International Journal of Computing and Digital Systems
ISSN (2210-142X)
Int. J. Com. Dig. Sys. 14, No.1 (Oct. 2023)
http://dx.doi.org/10.12785/ijcds/1401106

Network Traffic Analysis as a Strategy for Cache Management

Ramzi A. Haraty[1] and Nancy M. Rahal[1]

[1] Department of Computer Science and Mathematics, Lebanese American University, Beirut, Lebanon

Received 3 Jan., Revised 29 Jul. 2023, Accepted 28 Sep. 2023, Oct. 1 2023

Abstract: Given the significant revolution in data and technology and the increasing need to optimize system performance, it is imperative to transition towards unconventional caching methodologies in the context of contemporary technologies such as AI, edge computing, heterogeneous IoT, big data, and 5G mobile networks. This research paper presents a novel methodology for cache writing, wherein network traffic is carefully analyzed and examined, and data is categorized as highly accessed utilizing the probability distribution of a bell curve. The proposed approach focuses on analyzing network traffic and selectively writing data into the cache, unlike the old approach, which ignored network traffic and randomly added data into the cache. Empirical findings demonstrate favorable performance outcomes in comparison to conventional techniques, namely LIFO, FIFO, LRU, LFU, MFU, and MRU.

Keywords: cache management, network traffic, hit ratio, total delay.

1. Introduction

Data caching involves storing multiple copies of data or files in a temporary storage space, known as a cache, to facilitate faster retrieval. It plays a crucial role in various fields, including mobile computing, by utilizing prefetching techniques to store recently accessed data in local buffers for future use. When a user requests data that is already present in the cache, the client can immediately respond with the requested data, eliminating the need to access the server. This approach significantly reduces query time and conserves bandwidth, leading to improved system performance [1]. However, maintaining cache performance necessitates the adoption of a cache replacement algorithm when the cache becomes full. Caching techniques find application in databases, content delivery networks (CDNs), Domain Name Systems (DNS), session management, application programming interfaces (APIs), hybrid environments, web caching, available cache, integrated cache, and more.

In the field of data management, caching has emerged as a vital mechanism for enhancing system performance, particularly in mobile computing environments. By storing frequently accessed data locally, caching minimizes query delays and optimizes resource utilization. This paper focuses on the challenges and considerations associated with data caching in mobile computing environments, aiming to propose an effective cache replacement algorithm that maximizes cache performance.

The primary objectives of this research are as follows:

1. Improve Performance: The study aims to enhance the performance of data retrieval in mobile computing environments by leveraging caching techniques. By reducing query delays and conserving bandwidth, the proposed cache replacement algorithm seeks to optimize system performance and enhance user experience.

2. Address Limitations: Caching, while beneficial, presents certain limitations, such as cache size restrictions. This paper aims to tackle these limitations by introducing a cache replacement strategy that efficiently manages cache space, ensuring optimal data storage and retrieval.

3. Enhance Cache Consistency: Maintaining cache consistency is crucial to prevent data inconsistencies between the client cache and the server. The proposed cache replacement algorithm endeavors to maintain cache consistency by adopting appropriate strategies that synchronize cached data with the server's data.

4. Analyze Network Traffic: The research focuses on analyzing network traffic patterns to gain insights into data access frequencies and popular data items. By considering the probability distribution of a bell curve, the proposed algorithm intelligently categorizes data and determines which items should be stored in the cache to maximize the overall profit value.

The key contributions of this paper can be summarized as follows:

1. Novel Cache Replacement Algorithm: This research introduces a novel cache replacement algorithm specifically designed for mobile computing environments. Unlike traditional approaches that ignore network traffic patterns, the proposed algorithm leverages network traffic analysis to make informed decisions on data storage and eviction in the cache. By selectively caching highly accessed data items, the algorithm aims to improve cache hit rates and overall system performance.

2. The key contributions of this paper can be summarized as follows:

E-mail: rharaty@lau.edu.lb

3. Novel Cache Replacement Algorithm: This research introduces a novel cache replacement algorithm specifically designed for mobile computing environments. Unlike traditional approaches that ignore network traffic patterns, the proposed algorithm leverages network traffic analysis to make informed decisions on data storage and eviction in the cache. By selectively caching highly accessed data items, the algorithm aims to improve cache hit rates and overall system performance.

A. *Cache Replacement Algorithms*

When the cache is complete, a data victim needs to be selected. Instead of randomly selecting a block and negatively affecting system performance, cache replacement mechanisms like LRU, FIFO, LIFO, LFU, MFU, and MRU are adopted. Least recently used (LRU) considers that data not recently used for a long time will not be used in the future. Hence, the most recently used data are stored in the front of the cache, and the least recently in the back end. When the cache is full, and new data comes, a new entry will be placed on the top, and back-end data will be removed. With First in First Out (FIFO), the oldest data is found in the front and are replaced when the cache is full. So, when a mobile user cache is full, the first data entry stored in the cache is erased to make room for a new cached data item.

On the contrary, Last In First Out (LIFO) replacement algorithm evicts the most recently added data disregarding how many times it was accessed before. Then, according to Least Frequently Used algorithm (LFU), the least frequently used data object is removed to make room for new cached data when the cache reaches its maximum capacity. The data entry with the fewest count gets shuffled to the tail of the list. The opposite of LFU is the MFU replacement algorithm, where the data entry with the most accesses get shuffled to the tail of the list. When the cache is full, the most frequently used data object is removed to make room for a new cached data object. In contrast to LRU, the Most Recently Used (MRU) throw-outs the most recently used data first. When a cache is full, the most recently used data item is removed to give its place to a new cached data object.

B. *Scalable Asynchronous Cache Consistency Scheme – SACCS*

1) *Scheme and Key Features*

Consistency is a significant challenge when dealing with caches. Several cache consistency maintenance algorithms were created to tackle this issue. First, we have the stateless approach whereby the server (or mobile support station MSS in mobile system) is unaware of the client's content (or mobile user cache MUC). This approach employs simple database management schemes; however, it needs more scalability and the ability to handle user mobility and frequent disconnections. In contrast is stateful, which is scalable for large database systems. The authors in [1] introduced a third approach combining stateless and stateful, known as the Scalable Asynchronous Cache Consistency Scheme (SACCS). “SACCS mainly uses flag bits at the server cache (SC) and MU cache (MUC), an identifier (ID) in MUC for each entry after its invalidation, and estimated time-to-live (TTL) for each cached entry, and the conversion of all valid MUC entries into an uncertain state upon MU wakeup” [1]. In each zone, multiple MU communicate wirelessly via a local mobile support station (MSS). Each MSS belongs to a wired network connecting it to the original server.

In SACCS, the mobile support station (MSS) needs to recognize the legitimate state of MUC data items, as opposed to the asynchronous stateful method, which requires the MSS to identify all data objects for each MUC. SACCS, on the other hand, has an advantage over the stateless approach in that the server does not need to broadcast IRs to all MUs regularly, lowering the quantity of IR messages delivered over the network.

SACCS has four characteristics that make it a high-performance, scalable, and low-complexity algorithm. At SC and MUC, flag bits are used, and an ID is assigned to each data entry in MUC once it has been invalidated. When a MU reconnects (or wakes up), all valid data entries in MUC become uncertain, and each cached data entry has a TTL (time-to-live).

2) *Consistency and Maintenance Efficiency*

In SACCS, the MUC consistency is maintained using a flag bit for each data input. The flag is set to 1 if a MU retrieves a data entry from MSS, indicating that this data entry is located in a specific MU. When MSS receives an update message for that data entry, it broadcasts an IR to all MUs, and the flag bit is reset to 0. When an IR for data is broadcast, the data in a connected MU is set to an ID-only state or invalidated. The MU does not receive any IR if it is disconnected and all the data items in its cache are set to an uncertain state. Thus, SACCS maintains cache consistency across MSS and MUs.

