ΟΙΚΟΝΟΜΙΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ
ATHENS UNIVERSITY OF ECONOMICS AND BUSINESS
ΣΧΟΛΗ ΕΠΙΣΤΗΜΩΝ & ΤΕΧΝΟΛΟΓΙΑΣ ΤΗΣ ΠΛΗΡΟΦΟΡΙΑΣ
SCHOOL OF INFORMATION SCIENCES & TECHNOLOGY
ΜΕΤΑΠΤΥΧΙΑΚΟ ΣΤΑ ΠΛΗΡΟΦΟΡΙΑΚΑ ΣΥΣΤΗΜΑΤΑ
MSc IN INFORMATION SYSTEMS

MSc Thesis

TOPIC:

«Packet-level Caching in Information-Centric Networking»

FILTISAKOS SPYRIDON - POSTGRADUATE STUDENT

M317016

Supervised by G. Xylomenos

ATHENS, JULY 2019

Table of Contents

Table of Figures

Glossary

DoS	Denial of Service
DPI	Deep-Packet Inspection
DRAM	Dynamic Random-Access Memory
SRAM	Static Random-Access Memory
FIFO	First In - First Out
OPC	Object-oriented Packet Caching
ICN	Information-Centric Networking
CDN	Content Delivery Network
IPSec	Internet Protocol Security
DNSSec	Domain Name System Security Extensions
MTU	Maximum Transmission Unit
LRU	Least Recently Used
EXT	Extended File System
NTFS	New Technology File System
FAT	File Allocation Table
GFS	Google File System
NVM	Non-Volatile Memory
Ptrprev	Pointer - Previous
Ptrnext	Pointer - Next

Abstract

The Information Centric Networking (ICN) architecture is a clean state Internet infrastructure which, unlike the current host-centric communication model, focuses on the content itself, solving multiple issues in security, mobility, and data volume. Object-oriented Packet Caching (OPC) is a caching scheme that integrates object-oriented cache lookups with packet-oriented cache replacement. The OPC space allocation method in slow memory has some shortcomings which can be identified during the object eviction process. Moreover, one-access cache hits are not supported. Both issues are associated with the linked-list structure used in the slow memory, which must be traversed until the right chunk is found, creating this way an overhead. This thesis introduces a new slow memory organization for OPC which offers a faster eviction process with low overhead. The research process is described in detail, as several fields were studied and several alternatives were discarded before reaching the final solution.

1 Introduction

Network traffic is increasing every year. Network infrastructure and organization is racing to keep up with the ever-growing traffic requirements, in order to maintain up-to-expectations performance. Web caching, which takes place at in-network nodes, can contribute greatly to the reduction of load. It does not, however, eliminate all the problems, exhibiting significant scalability and flexibility issues. Information-Centric Networking (ICN) addresses most weaknesses and drawbacks that current cache schemes have, by using a content centric model instead of the current host-centric one. In ICN, information is fragmented into chunks that are assigned with unique names. Named chunks can be independently discovered, routed and forwarded. The information transportation shifts to receiver-driven operation and chunk requests and responses are introduced.

Further performance improvement comes with Object-oriented Packet Caching (OPC), a scheme which incorporated the effectiveness of packet-level caching along with the resource-efficiency of object-oriented caching. It aims at the improvement of router cache hit rates, without requiring additional storage resources. Moreover, OPC eliminates issues raised with packet caches, such as looped replacement and large object poisoning.

Issues still occur with OPC's implementation, mainly with procedures which require that a router's slow memory should be traversed. Object eviction and chunk retrieval procedures are rendered quite costly because they both involve several accesses to slow memory in order to be completed.

With these two problems resolved, OPC's capability as an object-oriented caching scheme can be maximized. This thesis proposes solutions to these exact problems.

2 Information Centric Networking (ICN)

2.1 Introduction

ICN a candidate for the future Internet architecture promising a new architectural framework. Since the Internet today is increasingly used for the transmission of the information rather than the communication between two points, ICN aims to meet present and future needs more effectively than today's Internet architecture. By naming the information on the Network Layer, ICN favors the development of caching on the network and multicasting mechanisms, thus facilitating the efficient and timely distribution of information to users. ICN can also address other constraints to which this online architecture is subject, such as mobility management, and security enforcement. There are four elements emphasized in this architecture:

- Information Naming
- Information Delivery
- Mobility
- Security

Bellow each element is further analyzed:

2.1.1 Information Naming

The role of the Internet has shifted from academic to serving universal interests of any kind. As a result, it is essential to support the distribution of large-scale information and content. Receiving information is currently under the spotlight, whether it is content or data, without having to access a particular computer environment (host or server). However, the current Internet design was designed so that the user needs to define in every application not only the information he or she intends to receive, but also the specific server that holds that information. As a result, unless auxiliary features are used, Internet-level networking mechanisms are unable to locate and transport information from the optimal location in which it is hosted. The ICN approach essentially decouples the information from its sources, in the sense that location and identification are clearly split. The information is identified, addressed and assigned independently of the location it resides at (Arianfar S., 2010) (Mohamed Diallo, 2011).

In the ICN architecture, each fragment of information is named, instead of specifying a pair of hosts that is comprised of a source and a destination. This shift from host naming to information naming has the positive effect that information retrieval is guided by the recipient. Unlike the Internet today, in which senders have exclusive control over the data to be exchanged, with ICN data can only be obtained if explicitly requested by the recipient. In ICN, after sending a request, the network is responsible to determine the best source that can provide the requested information. Thus, the routing operation of information requests is summarized in finding the best (closest) source that holds this information, based on a name, independently of where that information is located.

2.1.2 Information Delivery

Focusing on content-centric applications, requires the Internet to efficiently distribute vast amounts of information, as well as to manage unpredictable traffic surges, commonly referred to as flash crowds.

However, the Internet as it is today does not possess the mechanisms needed to handle the multitude of events and to effectively distribute the information. Under the current Internet architecture, the data to be transferred is treated as a series of bytes that have to be transferred from a source to a destination and therefore cannot realize any optimizations that would otherwise be possible (e.g., smart in-network caching, information replication at various points, information-aware traffic engineering). For the Internet to be able to fully utilize in-network storage capabilities, it must be expanded with mechanisms that can identify and retrieve the information from the optimal site where it resides (Group, 2010). The emergence of such techniques at the application level (e.g. Web caching) confirms that these concepts were an afterthought. These techniques have been implemented by Content Delivery Networks (CDNs) at the application level. However, the amount, location and destination of traffic cannot be predicted if flash crowds occur.

