CDCA: Transparent Cache Architecture to Improve Content Delivery by Internet Service Providers

Alwi M Bamhdi
Umm Al-Qura University, College of Computing, Al-Qunfudah, KSA

***Abstract*—The popularity of on-demand multimedia such as video streaming services has been rapidly increasing the overall Internet traffic volume in the world. As of the beginning of 2023, almost 82% of this global Internet traffic came from video transmission through on-demand online services, trending towards changing the Internet paradigm from location-based to content-based, culminating in a new paradigm of Information-Centric Networking (ICN). ICN focuses on content distribution based on name rather than location, allowing Internet Service Providers (ISP) to implement local content caching systems for faster delivery and reduced transmission delays and unnoticeable jitter or distortions. ICN can be implemented over a Software-Defined Networking (SDN) infrastructure. SDN enables flexible programming and implementation of forwarding packet rules within a network domain seamlessly. This paper proposes a hybrid architecture that combines ICN and SDN to create a transparent in-network caching system for content distribution over the traditional IP network. The architecture aims to improve the performance of Video-on-Demand (VoD) services for customers while efficiently utilizing network provider resources. A prototype called CDCA was developed and evaluated in a Mininet emulation environment. The results of the evaluation demonstrate that the CDCA hybrid architecture to create a caching system for content distribution enhances VoD service performance and optimizes network resource utilization.**

Keywords*—*Content caching; content delivery network; content search algorithm; information-centric networking; multimedia; network function virtualization; software defined networks

I. Introduction

The integration of the Web, ICN, SDN and NFV, juxtaposed the *archaic* Internet. By combining these technologies, there is the potential to enable seamless placement and retrieval of multimedia content by multiple users. The integration of these technologies allows internet network service providers to leverage local caching mechanisms to deliver content to multiple users simultaneously. This unique CDCA approach of a system represents a novel and innovative way of optimizing content distribution and improving the overall user experience. The specific details and benefits of this novel approach are further elucidated in this paper.

A. Summary of the CDCA Solution

This paper presents a content delivery-based caching framework architecture system called CDCA that typically resides inside network providers' premises, i.e., Internet Service Providers (ISP), using SDN, which is completely transparent for users, content provider applications and network providers. Transparency is applied to all actors: the user, the content provider and the ISP network, which neither needs to modify any application and network equipment. In essence, the SDN/NFV module within the CDCA architecture permits forwarding data packets to the content cache or to the content source server transparently.

The key features and advantages of the CDCA architecture provide for in-network caching. The architecture is applicable to any application communication protocol that follows the client-server model and identifies content using unique logical names, such as HTTP URLs. The combination of the SDN control plane, Proxy, and Cache components enables the orchestration of a distributed caching framework. CDCA ensures that content is transparently delivered as close to the user as possible without any modifications to communication protocols or user clients.

CDCA is designed with horizontal scalability and high manageability in mind, adopting the principles of the microservice architectural pattern style that structures an application as a collection of services. This design approach provides elevated deployment flexibility and high levels of availability, regardless of the network's operational state. Content management within the architecture is driven by rule-based policies, facilitated by distributed decision-making mechanisms and multiple caches distributed throughout the network. These components interact with the SDN centralized control plane to ensure efficient content delivery. Although this topic is relevant and interesting and outside the scope of this paper, it will be addressed in-depth in future research works.

CDCA is designed to be deployed and supported by the ISP premises on the client side. In CDN architecture, the service must be contracted and paid for by the content provider. And also, if the CDN server is not located within the ISP network, it will be not worth it, since it will have to pay for the high bandwidth requested by multiple users' content downloads.

B. Rest of the Paper

The CDCA architecture is distinct from a traditional Content Delivery Network (CDN) infrastructure. They are not comparable. The unique aspects and differences of the CDCA solution are elaborated in Section IV.

The rest of the paper is structured as follows. Section II covers the background to the concepts and issues related to the topic of this paper. Section III presents the literature review of related works. Section IV presents the CDCA

architecture and in Section V the evaluation and results from the experiments are presented. Section VI discusses some important open issues and challenges to consider for future research related to the CDCA architecture and its content delivery operations, and finally, Section VII concludes the paper.

II. BACKGROUND

It is a remarkable transformation of the Internet from its humble beginnings as a point-to-point communication network for users to becoming a ubiquitous and essential infrastructure for global everyday communication. Over its nearly 54-year history, commencing as ARPANET, the Internet has evolved and gained popularity, becoming a highly successful and effective communication system that continues to rapidly advance. With the increasing volume of traffic and the complexity of modern services such as file sharing, VoIP, social networking, e-commerce, online gaming, and multimedia streaming, the inadequacies of the Internet's original design have become apparent. The architecture of the Internet has undergone numerous amendments to accommodate these evolving needs, resulting in a progressively more complex system at each stage of its development. This growing complexity poses challenges in terms of implementation, maintenance, and management of new networking services and applications. The costs associated with these tasks continue to rise as the Internet becomes more sophisticated and intricate. However, innovation in information communication and internetworking philosophies helps to mitigate the phenomenon of ossification [1], whilst acknowledging the challenges posed by the limitations of the original design and recognizing the ongoing efforts to innovate and adapt to meet the demands of modern communication and networking needs.

The evolving nature of Internet applications, particularly the significant growth in video streaming services, including both live broadcasts and on-demand content is increasing exponentially. According to Cisco, it has seen a massive buildup of Internet traffic in the order of 4.8 zettabytes during 2022, which is over three times the 2017 rate, led by a combination of increased use of cloud computing, IoT device traffic, video viewing, and the sheer number of new users coming onboard every day. The video traffic constituted 82% of the total Internet traffic in 2021, a substantial increase from over 70% in 2017, excluding video exchanged through Peer-to-Peer (P2P) file sharing [2]. The forecast indicates that Internet video traffic will grow at a rate of more than 31% per year, while online gaming, which is part of the audio/video mix, is expected to grow at a rate of over 50% plus per year until 2025. Currently, most content distribution platforms handle content requests individually, resulting in a unicast delivery paradigm where each user receives content from a Content Delivery Network (CDN) infrastructure separately [43]. However, this approach overlooks the fact that much of the content requested by users is identical to content requested by others just moments ago. As a result, a substantial amount of redundant content is delivered repeatedly over the same network segment, leading to unnecessary strain on service providers' transmission capacity and bandwidth. This inefficiency becomes more significant as the number of users and unicast content continues to grow. Consequently, new approaches are needed to improve and optimize the efficiency of content distribution.

The ongoing modifications and improvements to the Internet's architecture to accommodate new applications demand faster infrastructure and versatile middleware. However, the modern web still faces many challenges due to incompatibilities inherited from the original design of the Internet. The shift in user behaviour is highlighted, with Internet users now seeking specific content *("what"*) rather than focusing on the location of that content *("where"*). This shift necessitates more than simple unicast communication for modern web applications [44]. As a result, network architectures need to be smarter and more flexible to support the exponential growth resulting from the dynamic nature of multimedia content availability and online delivery. This trend is expected to persist in the foreseeable future. To emphasize the magnitude of this growth, a comparison between the content data produced in 2008 amounted to 500 exabytes compared to the present-day zettabyte scale [3]. This example illustrates the substantial increase in content generation and consumption over the years. The need for smarter and more flexible network architectures to support the dynamic nature of content availability and delivery on the modern web is a must with the significant growth in content data generation and consumption.

In recent years, some proposals tried to change the current end-to-end IP packet networking and web search engines to innovate enhanced content-based network architecture, called Information-Centric Networking (ICN) [4]. ICN is based on the principle that the Internet should prioritize the data needed by users rather than focusing on the physical location from which the data can be retrieved. In contrast, the current Internet architecture is host-based and was initially designed to facilitate communication between a limited number of fixed computers and geographically dispersed users. Although ICN offers significant advantages, implementing it as a replacement for the existing Internet backbone would require a radical and impractical overhaul. Consequently, various research efforts have focused on adapting ICN architectures to operate within the constraints of the legacy Internet infrastructure. One approach involves integrating ICN with Software-Defined Networking (SDN) and the OpenFlow protocol, allowing for the implementation of ICN concepts while leveraging the programmability and flexibility of SDN to make it compatible with the current Internet backbone [5]–[8].

The concept of a *"programmable network"* originated as a result of the SDN principles and architecture. The concept of a programmable network was initially driven by the introduction of OpenFlow, an open protocol that enables the configuration of packet forwarding tables in switches. With OpenFlow, network users can actively modify these tables, providing a level of control over the network's behaviour [9]. By utilizing OpenFlow and similar technologies, a programmable network introduces an abstraction layer for switches, allowing the separation of the data and control planes. In this context, the switch functions as a hardware fabric that primarily focuses on transparently forwarding data.

The policy management for handling data content is then handled by software through the control plane functions and mechanisms. This decoupling of data and control planes enables greater flexibility and programmability in network management and configuration.

Over the last few years, Network Functions Virtualization (NFV) [10] has shown the most promising results in the development of advanced computer networking. NFV offers a new approach to developing network services by utilizing programmable software and virtualization means. It replaces traditional proprietary hardware network elements and appliances that perform various network functions, such as Network Address Translation (NAT), Intrusion Detection and Prevention System (IDPS), caching, and more. With NFV, these network functions are implemented as Virtualized Network Functions (VNFs) using software and deployed within Virtual Machines (VMs). This approach enables more flexible and efficient networking and network service deployment. By utilizing NFV, network services can be customized according to specific business needs, allowing for greater agility in serving the ever-changing demands of service providers and end-users. Additionally, NFV brings significant cost savings by eliminating the reliance on expensive proprietary hardware and enabling more efficient resource utilization.

III. Literature Review

Ooka et al. propose the OpenFlow-CCN, a system architecture joining Content-Centric Network (CCN) and OpenFlow mechanisms to achieve content end-to-end forwarding [6]. In their proposal, the content names are mapped to hierarchical structure hash values and the long prefix matching. In an OpenFlow network, the content packet is forwarded by a unique IP address based on the content name hash value. The architecture was evaluated on the Trema Controller in an OpenFlow network. Their proposal is quite interesting because it does not impose any modification to either the OpenFlow protocol or CCN. CDCA does not require a CCN infrastructure because it uses a traditional IP network.

Nguyen et al. propose an improvement in CCN caching strategy that uses SDN [7]. They implement a wrapper between CCNx software and OpenFlow switch to decode and hash the content name in CCN messages into parameters that an OpenFlow switch can forward, e.g., IP address or port number. They argue that the large naming space offered by these fields restricts the collision probability between two different contentnames. The evaluation shows that the wrapper does not affect forwarding performance but might have name-space problems. CDCA does not require a CCN infrastructure.

Chandra et al., proposed a caching architecture specifically for HTTP on an SDN infrastructure [11]. They concluded that while the OpenFlow protocol maintains an abstraction of control and forwarding planes, it presents challenges when dealing with ICN because it lacks content abstractions. In response to this limitation, Chandra et al. proposed an architecture that utilizes a unique Proxy and multiple caches distributed across the OpenFlow network. In comparison to the CDCA solution provided in this paper, it employs the concept of deploying a Proxy to determine whether content should be fetched from the Cache or the server and it introduces a distributed Proxy and Cache architecture to enhance scalability and resilience. This means that the CDCA Cache architecture is designed to accommodate scalability and provide better fault tolerance and resilience.