SACCS is also scalable and efficient. Each data object is associated flag bit that reduces the amount of IRs that must be broadcast. When data in MSS is updated, only the IR for the data with the flag bit set to 1 is broadcast. As a result, bandwidth usage is reduced.

This remainder of the paper is organized as follows: Section 2 presents related works. Section 3 presents the proposed algorithm. The experimental results are presented in section 4. And a conclusion is drawn in section 5.

2. Literature Review

Cache memory is an integral component of modern computing systems, bridging the gap between high-speed processors and slower main memory. Efficient cache management is essential to leverage the benefits of the cache hierarchy effectively. This literature review focuses on the latest advancements in cache management schemes and discusses their impact on system performance. Some of the replacement polices include:

- ARC (Adaptive Replacement Cache): The ARC algorithm combines the advantages of LRU (Least Recently Used) and LFU (Least Frequently Used) policies by adaptively adjusting the cache size based on workload behavior [2].
- CAR (Clock with Adaptive Replacement): CAR algorithm employs a clock-based replacement policy and dynamically adjusts the clock hand movement based on recent access patterns, thereby improving the cache hit rate [3].
- Prefetching Techniques: 3.1. StreamPrefetch: StreamPrefetch leverages data stream identification to accurately predict and prefetch data blocks, reducing cache misses and improving memory access latency [4].
- Markov Prefetcher: Markov Prefetcher utilizes Markov models to predict the next memory address to be accessed, enabling effective prefetching and reducing cache miss rates [5].
- Cache Partitioning: 4.1. CAT (Cache Allocation Technology): CAT enables cache partitioning at the hardware level, allowing different cache regions to be assigned to specific applications or threads, thereby providing better isolation and QoS guarantees [6].
- NUCA (Non-Uniform Cache Architecture): NUCA divides cache into multiple regions with varying access latencies, optimizing cache utilization and improving performance in multi-core processors [7].
- Adaptive Management Algorithms: 5.1. COSMIC (Contention-Oblivious Shared Memory Intelligent Controller): COSMIC dynamically adjusts the cache partitioning and replacement policies based on contention levels, enhancing fairness and performance in shared memory systems [8].
- PID-Controlled Cache Management: PID-Controlled Cache Management employs a proportional-integral-derivative (PID) controller to dynamically adjust cache allocation and replacement policies, achieving optimal cache utilization [9].
- *Delayed-Eviction* module, which will not execute eviction until the fetching is completed, ensuring that the accessed contents during the fetching period are not evicted. For the *Standard-Conflict* problem, we propose *Unified-Standard* module to measure the access probability of the requested content with a new value function as well as contents cached in the edge server. The module can ensure that the high-probability contents in access process will not be evicted in the following eviction process. The two modules are integrated together to form a two-layer caching framework named *Adele* [10].
- Least Partition Weight (LPW): LPW takes comprehensive consideration of different factors affecting system performance, such as partition size, computational cost, and reference count. The LPW algorithm was implemented in Spark and compared against the LRU as well as other state-of-the-art mechanisms and showed enhanced performance [11].

Other cache replacement strategies include:

A. Edge Caching

Implementing caching techniques at the edge of the network has been highly adopted lately because of its positive impact on system performance. Edge caching can improve the network's output, reduce energy consumption, improve the quality of service, and decrease system traffic. The authors in [12] highlights essential factors when implementing edge caching. In addition to deciding when, how, and what data items we need to cache, caching at the edge of the mobile network need to identify where to cache the content.

There are two main approaches to apply when thinking of edge caching. The first approach is caching data at the base stations, referred to as infrastructure caching [13] [14]. A different approach, named infrastructure-less caching, obliges the network to do caching on the user terminal. Either approach can improve system performance, and lower congestion and delay since prefetching of data is done at a lower level, reducing the load on the leading network. The first approach performs caching at the level of base stations; while the second method uses more than one type of caching. It can be device-to-device or push caching done with a single device. It is evident that by embracing edge caching, the load on the back-end link and network decreases, and communication is directly between base stations and the devices. According to [13], the main objectives of edge caching are "enhancing system throughput, cultivating energy efficiency, lessening backhaul load, increasing connectivity, and reducing service delay."

B. AI-Edge Computing and Caching

Computers and digitalized technologies, including the concept of the Internet of Things, have invaded our world and affected our daily lives. Physical and natural communication between people decreased and was substituted by virtual connections [15]. Lately, mobile computing has had a significant boost starting with the first generation's invention up to today's discoveries including the fourth and fifth generation in addition to LTE [16]. With the tremendous changes happening in IoT and mobile computing, adding to it the tremendous amount of data being processed and stored, it was crucial to introduce a new technology capable of maintaining good quality of service for the user. The suggested invention was heterogeneous IoT with edge computing. According to [17-18], both qualities of service and experience shall be improved with this approach. We are discharging the load from the back network and decreasing the weight from data warehouses. The main objective of heterogeneous IoT architecture is to be clear and able to distinguish between high-priority applications that are delicate to delay.

The architecture consists of heterogeneous IoT, edge cloud, and central cloud. The brain of this system is the cognitive engine found in the cloud. In addition, the cloud includes all the data and resources needed for communication. To release its load, an edge cloud was invented. Each edge cloud will have its router, base station, and a gateway that connects it to the core cloud called edge nodes. Caching is done intelligently at the level of edge. Heterogeneous IoT communicates with the edge cloud regarding data requests. The most important are the sensors for gathering and sending data like sensing sounds, acceleration, GPS, etc. [19].

C. Cache Management in 5G

When we say 5G or fifth-generation mobile networks, we automatically think of the high speed and massive amount of data being transmitted. Because of these two factors, 5G networks continually rely on caching and cache management policies to improve their performance. Because 5G networks include big data to be circulated, focusing on data when developing a design for such networks is a big failure. For example, this will increase network congestion if many users access or request the same data simultaneously. Hence, causing the system to jam. The authors in [20] proposed a request admission control (RAC) policy to efficiently control data to be cached, aiming to improve the system's capacity and benefit from maximum utilization of system resources. "Studies have demonstrated that by assuring the optimal use of limited resources, an efficient admission control mechanism can lower blocking or dropping probability as well as higher bandwidth utilization" [20].

The design of the proposed scheme is made up of units of base stations called "SBSs." On top of these, there are the macro-cell base stations or "MBSs." "SBSs" and "MBSs" are connected to the main network via a wired connection. The "SBSs" are responsible for increasing and enhancing the network's capability; whereas, the "MBSs" focus on controlling and handling the entire network using signals. "MBSs and SBSs use orthogonal spectrum bands to avoid inter-tier interference" [21]. When a data request is issued, the user is connected to a particular "SBS" holding a cached copy of the requested data. Otherwise, the user will be connected to "MB."

D. D2D Cluster-Caching for 5G Multimedia Mobile Networks

There are several types of data sent over the Internet. One of which is time-tolerant applications which are not affected by delays like sending an email. However, there is a type of data like buffering videos and images that cannot tolerate delays and are time-sensitive. These data are transmitted and exchanged in multimedia networks. The main obstacle with multimedia networks lies in controlling the delay time and time to transmit data from one network to another. In order to preserve the quality of service and experience for the user, a device-to-device technique was invented for a 5G network that is said to boost communication and preserve power and energy as well as minimizing the load on the back end network [22]. Therefore, multiple caching and mining policies have been developed to be paired with D2D links in mobile multimedia networks to address the previously stated issues of delay and traffic. "A heterogeneous statistical QoS-driven resource allocation approach using the D2D cluster-caching based system is applied to successfully address the aforementioned issues" [23].

