different popularity. Packet-level caching can thus improve caching performance. However, object-level popularity statistics cannot fully exploit packet-level caching advantages because using object popularity to estimate packet popularity is often inaccurate.

We illustrate this problem with a simple example in [Figure 1: see original paper]. In Figure 1(a), content object 1 has higher overall popularity than content

object 2. However, for portions A and B, packets in A exhibit higher popularity than those in B. Therefore, based on content object popularity statistics, A' s popularity may be underestimated, reducing the probability of caching these packets. In Figure 1(b), C' s sequence number is smaller than D' s, but D has higher popularity. In this case, prioritizing the caching of earlier packets in a content object (as in OPC) overestimates C' s popularity and caches C instead of the more popular D. Both scenarios degrade caching performance by storing lower-popularity content. This demonstrates that packet-level popularity statistics are crucial for improving caching performance but introduce excessive system overhead that hinders packet-level caching development.

---

3 System Model

3.1 Overview

For descriptive simplicity, this paper uses Named Data Networking (NDN), a typical ICN architecture, to demonstrate our caching mechanism. We consider an NDN network comprising multiple routers with limited-capacity Content Store (CS) caching structures. In NDN, two packet types exist: Interest packets and Data packets. Each Data packet has a name prefix, and clients send Interest packets containing this name prefix to retrieve corresponding Data packets. Intermediate routers forward requests hop-by-hop based on information stored in the Forwarding Information Base (FIB), which is established through name-based routing protocols (such as OSPFN or NLSR). During request processing, routers record path information in their Pending Interest Table (PIT) structure, enabling Data packets to return to requesters along the reverse path.

We assume Data packet sizes equal the network MTU (default 1500 bytes). NDN routers also maintain a Packet Statistics Table (PST) to record and update packet access popularity over time. Cached Data packets are stored in low-speed memory (e.g., DRAM or SSD), while index information resides in high-speed memory (e.g., SRAM). NDN packet names consist of an Object_Prefix and a Sequence number (Seq), where Seq indicates the packet' s order within the content object. For example, the name prefix */ndn/file/video/sample/57* represents packet 57 of object */ndn/file/video/sample*.

3.2.1 Packet Grouping

Typically, index entries are stored in high-speed memory using hash table structures, which offer O(1) time complexity for element operations (lookup, insertion, deletion). However, unlike object-level caching where the minimum storage unit is an entire object, packet-level caching stores individual packets. Since a content object usually comprises hundreds of chunks, numerous index entries are generated, requiring substantial high-speed memory. Nevertheless, different packets belonging to the same object share identical object prefixes, differing

only in their sequence number portion. A possible solution to reduce memory consumption is using a trie structure, but while tries save space by sharing object prefixes, their lookup efficiency is relatively low, especially with large prefix counts—for example, searching for prefix */ndn/file/video/sample/57* requires five matching operations.

To balance query efficiency and storage savings, we adopt a grouped-packet indexing approach. Based on the spatial locality principle—if a specific data block is requested at a particular time, nearby blocks are likely to be requested soon—we group N consecutive packets and designate the first packet' s name prefix as the group prefix. Cache index entries are created based on group prefixes. Each index entry includes a Bit-Map structure to identify whether specific packets within the group are cached. A bit value of 1 indicates the corresponding packet is cached, while 0 indicates it is not. Actual Data packets are stored in low-speed memory as unordered sets, which provide O(1) retrieval complexity. Index entries contain a pointer to the group's container in low-speed memory. Additionally, high-speed memory includes an intrusive list for sorting index entries to implement replacement functionality. [Figure 2: see original paper] illustrates the data structures used in the GPC scheme.

The core caching module operations include:

a) **Insertion**: The router creates an index entry based on the group prefix to which the packet belongs (if not previously created), then sets the corresponding Bit-Map bit to 1.

b) **Eviction**: The corresponding Bit-Map bit is set to 0. If all bits in the index entry' s Bit-Map become 0, the index entry is deleted.

c) **Lookup**: The router first searches the L1 index using the packet' s group prefix. If found, it checks the Bit-Map. If the corresponding bit is 1, the router accesses low-speed memory to retrieve the block data; otherwise, it forwards the request directly.