Georgopoulos et al. presented OpenCache, an in-network caching system designed specifically for Video-on-Demand (VoD) applications using OpenFlow technology [12]. OpenCache consists of two main components: the OpenCache Node (OCN) and the OpenCache Controller (OCC). The OCC is responsible for determining which videos should be cached, while the OCN handles the storage necessary for video caching. The experiments conducted on OpenCache have demonstrated positive results in terms of video start-up delay, external link usage, and video quality. However, it is noted that these experiments were conducted with a single video client, which may not fully represent the behaviour and performance of OpenCache in real-world production networks with multiple user accesses.

The CDCA concept and solution offered in this paper of finding the Cache that is nearest to the user overcomes the deficiencies from other comparative approaches in so far as making Cache decisions based on user requests in an *on-demand* fashion, rather than relying solely on operator decisions.

IV. CDCA: Content Delivery Cache Architecture

The CDCA architecture aligns with the SDN principle of having a centralized control plane, which is responsible for controlling and directing the forwarding of data packets within the network. In this architecture, the centralized control plane is designed to handle cacheable requests that adhere to the client-server model, similar to the distributed content Cache architecture embedded inside the provider datacenter [13], but going one step further by deploying the distributed content Cache system inside the ISP infrastructure closest to the end-user, offering better response time and server load-balancing. By leveraging the capabilities of the centralized control plane, the architecture enables efficient management of cacheable requests. The control plane can make decisions regarding content caching based on various factors, such as user demand, network conditions, or predefined rules. These decisions are then communicated to the versatile device responsible for managing the caching process. Overall, the architecture combines the benefits of SDN's centralized control plane with the ability to handle cacheable requests in a client-server model. This integration allows for effective management and optimization of caching operations within the network.

The CDCA operational framework architecture is shown in Fig. 1. It is based on two fundamental components: the Proxy and the Cache. The SDN Controller identifies a potential content flow, such as an HTTP request to a VoD provider via the destination IP address and TCP port information. Once identified, the content flow is redirected to the Proxy component. Within the Proxy, the traffic of interest (content) is mapped to the content re-director module

responsible for forwarding the content request to an appropriate Cache instance where the content is stored. The Cache instance can be located within the same network or distributed across multiple locations. By redirecting the content request to the Cache instance, the CDCA aims to retrieve the content from the Cache instead of fetching it from the original source, thereby reducing latency and network traffic. This mechanism improves the overall efficiency of content delivery by serving frequently requested content from nearby caches. This approach enables faster and more efficient content delivery by leveraging caching capabilities within the network.

Note that the Proxy searches in the DHT to find if the content is cached, then, forward the requests to the Cache or to the Content Server. Fig. 1 illustrates the Cache-missed example shown on the left side, and the successful Cache hit case shown on the right side.

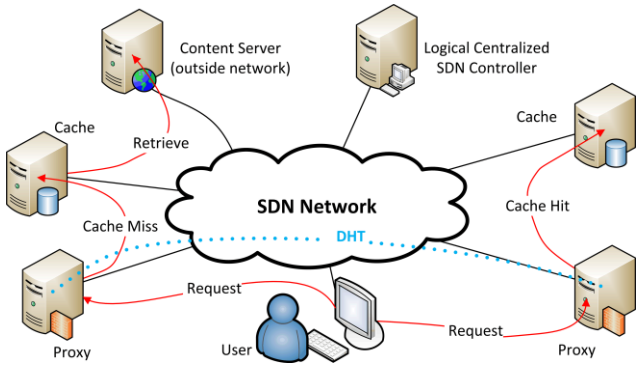

Fig. 1. CDCA system operational framework architecture.

The CDCA architecture implements an on-demand caching scheme for video-on-demand (VoD) services. In this architecture, larger content files can lead to longer transfer delays and higher processing overhead for caching operations, such as storing an entire file in the Cache and delivering it to the user while small files do not need chunking. To overcome this issue, large content files are divided into smaller parts called content chunks. Each chunk is handled independently, allowing for more efficient caching and delivery. This chunk-based delivery approach is specifically designed to handle the delivery of content in smaller segments rather than delivering the entire file at once. All VoD providers analyzed in this work used chunk-based delivery. This indicates that dividing content into smaller chunks is a common practice in the industry, likely due to the advantages it offers in terms of caching, delivery efficiency, and user experience. By adopting this chunk-based delivery approach and caching content chunks on-demand, the CDCA architecture aims to optimize the delivery of VoD content while minimizing transfer delays and processing overhead associated with caching operations.

Chunk-based caching [45] offers several advantages over file-based content caching. By dividing a content file into smaller chunks, it becomes possible to deliver different chunks from multiple caches, improving the efficiency of content delivery. One of the key benefits of chunk-based caching is increased storage efficiency. Instead of storing and replacing entire content files, the caching system can focus on individual chunks. This fine-grained approach allows for precise caching decisions, reducing the amount of storage required and improving overall Cache utilization. But the distribution of chunks within the network becomes a crucial decision in chunk-based Cache delivery since the placement of chunks across caches can significantly impact the performance and effectiveness of the caching system. Optimizing the distribution of chunks involves considering factors such as Cache proximity to users, network congestion, and popularity of specific content chunks. Efficient chunk distribution strategies typically enhance the caching system's ability to serve content quickly and reduce network traffic. Techniques such as content popularity analysis, adaptive caching algorithms, and dynamic chunk placement are employed to ensure effective distribution and retrieval of content chunks from caches. Furthermore, chunk-based caching has many advantages over file-based content caching. Different chunks of the same content can be delivered from multiple caches. Replacing some chunks instead of a whole content file may increase storage inefficiency. However, careful consideration must be given to the distribution of chunks within the network to optimize performance and achieve efficient content delivery.

In the CDCA architecture, the selection of caches is determined based on either the shortest path to the content or a set of custom rule-based policies. The forwarding decision is made by the Proxy that may receive information from the network management system and server state. Based on this information, the Proxy chooses the best Cache instance. For instance, when a content request is received by the Proxy, it checks if the requested content is already cached in a Cache instance. If a Cache hit occurs, meaning that the content is present in the Cache. This allows for fast and efficient content delivery without the need to retrieve the content from an external source. However, if the requested content is not present in any Cache, resulting in a missed Cache, the Proxy will then forward the request to the nearest Cache. The closest Cache is determined based on factors such as network proximity or predefined routing policies. This Cache, in turn, will request the content from the external Content Server, retrieve and Cache the content locally, and finally serve the user's request. This mechanism ensures that frequently requested content is cached closer to the users, reducing the need for content retrieval from remote servers and improving the overall response time and user experience. By leveraging caching and intelligent Cache selection, the architecture minimizes the latency associated with content delivery, optimizing the use of network resources and enhancing the efficiency of the system. In summary, the Proxy in the CDCA architecture determines Cache hits and misses for content requests. Cache hits allow for direct content delivery, while Cache misses triggering the retrieval of content from the closest Cache or the external Content Server, enabling efficient content caching and delivery to users.

In the CDCA architectural operating scheme, it is possible for multiple instances of the Proxy to be deployed across the network infrastructure “world”, where each instance is responsible for coordinating a “Cache island”, a group of Cache servers under same management domain. This

distributed deployment allows for efficient content delivery and load balancing. The SDN Controller plays a crucial role in the architecture by receiving network state information from the network management system. Based on this information, the SDN Controller makes forward decisions, determining the best Proxy instance to handle a content request. The goal is to minimize congestion, optimize transfer times, and enhance throughput by selecting the closest Proxy to the requesting user. Each Proxy instance is part of a Distributed Hash Table (DHT) in the overlay network. The DHT network facilitates the storage and retrieval of content-Cache instance mappings throughout the entire network. This distributed approach ensures that the mapping information is accessible and maintained across the network, enabling efficient Cache selection and content delivery. The Cache component, associated with each Proxy instance, is responsible for storing and serving cacheable objects triggered by any user's request within the network. When a cacheable object is requested, the Cache component determines if the object is already stored in the Cache. If it is present, the content is served directly from the Cache, minimizing latency. If the object is not in the Cache, the Cache component retrieves it from the appropriate source (e.g., external Content Server) and stores it in the Cache for future requests. By deploying this strategy of multiple Proxy instances, leveraging SDN-based forward decision-making, and utilizing a DHT overlay network, the CDCA architecture optimizes content delivery, reduces network congestion, and enhances the overall performance of the system.

The CDCA architecture was designed to accentuate high level of scalability and manageability at runtime to accommodate the dynamic nature of network environments. To achieve this, a logically centralized platform, such as a distributed SDN Controller system, is employed to provide scalability and resilience at the SDN layer [14]. While the specific analysis of the distributed SDN Controller system is not within the scope of the paper, it serves as a foundational component for the overall architecture.

Multiple instances of the Proxy and Cache components can be dynamically added or removed at runtime, facilitated by a cloud orchestration platform like OpenStack. This allows for an elastic and flexible solution where the system can adapt to changing demands and resource requirements. The management system, which could be integrated with the cloud orchestration platform, plays a vital role in monitoring the server's response time and making decisions regarding the deployment or termination of Proxy and Cache instances.

Similar to cloud management, the management of the Proxy and Cache instances follows principles of scalability and flexibility. The system can scale up or down based on the workload and user demands, ensuring optimal performance and resource utilization. Cloud management practices can serve as a reference for managing the Proxy and Cache components, but further exploration and research in this area are needed.

Of particular note to address the scalability issues mentioned in Section II, the CDCA architecture deploys many Proxy and Cache instances horizontally to assure a desired response time for any user by taking an integrated distributed Proxy approach to solve the unique Proxy limitation of the Chandra et al. proposal [11]. This CDCA approach directive offers several advantages in utilizing NFV in real-time network orchestration, particularly in optimizing runtime processing, transfer times, and scalability. Here are some benefits of using NFV in this context:

1) Flexibility and agility: NFV enables the virtualization of network functions, allowing them to be deployed and scaled as needed. This flexibility enables dynamic resource allocation and efficient utilization of infrastructure based on the current workload. It allows for quick deployment and adjustment of network services, leading to improved agility in responding to changing network demands.

2) Scalability: NFV provides scalability by allowing network functions to be dynamically instantiated and scaled according to the workload. During peak times, when network traffic is high, additional instances of network functions can be provisioned to handle the increased load, ensuring smooth operation and optimal performance. Similarly, during low-traffic periods, unnecessary instances can be scaled down or deactivated, saving resources.

3) Resource optimization: NFV allows for efficient utilization of hardware resources by consolidating multiple network functions onto virtualized infrastructure. This consolidation eliminates the need for dedicated hardware for each network function, leading to cost savings and improved resource utilization. Additionally, NFV enables the sharing of resources among different network functions, optimizing resource usage based on demand.