The architecture is divided into multiple clusters. Each cluster includes several connected users through a cellular or D2D link. First, the D2D broadcasting link connects the clustering head to the base station. The head itself is connected to multiple end users via a direct link. These users are called D2D users. The head is responsible for forwarding messages in both ways. In addition to ensuring that the data is sent with the same speed to all D2D users within the same cluster. An advantage of D2D users is that they can build a connection with each other, not with the BS only. The second type of user is connected using a wireless link, which we call cellular users. A vital advantage of the D2D method is that all D2D users can access a downloaded file or data previously cached in a D2D user, whether in the same cluster or not [23].

E. Collaborative Hierarchial Caching

As the name implies, caching with the collaborative hierarchical approach is based on multiple layers, mainly two: inter and intra levels [24]. This method of caching is primarily beneficial in 5G wireless networks. A decision should be taken to select the layer to do the caching of data. Regarding data communications, three methods are used: wired or wireless communication and the device to device transmission.

Wired links are established between the cloud and the routers responsible for caching. These routers are located in the first layer of the wireless edge network. These routers transmit data to each other using a wireless network through wireless fidelity or a cellular connection. Using a wired link, the cacheable routers of layer one are connected to the base station in layer two, which is connected to an end user via device to device connection. In contrast, mobile users communicate and share data using wireless cellular connections. These users are located in the third layer of the architecture. By adopting the collaborative caching approach based on caching layer, system performance will be enhanced by increasing the number of cache hits and reducing the number of hops in the system.

F. 5G Content Caching

In a 5G network, the data is not found exclusively with the content provider. Instead, this data is forwarded to the caching node or the base station. Thus, the user of the data requester will not have to wait for the data to be requested and provided by the leading content supplier. A copy of the requested data will be stored in the node to respond immediately. This content caching strategy in 5G caching has many advantages in terms of decreasing response waiting time and reducing system energy and power usage significantly. Hence producing a higher quality of service and better customer experience, which in turn leads to maximum user satisfaction [25]. There are several pros to applying caching technology in 5G, according to [26]. "First, it can cut the need for backhaul connections, resulting in a higher deployment density for nodes, better spectrum use, and higher network throughput." Second, network congestion will decline by storing a copy of the highly accessed data in a node. Bearing in mind that the data response time will drop significantly as well since the reply to the request will be performed on the node level instead of going back to the primary data supplier [27].

UE is the user gateway from which the network data is requested. The Access and Mobility Management Function or AMF is in charge of the management of users' portability and access to the network. The AMF is a control plane function in the 5G core network that handles connection and management mobility tasks. The module responsible for the last final access point in the 5G network is Radio Access Network or RAN. A RAN connects individual devices to other parts of a network through radio connections. It is a major component of modern telecommunications with different generations of mobile networking evolving from 1G through 5G. "Stackelberg game application function (S-AF) interacts with the 5G network's core network, network information processing, and the deployment of the cache optimization approach," according to [26]. To govern and regulate end-user sessions, the SMF module is used. To manage the user interface, the User Plane Function or UPF model is utilized, which includes a function for this matter. The UPF represents the data plane evolution of a Control and User Plane Separation (CUPS) strategy, which is a fundamental component of the 3GPP 5G core network(5GC). The UPF plays the most critical role in the process of data transfer.

Finally, the model which the data provided is located is called the DN. The architecture of the system includes data providers and wireless network operators. The network operator will specify the cost of the base station. Then the data supplier is notified. Based on the network operator type and the base station cost produced, a slot or portion of the content provider is allocated for end users' use. Thus, the end user will simply download or have access to the data immediately from the base station.

G. Cooperative Caching

1) Cooperative Caching in Wireless Sensor Networks (WSN)

Wireless sensor networks are made up of several nodes. These nodes are capable of interchanging and coordinating data. This is referred to as cooperative caching. The critical advantage of cooperative caching is that it allows vertices to have a collaborative cache room, reducing the cost of the communication link. Based on [28], several preconditions should be present for cooperative cache management to be efficient. First, the CDA. A cache discovery algorithm is needed to specify means to fetch and allocate the asked-for data to the adjacent nodes. Another fundamental requisite is the CAC or what is known as Cache Admission Control Algorithm. This algorithm identifies which data will be used later and must be cached. Besides, CRA is crucial in cooperative caching. When the cache is full, a cache replacement algorithm is needed, and the decision should be taken to choose a candidate item to be replaced. Finally comes the cache consistency algorithm, which validates the correctness of data written in the cache. The validation process analyzes the value of time-to-live (TTL).

2) Caching in Cooperative Zones (CCZ)

Caching in Cooperative Zones (CCZ) is based on dividing the network into zones. The main advantage is that the cost in local zones is low since adjacent nodes located in the same local zone share the cached data. Nodes are called sensor nodes, and the network is called a wireless sensor network. When a request for a data item occurs, the receiver will check if data is found in local memory. If found and valid, i.e., the consistency algorithm resulted in a valid copy, the data requested is provided, and a local hit happens. If not valid, the node will ask for a validated copy from an adjacent neighbour in the local zone. If data is not found locally, then local miss happens, and we need to search for the territory of the local zone. If data is found in one level neighbouring node, it will be retrieved and sent back to the requester resulting in a zone hit. Otherwise, a zone miss occurs, and we need to check and search for the requested data in the zones along the path of the original node. If data was found in other zones, a remote hit occurs. If not found, it will be requested from the data centers of the original node, and a global hit will occur. In cases of remote and global hits, the cache admission and cache replacement algorithms are used to provide data requested by the user [28].

H. IoT and Caching

1) IoT

Internet of things – IoT is considered the second most extraordinary transformation in the world of technology and science after the invention of the Internet. IoT is a simulated and uncontrolled network of interconnected objects and devices that exchange data and information according to a conventional and recognized set of rules. According to references [29] [30], the primary task of the Internet of Things is to identify, trace, track, observe and control devices intellectually and wisely. "It now appears in various domains; Smart Home, Wearable, Retail, Smart Cities, Healthcare, Agriculture, Automotive/Transportation, Industrial Automation, and Energy Management" [31]. Likewise, it grants easy access and involvement of a range of devices including "home appliances, surveillance cameras, monitoring sensors, actuators, displays, vehicles, and so on," according to the study in reference [18]. IoT was referred to as AAA: anytime, anywhere, anything. However, there is discord among scientists regarding the definition of IoT, resulting in multiple interpretations of the matter on business type and individual preference [32]. One of the main reasons that made IoT booming in the field of data science is its application and uses in almost all types of fields and domains.

The authors in [33] [34] demonstrate a generic list of areas where IoT is significantly used, such as "Home automation, medical aids, mobile healthcare, elderly assistance, intelligent energy management, and smart grids, automotive, traffic management and many others" [34].

IoT is capable of location detecting, distributing, controlling from a distance, and communicating securely and efficiently in addition to ad hoc networking. Recall the significant impact of IoT when applied to mobile computing. The author in [29] says that IoT can help mobile networks with "tracking, fleet management, traffic information system, disaster detecting and much more".

Three layers build up the internet of things architecture [35]. The lowest layer is the level that deals with nodes and circuits and is responsible for the instilled and internal communications. This is known as the hardware layer. Then comes the middleware layer, which is responsible for managing, processing, and storing data. The upper layer, known as the presentation layer, is the high-end layer, i.e., it contains advanced high-end tools that can flexibly work with several interfaces from different applications and provide data in one unified, comprehensive form.