Additionally, the investment on a CDN that includes and caters for all these possible cases is certainly not economically feasible, especially if we take into account the steady increase in user-generated information. For these reasons, it would be preferable to have a new Internet infrastructure that supports mechanisms within the network itself for efficient information retrieval. Thus, by presenting the ICN architecture, the network can now satisfy a request to receive information not only by locating the

original source but also by using in-network caches that keep copies of the requested information. This can be implemented without having to resort to additional costly solutions such as CDNs, because the network level in ICN works directly on named information.

Data packets in the ICN architecture get their name depending on the information they hold. As a result, they can be stored in caches and retrieved very easily, unlike in the classic Internet structure that requires costly techniques such as Deep Packet Inspection (DPI) (Zhaoguang Wang, 2011), (Ashok Anand A. G., 2008), (Ashok Anand V. S., 2009). In addition, by naming the information, ICN supports bundling requests for the same information, thus facilitating distribution to the relevant interested destinations via multicast forwarding.

2.1.3 Mobility

The existing Internet architecture, as was already mentioned, was designed for point-to-point communication between two nodes, with the node ID (IP address) being the main element referring to Hosts, with an IP address that belongs to a network. Surveys conducted for marketing purposes[1] indicate that by 2018 non-fixed nodes that have access to the Internet occupy fifty-eight (58) percent of web traffic while wire-traffic has a percentage of forty-two (42).

Wireless and mobile devices, while they possess the ability to easily switch from one network to another by changing their IP addresses, are unable to achieve continued connectivity in motion, an issue that is now an increasingly important requirement for the future Internet. To address this problem various solutions such as Mobile IP have been proposed; they are considered to be temporary and ineffective due to issues such as triangular routing, valley routing, and exit policy violations.

The mobility problem of the nodes is solved by the publish-subscribe (Patrick Th. Eugster, 2003) model of ICN, in which users interested in information are subscribed in the sense that they are interested in this information, while the users who offer the information publish it, advertising it on the network. The procedure of matching subscriptions to publications is implemented by a rendezvous function performed by brokers within the network that hold this responsibility. The effectiveness of this

[1] Mobile Vs. Desktop Usage In 2019 (https://www.perficientdigital.com/insights/our-research/mobile-vs-desktop-usage-study)

communication model stems from the fact that communication between publishers and subscribers is asynchronous.

Publishers may issue information before receiving a subscriber request, and subscribers can start broadcasting requests immediately after the announcement that this information is available. Also, there is usually no reporting between publishers and subscribers, so publishers do not keep track of how many subscribers receive a post for a specific piece of information.

Respectively, subscribers do hold any records on how many different publishers provide the information they requested (Patrick Th. Eugster, 2003). These properties are the key to solving the mobility problem of nodes, as mobile nodes can simply reissue subscriptions by sending messages of interest for the information they require after a handoff, and the network can redirect these subscriptions to internal caches instead of requesting it from the original publisher.

2.1.4 Security

The Internet was designed to operate in a completely reliable and utopian environment where concepts and actions such as user authentication and data, integrity and privacy were not considered as primary requirements.

Additionally, the Internet was designed to forward any traffic coming into the network, which led to an imbalance of power between senders and recipients. These features allowed spammers, hackers, and generally malicious users to perform Denial of Service (DoS) attacks while covering their tracks effortlessly. To address such malicious attacks, mechanisms such as firewalls, spam filters, etc., as well as new security protocols (e.g., IPSec, DNSSec) have been developed to supplement some other pre-existing ones.

However, these mentioned solutions do not manage to penetrate deep into the grid, and malicious data manages to survive. The end-to-end Internet principle has prevented the deployment of security and reliability mechanisms deeper into the network where they would be more effective both in avoiding or compromising and eventually blocking malicious attacks.

In the ICN architecture, there is no data flow, unless a user has explicitly requested to receive a specific piece of information. As a result, the amount of spam traffic is

significantly reduced. Additionally, in ICN architectures that certify the name of the information, malicious data filtering can be achieved even by in-network mechanisms. Finally, the distinction between users requesting information and those who are processing information, is in itself a novelty that can lead to more effective measures against denial of service (DoS) attacks. This can be implemented by adding an indirection point where separation between the two entities takes place. ICN can assess requests for specific pieces of information in that point before they reach their destination. This approach, known as indirection, in addition to defending against DoS attacks, can also provide user privacy, due to the fact that the publisher does not need to know about the identities of subscribers.

2.2 ICN Foundations for Object-oriented Packet Caching

In ICN bibliography the terms chunk and packet are interchangeable and refers to the Maximum Transmission Unit (MTU), which is the largest packet allowed. There are a few ICN implementations and in most of them the information travels within the network as a set of the above-mentioned data chunks, which carry a unique identifier. The identifier usually consists of the name of the content followed by the packet's rank number, and is placed in the header of the packet, thus relieving the network from the computational cost needed to identify identical packets. In ICN, should two packets or more have the same ID, then they statistically contain the same content.

The ICN transport protocols are mainly receiver-driven (Giovanna Carofiglio, 2012), (Yannis Thomas C. T., 2014). A transmission is conducted via multiple independent chunk transmissions. Each transmission is triggered by a specific packet request and completed by the transmission of the corresponding data packet.

This model lays the foundation for the exploitation and use of on-path caches, using cache buffers or buffer queues, which are placed on the ICN routers as temporary warehouses, thus enabling them to directly handle packet requests for the packets that have been stored.

Research has examined the ubiquitous caching merits (Zhongxing Ming, 2012), (Sumanta Saha, 2013), (Ioannis Psaras, 2012), (Somaya Arianfar, 2010.) by trying to increase its performance by aggregating the capabilities of each cache on each on-path router. However, experience has shown that the distribution of content is not uniform. Thus, some authors argue that by caching on super-nodes located at the edge of the

network it is possible to achieve almost the same performance as caching on all nodes (Seyed Kaveh Fayazbakhsh, 2013)

There are also arguments that establish the idea caching in a subset of nodes that meet certain central requirements, which determine how central the node is (Wei Koong Chai, 2013)

There is a research that deals with the specifics of the ICN packet cache structure (Rossini G., 2014). It suggests a two-level cache model to improve response time. Specifically, it suggests that groups of chunks have to be presented from a slow memory (SRAM) to a fast (DRAM), so that a better response time on consecutive successive requests can be achieved. However, authors suggest this design is suitable only for edge routers, due to the storage requirements and the static content directory it requires.

In-network routers SRAM and DRAM should be used for wire-speed operation. Most of the different approaches simply consider a Least Recently Used (LRU) replacement policy (Sumanta Saha, 2013), (Ioannis Psaras, 2012), (Seyed Kaveh Fayazbakhsh, 2013), (Wei Koong Chai, 2013) or other innovative policies for the even distribution of cached content along the paths (Zhongxing Ming, 2012), (Somaya Arianfar, 2010.), (Mikhail Badov, 2014) without taking into account that router cache performance can be limited by the size of their fast memory.