4) Faster deployment and service innovation: NFV decouples network functions from proprietary hardware, allowing them to run on general-purpose servers or cloud infrastructure. This decoupling simplifies the deployment process and reduces the time required to introduce new services or update existing ones. It enables service providers to rapidly deploy and scale network functions, promoting faster innovation and time-to-market for new services.

5) Cost efficiency: NFV can result in cost savings by reducing the need for expensive proprietary hardware appliances. Instead, virtualized network functions can be run on standard servers or cloud infrastructure, which are typically more cost-effective. NFV also enables service providers to adopt a pay-as-you-grow model, scaling their infrastructure based on actual demand, thereby optimizing costs.

By leveraging NFV in real-time network orchestration, service providers can achieve optimum runtime processing, reduce transfer times, and achieve high scalability, allowing them to efficiently handle variable network workloads and provide a better quality of service to their users.

Overall, the CDCA architecture aims to provide a scalable and manageable solution by exploiting distributed SDN controllers, dynamic deployment of Proxy and Cache instances, and integration with cloud orchestration platforms. Future work can delve deeper into the management aspects,

drawing inspiration from cloud management practices to enhance the efficiency and effectiveness of the system.

Algorithm 1 gives the core logical idea behind the operational aspects of the CDCA architecture, describing how the Proxy and Cache instances work together. Both components have a specific task in the processing of user requests. The Proxy is responsible for identifying and deciding *where* and *which* requests are going to be cached, while the Cache is responsible for providing the storage resources in order to Cache the contents according to the network policy. The details of the operational aspects of each component are described in the next subsections.

Algorithm 1: CDCA architecture operation

```
The user triggers a content request;
The SDN Controller forwards the request to the closest Proxy cPrx;
cPrx checks its shared index if there is a content copy under its domain;
if there is a content copy then
    cPrx forwards the request to the Cache cCh thatholds the content copy;
    cCh delivers the cached content to the user;
end
else
    cPrx forwards the request to the closest Cache cCh;
    cCh retrieves the content from the original ContentServer;
    cCh delivers the content to the User;
    cCh stores the content for next User;
    cCh notifies cPrx that a new content has beencached;
    cPrx updates the index;
end
```

A. Content Forwarding

The dependency on the SDN infrastructure in the CDCA solution for content dispatching offers several advantages, particularly in terms of efficiency and transparency. Some key points related to this dependency:

1) Transparent request forwarding: SDN allows for the transparent forwarding of content requests to cacheable content and Proxy instances on the network. By leveraging the OpenFlow protocol or similar SDN technologies, it becomes possible to create traffic flows on the switches that map specific TCP or UDP ports of applications. This enables the SDN Controller to direct requests to the appropriate Cache or Proxy without modifying the IP packet header. This transparency ensures that the communication between clients and the requested content or Proxy remains seamless and unaffected.

2) Efficient traffic steering: With the help of SDN, traffic can be efficiently steered to the desired destinations. By leveraging the programmability and control capabilities of SDN, the SDN Controller can dynamically analyse network conditions, load distribution, and Cache availability to make intelligent decisions on how to direct traffic. This enables the system to optimize content delivery by sending requests to Cache instances or the closest Proxy, minimizing latency and improving overall network performance.

3) Flexibility and adaptability: The use of SDN provides flexibility and adaptability to the solution. Since the forwarding, behaviour of switches can be dynamically controlled by the SDN Controller, changes in the network topology or caching infrastructure can be easily accommodated. New Cache instances or Proxy nodes can be added, removed, or reconfigured without requiring changes in the underlying network infrastructure. This flexibility allows the solution to scale and adapt to changing demands and evolving network conditions.

4) Enhanced network visibility and control: SDN offers centralized network management and control, providing enhanced visibility and control over network traffic. By having a centralized SDN Controller, network administrators can monitor and manage content dispatching, Cache utilization, and overall network performance from a single point of control. This centralized control enables efficient decision-making and troubleshooting, leading to improved network efficiency and performance.

Typically, the SDN infrastructure for content dispatching brings efficiency, transparency, flexibility, and enhanced control to the CDCA solution. By using SDN technologies like the OpenFlow protocol, it becomes possible to transparently forward requests to cacheable content and Proxy instances on the network without the need to modify IP packet headers. This enables efficient traffic steering and dynamic adaptability, leading to improved content delivery and network performance.

In the CDCA architecture, two approaches are used for creating flows in the SDN switches: proactive and reactive [15]. Here are the characteristics and considerations associated with each approach:

1) Proactive approach: In the proactive approach, the OpenFlow Controller configures all the necessary flows from the users to the nearest Proxy instance before any request is made to a server. This means that the flows are pre-installed in the switches based on anticipated traffic patterns. The advantage of this approach is that it avoids the need for switches to request the Controller for each new flow, thereby reducing forwarding delay. However, one drawback is that all data packets are routed through the same network segment or domain path, which can potentially lead to congestion on that path.

2) Reactive approach: In the reactive approach, flows are created on-demand, meaning that they are installed in the switches only when a request is made by a user. When a switch receives a packet for which there is no pre-installed flow, it sends a request to the Controller, which then installs the appropriate flow and forwards the packet accordingly. This approach introduces some latency as switches need to

contact the Controller for each new flow. However, it provides the opportunity to use load-balancing techniques to set alternate paths, thereby reducing congestion and improving network performance.

In the CDCA architecture, the Controller sets a path to the nearest Proxy instance in a reactive manner, meaning that flows are installed on-demand as requests are made by users. This allows for dynamic path determination based on the current network conditions and load distribution. On the other hand, the path from the user to the Cache or external network is set proactively, meaning that the necessary flows are pre-installed to optimize the forwarding of user traffic. This proactive approach helps minimize latency and optimize link usage. Remember that the NFV module establishes a new path for each new user request, providing scalability and load balancing.

By combining the proactive and reactive approaches, the CDCA architecture aims to achieve an efficient and balanced network operation. The reactive approach mapping users' requests to the nearest Proxy minimizes latency by dynamically determining the optimal path based on current network conditions. Meanwhile, the proactive approach setting the path from the user to the Cache or external network to ensure efficient forwarding without switches requires the Controller the path for each new flow [15].

It is important to note that the choice between proactive and reactive approaches may depend on specific network conditions, traffic patterns, and performance requirements. Both approaches have their advantages and trade-offs, and the decision should be made based on the specific needs and constraints of the network deployment.

When the SDN Controller receives an HTTP request, it sets a flow from the user to the Proxy. The forwarding is based on the destination IP address, i.e., the Content Provider IP, destination HTTP port (typically port 80) and the user's IP address to select the best Proxy to be used, e.g., closest to the user. The Proxy element is necessary because the SDN only analyses IP header fields, not HTTP requests. The Proxy analyzes the HTTP GET header and checks if the content is cached in its DHT index table. Then, it forwards, i.e., sets the flow in network switches, to connect the user to Cache. From this moment onwards, the user interacts only with the Cache until a new request is done, minimizing Proxy processing. In big content delivery, e.g., VoD services, a user maintains a long-time connection with a Cache.

Once a request arrives at a Proxy, it performs a deep inspection of the request to decide to which Cache this request should be forwarded. This inspection is possible for requests using the HTTP protocol, as the packet payload can be read by the Proxy. However, for requests using the HTTPS protocol, which provides encryption and security, performing deep inspections on the packet payload is not feasible and would risk breaching security.

When the Proxy needs to determine the path from the user to the closest Cache, it sends a command to the SDN Controller using a Representational State Transfer (REST) Application Programming Interface (API). The API allows the Proxy to communicate with the SDN Controller and provide the necessary information to install the required flows on the switches that belong to the network path connecting the user to the closest Cache.

This process of sending commands to the SDN Controller and installing flows on the switches ensures that the traffic is directed efficiently and transparently, optimizing the content delivery process. Fig. 2 provides an illustration of this timeline flow process, highlighting the interactions between the Proxy, SDN Controller, and switches in the network path as follows:

1) Content request identification: The SDN Controller identifies content requests from users within the network.

2) Forwarding to proxy: The SDN Controller forwards the content requests to the Proxy component responsible for handling Cache-related operations.

3) Checking cache availability: The Proxy checks if the requested content is already cached within the network.

4) Forwarding to cache: If the content is already cached, the Proxy forwards the request to the appropriate Cache instance that stores the content. This allows for direct content delivery from the Cache to the user.

5) Flow path configuration: The SDN Controller configures a data flow path from the Cache to the user, establishing a direct transmission route for efficient content delivery.

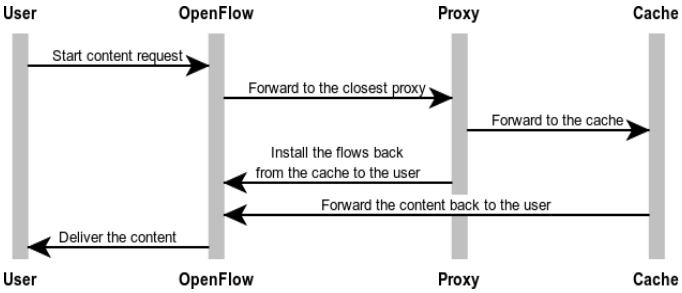

Fig. 2. Content forwarding sequence flow steps.

The CDCA approach ensures a transparent Cache handling for users without requiring any changes to their application implementations. The underlying transport protocol, TCP, necessitates that both the Proxy and Cache nodes are aware of connection handling details, as the connection state needs to be exchanged between them by SDN Controller that knows all Cache and clients address [16]. This awareness allows for the seamless transfer of connections from the Proxy to the Cache.

The use of SDN in implementing a transparent forwarding mechanism is also discussed in the work by Koulouzis et al [17]. Their research focuses on enhancing the transfer of data-intensive scientific applications by leveraging SDN network programmability.

By incorporating SDN principles and applying the REST API and SDN architecture, the CDCA architecture enables dynamic and controlled flow management, allowing for efficient content caching and delivery while respecting security considerations and the limitations imposed by the HTTPS protocol.

The release of HTTP/2 introduces improvements and new challenges to the Web. Its specification was published in May 2015 [18], and its adoption has been growing. The HTTP/2 decreases latency and improves web browsers' load speed and video delivery by implementing many new features [19]: (1) compression of HTTP headers; (2) inclusion of HTTP/2 Server Push; (3) multiplexing and pipelining of multiple requests over a single TCP connection; and, (4) fixing of blocking problem in HTTP/1.1. The HTTP/2 specification recommends cryptography using TLS 1.2, but it is not mandatory. For video streaming, HTTP/2 Server Push relieves the Proxy processing, and the video chunks should be delivered unencrypted to reduce the encryption overhead of video files.

The CDCA architecture supports HTTP/1.1 and HTTP/2 offering seamless smooth ambiance effect video viewing without side effects, because the request are initiated at the client side. In HTTP/1.1, with no Server Push, the performance is lower than HTTP/2. In general, most of the contemporaries, like CDCA researcher, consider encrypted video to be unfeasible for normal viewing because each browser would need a different crypt-key per connection. However, if cryptography is required for whatever reasons, CDCA system is easily capable of providing any publicly available standard encryption/decryption module such as a Deep Packet Inspection (DPI) module, which will be addressed in another article.