2) *IoT and Edge Computing*

Mobile computing is considered a solid foundation for the invention and creation of the Internet of Things. The reason behind that is that the earlier was well known for its applications and technologies in linking everything (objects, things, etc.) to the internet, as well as being ideally used in applications based on time. However, for non-time sensitive applications, SDN or Software Defined Network was suggested in [31]. Centralization of data is a crucial concept in mobile computing, which causes a significant drawback in terms of performance degradation because of limited battery life and computing power. The issue of data centralization is addressed by introducing a new concept in network computing, referred to as edge computing. Edge computing means forcing or pushing all the computing functions to the network edges instead of concentrating them in the center. This can significantly improve performance by minimizing latency and delay time needed to fetch data. Also, it will help increase the accessible band [21, 32]. This is precisely what Cisco created. A new concept of edge computing, called fog computing, was introduced to enhance and improve system performance [21, 36].

In contrast to mobile computing, Fog computing legitimizes programs and software of devices to run directly at the network edge. The devices on which these applications run must be in the IoT. Fog computing is powerful in terms of the number of connected devices it can handle. The applications will execute on the network edge as long as these devices are in the IoT. That is why fog computing pairs well with IoT scenarios [37].

3) *IoT-based NDN and Hybrid Cache Management*

Internet Protocol IP is considered limited in terms of data distribution when it comes to applications based on the Internet of Things. Hence, the Named Defined Network, known as NDN, was adopted. The NDN paradigm is identified as a key enabler of 6G network architectures aimed at improving content distribution. NDN is an information centric networking (ICN) architecture that promotes a communication model directly based on topology-independent content names, instead of internet protocol (IP) addresses.

Where to place caching program and what the adopted technique used for replacement when the cache is full are believed to be critical factors that are highly needed for IoT networks. To handle and tackle internet issues in terms of data delivery, the number of hop counts and percentage of hit ratio, and improving system performance in general, an innovative strategy was created to enhance performance in networks, specifically in IoT environments. A hybrid cache management technique is adopted in the Name Defined Networks. NDN networks consist of several cacheable nodes. These nodes are responsible for storing and fetching data each time a data request is explicitly sent in IoT schemes resulting in a high hit ratio due to fast identification of data accessibility. In addition, the NDN hybrid cache system helps reduce energy consumption and various benefits can be achieved via the IoT, including "quick data availability, reduced energy and power consumption, reduced network resource usage, reduced network congestion, and effective bandwidth utilization" [38].

4) *Caching in ICN*

a) *Information–Centric Network – ICN*

As the name implies, an information-centric network of ICN is based entirely on information and material of the customer regardless of the latter's location or type of requester. "This concept provides unique and location-independent content names, in-network caching, and name-based routing." With ICN retrieving and accessing data is easy because of caching, in addition to the significant advantage, which is distributing the load between the customer and producer. Because ICN does not depend on location, the traffic and congestion are reduced. An information-centric network is considered an ideal solution for IoT environments, particularly for named data networking (NDN) [31].

b) In-Network Caching Strategies

As stated above, the named defined networks are based on cacheable nodes. These nodes are part of the request and response paths when data is requested. Based on the adopted caching policy, each node decides whether it should store a copy of the requested data. The first type is the LCE method or as known as Leave Copy Everywhere [39 – 41]. Each cacheable node residing on the requesting path will save a copy of the requested data. Second is the Leave Copy Down (LCD) technique. With LCD, two nodes will save a copy of the requested data in contract to LCE. A copy is saved in the gateway and a second copy is on the returning path to the client. Then comes the edge-caching approach [42], which is the opposite of LCD. The cached data is saved on the final doorway node on the path back to the user. Another in-network caching strategy is the consumer-cache strategy [43] is an alternative to edge-caching. The cached data is saved on the first node on the return path to the client.

Next comes the between-ness centrality. This caching strategy is based on counts. Nodes with the highest number of counts will store a copy of cached data. If node x is part of connecting two other nodes in the network, the count for node x is incremented. The node with the highest count is responsible for storing a copy of data. This count is referred to as the "between-ness centrality metric" [44]. Finally, the last type of caching method in ICN is the ProbCache caching approach [44] which is based on probability. The probability is calculated based on the inversely proportional link of the distance between the user and the creator. The node with the highest probability is the node to store a copy of the cached data.

I. Big Data

The number of internet and technology users has increased dramatically in the past years due to the widespread of social networking platforms and wireless mobile applications. A vast amount of information is generated, processed, and stored every second. This is known as big data. Big data comes in different types and from different resources. Data can be structured, semi-structured, or unstructured; may come from databases, logs, social networks, documents, etc. Big data is empirical because it allows businesses to improve by making the right decisions by collecting data and studying and analyzing previous and current behaviors. Despite its advantages, big data has many drawbacks. Scalability is a major downside. While all applications, systems, and platforms are oriented towards expansion in data and information, and to solve the issue of robustness, it is required to become software-defined. This problem occurs in wireless mobile frameworks because of their limited architecture. In this context, fog computing was an ideal solution whereby edge devices and the cloud operate similarly [45 - 46]. A popular example where big data is highly famous and used is Hadoop; it proved to be a successful example of "big data management software offering software solutions in terms of cost, processing capabilities, analytic techniques compared to traditional tier-one database infrastructures" [45]. Hadoop adopted edge or fog computing. It pushes data toward users and forces the required applications to the edge, increasing customer satisfaction. Edge computing and big data work as follows. "A big data platform installed at the core site is in charge of monitoring and forecasting user demand, and cache-enabled BSs store the strategic contents that the big data platform has forecast."

The cache-enabled architecture includes a big data platform and cache-based station. So instead of accessing the core router, which will access the cloud and hence increase network traffic and congestion, leading to bottlenecks, the big data platform will include a database and cluster computing system that locates or caches data based on machine learning and analyses of users' behaviors.

This literature review has provided an overview of the latest cache management schemes, covering replacement policies, prefetching techniques, cache partitioning, and adaptive management algorithms. The discussed schemes have demonstrated improvements in cache performance, hit rates, and system throughput. However, further research is needed to explore their scalability, adaptability to diverse workloads, and integration with emerging architectures.

3. THE APPROACH

When the caching concept was first introduced, minimizing and decreasing network congestion traffic was the primary purpose. The system will respond immediately with a copy of the requested data if found or stored in the small memory known as a cache. Hence reducing delay from one side and decreasing circulation of requests in the network from the other, leading to less congestion and network traffic. Despite its advantages, caches have some significant drawbacks. Limitations in size led to the introduction of cache replacement algorithms that aimed to improve cache work in case the latter was full. Several algorithms were presented, and two or more replacement algorithms were combined to maximize the efficiency of the cache. Another area for improvement in the cache was consistency. To assure data in the cache are reliable, i.e., consistent with their original copies in main memory, numerous consistency schemes were adopted like stateful and stateless methodologies. An approach combining stateful and stateless methods called Scalable Asynchronous Cache Consistency Scheme (SACCS) was introduced to improve system performance. SACCS was combined with known replacement algorithms like FIFO and MFU to maximize system performance [1].

We currently live in a world that is shifting towards analyzing and processing data continuously. A massive amount of data is created every day. "Data is the new oil," a quote by Branka Vuleta reflecting today's global economy and our digital lifestyle. With all this being said, we can no longer stick to the traditional methods of caching and cache replacement algorithms. We need to maximize the work of a cache to improve system performance, which will lead to end-user satisfaction. Previously with SACCS and all the replacement algorithms, data was written to cache randomly. In our work, we will adopt a different writing method to cache. Updating the cache will be based on analyzing network traffic to identify winner data to be allocated in the cache. With the newly adopted approach, system performance improved significantly. This was experimented with by testing the cache hit ratio, cache miss ratio, and access delay. Experimental results will be represented in section 4 of this paper.