2.3 ICN issues that led to OPC development

Object-oriented packet caching's main design purpose was to address ICN's design issues by focusing on aspects that can be improved. These issues are described below:

2.3.1 Limited storage resources

Typically, a cache policy design is implemented through a hash-table structure that spreads across the slow and fast system memory. The system's fast (but expensive) memory stores the hash-table and assigns a hash identifier to a pointer that shows the packet data in the slow (but inexpensive) system memory (Somaya Arianfar, 2010.), (Anirudh Badam, 2009). Most cache policy designs assume have 1500-byte chunks and LRU records of at least 32 bytes (Anirudh Badam, 2009). The ratio of fast and slow memory entries is one-to-one, leading to the requirement that the ratio between sizes of the fast and slow memory is about 1:46. The DRAM memory of a router many times

larger in size than an SRAM single-chip, resulting in the use of only a small percentage of the available storage space on the network to be indexed at the packet-level, as there is not enough SRAM for indexing packets. Possible solutions raise other issues such a caching granularity and inability to achieve wire-speed operation.

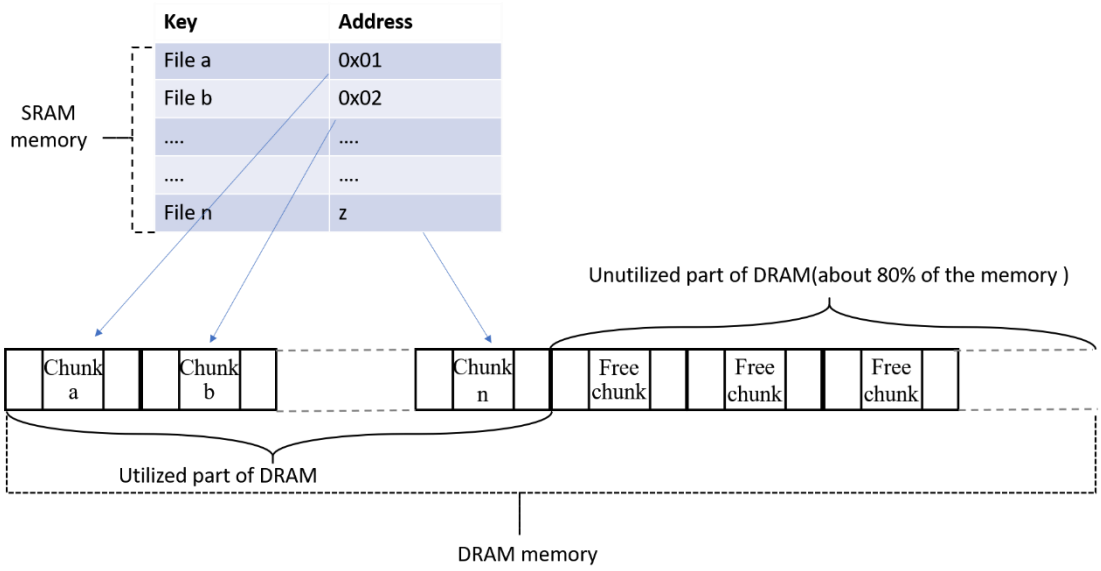

Figure 1. SRAM – DRAM pairing: Each SRAM object points to a single chunk in DRAM.

2.3.2 Looped replacement

Unlike object-caches, packet-caches, depending on the replacement policy and the access pattern, may contain only one chunk of an object. In most applications, an object's packets are requested in ascending sequential order. In a cache with LRU replacement policy, this results in the first packets of the object being deleted from the cache before the last packets, since they are stored longer in memory.

This effect is called looped replacement and can occur in any cache size if the LRU policy is used, since the object packets are sequentially requested and requests for the same object are not so frequent.

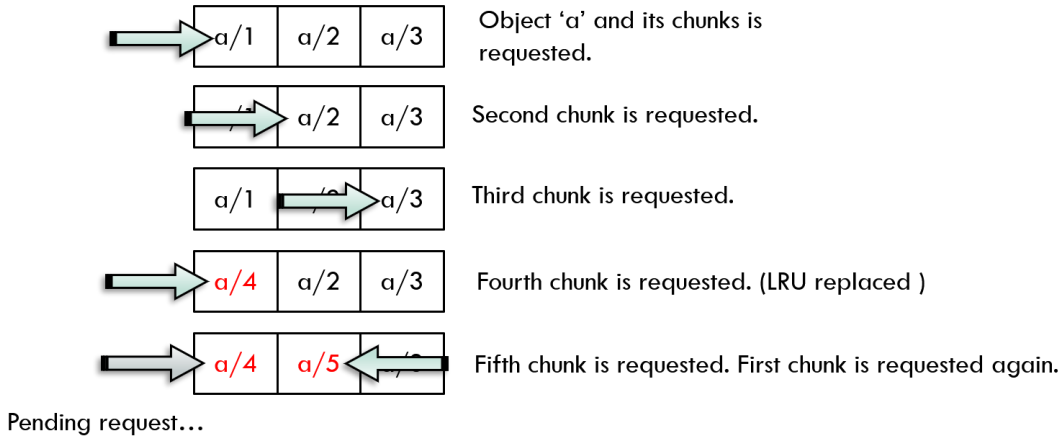

Figure 2. Looped Replacement: When the first chunk is evicted and then requested again a loop of evictions and insertions begins until the last chunk is retrieved.

2.3.3 Large object poisoning

Small in-network caches struggle with the management of large, but unpopular objects. Cache designs, either at the packet level or at the object level, using the LRU replacement policy, store all the chunks of each object regardless of how popular they are. This can significantly reduce cache performance, especially in cases where large files that occupy a significant amount of memory space are involved. This wastes significant cache resources, since some stored content is not actually useful.