3.2.2 Request Filtering

As previously discussed, due to caching capacity constraints, a single router cannot satisfy most user requests. To accelerate packet forwarding, we utilize an on-chip Bloom Filter to reduce external memory accesses. A Bloom Filter is a space-efficient, high-performance data structure for fast set membership testing. We synchronize index information to the Bloom Filter. Standard Bloom Filters only support insertion operations; to support deletion, we employ an extended structure called Counting Bloom Filter. All L1 index entry operations are synchronized to the Bloom Filter. Since L1 index entries are based on group prefixes, the number of corresponding elements stored in the Bloom Filter is substantially reduced. For example, for a router with 4GB DRAM, using a group size of 32 and a Bloom Filter false positive probability of 0.05 requires only 273KB to store group prefixes. Storing all packet prefixes would require

approximately 8.5MB for the Bloom Filter. Clearly, grouped-packet caching reduces both L1 index entry count and required Bloom Filter capacity. The required Counting Bloom Filter capacity can be calculated using Equation (1):

≪ MATH_1 ≫

where *C_storage* represents low-speed memory capacity, *packet_size* represents packet size, *group_size* represents group size, and represents false positive probability.

When user requests arrive, the router first tests the Bloom Filter to determine whether the group prefix exists in the L1 index. If true, the router accesses external high-speed memory for subsequent operations; otherwise, it forwards the request directly. This filters the majority of user requests and dramatically reduces external memory accesses. Figure 3: see original paper describes the Interest packet processing flow.

3.2.3 Group-Level Popularity Statistics

Packet-level popularity statistics are difficult to implement due to enormous overhead, particularly storage consumption. We therefore propose group-level popularity statistics. Based on spatial locality, packets with sequential proximity exhibit similar request popularity. Consequently, we use group popularity instead of packet popularity to assist routers in making caching decisions. Routers record group popularity information in the PST, creating indexes based on group prefixes. This substantially reduces popularity statistics overhead.

When Data packets arrive, the router looks up the PST to obtain the popularity of the group to which the packet belongs. If its popularity value exceeds a threshold, the router stores the Data packet; otherwise, it only forwards the Data packet. The threshold is defined as:

≪ MATH_2 ≫

where *Min(popularity)* represents the minimum popularity among content currently stored in the CS, and is a tuning parameter that adapts to dynamic changes in content popularity within the network. Smaller values suit scenarios with rapidly changing content popularity. Figure 3: see original paper describes the Data packet processing flow.

---

4 Performance Evaluation

4.1 Experimental Setup

To evaluate our scheme' s practical performance, we implemented it in ndnSIM, a module based on NS-3 that incorporates NDN data structures and forwarding

logic. We added data structures and modified forwarding logic to realize the GPC scheme. In our evaluation, we defined three schemes with different group sizes (8, 16, and 32), named GPC-8, GPC-16, and GPC-32. We compared these three GPC strategies against the following packet-level caching schemes:

a) **CEE (Cache Everything Everywhere)**: Every caching device caches any packet passing through it.

b) **OPC (Object-oriented Packet Caching)**: An object-oriented packet caching strategy that establishes indexes at the object level and continuously stores the first N packets of a content object.

We assume user requests follow a Zipf distribution and set the default Zipf parameter to 1.0. To examine the impact of Zipf parameters on caching performance, we vary its value from 0.7 to 1.3. We assume the number of packets comprising a content object follows $N(1000, 250)$, with actual sampled packet counts distributed in the interval [68, 2026]. We use traffic generators to create packets with variable popularity and construct experimental topologies using both BA and WS models. The experimental topology consists of 100 nodes, including one randomly selected content server node and 66 edge router nodes, with the remaining nodes serving as intermediate routers. Each node' s cache capacity is set to 1000 packets, the Bloom Filter default capacity is 150 bytes (varied from 90–210 bytes to examine capacity' s filtering effect), and the request rate is 200 requests/second. Except for Zipf distribution parameters, chunk count, and Bloom Filter capacity, all other parameters remain fixed at default values. describes the main experimental parameters.