A cache hit occurs when the data requested by the user or application is already found in the cache. It provides a fast reply since data is retrieved by reading the cache memory. To evaluate cache performance, the cache hit ratio is calculated. It is equivalent to the number of cache hits divided by the total of cache hits and cache misses (or 1 - cache miss ratio). Similarly, as the name implies, a cache miss happens when the requested data is not found in the client's cache. The client only requests the data from the server (main memory). The client will retrieve the requested data from the server, keep a copy of it in its local cache for future requests, and then forward the data item to the client. The Miss ratio is calculated to evaluate the performance. It corresponds to the number of cache miss divided by the total of cache hit and cache miss (or 1 - cache hit ratio).

To improve performance, i.e., increase cache hit (decrease cache miss), the new approach focuses on analyzing network traffic and selectively writing data into the cache, unlike the old approach, which ignored network traffic and randomly added data into the cache.

Network traffic follows a log-based probability distribution based on a bell curve. In this distribution, around 70% of system traffic goes to 30% of the data. The remaining 30% of the traffic goes to 70% of the data. This means that data is categorized into two groups: highly-requested and not-highly-requested. Highly requested data comprise 70% of network traffic, and not-highly-requested data make up the remaining 30%. Simply this approach will check and identify data before adding it to the cache. If the data is highly requested and is not already in the cache, it will be written into the cache. However, if the data is rarely accessed and not highly requested, it will be ignored and not added to the cache. We are automatically increasing the hit rate by adopting this approach since we are filling up the cache with highly accessed and frequently requested data items. Furthermore, the red curve in figure 1 below highly describes the behavior of network traffic, similar to that of a log probability distribution. Zones 1 and 3 represent 30% of network traffic, including 70% of the data. These data are labelled as not-highly-requested data. Zone 2 represents 70% of network traffic and includes highly-requested data (30% of the data). We are greatly interested in these data items and will write to cache to increase the cache hit and improve system performance.

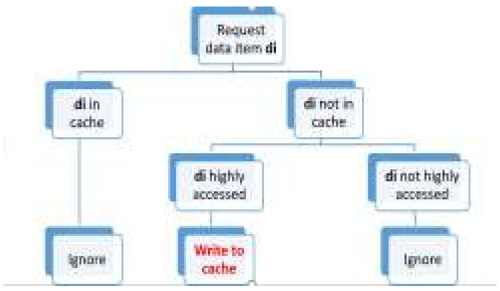

Figure 2. Red Curve illustrating the behaviour of network traffic

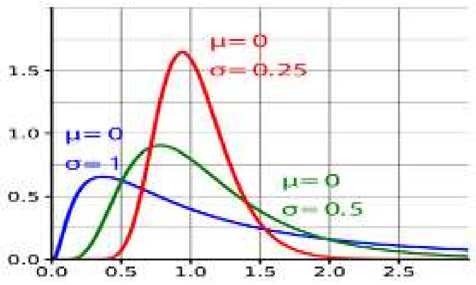

Figure 1. network traffic analysis flow

Based on the bells curve and the log-based probability distribution, data were classified as either highly accessed or not to identify whether to write it to cache or ignore it. The below flowchart of figure 2 illustrates the way the approach works.

When a data item is requested, the approach checks whether the requested data is highly accessed based on network traffic analysis. If so, data is written into the cache if not already in the cache. If data is not highly accessible, it will be ignored and not added to the cache. To illustrate, the suggested approach depends on five critical procedures illustrated in the tables below. Foremost, a table is created to store the frequency of access for each data point. Each time a request is sent to access a particular data point, the frequency table is updated to reflect the change. Ideally, the newly added data points will be located at the bottom because they have zero access. As more requests are received with time, the frequency of the data requested will be updated, and consequently, the data point will have its proper positioning. Accordingly, the decision to cache or not will be performed. The process of updating the frequency of the data point is shown in table 1 below.

TABLE I. UPDATE FREQUENCY PROCEDURE

	Data Point Frequency Update Method
	Begin
1	UpdateFrequency(data_point):
2	frequency = GetDataPointFrequency(data_point)
3	frequency += 1
4	SetDataPointFrequency(data_point, frequency)
	End

The current frequency value for the data item is identified by calling the GetDataPointFrequency(data_point) method. Then this value is incremented by one and updated using the SetDataPointFrequency(data_point, frequency) method. To keep data in the cache consistent and updated, the method shown in table 2 below should be called periodically. To update the cache table, we need to know the probability of the data item. This is done using the GetProbability(data_point) function. The procedure checks if the probability is less than 0.3 (i.e., a data item is not highly accessible according to the adopted approach and bell's curve). Hence, data should be removed from the cache.

TABLE II. UPDATE CACHE TABLE PROCEDURE

	Cache Table Update Method
	Begin
1	for data_point in data_points:
2	probability = GetProbability(data_point)
3	if probability < 0.3:
4	RemoveFromCache(data_point)
	End

The get-all frequencies method of table 3 is used to view and retrieve the frequencies of all data items. The frequency value is retrieved by using GetDataPointFrequency(data_point). Each data point has a count that is incremented when accessed. The method would return how many times a data point has been accessed. The method will return a zero value if the data item does not exist.

TABLE III. GET ALL FREQUENCIES PROCEDURE

	Get All Frequencies Method
	Begin
1	GetAllFrequencies:
2	frequencies = []
3	index = 0
4	for data_point in data_points:
5	frequency = GetDataPointFrequency(data_point)
6	frequencies[data_point] = frequency
7	return frequencies
	End

Upon the arrival of the request for a data item *di*, we are assuming we have a history of network traffic. We need to analyze the traffic of this data and see if it is highly accessed. This is done based on the bell curve, and since 70% of network traffic includes highly accessed data, we check if the requested data has a probability greater or equal to 0.7, i.e., it lies in the area of highly accessed data in the bell curve. This is decided based on the output of getting probability and should cache methods illustrated in tables 4 and 5, respectively. The get probability method will retrieve all the frequencies using the GetAllFrequencies() method and then sort them using the sort() method. If the data item is at the 30% bottom, i.e., it is not highly accessed and should not be cached, it will be identified by returning its position.

TABLE IV. GET PROBABILITY PROCEDURE

	Get Probability Procedure
	Begin
1	GetProbability(data_point):
2	frequency = GetDataPointFrequency(data_point)
3	all_frequencies = GetAllFrequencies()
4	sort(all_frequencies)
5	# data point is at the bottom 30%
6	return position(frequency, all_frequencies)
	End

The should cache method is responsible for the following. If the retrieved probability value for the get probability method is below 0.3 (i.e., the data point is located in the 30% bottom), the method points out that data should not be cached. If it is not the case (i.e., the data item is located in the top 70%, which means it should be cache), the system will check if there is space in the cache to add this item. If the cache is not full, the data item will be added. Otherwise, we need to check if we can replace the data to be cached with an existing cached item. We need to replace an existing cache item with a lower probability than the new one. Else, all cached data have a higher probability. Hence the data point will not be cached.

TABLE V. SHOULD CACHE PROCEDURE

	Should Cache Procedure
	Begin
1	ShouldCache(data_point):
2	probability = GetProbability(data_point)
3	if probability < 0.3:
4	return False
5	if CacheIsNotFull:
6	return True
7	if CacheIsFull:
8	for cached_point in cache:

	Should Cache Procedure
9	prob = GetProbability(cached_point)
10	# replace an existing cache item that has a lower probability with the new item
11	if probability > prob:
12	return True
13	# all cached data have a higher probability, so the data point will not be cached
14	return False
	End

Finally, if should cache method result was to write the data item into the cache, the cache method displayed in table 6 is responsible for this task. Item will be inserted to cache if cache capacity permits (i.e., the cache is not full). Otherwise (i.e., if the cache is full), we need to find a cached item with a probability lower than the new data item we need to add. In other words, the probability of the data item we need to write to the cache should be greater than the lowest cached item's probability.