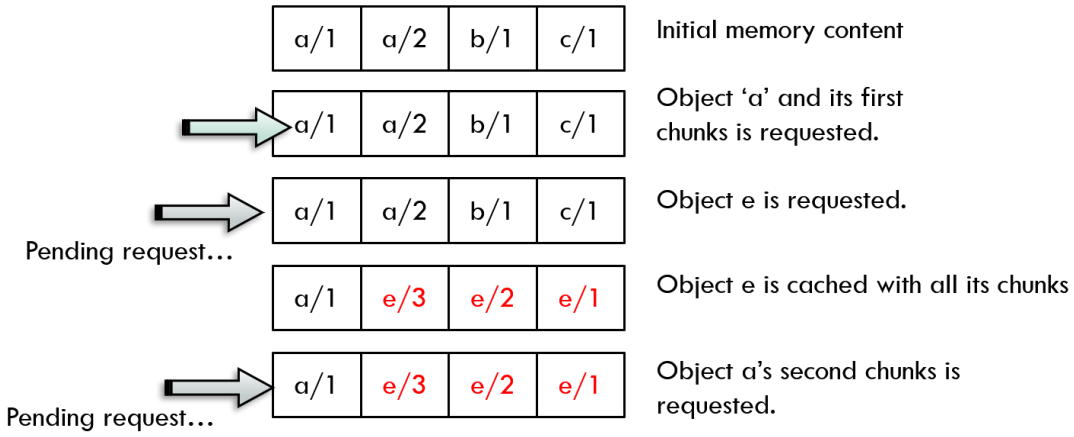

Figure 3 Large object poisoning: When object e is requested and cached, it evicts more popular chunks of other objects.

3 Object-oriented Packet Caching (OPC)

3.1 OPC scheme characteristics

OPC design focuses and effectively addresses the weaknesses of ICN packet caches. It combines the effectiveness of object-oriented caching searches with a packet-level replacement policy. Based on the observation that most applications request the packets of an object in ascending sequential order, the initial segment of an object is always stored, from its first chunk until its n-th, without any gaps. Therefore, any partially stored objects are always represented by their first n packets, thus avoiding looped replacement. The OPC lookup index contains the object's name and a counter for each stored object. This counter indicates the number of chunks stored in the cache and it is called *last chunk id*, thus providing the identification for the cached object's last packet. If a request for an object with a sequence number/rank less than or equal to the last chunk id is received, then the cache has this packet stored and can serve directly this request. Otherwise, when a request with a higher chunk rank than the last chunk id is received, then the cache forwards the request to the actual destination. This technique reduces indexing costs to one record for each (partially) stored file or, depending on the file size, close to that of an ordinary LRU packet cache

3.2 OPC architecture and data structures

An OPC node maintains three data structures: two for the insertion and chunk lookup processes, and one for the eviction process.

The first two structures are called Layer 1 (L1) and Layer 2 (L2) indexes and organize the data at the object-level and the cache-level, respectively. In particular, the L1 index is stored in the fast memory (e.g., SRAM) and is implemented as a fixed-size hash table, with a single record for each cached object. Each entry in L1 assigns a content identification with a pair of values: the rank / order of the last chunk id of that object and a pointer pointing to the last chunk of the object stored in the L2 index (Ptr_{mem}). The L2 index on the other hand is essentially an array in the slow memory (e.g. DRAM) which contains the cached chunks for each of the objects in sequential order.

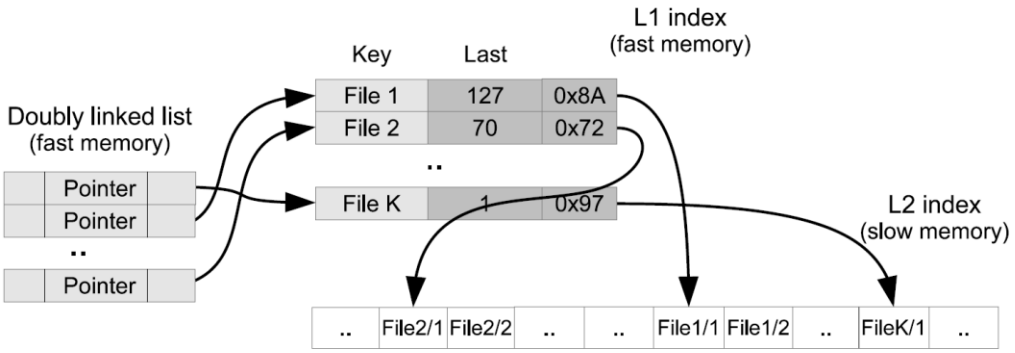

Figure 4. Data structures used by OPC. source: Object-oriented Packet Caching for ICN.

Chunks of the same object are not stored in contiguous memory space, as the total number of chunks per object is variable. They form instead a linked list, in which the last chunk of the object is the starting point. Therefore, each chunk slot in L2 consists of a data chunk and a pointer Ptr_{prev} to the previous chunk of the same object. The combination of Ptr_{mem} of L1 structure and Ptr_{prev} of L2 forms a linked list per object, where the head of the list is the last chunk of the object. Moreover, one global pointer, Ptr_{free} points at a list of available chunks, which are also linked via their Ptr_{prev} pointers. When a new data chunk is received, with the assistance of the L1 index it is verified whether the object for this chunk is stored in the cache and the incoming chunk is the next one in the sequence. If this happens to be true, it is stored in the L1 index and the last chunk id is increased by 1. If the object is not stored and the chunk is the first chunk of the object, then we store the chunk in the L2 index and create a new record in the L1 index, with the last chunk id being equal to 1 and Ptr_{mem} pointing to the chunk stored in L2. Otherwise, the chunk is ignored.

The third data structure in the OPC is a double-linked list which is utilized for the classification of objects according to the replacement policy adopted. This list is also stored in the fast memory and indicates the least important (depending on the policy) object stored in the OPC cache. This object is to be removed when the cache is full and memory space needs to be released. In the OPC implementation, the replacement policy used is the LRU, and therefore the items are ranked by the double-linked list according to which one was used more recently, thus removing the objects that have not been used for the most time from the cache. However, the OPC design is not limited to a replacement policy, so stored content can be organized with either LRU, LFU or FIFO

structure. If an object is to be removed from the cache due to lack of space in the L1 index, then both the L1 record and all the chunks that it points to in the L2 are retrieved. If removal is due to lack of space in the L2 index, only the last chunk of the selected record is retrieved and the L1 record is updated by decreasing the last chunk id by 1.

3.2.1 Caching algorithm behavior

To verify that OPC always keeps stored the initial chunks of a file, the following packet replacement algorithm was implemented: OPC inserts a chunk with rank / order *i* in the cache whether it is the first chunk of that object or there is already stored a chunk with rank *i - 1*. In the first case, when chunk *i* is the first chunk of the object, a new index record is created for the corresponding content. During the second case, that is, when the *i-1* chunk has already been stored for an object, the last chunk id is updated. This feature guarantees that at any time the cache will always hold the first piece of each object, without any gaps. If there is no space in the slow memory to store a new chunk, then an object-level LRU list is used in the selection of an object in the cache and then the last chunk of the selected object is deleted, so that the cache again keeps stored the first chunks of an object without spaces in between. If there is no space in the fast memory for a new object, then the index record for this object in the queue / end of the LRU is deleted along with the corresponding chunks of the object in the slow memory. Specifically, the entry pointed at by Ptr_{free} is used, and Ptr_{free} is modified to point to the next free chunk. The new chunk is linked to the list of the appropriate object by modifying its $Prev_{ptr}$ to the previous head of that object's list and updating the Ptr_{mem} of that object to point at the new chunk.