Content identification is the initial most important task performed in the Proxy. The architecture does not Cache all kinds of information, but only large and reusable content, e.g., video and music on demand, software installers, or sizable images.

The most important, big, and reusable content object application on the Internet is VoD. A plethora of VoD providers exist. All of them use similar approaches to provide a variety of content delivery services. Most of them use Flash and HTML5 as video player. In the CDCA experiment, HTML5 was chosen since it is better for identifying video characteristics. The video stream encoder is mostly H.264/MPEG-4, and the video stream is not encrypted to reduce server and client overhead. This fact is very important for the CDCA architecture because there are some outstanding security issues regarding encrypted content, which should be readable only by the first user. And, of course, we suppose the multimedia videos are royalty-free to permit distribution to all users, which, in reality, might be different in business-oriented provider networks with regard to payment to owners of the content or to their nominated agents, such as the service provider.

Currently, the transport protocol for video streaming on the Internet is accomplished by using either TCP or UDP. The delivery of live video streaming with *on-the-fly* encoding, like IPTV, is mostly UDP based, but the delivery of pre-encoded video, called VoD, essentially uses TCP [20]. In the CDCA architecture, the focus is on VoD services over TCP. For the evaluation, YouTube is considered since it is the most prominent VoD portal, which handles more than several billion video streams daily. While the YouTube site itself operates over HTTPS, the actual video stream requests and delivery are performed using HTTP. This allows for the identification of unencrypted HTTP requests and the corresponding HTTP objects are easily identified using pattern-matching techniques [21]. It is worth noting that the architecture is designed to handle TCP-based VoD services and can be adapted to support other VoD providers with minor modifications. The future intention is to analyze and adapt the CDCA solution for various VoD platforms, applying the same approach used for YouTube.

For instance, considering a YouTube video URL, the web page provides the link for different formats, e.g., *href="http://www.youtube.com/watch?v=yXc8KCxyEyQ"* for standard format, *href="http://m.youtube.com/watch?v=yXc8KCxyEyQ"* for hand-held devices, and also for specific devices, like Android, *href="android-app://com.google.android.youtube/http/www.youtube.com/watch?v=yXc8KCxyEyQ"*.

It seems that Google implements some techniques to restrict the direct access to video in a lot of ways. It is possible to see some interesting fields in the URLs, e.g., "?part=" and "?range=," which permit identifying of each downloaded chunk and storing it in the correct order. The process to retrieve a Youtube video requires reverse engineering to extract chunk information from the URL, a technique used by some "Youtube downloaders", which is very common nowadays.

YouTube employs various mechanisms, including DNS translation and URL redirection, to optimize the delivery of video content. These mechanisms help distribute the video's chunk files across multiple caching servers, allowing for load balancing of the transmission.

The YouTube video delivery name is composed of three key components [22]:

1) Video ID space: Each video on YouTube is assigned a unique video ID, which serves as an identifier for the specific video content.

2) Hierarchical logical video server: YouTube utilizes a hierarchical logical video server structure to organize and manage video content. This structure helps ensure efficient content management and delivery across the platform.

3) Physical server cache hierarchy: The physical server Cache hierarchy represents the distribution of caching servers used by YouTube. The chunk files of the videos are stored in these caching servers, which are strategically located to optimize content delivery and minimize latency for users.

By employing DNS translation, URL redirection, and a well-structured video delivery name, YouTube can efficiently distribute and deliver video content, providing a seamless streaming experience to its vast user base as:

a) YouTube Video Id Space: Each YouTube video is uniquely identified using a fixed-length flat identifier with random (nonsensical) characters, e.g., something like *yXc8KCxyEyQ*.

b) Hierarchical Cache Server DNS Namespaces:

YouTube defines multiple DNS namespaces representing a group of logical video servers related to the video format and resolution. The DNS namespace forms a hierarchical structure of logical video servers that are mapped to an IP address where the video file is stored, e.g., http://b8u-4vge.googlevideo.com

c) Physical Cache Servers: In the case of YouTube, the logical Cache servers are represented by a unique logical namespace in the DNS names. However, each DNS resolution of this namespace can map to a large set of IP addresses, which correspond to the physical Cache servers. For example, if we consider the hostname *http://b8u-4vge.googlevideo.com*, the primary namespace represents the logical Cache server. However, depending on factors such as the user's location and the load balancing policy in place, multiple IP addresses can be associated with this hostname. These IP addresses correspond to the physical Cache servers that store and serve the video content.

YouTube uses HTTP to deliver videos to users in order to reduce the cryptography overheads. Even under an HTTPS connection, the HTTP request can be easily seen. Analyzing the URL, we can identify the video name (Video Id Space) and the format and resolution for a specific device (Hierarchical Cache Server DNS Namespaces). In the CDCA architecture, the *Physical Cache Server* component does not give away any useful information.

B. Proxy Design

The Proxy component acts as an intelligent intermediary between users and Cache nodes, ensuring efficient content delivery and load balancing within the network. It plays a crucial role in the CDCA architecture as it handles users' requests and ensures the delivery of the requested content to the nearest available Cache. Its main function is to determine the appropriate Cache node to serve the content based on a mapping index.

When a user makes a content request, the Proxy checks whether there is an existing copy of the requested content in the network. If a copy is available, the Proxy forwards the request to the Cache node that holds the content. This minimizes the latency and improves the response time for the user. In cases where the requested content is not available in any Cache node, the Proxy identifies the Cache node closest to the user and forwards the request to that node. This Cache node becomes a aspirant for holding a copy of the requested content. If there are multiple Cache nodes with the same distance from the user, the Proxy employs a round-robin operation. This load-balancing technique distributes the requests among the Cache nodes equally, ensuring efficient utilization of resources and preventing any single Cache node from becoming overloaded.

Since it is possible that many Proxies exist over the network, each Proxy instance participates in a DHT to share the content index. This DHT serves as a Distributed Forwarding Unit (DFU) that allows efficient content lookup and forwarding among the proxies.

While there are multiple Proxy instances distributed throughout the network, it is important to note that each user's requests will always be handled by the same Proxy node nearest to it. This selection of the closest Proxy node to the user ensures proximity-based routing and minimizes latency in the content delivery process.

By maintaining the user's requests consistently routed to the same Proxy node, it offers several advantages. Firstly, it provides a predictable and reliable user experience as the user interacts with the system through a specific Proxy node. Secondly, it allows the Proxy node to maintain the context and state of the user's requests, facilitating personalized content delivery and improving overall efficiency.

The sharing of the DHT among the Proxy instances enables efficient content indexing and lookup. When a user request arrives at a Proxy node, it can quickly query the DHT to determine if the requested content is available in the network and identify the Proxy node that holds a copy of the content. This ensures effective content retrieval and delivery to the user.

The forward decision is taken by the SDN Controller that receives the network state information from the network management system, choosing the best Proxy instance. Because any new user's request is processed by the SDN Controller, it should choosethe optimal Proxy *on-the-fly* based on the network status. Generally, the closest Proxy will be the perfect solution because it will have the minimum delay. It also does not store any content and only looks at a DHT table. However, if one Proxy processes more requests than another, it is possible to redirect the requests to load balance the system considering network and server status.

The Proxy and Cache instances maintain a virtual overlay network configuration schema, which enables them to exchange Cache states and commands seamlessly. When a Proxy decides to forward a user's request to a Cache instance, it sends a command to the chosen Cache, specifying the request and the user. This command instructs the Cache to serve the requested content to the user.

Upon sending the command, the Proxy listens for an acknowledgment from the Cache. This acknowledgment confirms that the content has been successfully cached by the Cache instance. It allows the Proxy to update its index, which maps content names to Cache instances. This updated index enables the Proxy to efficiently forward future requests for the same content to the same Cache instance, enhancing caching effectiveness and reducing content retrieval latency.

Additionally, the acknowledgment may provide information about the content's lifetime. The Cache instance specifies the maximum duration for which the content will be cached. Once the Cache lifetime expires, the corresponding index entry for the content is automatically removed from the DHT, ensuring that outdated content is not unnecessarily stored in the network.

By maintaining an efficient and synchronized index across Proxy and Cache instances, the architecture optimizes content caching, improves content availability, and ensures that the most relevant content is stored and served efficiently.

The content caching mechanism is designed to be transparent and efficient, allowing any Proxy to forward a request to a Cache instance that already holds the requested content. This means that even if a Proxy is responsible for managing a specific Cache instance, it can still forward requests to other Cache instances that have the desired content.

As an example, for instance, where Proxy 'A' manages Cache 'A' and Proxy 'B' manages Cache 'B', if a user near Proxy 'A' requests content 'C', Proxy 'A' would handle the request. However, since Proxy 'A' knows that Cache 'B' has a copy of content 'C' (as indicated by the DHT index), it would forward the user's request to Cache 'B' to fetch the content. This behaviour enables efficient utilization of the caching infrastructure, as any available Cache instance can serve content to users, regardless of the Proxy managing it, thus optimizing content retrieval and delivery, ensuring that users can access the content efficiently from the nearest available Cache instance. This approach enhances the overall performance of the caching system and improves user experience.

Usually, choosing the optimal Cache or deciding to fetch content from an external server is most desirable in content delivery systems. The choice between fetching content from an in-network Cache or from an external server depends on various factors, such as Cache proximity, network conditions, content availability, and delivery requirements.

In the CDCA architecture, the decision of whether to fetch content from an in-network Cache or from an external server is not explicitly addressed. However, future works and research can focus on developing intelligent policies or algorithms to determine the best Cache to use within the network or when it is more efficient to fetch content from an external server. These policies could take into account factors such as Cache proximity, network congestion, content popularity, Cache availability, and other performance metrics. By considering these factors, the system can dynamically adapt and make better decisions based on the current network conditions and optimize content delivery for the best user experience.

Overall, this Proxy design scheme enhances the scalability, reliability, and performance of the CDCA architecture.

C. Cache Design

The Cache component plays a vital role in serving user requests and optimizing content delivery. When a request reaches a Cache instance, it first checks its local memory using a Hash Table for the fast lookup to see if the requested content is already stored. If the content is found in the Cache, it can immediately send the requested content back to the user through the previously configured OpenFlow path. This enables fast and efficient delivery of content from the Cache without the need to retrieve it from the original server.

However, if the content is not available in the Cache, the Cache acts as a Proxy and forwards the user's request to the target server specified in the request. This behaviour is similar to a standard client-server interaction without caching. When the target server responds to the request, the Cache immediately sends the response back to the user.

Additionally, the Cache performs a deep packet inspection on the response received from the target server. This inspection helps determine if the response is cacheable or not. The Cache examines the nature of the protocol being used (such as HTTP) and looks for specific headers, like the Cache-Control header in the case of HTTP. The Cache-Control header provides instructions to consumers (in this case, the Cache) on how the response can be cached, including the duration for which it can be cached or whether caching is prohibited by the server.