4.2 Performance Metrics

We define the following performance metrics to evaluate GPC' s practical performance:

a) **Server Load Reduction Ratio (SLRR)**: Describes the ratio by which the caching system reduces content server workload:

≪ MATH_3 ≫

where *S_counts* represents the total number of requests responded to by the server, and *R_counts* represents the total number of user requests.

b) **Average Hop Reduction Ratio (AHRR)**: Reflects the ratio of hop count reduction for user requests due to the caching system:

≪ MATH_4 ≫

where *hops_c* represents hops between the cache-hit node and the requesting user, *hops_nc* represents hops between the requesting user and the content server, and *R_counts* represents the total number of user requests.

c) **Average Cache Hit Ratio (ACHR)**: Reflects the average hit rate across all caching nodes:

≪ MATH_5 ≫

where *Hit_counts_r* represents the number of user requests hitting at router r, and *R_counts_r* represents the total number of user requests received by router r.

d) **Link Load Rate**: Reflects traffic load on specific links during a given period.

e) **Storage Access Reduction Rate (SARR)**: Reflects the ratio of memory access reduction due to Bloom Filter usage:

≪ MATH_6 ≫

where *F_counts_r* represents the total number of user requests filtered by the Bloom Filter at router r, and *R_counts_r* represents the total number of user requests received by router r.

f) **Index Occupancy**: Reflects the high-speed memory capacity occupied by cache indexes at specific nodes.

4.3 Results Analysis

We conducted extensive simulation experiments across multiple topologies, analyzing how varying the Zipf distribution parameter (0.7–1.3) affects caching performance under different strategies. Results demonstrate that the GPC strategy significantly outperforms other schemes across defined metrics while substantially reducing memory consumption.

4.3.1 Caching Performance Figure 4: see original paper–(c) show the impact of Zipf parameters on caching performance. As increases, all three performance metrics improve because higher Zipf parameters increase the popularity of hot content, facilitating better caching system performance. As shown in [Figure 4: see original paper], our scheme outperforms comparison schemes across all three metrics. Notably, GPC-8 achieves optimal caching performance because smaller group sizes yield more precise popularity values, enabling more accurate caching decisions. Compared to OPC, GPC-8 achieves average improvements of 29.53%, 48.86%, and 82.36% across the three metrics. Compared to CEE, performance improvements reach 37.65%, 105.99%, and 125.19% respectively.

4.3.2 Link Load Rate We evaluate caching strategies' impact on link load rates by collecting and analyzing link data transmission states. [Figure 5: see original paper] shows traffic load rates for the top 10 highest-load links under

different caching strategies. Scenarios using GPC-8 exhibit significantly lower link load rates than other strategies. For this metric, GPC-8 achieves average improvements of 40.45% and 19.08% over CEE and OPC, respectively.

4.3.3 External Memory Access [Figure 6: see original paper] demonstrates the Bloom Filter' s effectiveness. With the Bloom Filter, most requests are forwarded directly without accessing external memory. As Bloom Filter capacity increases, forwarding performance improves, though the performance curve is convex, indicating diminishing improvement rates. This shows that Bloom Filter effectiveness has an upper bound, and capacity should not be increased indefinitely for better filtering.

4.3.4 Index Occupancy [Figure 7: see original paper] shows the relationship between low-speed memory capacity and index size. Assuming each index entry occupies 40 bytes and packet size is 1500 bytes, the Bit-Map sizes for GPC-8, GPC-16, and GPC-32 are 1, 2, and 4 bytes respectively. For GPC-8, one index entry can represent up to 8 packets, while a CEE index entry can only identify one packet. Therefore, increasing group size reduces index entry count and consequently high-speed memory usage.

---

5 Conclusion

This paper proposes a novel packet-level caching optimization approach that addresses scalability issues in NDN architecture from multiple perspectives. The strategy divides packets comprising content objects into multiple groups and uses group prefixes to establish index entries, reducing high-speed memory consumption. Additionally, an external Bloom Filter filters invalid cache queries to accelerate packet forwarding rates. Furthermore, the scheme optimizes caching decisions based on group-level popularity statistics to improve overall caching performance. Experimental results demonstrate that the proposed scheme outperforms existing solutions in server load reduction ratio, average hop reduction ratio, and average cache hit ratio while significantly improving resource utilization.

Future work will focus on refining this approach and deploying it on actual NDN platforms for further testing and application.