3.2.2 OPC issues

When an entire object is to be evicted, the whole list for this object must be traversed. The Ptr_{free} pointer is made to point at the head of the evicted object's list and then we traverse the list following its Ptr_{prev} pointers until its last pointer is reached and modified to point at the previous head of the free list. Essentially, we place the evicted object's chunks at the head of the free list. Each hop requires a slow memory access for that node which we assume requires 55ns, while SRAM access is assumed to require 0,45ns.

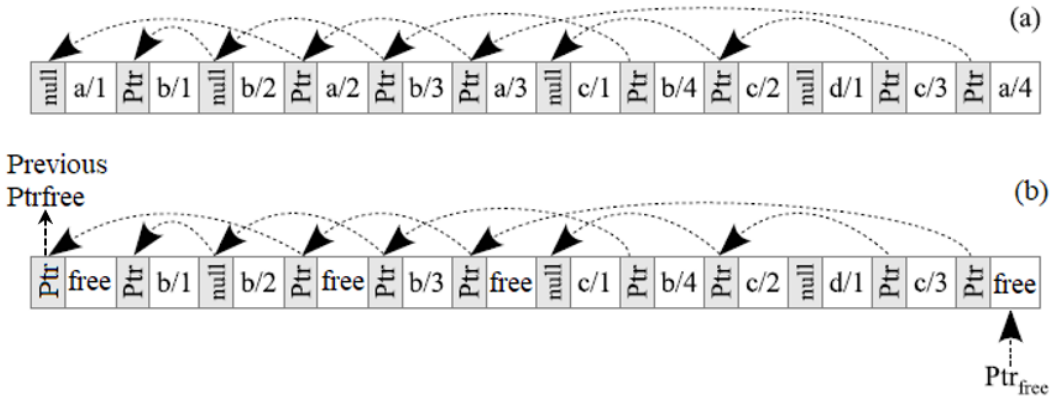

Figure 5. (a) 'a' object's chunks before eviction (b) freed chunks after eviction

The main overhead of the OPC resides in the fact that One-access cache hits are not supported. When a cached chunk needs to be retrieved, OPC must follow the object's linked list from the last stored chunk, until the right chunk is found.

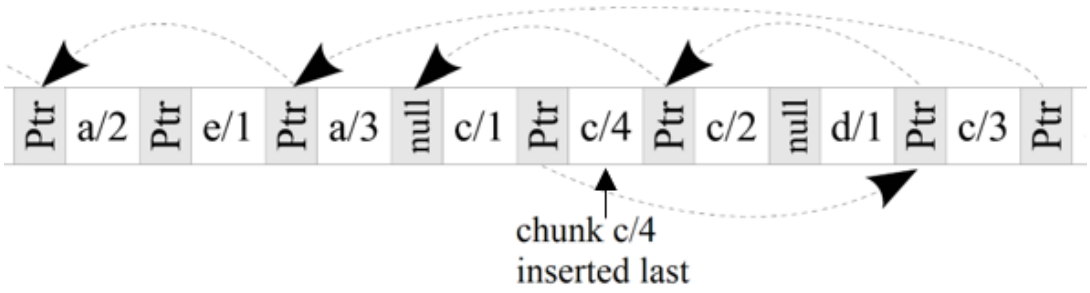

Figure 6. L1 pointer to object's last inserted chunk in DRAM.

4 Research Process

Data organization, storage and retrieval were the subject of this thesis in an abstract, simplified definition. In this context, the following data-related fields where researched in order to obtain a complete view of this technological and scientific field.

4.1 Filesystems

Filesystems was the first field that was researched as it contains the basic principles of how data is stored and retrieved in a storage medium. Advanced space management techniques utilized in filesystems could provide an efficient solution to OPC issues. Moreover, there is a multitude of filesystems for networks of servers. This fact along with the web's architecture sets the basis for forming a solution that includes more than one node on the network. Among the systems that were researched were all extended file systems (EXT), New Technology File system (NTFS), File Allocation Table (FAT) and Google file system (GFS).

Additionally, other less known file systems, which were extensively analyzed in scientific papers were included in the study. Ramcloud, a log-structured filesystem for DRAM-based storage and NOVA (Jian Xu, 2016), again a log-structured file system for hybrid volatile and non-volatile memory systems. These two filesystems stood out because they had both DRAM type random access memories as the main component that stored its structures. Borrowing from their eviction and space saving techniques, the first solution was formed.

NOVA has a dynamic memory structure formed from several 2MB block arrays. Pointers were set to indicate the first and the last part of an object, with the last being the newest. Each part of the object pointed to the next part of the same object. By adopting this structure, the original OPC linked list structure would be kept and the extra pointer would return the first position of the object for a faster eviction process. The SRAM memory though would change structure, as it would store pointers for the static block array plus the pointers for the beginning and the last part of each object in memory. The eviction method included the vacation of a whole block by moving the parts of the block that would be kept, together with other parts of the same object, thus eventually forming blocks that consist only from parts of the same object, giving the

opportunity to retrieve the entire object, speeding up the hit process. Eventually, this approach was dismissed as it would be costly in memory resources for SRAM and overhead was caused from moving whole blocks of data chunks in slow memory. The fact that the eviction process would occur whenever the node memory was at least at 90% capacity was also a risk for the system bandwidth to run out.

Ramcloud with a two-level cleaning policy in conjunction with the eviction process described above would be a solution. Apart from the regular eviction processes which would be adapted to a router's node specifications, another process called "compaction" would take place when a certain threshold of memory utilization was reached (Stephen M. Rumble, 2014). That process would initiate a compaction for the block arrays which are partially empty, moving and pairing their data so that the memory free some resources.

This solution was also abandoned as it shifted the overhead from the memory to the node's CPU. The computational capacity of each node's resources would be drained from the constant checks that the router should perform to verify if the threshold it reached.

4.2 Non-volatile and volatile storage organization.

A focus on the non-volatile (NV) and volatile (V) storage organization was the next step for the research. Techniques for efficiently managing NVMs and hybrid systems which utilized a combination of disk and RAM to create persistent memory structures. Anything that could be used efficiently for OPC improvement with a similar structure and alterations that eliminate the current structure's benefits would be avoided. Nothing with potential impact for our research could be retrieved from this field.