When the Cache determines that the fetched content is cacheable, it immediately notifies the Proxy that it now holds a copy of the content. This ensures that subsequent requests for the same content can be efficiently served from the Cache. Additionally, the Cache may include a Cache lifetime value, which indicates the maximum duration for which the content should be considered valid in the Cache. The determination of the Cache lifetime value depends on the characteristics of the media, such as its freshness requirements or expiration policies.

In cases where the content is deemed non-cacheable, such as very small content that may not benefit from caching, the Cache simply sends the response directly to the user without storing it. This prevents unnecessary utilization of Cache space and avoids the need for DHT updates related to that particular content.

Setting the Cache lifetime value appropriately is crucial for optimizing system performance. It allows balancing between serving stale content and the overhead of fetching fresh content from the server. The specific Cache lifetime values and policies can be defined by the content provider according to their own requirements and policies.

D. Cache Management

Separating the data storage layer from the control function of Cache instances offers flexibility and scalability in the deployment of the caching system. By decoupling these two functions, each Cache instance can operate independently without interfering with the others in the storage layer.

This separation allows for various deployment scenarios, each with its own characteristics. For example, multiple Cache instances can be deployed on different physical servers or virtual machines, enabling distribution across different geographical locations or network segments. Each Cache instance can have its own storage resources, such as disk space or memory, dedicated to serving content requests. For instance, an ISP may deploy a number of caches over their network, configuring 70% of the instances to use their RAM memory while the other 30% are configured to use SSD disks. The decision about the type of storage that will be used by the Cache instances and where they are deployed is a decision that must be taken by the network operator since there are many strategical decisions involved.

The separation of data and control also enables the implementation of different caching strategies or policies within each Cache instance. This means that each Cache

instance can have its own set of rules and algorithms for content eviction, replacement, or caching optimization, tailored to specific requirements or objectives, allowing for efficient and independent operation of each Cache instance within the caching system.

The decision of whether to store content in memory or on local disks within a Cache instance is an important aspect that can significantly impact caching performance and efficiency. Both options have their advantages and considerations.

Using an in-memory approach provides faster access to cached content due to the high speed of memory. It is particularly suitable for frequently accessed or hot content that requires low-latency delivery. However, memory capacity is typically limited compared to disk capacity, which means that the Cache can store a smaller amount of content in memory. This can lead to a higher likelihood of Cache eviction for less frequently accessed or cold content, resulting in potential Cache misses and increased latency.

On the other hand, storing content on local disks provides a larger storage capacity, allowing the Cache to accommodate a larger volume of content. This is advantageous for caching less frequently accessed or larger files. Disk-based storage can be especially useful for VoD (Video on Demand) services that deal with pre-encoded videos, where the content size can be substantial. However, accessing content from disks is generally slower compared to memory, which can introduce additional latency.

The choice between in-memory and disk-based storage depends on various factors, including the nature of the content, the expected workload, and the network administrator's policy. If the network administrator prioritizes fast access to frequently accessed content, an in-memory approach might be preferred. Conversely, if accommodating a larger volume of content is crucial, disk-based storage is more suitable.

Additionally, the eviction policy is an essential consideration in Cache design. It determines how content is selected for eviction when the Cache reaches its capacity limit. There are various eviction algorithms, such as LRU (Least Recently Used), LFU (Least Frequently Used), and Random Replacement, each with its own trade-offs in terms of Cache efficiency and performance. The network administrator can choose the most suitable eviction algorithm based on factors such as content popularity, access patterns, and the desired Cache hit rate.

The CDCA architecture is flexible such that it can be configured to use any eviction algorithm by an open API, i.e.; the network administrator can choose the best algorithm according to the user’s profile. Cache eviction policies are well discussed in Balamash [23] and Wang [24].

It is possible to use a hybrid approach when dealing with the decision to store the content in memory or on disk. For instance, a hybrid approach combining both memory (M1) and disk (M2) storage can provide a balance between fast access and larger storage capacity. This approach takes advantage of both memory and disk to optimize caching performance.

In the hybrid approach, frequently accessed or hot data is stored in the M1 Cache, which is the faster memory component. This ensures that popular content is readily available for fast retrieval and reduces latency for frequently requested items. The M1 Cache acts as a high-speed Cache tier that can quickly serve content without accessing the slower disk storage.

On the other hand, less frequently accessed or cold data is stored in the M2 Cache, which resides on a disk. The M2 Cache provides a larger storage capacity compared to memory, allowing the Cache to accommodate a broader range of content. Although accessing content from the M2 Cache may introduce additional latency, the presence of frequently accessed items in the M1 Cache minimizes the impact on overall performance.

To optimize the hybrid approach, a managing schema algorithm is employed to determine which data should reside in the M1 Cache and which should be stored in the M2 Cache. This algorithm can monitor access patterns, frequency of requests, and other relevant metrics to make informed decisions about data placement. For example, if a content item in the M2 Cache starts to experience increased access frequency, the managing schema algorithm can dynamically promote it to the M1 Cache to improve access time.

The reconfigurability of the Cache instance is a valuable feature that allows for dynamic adjustments and fine-tuning of the Cache's parameters to meet changing demands and optimize performance. By deploying the Cache instance within a Virtual Machine (VM), it becomes possible to modify various aspects of the Cache configuration during runtime.

One such configurable parameter is the storage capacity, which includes both memory and disk space. As the workload increases and the Cache approaches its capacity limit, it may become necessary to adjust the available storage resources to accommodate additional data. This can be achieved by dynamically increasing the memory allocation or expanding the disk space assigned to the Cache VM.

In addition to storage capacity, other parameters such as eviction policies can also be tuned. The eviction policy determines which content items are evicted from the Cache when it reaches its capacity limit. By adjusting the eviction policy, the Cache manager can prioritize certain content or employ different strategies to optimize Cache utilization and improve hit rates.

When the Cache becomes saturated and the eviction rate surpasses a predefined threshold, the Cache manager can take proactive measures to address the situation. This may involve automatically attaching a new VM to distribute the caching load, increasing the available memory to accommodate more content in the M1 Cache, or modifying the eviction policy to better manage the Cache's content.

The ability to make such adjustments dynamically (on-the-fly) and in a dynamic manner allows the Cache instance to adapt to varying workloads and optimize its performance in real time. This flexibility ensures that the Cache can efficiently handle increasing demands and effectively utilize available resources, ultimately enhancing the overall

efficiency and responsiveness of the caching system.

E. Cache Policy Management

Designing a system that can cater to various business needs and network traffic patterns requires careful consideration of policy management. Different organizations and network environments may have specific requirements, priorities, and constraints that need to be taken into account. Therefore, the system project incorporates design strategies to enable efficient content delivery while also accommodating complex real-world policies.

Policy management encompasses various aspects, including caching policies, eviction policies, load balancing policies, content placement policies, and more. These policies define how the system operates, makes decisions, and prioritizes tasks. By incorporating flexibility in policy management, the system can be customized and tailored to meet specific requirements.

One important design strategy is to provide configurable parameters and APIs that allow administrators or users to define and modify policies according to their needs. This flexibility empowers organizations to adapt the system to their unique business rules and network traffic patterns. For example, administrators can define caching policies based on content popularity, user preferences, or other relevant factors.

Furthermore, the system project may offer a range of pre-defined policy templates or algorithms that serve as a starting point for administrators to choose from. These templates can be based on industry best practices or research findings, providing guidance for policy selection. Administrators can then fine-tune and customize these templates to align with their specific requirements.

Additionally, the system may provide monitoring and analytics capabilities to gather data on network traffic, content usage patterns, performance metrics, and other relevant information. This data can be used to evaluate the effectiveness of existing policies and make informed decisions for policy adjustments or optimizations.

By considering policy management as an integral part of the system design, the project aimed to provide a flexible and adaptable solution that can cater to diverse business needs and network scenarios. This approach acknowledges that different organizations may have unique policies and requirements, and it offers the means to configure and manage these policies effectively to achieve optimum performance and content delivery outcomes.

The policy management function being performed on the Proxy component is a logical and efficient choice. As the decision-maker responsible for managing a set of caching instances, the Proxy is well-positioned to handle policy management tasks. Since all requests from a subset of network nodes pass through the Proxy, it has the necessary visibility and control to implement and enforce caching policies effectively.

By extending the Proxy's content inspection capabilities, it becomes possible to deeply analyse the content's data and incorporate manageable aspects into the policy management process. This allows the Proxy to make intelligent decisions about which content should be cached based on specific criteria or conditions.

In the current stage of proposal development and experimentation, the system has addressed five caching policies to provide adaptability to different workloads commonly encountered in practice:

1) Cache everything: This policy implies caching all content without any specific filtering or criteria. It ensures that all requested content is stored in the Cache for future retrieval.

2) Cache only content whose name matches any given regular expression set: This policy allows administrators to define a set of regular expressions to match content names. Only content with names that match these expressions will be cached, while others will be bypassed.

3) Cache only content whose size matches certain file criteria: This policy focuses on caching content based on their file size. Administrators can define specific criteria (e.g., minimum or maximum file size) to determine which content should be cached.

4) Cache only content served by a specific set of target domains: This policy restricts caching to content served by designated domains. Administrators can specify a list of target domains, and only content from these domains will be eligible for caching.

5) Cache-only content of a given type (audio, video, etc.): This policy enables selective caching based on content types. Administrators can specify the types of content (e.g., audio, video, images) that should be cached, while excluding others.

The system utilizes a pipeline processing approach to handle multiple sets of distinct policies in an efficient manner. This approach allows the policies to be applied in a combinatorial fashion, starting from the most restrictive to the least restrictive that serves as a set of policy filters to derive an optimal management solution.

When a request reaches the Proxy, it undergoes a deep inspection phase to gather relevant information about the content. Following this, the request is processed through the policy pipeline, which consists of sequentially applying the defined policies to determine whether the content should be cached or not.

The policy pipeline filters the requests based on the defined rules, allowing or denying caching of the requested content. If a request is denied by any policy rule in the pipeline, the Proxy sends a forward command to the Cache instance without expecting Cache confirmation or acknowledgment. In this case, the content will have a flag indicating that it cannot be cached by that specific Cache instance.

The algorithm presented in Algorithm 2 serves as an example of how the policy pipeline operates for Caching, determining the caching behaviour based on the policies:

1) Initialize the request
2) Perform deep inspection on the request

3) Set caching_allowed = True

4) For each policy in the policy pipeline: a. Apply the policy rule to the request b. If the policy denies caching: - Set caching_allowed = False - Break the pipeline loop

5) If caching_allowed is True: a. Forward the request to the closest Cache instance to the user b. Expect Cache confirmation/acknowledgment

6) If caching_allowed is False: a. Send a forward command to the Cache instance b. Set the content's "*cacheability"* flag to indicate no caching

By processing requests through this policy pipeline, the system can effectively filter and determine the caching behaviour based on the defined policies. The pipeline approach allows for flexible and customizable policy management, enabling administrators to derive an optimal caching solution based on their specific requirements and policies.