References

[1] Cisco. Cisco visual networking index: forecast and methodology, 2016–2021 [EB/OL]. (2017) [2017-07-15]. https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/complete-white-paper-c11-481360.html

[2] Wang Lan, Bayhan S, Kangasharju J. Optimal chunking and partial caching in information-centric networks [J]. Elsevier Science Publishers B. V., 2015, 61(C): 48-57.

[3] Jacobson V, Smetters D K, Thornton J D, et al. Networking named content [C]// Proc. of ACM CoNEXT. 2009: 1-12.

[4] Koponen T, Chawla M, Chun B, et al. A data-oriented (and beyond) network architecture [C]// Proc. of ACM SIGCOMM. 2007: 181–192.

[5] FP7 SAIL project [EB/OL]. http://www.sail-project.eu/

[6] Fotiou N, Nikander P, Trossen D, et al. Developing information networking further: from PSIRP to PURSUIT [C]// Proc. of International Conference on Broadband Communications, Networks and Systems. Berlin: Springer, 2010: 1-14.

[7] Chai W K, He Diliang, Psaras I, et al. Cache less for more in information-centric networks [C]// Proc. of NETWORKING. 2012: 27-40.

[8] Psaras I, Chai W K, Pavlou G. Probabilistic in-network caching for information-centric networks [C]// Proc. of ACM SIGCOMM Workshop on ICN. New York: ACM Press, 2012: 55-60.

[9] Zhang Meng, Luo Hongbin, Zhang Hongke. A survey of caching mechanisms in information-centric networking [J]. IEEE Communications Surveys & Tutorials, 2015, 17(3): 1473–1499.

[10] Bernardini C, Silverston T, Festor O. MPC: popularity-based caching strategy for content-centric networks [C]// Proc. of IEEE International Conference on Communications. 2013: 3619-3623.

[11] Wang Wei, Sun Yi, Guo Yang, et al. CRCache: exploiting the correlation between content popularity and network topology information for ICN caching [C]// Proc. of IEEE International Conference on Communications. 2014: 3191-3196.

[12] Ren Jing, Qi Wen, Westphal C, et al. MAGIC: a distributed max-gain in-network caching strategy in information-centric networks [C]// Proc. of IEEE INFOCOM. 2014: 470–475.

[13] Wu Haibo, Li Jun, Zhi Jiang. MBP: a max-benefit probability-based caching strategy in Information-Centric Networking [C]// Proc. of IEEE International Conference on Communications. 2015: 5646-5651.

[14] Hu Xiaoyan, Gong Jian, Cheng Guang. Enhancing in-network caching by coupling cache placement, replacement and location [C]// Proc. of IEEE International Conference on Communications. 2015: 5672-5678.

[15] Banerjee B, Seetharam A, Tellambura C. Greedy caching: a latency-aware caching strategy for information-centric networks [C]// Proc. of IFIP Networking. 2017.

[16] Thomas Y, Xylomenos G, Tsilopoulos C, et al. Object-oriented packet caching for ICN [C]// Proc. of International Conference on Information-Centric Networking. 2015: 89-98.

[17] Wang Lan, Hoque A, Yi Cheng, et al. OSPFN: an OSPF-based routing protocol for named data networking [R]. University of Memphis and University of Arizona, 2012.

[18] Hoque A, Amin S, Alyyan A, et al. NLSR: named-data link state routing protocol [C]// Proc. of ACM SIGCOMM Workshop on Information-Centric Networking. New York: ACM Press, 2013: 15-20.

[19] Afanasyev A, Moiseenko I, Zhang Lixia. ndnSIM: NDN simulator for NS-3, NDN-0005 [R]. NDN, 2012.

[20] Henderson T R, Roy S, Floyd S, et al. ns-3 project goals [C]// Proc. of Workshop on ns-2: the IP Network Simulator. New York: ACM Press, 2006.

[21] Breslau L, Cao Pei, Fan Li, et al. Web caching and Zipf-like distributions: evidence and implications [C]// Proc. of the 18th Joint Conference of the IEEE Computer and Communications Societies. 2002: 126-134.

[22] Barabási A, Albert R. Emergence of scaling in random networks [J]. Science, 1999, 286(5439): 509-512.

[23] Watts D. J, Strogatz S. H. Collective dynamics of small world networks [J]. Letters to Nature, 1998, 393: 440-442.

Note: Figure translations are in progress. See original paper for figures.