4.3 Data replacement Algorithms.

"Data replacement algorithms" as a term refers to memory allocators, whether they focus in data eviction or in data insertion. Performance was a major factor in this research area as it was directly linked with the thesis' purpose. Any survey, case study or research that fit the criteria could contribute to further the research. Yet, most of the papers referred to memory allocators to be used in file systems and the majority of them

was less complex than the ones used the filesystems which were previously researched, thus not contributing drastically to the study.

4.4 Cache replacement policies.

Cache algorithms, also called cache replacement algorithms or cache replacement policies were researched in order to optimize the data eviction scheme. Currently, OPC uses an LRU scheme to determine which object will be partly or entirely evicted. As stated in the paper, a FIFO algorithm would have interchangeable results in the decision process. However, other cache replacement algorithms were researched in order to ascertain that other algorithms could be considered as a more efficient solution. Multi queue (MQ), Adaptive replacement Cache (ARC) and Clock with adaptive replacement (CAR) and a variation called “Compact CAR” (Atsushi Ookaa, 2016) were examined but eventually the idea of their use was dropped because these algorithms did not provide substantial improvement for the two problems this research wanted to solve.

4.5 Buffer Algorithms

Circular buffer was mainly taken into account to as its concept was the inspiration to produce the final solution for data structure in slow memory (Cooke, 2014). Bucket was also considered to be used in the first NOVA & Ramcloud aborted solution (Fisk, 2005).

5 Thesis proposal

In my implementation of OPC, each chunk slot in L2 has an extra chunk pointer Ptr_{next} to the next object chunk, apart from the $Ptr_{prev,}$ thus forming a doubly linked list for the chunks of each object. Each chunk points to its next and previous chunk, apart from the first chunk of each object where Ptr_{prev} points to the chunk itself. Moreover, the last stored chunk of each object has the Ptr_{next} pointing to the first chunk of the same object, thus forming a semi-circular structure. The Ptr_{prev} for the first chunk points at itself for reasons described in "chunk storage process" section below. Ptr_{free} is also modified to consist of two pointers, PTR_{first} and PTR_{last}, pointing at the first and last chunk in the free list, respectively. An example of the L2 structure is shown in the following figure.

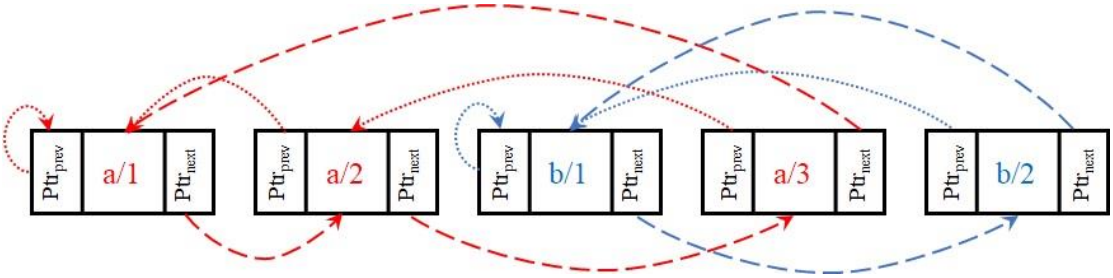

Figure 7. Proposed implementation of OPC L2 structure.

5.1 Improving cache eviction

The first problem that has arisen with OPC is that during the process of eviction of an object in its entirety, the entire linked list residing in DRAM needs to be traversed, rendering this procedure costly. Each hop to retrieve the previous chunk of the object costs a DRAM access, eventually adding up for *n* DRAM accesses for each currently stored chunk, where *n* is the total number of stored chunks.