TABLE VI. CACHE PROCEDURE

	Cache Procedure
	Begin
1	Cache(data_point):
2	if CacheIsNotFull:
3	InsertToCache(data_point)
4	else:
5	for cached_point in cache:
6	prob = GetProbability(cached_point)
7	# replace an existing cache item that has a lower probability with the new item
8	if probability > prob:
9	# replace cached_point with the new data_point
10	ReplaceInCache(cached_point, data_point)
11	return
	End

4. Experimental Results

For testing purposes, every experiment was performed ten times with a fixed number of requests (1000 requests) for the six algorithms. Four factors were used to evaluate and test the efficiency of the adopted approach: writing to cache based on network traffic analysis—the number of cache hits and misses, delay time, and percent of delay improvement. The experimental results are compared with the old traditional approach, writing randomly to cache. By definition, a cache hit is when the data requested for processing is located and found in cache memory; whereas cache miss is the reverse, i.e., data requested is not in cache memory. If the cache hit is (*x*), a cache miss is computed using (*1-x*) and vice versa. In case of a cache miss, the system must retrieve the requested data from the main memory. The time needed to do that is known as delay time. The Delay time in cache hit is zero. In our work, we use the terms delay and total delay. Delay time is the actual delay when utilizing a cache strategy which is writing to cache based on network traffic. However, the total delay time is when not using any caching strategy, i.e., writing to cache randomly. To evaluate the attained improvement of the new approach, the incremental improvement in the delay is calculated using: *Total improvement = 1 – (delay/total delay)*. The below figures illustrate the experimental results of the adopted approach of analyzing network traffic to write efficiently to cache. Figure 3 below shows that all algorithms performed better with the network traffic analysis approach with a maximum cache miss ratio of 69.6 (LIFO replacement algorithm). FIFO and MFU performed the best, with the lowest cache miss ratio of 59.6. Then comes LFU (60.52) and LRU (60.69); finally, MRU has a 67.37 cache miss ratio. In the previous approach, FIFO performed the best with a cache miss ratio (of 69.1).

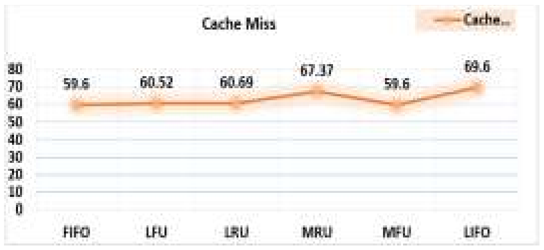

Figure 3. Cache Hit Ratio

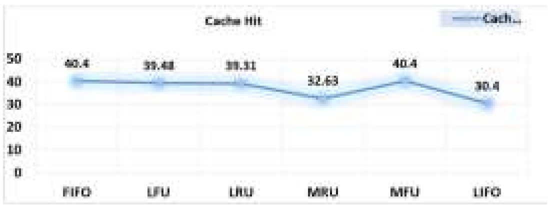

Figure 4. Cache Miss Ratio

In figure 5, a comparison between time delay and total time delay shows the efficiency of the adopted approach. This is reflected by dramatic improvement and a decrease in delay in all the replacement algorithms. As shown below, LRU has 14616.3 delay time with the new approach vs. 24262.5 with the old approach, resulting in an improvement of 39.75 %, as represented in figure 6. FIFO had the highest delay improvement percentage, 40.28%, with a delay of 14758.4 compared to 24712.9 with writing randomly to cache. MFU delay time improved by 40.21%, as shown in figure 6. Its delay time with writing to cache based on the network traffic approach improved to 14698.8 compared to 24586.3 with the old approach. LRU and LFU come next with 39.75% and 38.27% of improvement, respectively. In a nutshell, the delay time for all replacement algorithms improved significantly by a minimum of 30% (LIFO) and a maximum of 40.28% (FIFO).

As a result, the newly adopted approach of this study has experimentally proven its efficiency based on the positive outcomes of all testing factors. The cache hit ratio and cache miss ratio improved significantly, with all replacement algorithms bypassing the best results of the old approach (FIFO + SACCS). With the improved cache hit ratio and cache miss ratio, the delay time improved vividly.

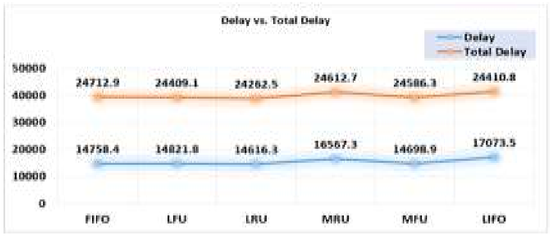

Figure 5. Delay vs. Total Delay in msec

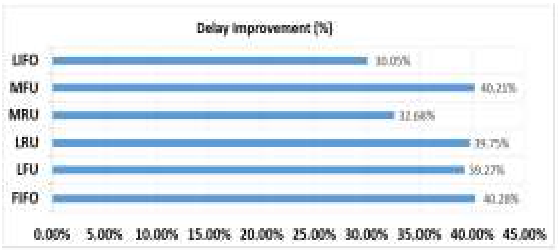

Figure 6. Percent in Delay Improvement

Analyzing traffic in a cache replacement strategy surpasses other cache replacement strategies due to its ability to leverage fine-grained information about data access patterns, resulting in improved cache performance. By analyzing traffic, the replacement strategy can effectively identify frequently accessed data blocks and prioritize their retention in the cache, while evicting less frequently accessed or cold data. This approach enables the cache to adapt dynamically to workload variations, reducing cache misses and improving overall cache hit rates. Unlike traditional replacement strategies that rely solely on limited information such as recency or frequency of data accesses, traffic-based analysis considers the temporal and spatial locality of data references, enabling more accurate predictions of future data accesses. Furthermore, analyzing traffic provides insights into the access patterns of different data types or application domains, allowing the cache replacement strategy to tailor its decisions to specific workloads, thus optimizing cache utilization and system performance. Overall, the incorporation of traffic analysis in a cache replacement strategy offers a more comprehensive and sophisticated approach that outperforms other strategies by exploiting detailed knowledge of data access patterns for efficient cache management.

CONCLUSION

Caching plays a pivotal role in enhancing data retrieval performance within mobile computing environments. Due to the wireless communication between the mobile device and the server, numerous challenges arise, including intermittent connections, limited uplink bandwidth, and constrained client resources. Caching at the mobile client alleviates these issues, albeit with certain limitations. Various approaches, such as stateful, stateless, and SACCS methodologies, have been proposed to maintain cache consistency. However, the cache size limitation remains a challenge. To address this limitation, replacement strategies have been suggested.

In this study, we introduce a novel cache replacement algorithm based on network traffic analysis and the categorization of data according to a bell curve's probability distribution. Our proposed approach focuses on analyzing network traffic and selectively storing data in the cache, departing from the previous approach that disregarded network traffic and randomly added data to the cache. The primary objective is to maximize the overall profit value of cached data items. Comparative evaluations with other strategies demonstrate superior performance of our algorithm.

Future work in cache management research will include comparative evaluations of our proposed cache replacement strategies with the latest cache replacement strategies available in the literature. This comparison aims to determine the relative performance and effectiveness of different cache replacement schemes.