In some cases, certain requests cannot enter the processing pipeline until they have been served by the originating server. For example, in HTTP requests, the requester may allow the receipt of both text and video responses, but the actual response will determine the content type by inspecting the ContentType header. Similarly, the content's size may not be known until the response has been completely received.

In such cases, the same processing pipeline can be applied at the Cache instances, but only in specific scenarios where the request can be fully inspected and the necessary information is available. The Cache instances can run the policy pipeline to determine the caching behaviour based on the received response.

It's important to note that if the Proxy's flag command allows caching, it prevents the Cache from overriding any previous decision made by the Proxy. This ensures that the caching behaviour determined by the Proxy is maintained and not altered by the Cache instances.

By allowing the processing pipeline to run at both the Proxy and Cache instances, the system can ensure consistent caching decisions and policies across the network, taking into account the specific characteristics and information available at each stage of the request-response cycle.

Algorithm 2: Policy processing method

```
Data: The content request cRqst; and a set of filters
      polFltr[]
Result: 1 if the content should be cached, 0 otherwise
foreach p in polFltr[] do
    if p(cRqst) == 0 then
        return 0;
    end
end
return 1;
```

F. System Scalability

Scalability is a crucial aspect of the CDCA architectural system, and it involves the SDN Controller, proxies, and caches. To ensure scalability and address availability concerns, the architecture allows for the deployment of new instances of these components in the network on the fly. One key factor in achieving scalability is the use of a stateless OpenFlow control. This enables simple load balancing across multiple Controller devices, ensuring that the control plane can handle increasing demands and distribute the workload effectively [9]. By distributing the control plane functionality, scalability and redundancy are improved, as multiple controllers can handle the control tasks in a distributed manner. In the context of SD-ICN, scalability becomes an even more significant concern due to the introduction of in-network caching and content-based communication. To address the control plane scalability challenge, Gao et al. propose the Scalable Area-based Hierarchical Architecture (SAHA) [25]. SAHA is designed to handle the control plane scalability problem specific to SD-ICN environments and provides a hierarchical architecture that enables efficient management and scalability.

Several distributed architectures have been CDCA to enhance OpenFlow scalability and redundancy. Examples include Disco [26], ElastiCon [27], and Onos [28]. These architectures aim to distribute control plane functionality, improve scalability, and provide redundancy mechanisms to ensure high availability.

By leveraging these scalable and distributed architectures, the CDCA system can handle increasing demands, distribute control tasks effectively, and provide redundancy to ensure system availability. This enables the system to accommodate a growing number of users, requests, and caching instances while maintaining efficient control plane operations.

Analyzing the Proxy design, it is possible to notice that its main idea is to simply forward the request to the appropriate Cache, thus it relies on the DHT index to get all information it needs about the caching state, which is spread over all Proxy instances of the network topology. The most costly operation that the Proxy does (determine where content was previously cached) basically relies on a DHT lookup operation, which is the operation that can constrain the Proxy scalability. Since the chosen DHT implementation is in conformance with Chord [29], a lookup operation needs just $O(logN)$ messages to find any key in the table and $O(log^2N)$ messages to update any key, where N is the number of proxies deployed in the network. While the Proxy relies on a DHT to find the appropriate Cache to forward requests, the Cache itself is a simple solution that only relies on a Hash Table to look up cached contents locally, so all its operations require $O(1)$ time. The main concern about the Cache is simply its memory capacity, which obviously will limit the amount of data that can be stored in that network node.

Along with the individual characteristics of each component, the overall architecture is also important. The decision of using a microservice architectural pattern was furthermore taken based on the scalability opportunities that such design offers. Many modern systems built incloud environments take this same direction when scalability is a major nonfunctional requirement. The main characteristic of microservices comes from its own definition, which states that a microservice should do one thing, and do it well. Such an

idea allows that a system with multiple functional components can be developed, tested and deployed separately. It makes easy as operational actions that must be taken in response to business and network requirements. The CDCA solution has these characteristics, allowing, for instance; the number of proxies, caches, or controllers can be changed over time independently of each other to satisfy any demand.

The CDCA architecture has been designed with flexibility in mind, and considering how the architecture's components could be used and reused in real-world production provider networks. The architecture considers the business needs, the technical issues, and the most important constraints that concern network providers on a daily basis. As previously mentioned, the idea to use SDN and NFV brings several deployment possibilities, and the architecture has been designed to accommodate these different deployment possibilities. The network administrator can deploy the system according to the exact business and technical needs. The idea is to create autonomous *islands* responsible for handling the in-network content caching in a specific network segment/domain. These islands may contain several Cache instances and few Proxy instances, each of them operating independently, yet sharing the same data content across the DHT index at the proxies. The network segment/domain could be composed of any arbitrary set of users and/or network devices, each of them sharing common features, like geographical region, traffic patterns, etc.

V. Result Analysis of the CDCA System

The experiments have been conducted to evaluate the effectiveness of the CDCA architecture and prototype operational system. By conducting these experiments, one can gain valuable insights into the performance, scalability, and feasibility of deploying the system in real-time production networks.

Evaluating the system in a realistic environment helps identify any potential challenges, bottlenecks, or areas for improvement. It also allows you to gather empirical data on the system's performance, such as response times, caching efficiency, and resource utilization. By conducting experiments and gathering insights, one can refine and optimize the CDCA architecture, ensuring that it meets the requirements and expectations of real-world deployment scenarios. Additionally, sharing the results of these experiments and promoting further research and development in this subject area can contribute to the advancement of the field and drive innovation in content delivery networks beyond today's technical achievements.

In all experiments, the CDCA solution is compared against a *legacy* network, i.e., a traditional Internet environment where the content is delivered directly from the content server to the user.

The main objective of the experiment is to evaluate the effectiveness of the users, considering the network provider and the content provider resources. To this extent, the first experiment aims to analyze how long a set of users would wait to retrieve arbitrary contents with different chunk sizes from an external server. The second experiment aims to verify how many data packets and bytes are exchanged within the network provider's infrastructure when requests for contents with distinct sizes are performed by several users. Finally, the third experiment checks the throughput at the content provider's server at different hit rates and Cache storage capacities when requests for content with distinct sizes are performed by many users.

A. Experimental Environment Evaluation

The evaluation of the prototype using an emulation methodology provides a controlled and reproducible environment to assess the performance and behavior of the CDCA architecture. Mininet, a virtualized network platform, was chosen as the basis for the evaluation, offering the ability to create interconnected virtual devices such as hosts, switches, and controllers [30].

The evaluation scenario of the topology experiments was implemented within a c3.2xlarge Amazon EC2 VM, which provided sufficient resources including 8 virtual CPUs, 15GB of RAM, and 2 x 80GB of SSD storage. The virtualized devices in Mininet communicated with each other via virtual interfaces, enabling the execution of real protocol stacks in a virtual network.

The network topology used in the experiment is composed of one content server, one SDN Controller and six islands with three users each as shown in Fig. 3. The network driver and switching delay considered in Mininet environment are not shown. The latency stated in the figure is only the fiber propagation delay.

To control the traffic flow in the network, an OpenFlow Controller was employed. The Floodlight OpenFlow Controller was selected for its simplicity and development flexibility, which facilitated the implementation and management of the network environment [31]. The evaluation took into account the latency of a 1 Gbps Ethernet board driver (100 µs) and the switching latency in the Linux Open vSwitch. These latency values were set to be greater than those typically found in real network infrastructures to ensure the worst case scenario for evaluation purposes of the results.

The flexibility of Mininet allowed configuring parameters such as link bandwidth and delaying strategies, enabling the emulation of various network conditions and scenarios similar to production networks. This flexibility enhanced the accuracy and applicability of the evaluation results.

It is worth mentioning that the evaluation utilized OpenFlow version 1.0, as it was deemed sufficient for assessing the CDCA architecture. Subsequent versions of OpenFlow did not offer any new features or fields that would significantly impact the evaluation of the proposal.

By conducting the evaluation in this controlled environment, the researchers were able to gather data on the performance, scalability, and feasibility of the prototype. These insights help validate the effectiveness of the CDCA architecture and provide valuable information for further refinement and improvement. It has allowed for a comprehensive assessment of the CDCA architecture's capabilities in a realistic network setting.

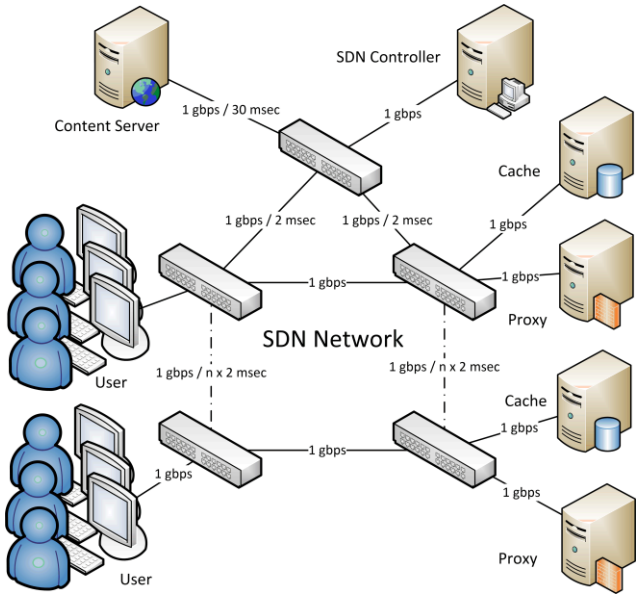

Fig. 3. Evaluation topology of the experiment.

The experiment involved the initiation of multiple applications running over HTTP, encompassing a variety of content types ranging from plain text websites to multimedia video streams. The content response chunk sizes were varied in the range of 10 to 3200 kilobytes.

The methodology employed in the experiment drew inspiration from the work of Augustin et al. [32], who extensively analysed the bandwidth usage of Web 2.0 applications. By adopting a similar approach, the experiment aimed to evaluate the CDCA architecture's ability to handle a diverse range of applications and their associated bandwidth requirements. This experimental method was selected because it demonstrated success in addressing a wide spectrum of applications, including email and on-demand video streams. This approach also permitted gathering meaningful statistics and insights into the performance and efficiency of the CDCA architecture in handling different types of content and application scenarios. It also allowed conducting experiments with various content response chunk sizes and a diverse set of applications to assess the scalability, efficiency, and effectiveness of the CDCA architecture in accommodating different bandwidth demands and traffic patterns as well as providing valuable insights into its performance and potential benefits in practical deployment scenarios.

In the experiment, each user was implemented in Java using Apache's HTTP Client library to perform HTTP requests. Each user was single-threaded, and all users concurrently sent their requests.

To simulate realistic user behaviour, the Cache hit rate was set at 70%, meaning that 70% of the requests made by the users were expected to be found in the Cache and result in Cache hits. The CDN's Cache hit rate is variable and depends on the content and user profile. The literature typically regard CDN's hit rate estimate to be in the range from 60% to 90%, so an intermediate value was used. As we used emulation, the chunks were randomly generated and the client's hit rate was set to 70%.