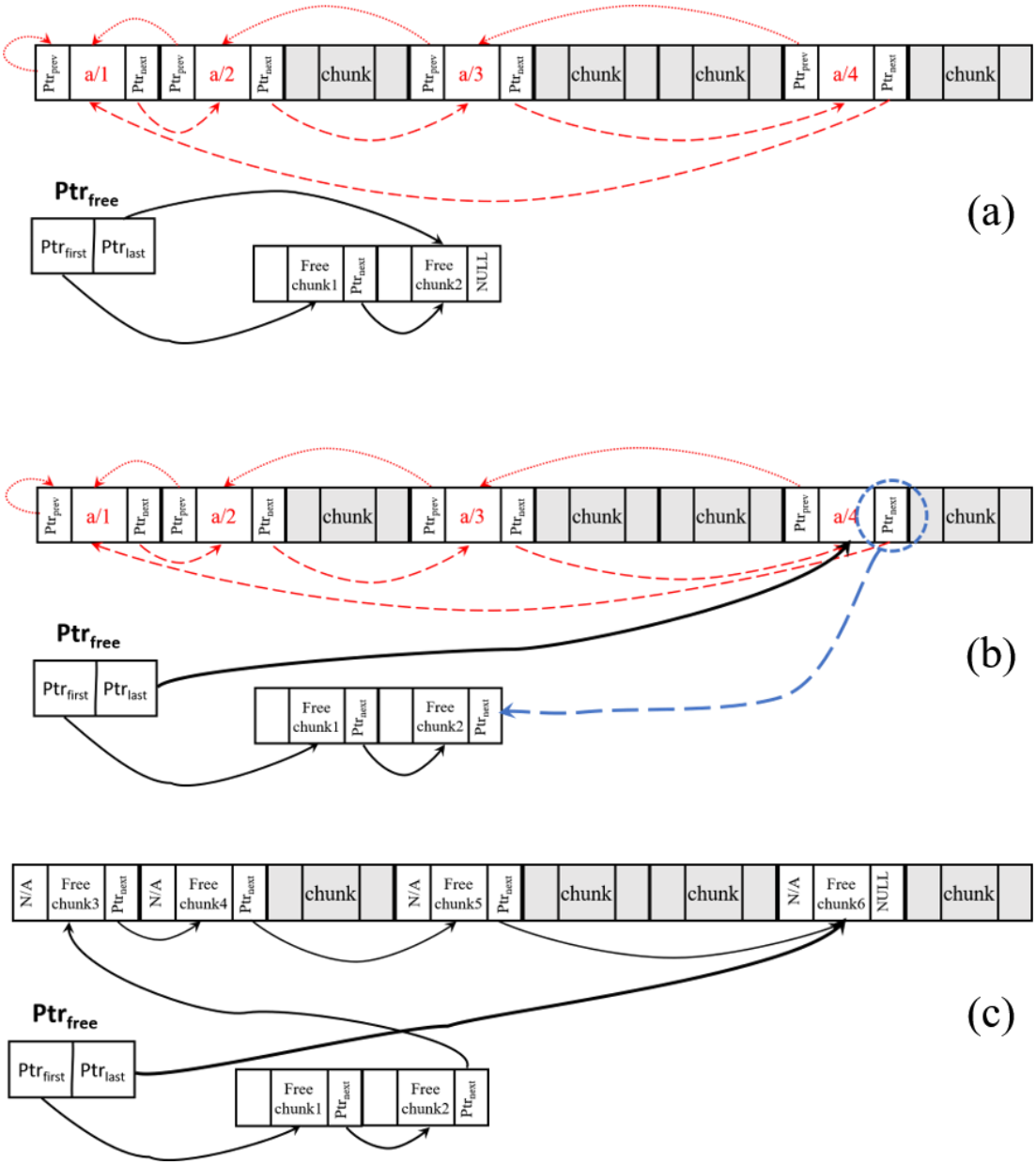

Figure 8. Modified object eviction in L2. (a) Ptr_{free} before the eviction of object a and its chunks in L2. (b) The Ptr_{last} of Ptr_{free} is made to point at the last chunk of a (chunk 'a/4') and the Ptr_{next} of chunk 'a/4' is copied to the Ptr_{next} of the last free chunk, making it point at the first chunk of the object that is being evicted. (c) The Ptr_{next} of the last chunk is updated to null.

To solve this problem, we exploit the semi-circular structure of the per-object lists, to add the entire cached object to the free list without traversing any lists. Specifically, as shown in the figure above, the Ptr_{last} pointer of the free list is made to point at the last chunk of the evicted object, but then, instead of traversing the object to find its first chunk, we use the Ptr_{next} pointer of the last chunk to directly find the first chunk, and modify the pointer at the end of the free list to point at that chunk. Finally, we set the Ptr_{next} pointer of the last chunk to null, to show that this is the end of the free list.

In this implementation, the Ptr_{free} array is traversed using $Ptr_{next.}$ and its inventory is emptied in FIFO fashion. This whole procedure reduces the cost from 1 SRAM + n*DRAM to 1 SRAM + 1 DRAM (we only read and modify the pointers at the last chunk in L2), while the number of the stored chunks does not affect the procedure.

Single chunk eviction procedure is also altered with the new L2 structure. In this situation, an overhead has been created because: when we remove the last chunk, so as to avoid the problems that OPC solves (looped replacement), we update the Ptr_{next} of the last chunk to null and then we use Ptr_{prev} to update the Ptr_{next} of the previous chunk to point at the first chunk of the object, in order to keep the circular structure. This procedure raises the access cost of a single chunk eviction from 1 SRAM + 1 DRAM to 1 SRAM + 2 DRAM, since we need to access two chunks in L2.

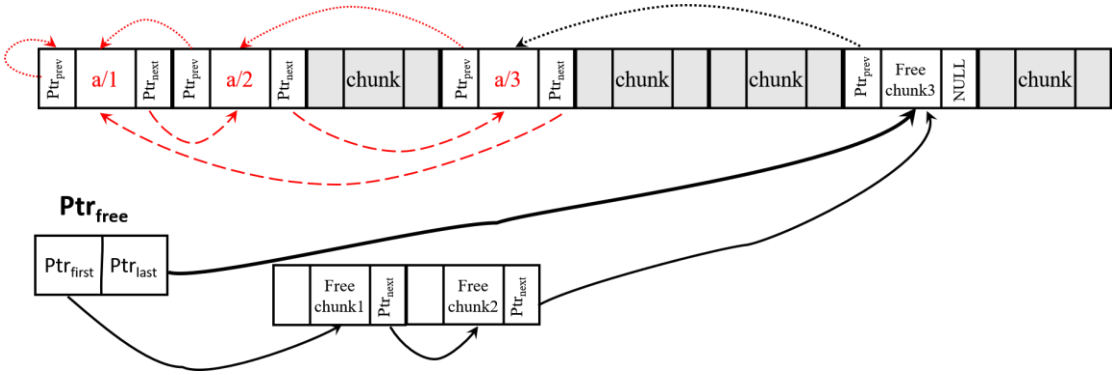

Figure 9. Single chunk eviction

5.2 Reducing cache hit overhead

The second problem that needs to be addressed is that One-access cache hits are not supported, since an object's linked-list must be followed until the right chunk is found, creating an overhead. This problem is partly solved by improving the way the linked list is traversed, exploiting the fact that the object lists are now bi-directional. When the Interest packet is received and it is confirmed that the requested object's chunk is indeed cached, an additional check is performed to assess whether the requested chunk's rank is less than half than the last_chunk's id rank; that is, we check whether the requested chunk is in the first or the second half of the linked list. If the requested chunk is in the first part, we start from the first chunk and the list is traversed using the Ptr_{next} pointers. Otherwise, the linked list is traversed just like in the previous implementation using Ptr_{prev} and starting from the last chunk. Essentially, we cut the size of the list to be

traversed in half. If the "last_chunk_id/2" value is less or equal than the chunk_id requested, then the cost of the accesses will be 1 SRAM + m * DRAM where *m* <= n/2 (by using Ptr_{prev}). If the last_chunk_id/2" value is greater than the chunk_id requested, this means that a low rank chunk need to be fetched, so Ptr_{next} will be used to traverse the list. In this case, the cost is 1SRAM + y * DRAM time where *y* < n/2. The last chunk will be accessed in both scenarios, thus creating a negligible overhead of one additional access in the second scenario.

5.3 Updated chunk storage process

When the first chunk is placed in the memory, both Ptr_{prev} and Ptr_{next} point at itself. The second chunk will have its Ptr_{prev} point at the previous chunk and update its Ptr_{next} to point to the previous chunk's Ptr_{next} which in this case is the first chunk. Then the previous chunk will be accessed and its Ptr_{next} will be updated to point to the newly placed chunk. The same process is repeated during the insertion of the third and each other chunk. In this manner, the pointer to the first chunk will be always stored in the Ptr_{next} of the last chunk, ensuring that the structure remains semi-circular. This procedure also creates an overhead compared to its predecessor raising the access cost from 1 SRAM + 1DRAM to 1 SRAM + 2 DRAM.

5.4 Gains and Costs Overview

To recapitulate the improvements of OPC with the changes proposed in this thesis, without disregarding the overhead created in a few circumstances, they are listed below:

- Single chunk insertions and evictions used to cost 1SRAM + 1DRAM accesses, while in our implementation the cost has been slightly raised to 1SRAM + 2DRAM accesses.
- Whole object evictions used to cost 1SRAM + n*DRAM accesses, where n is the number of stored object chunks. In our implementation this cost is reduced to 1SRAM + 2DRAM accesses.
- Packet access used to cost 1SRAM + m*DRAM accesses where m is the number of hops followed in the linked list from the last stored chunk to the requested one. Although the cost formula remains the same, the number of hops needed

to traverse the requested chunk is decreased up to half of the value of the previous *m.* In our implementation *m* take values [1, m/2] as less hops are needed to traverse the semi-circular list if the chunk requested has a low rank.