By conducting a comprehensive analysis of the newest cache replacement strategies, we can gain insights into their strengths, limitations, and potential areas of improvement. This evaluation will involve benchmarking the different replacement policies against a wide range of workloads and assessing their cache hit rates, miss rates, and overall system performance. The comparative study will provide valuable information to determine which cache replacement strategies outperform others in specific scenarios. It will enable us to identify the most efficient and effective strategies that can be adopted in various computing environments, such as high-performance computing, cloud computing, and embedded systems.

Additionally, the comparative evaluation will help identify the impact of different cache replacement strategies on system-level metrics, such as energy consumption, resource utilization, and response time. This holistic analysis will enable researchers and practitioners to make informed decisions when selecting cache replacement strategies for specific applications or computing systems.

Furthermore, the comparison of cache replacement strategies will facilitate the identification of potential research directions and areas for further improvement. It may reveal opportunities to develop novel hybrid or adaptive cache replacement approaches that combine the strengths of multiple existing strategies to achieve even better performance.

REFERENCES

[1] Haraty, R. A., & Nahas, L. H. (2018). A Recommended Replacement Algorithm for the Scalable Asynchronous Cache Consistency Scheme. Proceedings of the 7th iCatse International conference on IT Convergence and Security, Seoul, Korea, 25-28 September 2017.

[2] Megiddo, N., & Modha, D. S. (2003). ARC: A Self-Tuning, Low Overhead Replacement Cache. In USENIX Annual Technical Conference (pp. 1-14).

[3] [3] Jiang, H., Zhang, X., & Li, H. (2012). CAR: Clock with Adaptive Replacement. IEEE Transactions on Computers, 61(1), 5-18.

[4] [4] Liu, J., & Chung, S. (2018). StreamPrefetch: Exploiting Data Stream Identification for Efficient Prefetching. ACM Transactions on Architecture and Code Optimization, 15(4), 1-25.

[5] [5] Gao, Y., & Lilja, D. (2011). Markov Prefetcher: An Accurate Next-Address Predictor for the Memory Hierarchy. IEEE Transactions on Computers, 60(9), 1285-1301.

[6] [6] Intel Corporation. (2015). Intel® Cache Allocation Technology (CAT): A new performance capability for the Intel® Xeon® Processor E5 Family. Retrieved from https://software.intel.com/content/www/us/en/develop/articles/intel-cache-allocation-technology.html

[7] [6] Hsieh, S., & Jouppi, N. P. (2006). Understanding the Interactions of Pthreads in Speculative Parallelization. In Proceedings of the 2nd Annual Workshop on the Interaction between Compilers and Computer Architectures (pp. 1-9).

[8] [8] Li, Q., Gao, M., & Zhang, Z. (2018). COSMIC: A Contention-Oblivious Shared Memory Intelligent Controller. In Proceedings of the 27th International Conference on Parallel Architectures and Compilation Techniques (pp. 131-143).

[9] [9] Yuan, K., & Niu, X. (2012). PID-Controlled Cache Management for Embedded Systems. Journal of Computers, 7(7), 1651-1660.

[10] [10] Pengmiao Li, Yuchao Zhang, Huahai Zhang, Wendong Wang, Ke Xu, Zhili Zhang, A delayed eviction caching replacement strategy with unified standard for edge servers, Computer Networks, Volume 230, 2023, 109794, ISSN 1389-1286, https://doi.org/10.1016/j.comnet.2023.109794.

[11] [11] Hui LI, Shuping JI, Hua ZHONG, Wei WANG, Lijie XU, Zhen TANG, Jun WEI, and Tao HUANG, LPW: an efficient data-aware cache replacement strategy for Apache Spark. Science China – Information Sciences, Vol. 66, 112104:1–112104:20, January 2023.

[12] [12] J. A. Stankovic, “Research directions for the Internet of Things,” IEEE Internet Things J., vol. 1, no. 1, pp. 3–9, Feb. 2014.

[13] [13] T. X. Tran, A. Hajisami, and D. Pompili, “Cooperative hierarchical caching in 5G cloud radio access networks,” IEEE Network Magazine, August 2016.

[14] [14] M. Zhang, H. Luo, and H. Zhang, “A Survey of Caching Mechanisms in Information-Centric Networking,” IEEE Commun. Surveys & Tutorials, vol. 17, no. 3, 2015, pp. 1473–99.

[15] [15] L. Zhou et al., “When Computation Hugs Intelligence: Content-Aware Data Processing for Industrial IoT,” IEEE Internet of Things J., vol. 5, no. 3, June 2018, pp. 1657–66.

[16] [16] D. Wu et al., “Optimal Content Sharing Mode Selection for Social-Aware D2D Communications,” IEEE Wireless Commun. Letters, vol. 7, no. 6, 2018, pp. 910–13.

[17] [17] Y. Hao, Y. Miao, L. Hu, M. S. Hossain, G. Muhammad, and S. U. Amin, "Smart-Edge-CoCaCo: AI-Enabled Smart Edge with Joint Computation, Caching, and Communication in Heterogeneous IoT," in IEEE Network, vol. 33, no. 2, pp. 58-64, March/April 2019, DOI: 10.1109/MNET.2019.1800235.

[18] [18] X. Wang, Y. Han, C. Wang, Q. Zhao, X. Chen and M. Chen, "In-Edge AI: Intelligentizing Mobile Edge Computing, Caching and Communication by Federated Learning," in IEEE Network, vol. 33, no. 5, pp. 156-165, Sept.-Oct. 2019, doi: 10.1109/MNET.2019.1800286.

[19] [19] Y. Hao, Y. Miao, L. Hu, M. S. Hossain, G. Muhammad, and S. U. Amin, "Smart-Edge-CoCaCo: AI-Enabled Smart Edge with Joint Computation, Caching, and Communication in Heterogeneous IoT," in IEEE Network, vol. 33, no. 2, pp. 58-64, March/April 2019, DOI: 10.1109/MNET.2019.1800235.

[20] [20] M. F. Ahmad and M. Arif Hossain, "Mobility Aware Cache Management in 5G Future Generation Wireless Communication System," 2019 1st International Conference on Advances in Science, Engineering and Robotics Technology (ICASERT), 2019, pp. 1-6, DOI: 10.1109/ICASERT.2019.8934710.

[21] [21] S. Zhang, N. Zhang, P. Yang, X. Shen, "Cost-effective cache deployment in mobile heterogeneous networks." IEEE Transactions on Vehicular Technology 66.12 (2017): 11264-11276.

[22] [22] Y. Shen, C. Jiang, T. Q. S. Quek, and Y. Ren, “Device-to-device-assisted communications in cellular networks: An energy efficient approach in downlink video sharing scenario,” IEEE Transactions on Wireless Communications, vol. 15, no. 2, pp. 1575–1587, Feb. 2016.

[23] [23] X. Zhang and J. Wang, "Heterogeneous Statistical QoS-Driven Resource Allocation for D2D Cluster-Caching Based 5G Multimedia Mobile Wireless Networks," 2018 IEEE International Conference on Communications (ICC), 2018, pp. 1-6, DOI: 10.1109/ICC.2018.8422701.

[24] [24] X. Zhang and Q. Zhu, "Collaborative Hierarchical Caching over 5G Edge Computing Mobile Wireless Networks," 2018 IEEE International Conference on Communications (ICC), 2018, pp. 1-6, DOI: 10.1109/ICC.2018.8422371.

[25] [25] R. Torre, I. Leyva-Mayorga, S. Pandi, H. Salah, G. T. Nguyen, and F. H. P. Fitzek, “Implementation of network-coded cooperation for energy efficient content distribution in 5G mobile small cells,” IEEE Access, vol. 8, pp. 185964–185980, 2020.