To optimize network resource utilization, each client used HTTP pipelining, which allows multiple requests to be sent over a single TCP connection without waiting for individual responses. This approach maximizes the use of network capacity by reducing the overhead of opening and closing TCP connections for each request. Each client was configured to have a maximum of 10 pipelined requests without responses, meaning that a client could have several pending requests in transit simultaneously, even though each client was single-threaded.

The perceived delay experienced by users was measured as a roundtrip time, starting from the moment a user sent an HTTP request until the corresponding response was received by the same user. This metric captured the overall time required for a user to receive a response and reflected the user's perceived delay in accessing the requested content.

To gather statistics on bytes and data packets, the OpenFlow Controller was utilized. After each experiment round, the OpenFlow Controller obtained the counter values for each port of the OpenFlow switches through StatsRequest messages [33]. This allowed collection of information on the amount of data transmitted and the number of packets exchanged within the network under realistic conditions. This approach provided insights into the system's ability to handle concurrent user requests, optimize network resources, and deliver content with reduced perceived delay.

In the first experiment, caches with a capacity of 1GB were used. This means that each Cache instance had the ability to store up to 1GB of content. The experiment aimed to evaluate the system's performance and effectiveness with this limited Cache capacity.

The second experiment involved two Cache instances, each with a capacity of 1GB. This setup allowed for a total Cache capacity of 2GB. By increasing the number of Cache instances, the system aimed to assess the impact of distributed caching on performance and content availability.

In the third experiment, Cache capacities varied from 2GB to 6GB. This range of capacities allowed the researchers to investigate the scalability and performance of the system as the Cache capacity increased. The experiment aimed to understand how increasing Cache capacity influenced Cache hit rates, perceived delay, and overall system efficiency.

In all three experiments, the caches were configured to use the Least Frequently Used (LFU) eviction policy. LFU is a Cache replacement policy that selects the least frequently used content for eviction when the Cache reaches its capacity limit. This policy is based on the assumption that content popularity is a significant factor in Cache usage, and frequently accessed content is more likely to be accessed again in the future.

Additionally, the caches in these experiments used an in-memory approach, meaning that the content was stored in the Cache's memory rather than on disk. This allowed for faster access times but limited the overall storage capacity compared to using disk-based storage.

The decision to use an LFU eviction policy and an in-

memory approach was based on previous research by Famaey et al. [34], which highlighted the effectiveness of popularity-based Cache replacement strategies for Video-on-Demand (VoD) services. Adopting LFU and an in-memory approach, allowed them to align the experiments with existing literature for comparison purposes and leverage the benefits of these strategies in their system.

B. Experimental Results

In the first experiment, the user's average delay when requesting content from a server was evaluated. Each experiment was executed 10 times with different chunk sizes and hit rate of 70%, showing overall results of an average of 95% confidence interval as shown in Fig. 4. It demonstrates the effectiveness of the CDCA architecture in improving the user's delay perception.

The graph illustrates that the CDCA architecture led to a significant improvement in the user's delay, regardless of the content's size. On average, the delay was reduced by nearly 75% compared to traditional approaches. This improvement indicates that the CDCA architecture effectively optimizes content delivery and reduces the perceived delay from the user's perspective.

Furthermore, the result for 3200 kilobytes of contents showed an even higher improvement of almost 80%. This suggests that the CDCA solution not only enhances the user's delay but also reduces network traffic. By utilizing caching and efficient content delivery mechanisms, the CDCA architecture minimizes the need for repeated content requests, leading to reduced network congestion and improved overall performance.

These findings support the effectiveness of the CDCA architecture in improving user experience by reducing delay and optimizing content delivery, irrespective of the content's size.

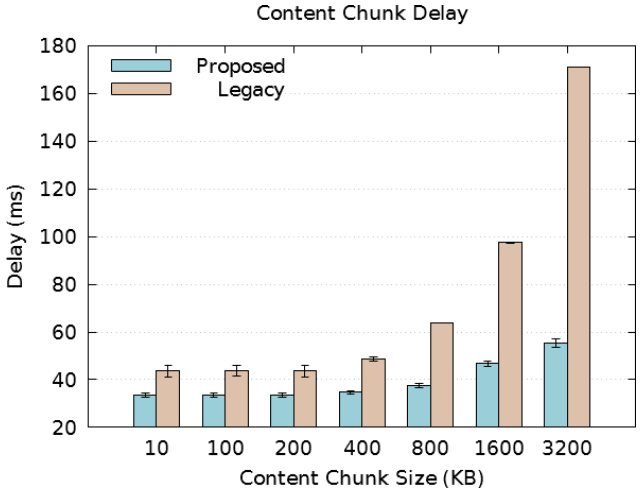

Fig. 4. User's observed delay with responses with different chunk sizes.

Fig. 5 displays the number of bytes transferred after the execution of the second experiment of the sum total of the number of bytes transferred through all switches at each network interface within the network topology. It compares the CDCA solution against traditional approaches. Both values, for all content sizes, show a reduction in the number of bytes transferred when using the CDCA solution. This reduction can be attributed to the architecture's ability to deliver content closer to the users through caching. By caching content in proximity to the users, the need to transfer the same content repeatedly over the network is minimized. This result in a more efficient utilization of network resources and a reduction in the overall data transferred.

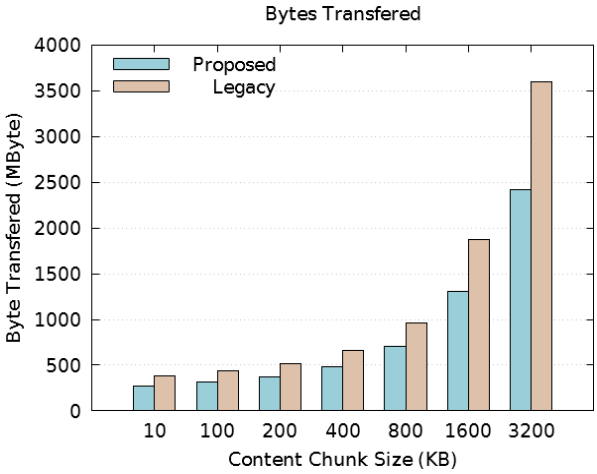

Fig. 5. Sum total of the number of bytes transferred through all switches.

The results show that the CDCA architecture effectively reduces the amount of data transferred, leading to more efficient network resource utilization and improved Quality of Experience (QoE) for users by reducing delay, as observed in the previous experiment, but also optimizes the use of network resources. The results highlight the positive impact of the CDCA architecture by minimizing data transfer and efficiently delivering content, the CDCA solution helps networks become more effective, ultimately reducing operational costs associated with bandwidth usage.

Fig. 6 and Fig. 7 show the results of the third experiment respectively.

From Fig. 6 it can be observed that operating with 2 caches storing 1GB of data the Cache hit rates increases and the throughput at the server decreases. This trend indicates that the CDCA architecture effectively reduces the number of requests which reaches the destination server as the Cache hit rate increases. This reduction in server requests is independent of the chunk size of the content. The results demonstrate that the caching mechanism of the CDCA architecture successfully offloads traffic from the server, improving its throughput.

Fig. 7 focuses on the influence of Cache capacity on server throughput at different Cache hit rates with response sizes of 3200 kilobytes. The graph shows that Cache capacity plays a significant role in the server's throughput, particularly at higher Cache hit rates. For example, at a hit rate of 40%, there is a noticeable difference of around 30 Mb/sec in server throughput when the Cache capacity increases by just 4GB. This difference becomes even more significant, reaching 100 Mb/sec, as the hit rates increase to 80%. These findings highlight the importance of Cache capacity in effectively reducing the load on the server and improving its throughput.

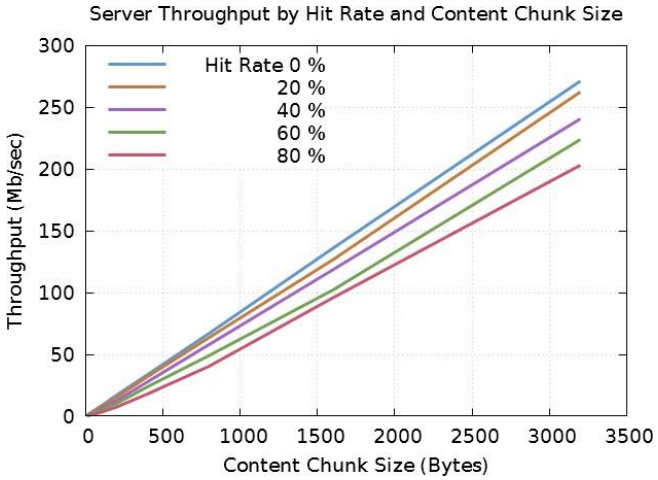

Fig. 6. Server throughput at different Cache hit rates and response sizes.

With a larger Cache capacity, more content can be stored and served directly from the Cache, resulting in a reduced burden on the server. As a result, the CDCA architecture demonstrates its ability to alleviate the server's load and improve its performance, especially when higher Cache hit rates are achieved. The CDCA architecture successfully compels caching to offload traffic from the server and optimize its performance, leading to more efficient content delivery and enhanced network scalability.

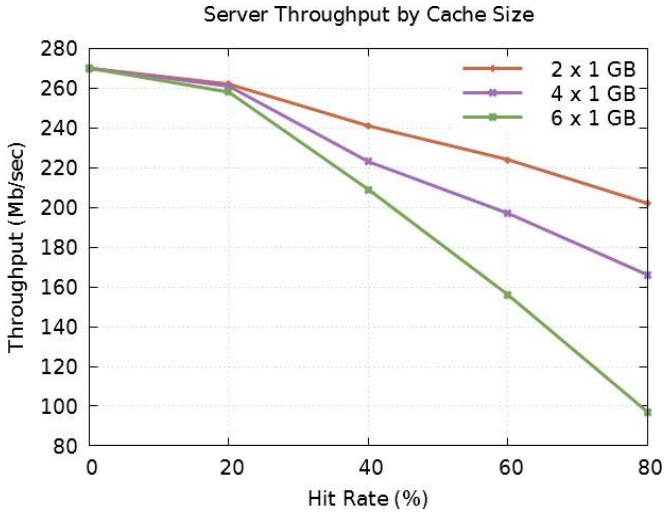

Fig. 7. Server throughput with different Cache hit rates and capacities.

The consistent improvement patterns observed in both users' QoE metrics and network metrics, regardless of the content size, are indeed interesting findings. The CDCA solution, by caching content near users at high speeds, effectively reduces transmission time and improves overall performance, regardless of the size of the content being delivered.

This result suggests that the CDCA architecture efficiently utilizes caching mechanisms to deliver content to users, irrespective of the content's size. The proximity of the cached content to the users, combined with the optimized delivery process, minimizes the impact of content size on transmission time. Consequently, users experience similar levels of improvement in QoE metrics, such as reduced delay and improved perceived performance, regardless of whether they are accessing small or large content.

Furthermore, improvements were also observed regarding server throughput and data transfer, reinforcing the effectiveness and validation of the CDCA architectural solution. By offloading traffic from the server and optimizing content delivery through caching, the architecture efficiently utilizes network resources, leading to reduced server load, decreased transferred bytes, and improved overall network operational efficiency while optimizing network performance and resource utilization.