6 Conclusions and future work

In conclusion, the two main drawbacks of OPC scheme have been addressed, with a solution that creates a small overhead for the slow memory capacity (one extra pointer per chunk). Moreover, the object eviction process has been greatly improved access-wise, while the chunk retrieval process was made more efficient in a sense that it requires half of the previously needed performance.

The experimental evaluation of the algorithm presented in this thesis can implemented with a network simulator as future work. In addition, the OPC scheme has further room for improvement to minimize the number of accesses in the slow memory as much as possible.

7 References

Anirudh Badam, K. P. (2009). HashCache: Cache Storage for the Next Billion. *Proceedings of the 6th USENIX Symposium on Networked Systems Design and Implementation*, (σσ. 123–136). Boston, MA, USA.

Arianfar S., N. P. (2010). On content-centric router design and implications. *CoNEXT International Conference On Emerging Networking Experiments And Technologies.* Philadelphia, Pennsylvania: ACM.

Ashok Anand, A. G. (2008). Packet caches on routers: The implications of universal redundant traffic elimination. *Proceedings of the ACM SIGCOMM 2008 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications.* Seattle, WA, USA.

Ashok Anand, V. S. (2009). SmartRE: an architecture for coordinated network-wide redundancy elimination. *SIGCOMM '09 Proceedings of the ACM SIGCOMM 2009 conference on Data communication* (σσ. 87-98). Barcelona, Spain: ACM New York, NY, USA ©2009.

Atsushi Ookaa, S. E. (2016). Compact CAR: Low-Overhead Cache Replacement Policy for an ICN Router. *IEICE Transactions on Communications*, 1366-1378.

Borislav Djordjevic, V. T. (2012). Ext4 file system in Linux Environment: Features and Performance Analysis. *INTERNATIONAL JOURNAL OF COMPUTERS*, 288-293 .

Cooke, S. (2014, Sept 19). *The Bip Buffer - The Circular Buffer with a Twist.* Ανάκτηση από CODE PROJECT: https://www.codeproject.com/Articles/3479/The-Bip-Buffer-The-Circular-Buffer-with-a-Twist

Enge, E. (2018, May 1). *MOBILE VS. DESKTOP USAGE IN 2019*. Ανάκτηση από Preficient Digital: https://www.perficientdigital.com/insights/our-research/digital-personal-assistants-study

Fisk, D. (2005). *Programming with Punched Cards.* Ανάκτηση από /www.columbia.edu: http://www.columbia.edu/cu/computinghistory/fisk.pdf

Giovanna Carofiglio, M. G. (2012). ICP: Design and evaluation of an Interest control protocol for content-centric networking. *2012 Proceedings IEEE INFOCOM Workshops.* Orlando, FL, USA: IEEE.

Group, E. C. (2010, March). *Fundamental Limitations of Current Internet and the path to future Internet.* Ανάκτηση από http://www.future-internet.eu/publications/view/article/fundamentallimitations-

Ioannis Psaras, W. K. (2012). Probabilistic in-network caching for information-centric networks. *ICN '12 Proceedings of the second edition of the ICN workshop on Information-centric networking* (σσ. 55-60). Helsinki, Finland: ACM New York, NY, USA ©2012.

Jian Xu, S. S. (2016). NOVA: A Log-structured File System for Hybrid Volatile/Non-volatile Main Memories. *Proceedings of the 14th USENIX Conference on File and Storage*

Technologies (FAST '16) (σσ. 323-338). Santa Clara, CA, USA: USENIX Association Berkeley, CA, USA ©2016.

Mikhail Badov, A. S. (2014). Congestion-aware caching and search in information-centric networks. *ACM-ICN '14 Proceedings of the 1st ACM Conference on Information-Centric Networking* (σσ. 37-46). Paris, France : ACM New York, NY, USA ©2014.

Mohamed Diallo, S. F. (2011). Leveraging Caching for Internet-Scale Content-Based Publish/Subscribe Networks. *2011 IEEE International Conference on Communications (ICC).* Kyoto, Japan: IEEE.

Patrick Th. Eugster, P. A.-M. (2003). The many faces of publish/subscribe. *ACM Computing Surveys (CSUR)*, 114-131.

Rossini G., R. D. (2014). Multi-Terabyte and multi-Gbps information centric routers. *IEEE* (σσ. 181–189). Toronto, ON, Canada: IEEE.

Seyed Kaveh Fayazbakhsh, Y. L. (2013). Less pain, most of the gain: incrementally deployable ICN. *SIGCOMM '13 Proceedings of the ACM SIGCOMM 2013 conference on SIGCOMM* (σσ. 147-158). Hong Kong, China: ACM New York, NY, USA ©2013.

Somaya Arianfar, P. N. (2010.). Packet-level Caching for Information-centric Networking. *Proc. of the ACM ReArch Workshop.*

Stephen M. Rumble, A. K. (2014). Log-structured Memory for DRAM-based Storage. *Proceedings of the* (σσ. 1-16). Santa Clara, CA USA: USENIX Association Berkeley, CA, USA ©2014.

Sumanta Saha, A. L.-J. (2013). Cooperative caching through routing control in information-centric networks. *2013 Proceedings IEEE INFOCOM* (σσ. 100–104). Turin, Italy: IEEE.

Wei Koong Chai, D. H. (2013). Cache "less for more" in information-centric networks. *Computer Communications*, 758-770.

Yannis Thomas, C. T. (2014). Accelerating File Downloads in Publish Subscribe Internetworking with Multisource and Multipath Transfers. *WTC 2014; World Telecommunications Congress 2014.* Berlin, Germany: VDE.

Yannis Thomas, G. X. (2015). Object-oriented Packet Caching for ICN. *ACM-ICN '15 Proceedings of the 2nd ACM Conference on Information-Centric Networking* (σσ. 89-98). San Francisco, California, USA: ACM New York, NY, USA ©2015.

Zhaoguang Wang, Z. Q. (2011). An untold story of middleboxes in cellular networks. *SIGCOMM '11 Proceedings of the ACM SIGCOMM 2011 conference* (σσ. 374-385). Toronto, Ontario, Canada: ACM New York, NY, USA ©2011.

Zhongxing Ming, M. X. (2012). Age-based cooperative caching in Information-Centric Networks. *2012 Proceedings IEEE INFOCOM Workshops* (σσ. 268–273). Orlando, FL, USA: IEEE.