[26] [26] Yanfang Zha, "Key Technologies of Cache and Computing in 5G Mobile Communication Network", Wireless Communications and Mobile Computing, vol. 2021, Article ID 3099272, 11 pages, 2021. https://doi.org/10.1155/2021/3099272

[27] [27] A. Gupta and R. K. Jha, “A survey of 5G network: architecture and emerging technologies, ” IEEE Access, vol. 3, pp. 1206 – 1232, 2015

[28] [28] Charan, Piyush & Usmani, Tahsin & Paulus, Rajeev & Saeed, Syed. (2018). A Cooperative Cache Management Scheme for IEEE802.15.4 based Wireless Sensor Networks. International Journal of Electrical and Computer Engineering (IJECE). 8. 1701. 10.11591/ijece.v8i3. pp1701-1710.

[29] [29] S. Chen, H. Xu, D. Liu, B. Hu, and H. Wang, "A Vision of IoT: Applications, Challenges, and Opportunities With China Perspective," in IEEE Internet of Things Journal, vol. 1, no. 4, pp. 349-359, Aug. 2014, DOI: 10.1109/JIOT.2014.2337336.

[30] [30] L. Atzori, A. Iera, and G. Morabito, ‘‘The Internet of Things: A survey,’’ Comput. Netw., vol. 54, no. 15, pp. 2787–2805, Oct. 2010.

[31] [31] Maroua Meddeb, Amine Dhraief, Abdelfettah Belghith, Thierry Monteil, Khalil Drira. How to cache in ICN-based IoT environments? ACS/IEEE International Conference on Computer Systems and Applications (AICCSA 2017), Oct 2017, Hammamet, Tunisia. HAL-01575386

[32] [32] Gupta D, Rani S, Ahmed SH, Verma S, Ijaz MF, Shafi J. Edge Caching Based on Collaborative Filtering for Heterogeneous ICN-IoT Applications. Sensors (Basel). 2021 Aug 15;21(16):5491. DOI: 10.3390/s21165491. PMID: 34450933; PMCID: PMC8402262.

[33] [33] K. Shafique, B. A. Khawaja, F. Sabir, S. Qazi, and M. Mustaqim, "Internet of Things (IoT) for Next-Generation Smart Systems: A Review of Current Challenges, Future Trends and Prospects for Emerging 5G-IoT Scenarios," in IEEE Access, vol. 8, pp. 23022-23040, 2020, DOI: 10.1109/ACCESS.2020.2970118.

[34] [34] A. Zanella, N. Bui, A. Castellani, L. Vangelista and M. Zorzi, "Internet of Things for Smart Cities," in IEEE Internet of Things Journal, vol. 1, no. 1, pp. 22-32, Feb. 2014, DOI: 10.1109/JIOT.2014.2306328.

[35] [35] Salman, O.; Elhajj, I.; Kayssi, A.; Chehab, A. Edge computing enabling the Internet of Things. In Proceedings of the 2015 IEEE 2nd World Forum on Internet of Things (WF-IoT), Milan, Italy, 14–16 December 2015; pp. 603–608.

[36] [36] F. Bonomi, R. Milito, J. Zhu, S. Addepalli, "Fog computing and its role in the internet of things," MCC workshop on Mobile cloud computing, August 2012.

[37] [37] Lee, K.; Kim, D.; Ha, D.; Rajput, U.; Oh, H. On security and privacy issues of fog computing supported Internet of Things environment. In Proceedings of the 2015 6th International

Conference on the Network of the Future (NOF), Montreal, QC, Canada, 30 September–2 October 2015; pp. 1–3.

[38] [38] M. A. Naeem, T. N. Nguyen, R. Ali, K. Cengiz, Y. Meng, and T. Khurshaid, "Hybrid Cache Management in IoT-Based Named Data Networking," in IEEE Internet of Things Journal, vol. 9, no. 10, pp. 7140-7150, May 15, 2022, DOI: 10.1109/JIOT.2021.3075317.

[39] [39] B. Alahmri, S. Al-Ahmadi and A. Belghith, "Efficient Pooling and Collaborative Cache Management for NDN/IoT Networks," in IEEE Access, vol. 9, pp. 43228-43240, 2021, DOI: 10.1109/ACCESS.2021.3066133.

[40] [40] Boubakr Nour, Kashif Sharif, Fan Li, Hassine Moungla, Ahmed E. Kamal, et al. NCP: a near ICN cache placement scheme for IoT-based traffic class. GLOBECOM 2018: IEEE Global Communications Conference, Dec 2018, Dubai, United Arab Emirates. pp.1 - 6, ⟨10.1109/GLOCOM.2018.8647629⟩. ⟨hal-02049991⟩

[41] [41] L. Saino, I. Psaras, and G. Pavlou, "Icarus: a caching simulator for information-centric networking (ICN)," in Proceedings of the 7th International ICST Conference on Simulation Tools and Techniques, ser. SIMUTOOLS'14. ICST, Brussels, Belgium, Belgium: ICST, 2014.

[42] [42] S. K. Fayazbakhsh, Y. Lin, A. Tootoonchian, A. Ghodsi, T. Koponen, B. Maggs, K. Ng, V. Sekar, and S. Shenker, “Less pain, most of the gain: Incrementally deployable ICN," SIGCOMM Comput. Commun. Rev., vol. 43, no. 4, pp. 147–158, Aug. 2013.

[43] [43] M. Meddeb, A. Dhraief, A. Belghith, T. Monteil, and K. Drira, “Cache coherence in machine-to-machine information centric networks,” in Local Computer Networks (LCN), 2015 IEEE 40th Conference on, Oct 2015, pp. 430–433.

[44] [44] W. K. Chai, D. He, I. Psaras, and G. Pavlou, “Cache” less for more” in information-centric networks,” in Proceedings of the 11th International IFIP TC 6 Conference on Networking - Volume Part I, ser. IFIP’12. Springer-Verlag, 2012, pp. 27–40.

[45] [45] Zeydan, Engin & Bastug, Ejder & Bennis, Mehdi & Kader, Manhal & Karatepe, Alper & Er, Ahmet & Debbah, mérouane. (2016). Big Data Caching for Networking: Moving from Cloud to Edge. IEEE Communications Magazine. 54. 10.1109/MCOM.2016.7565185.

[46] [46] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, “Fog computing and its role in the internet of things,” in Proceedings of the first edition of the MCC workshop on Mobile cloud computing, Helsinki, Finland, August 2012.

Ramzi A. Haraty is an associate professor of Computer Science in the Department of Computer Science and Mathematics at the Lebanese American University in Beirut, Lebanon. He is a program evaluator (PEV) for CSAB/ABET. He also serves as the Secretary of the Syndicate of Computer Sciences in Lebanon. He received his B.S. and M.S. degrees in Computer Science from Minnesota State University - Mankato, Minnesota, and his Ph.D. in Computer Science from North Dakota State University - Fargo, North Dakota. His research interests include database management systems, artificial intelligence, and multilevel secure systems engineering. He has well over 110 books, book chapters, and journal and conference paper publications. He supervised over 110 dissertations, theses and capstone projects. He is a member of the Association of Computing Machinery (ACM), ACM SIGAPP, Institute of Electronics, Information and Communication Engineers, the International Society for Computers and Their Applications, and the Syndicate of Computer Sciences in Lebanon.

Nancy Rahal is currently serving as an institutional analyst within the office of Institutional Effectiveness and Decision Support at the American University of Beirut. Previously, she worked as a Data Analyst in the Enrollment Management Unit at the Lebanese American University. Nancy holds a Bachelor of Science and Master of Science degrees in Computer Science, with a minor in Mathematics, from the Lebanese American University of Beirut – Lebanon. Her research focuses on various areas including database management, caching, analytics, BI tools and data visualization.