C. Discussion of the Results

In all three experiments, we can notice that the CDCA architecture has been effective in reducing the user's perceived delay, reducing the network data transfer, and reducing the network traffic at the content provider's server. The improvement on the user's QoE is intrinsically connected to the network data transfer reduction since fewer network segments need to be traversed in order to serve the content requests. As a result, if several requests are being handled by the Cache nodes spread over the network, then fewer requests need to be forwarded to the external content server, reducing the amount of data exchanged.

Fig. 7 provides valuable insights into the impact of Cache capacity on the system's performance. As the Cache capacity increases, the hit rate improves, resulting in a higher proportion of content being served from the Cache instead of the content server. This reduces the load on the server and improves overall throughput. Conversely, when the Cache capacity is low, frequent evictions occur, leading to more requests being forwarded to the content server. This not only increases the load on the server but also decreases the overall throughput, as observed in the results.

The findings suggest that the Cache capacity should be carefully considered during system deployment. Insufficient Cache capacity may result in higher eviction rates and increased dependence on the content server, ultimately affecting user experience and server performance. It is important to allocate an appropriate amount of Cache storage to accommodate the expected workload and ensure efficient content caching.

Furthermore, implementing a policy that selectively caches specific types of content, such as popular videos or audio, can optimize Cache utilization and prevent waste of resources. By focusing caching efforts on high-demand content, the Cache capacity can be effectively utilized to serve the most frequently requested content, enhancing overall performance and reducing the strain on the system.

These insights highlight the importance of careful Cache capacity planning and policy management in real-world deployments. By considering the workload characteristics, content popularity, and resource constraints, ISPs can design caching solutions that maximize the benefits of caching while efficiently utilizing Cache resources.

As depicted in Fig. 4 and Fig. 7, bigger content chunk

sizes and Cache storage leads to better results, indicating that large caches improve the system's performance. Nowadays, storage capacity is not a considerable problem since the ever-increasing capacity and decreasing prices of RAM and Flash memory provide affordable storage for huge capacity data (or files) resulting in superior overall system performance. And also, the system performance could be influenced by the Cache eviction policy used.

An initial approach for the ontology taxonomy utilizes four categories: (1) Content Type, (2) Content Identification, (3) Content Location, and (4) Content Chunk.

Content Type considers the content media type. There are two main categories of multimedia: time-sensitive media, e.g., video and audio, and non-time-sensitive media, e.g., documents and software. In the CDCA architecture, the time-sensitive content has to be treated in a different way in order to maintain the seamless delivery rate and guarantee the user real-time QoE. The video and audio parameters, i.e., resolution and CODEC, are used to quantify the required media rate in order to set the network provision once the end-user file is selected, preferably through the ontology search and found mechanism.

Content Identification is used to identify the content name, version, date and hash. This information is important for checking the content version to ensure that the most updated content version is delivered to the end user.

Finally, the use of multiple SDN controllers, proxies and caches, provides scalability by growing infrastructure horizontally. The Proxy and Cache implementation using a stateless *microservice* framework helps to meet all the scalability requirements. However, further studies will be necessary to determine any limitations in a large-scale deployment.

VI. Open Research Issues, Challenges and Future R&D Directions

The CDCA architecture provides a transparent Cache system to improve the delivery of content objects inside an ISP infrastructure. Although outside the scope of this paper, there are still some open research issues that need to be considered before the CDCA system could be safely and practically deployed.

A. Ontology Issues

The engine should offer the end-user the ability to find any content in a topic-specific manner, within a very short response time. In a real-life service production environment, it is possible for the network provider to host millions of content objects. To find topic-specific content with the correct version and date, it is necessary to organize the content index in an effective and efficient way, which should be approached through the use of some kind of dynamically updating knowledge-based ontology and deep machine learning techniques [35].

Several works have investigated the theory and practice of the semantic web and CDCA ontologies to organize the content classification [36]. However, most of these proposals focused on classifying the content to help the end-user find specific information, e.g., sport, business, travel, and so on. Nevertheless, in the CDCA solution, this kind of classification alone is not very useful because, for resourceful Cache management, it is not relevant whether the video is about sport or about travel, because the resolution, transmission rate and content chunk size are far more critical and significant for the optimization, sustainability and scalability. This does not mean that the content sought by the end-user is not important, but the selected optimum delivery of cached parts is the main consideration. The twin objectives of optimal seeking and delivery are crucial in formulating ontology.

Content-Location is important for finding the best geographic content distribution location point in order to improve the content QoE and the load balancing criteria during transmission.

The Content Chunk defines how the content chunk is divided and organized to improve the overall delivery and system performance. The content chunk is an atomic particle and it is the main element in the design of the CDCA architecture. The definition of its length is important to satisfy optimum delivery and system performance.

In a content-based network, the same content may have different names due to various reasons such as alternate naming strategies, load balancing, content distribution strategies, or user location-based routing. This can result in duplicate copies of the same content being stored in caches, leading to wastage of memory and storage resources.

To address this issue and avoid unnecessary hosting and caching of duplicate content, a hash mechanism can be employed. By calculating a hash value based on the content's data, such as using hash functions like MD5 or SHA-1, it becomes possible to determine whether multiple requests refer to identical content or not. The hash value serves as a unique identifier for the content, regardless of its name or location.

When a request is received, the system can calculate the hash value of the requested content and compare it with the existing Cache entries. If a matching hash value is found, it indicates that the content is already cached, and there is no need to store another copy. Instead, the existing cached copy can be retrieved and served to the requesting user.

By using a hash mechanism, the CDCA architecture can effectively identify and eliminate duplicate copies of content, thereby optimizing memory and storage resources in caches. It ensures that only one copy of the content is stored, regardless of the different names or variations associated with it due to load balancing or other factors. This approach helps in reducing storage overhead and improving the overall efficiency of the content delivery system.

B. Optimization Issues

Although the CDCA architecture shows improvements in the response time of content requests and in the reduction of traffic from an external content provider, many aspects can be improved, as discussed in Section IV-C. We also envisage that it is possible to improve the content inspection algorithm to optimize searching and the application of a load-balancing mechanism. Although the system seems to

be scalable, further studies are necessary to determine any limitations in a large-scale system.

As noted earlier, the CDCA architecture utilized OpenFlow 1.0 because more recent versions do not offer any new field to improve the CDCA system. However, if a future version of the OpenFlow protocol implements matches in the HTTP Content-Type field, it will be able to forward certain specific MIME requests, for example, "video/mp4", direct to the Cache, avoiding the need for a Proxy. Even so, the content lookup should be done by the OpenFlow Controller instead of the Proxy. The lookup inside the switch diverges from the OpenFlow philosophy to maintain simplicity.

The issues mentioned in Section IV-C can be resolved by the configurable architecture. The configurable software approach means that the network administrator can deploy new VMs or adjust VM configuration, i.e., allocate memory and disk capacity when needed. This function can be accomplished by a cloud management and orchestration system. The CDCA prototype provides an open API that offers the capacity to change the eviction algorithm *on-the-fly*. The evaluation utilized the Least Frequently Used (LFU) eviction algorithm over RAM memory, as described in Section IV-D. In future research work it will compare the system behavior using various cache policies.

A critical issue, that should be investigated in the future, is the popularity prediction of User Generated Content (UGC). This is a valuable tool for content providers and advertisers. As the cached content is delivered inside the own ISP, the content provider could not get access to the user's profile. An interesting approach is proposed by Figueiredo et al [37].

It tackles the popularity prediction trend of a UGC object as early as possible to infer the user behavior. The results obtained by using YouTube datasets show an improvement of 38% in classification effectiveness, compared to the baseline approaches. Using this approach, the ISP can collect the user'sinformation and notify the content provider.

C. Security Issues

The CDCA system deployment within an ISP opens an opportunity with varying degrees of risk for external and internal security attacks of various kinds. It is possible that the CDCA critical infrastructure is susceptible to Distributed Denial of Service (DDoS) attacks affecting server provision and slowing down (or completely shutting down) the service, thereby frustrating the end user [38]. The distributed and open structure of a Cache system and its associated services can make it an attractive target for potential cyber-attacks. As with any system connected to the Internet, it is important to consider security measures to protect against intruders and mitigate the risks associated with cyber-attacks. These intruders can masquerade and manipulate various types of multimedia content, and therefore a supporting set of safety measures and security mechanisms and services would be needed to prevent intrusions and breaches. This would require a versatile, collaborative Intrusion Detection and Prevention System (IDPS) which must be flexible enough to guarantee smooth optimized streaming throughput flows with near-zero (minimum) glitches.

Given the openness, and transparent nature of the mix-mode multimedia content delivery caching SDN architecture, traditional IDPS mechanisms would be fundamentally inefficient and ineffective [39]. In particular, it would be extremely difficult to detect intrusions in transparent multimedia content, hence preventing subsequent intrusions without employing a smart IDPS. It must involve advanced machine learning and computational intelligence techniques and the use of the five fundamental principles of autonomic self-management computing, knowledge base and ontology, risk management, fuzzy theory, and advanced artificial intelligence techniques [40] to leverage and satisfy the detection and prevention security capabilities of a Cache system. By incorporating these advanced techniques and concepts, the Cache system can enhance its ability to detect and prevent cyber-attacks, improve threat response mechanisms, and optimize overall security operations [41] [42]. However, it's important to consider the specific requirements and constraints of the Cache system via proper risk assessment and adapt these techniques accordingly to achieve effective and efficient operational and security outcomes.

HTTP/2 includes encryption as an optional facility. It is not mandatory because encryption can result in unnecessary jitter and distortion of the smooth viewing flow of videos. Many experts consider encryption unfeasible for ordinary large run-of-the-mill content delivery of videos. However, when and where cryptography is required, we also observe other major safety measures (security, privacy, trust, ID management, Digital Rights Management (DRM), digital *blockchaining* (virtual currencies), audit, digital forensic, non-copyediting, copyright infringement, permission to use, royalties, payments, etc.), and in particular issues related to the hiding of secret information using steganography will have to go beyond HTTP/2 specification. The complexities of these sets of safety measures are best accommodated by a comprehensive ontology.

D. Copyright Infringement and Payment Issues

Another issue of importance and concern is that the CDCA architecture considers all content to be public without any restriction to distribution, which, however, makes the CDCA architecture unsuitable to deliver protected and paid content, such as required by some VoDs. Another example would be the rights of the content owner or their agent to be paidfor documentaries, films, or music. Such content should typically be encrypted and the VoD provider can provide a temporary digital security key to the subscriber to decrypt the content for a certain period of time based on a payment scheme and without allowing copying of the content for further illegal distribution. This can be potentially achieved by activating the signature timeout period in conjunction with monitoring if the user attempts to copy the content. After this expiry period, the key is invariably disallowed, requiring the subscriber to renew the key to access the content. Implementing these security and protection measures requires careful consideration of technical, legal, and business aspects. It involves collaboration between content providers, payment service providers, DRM vendors, and security experts to

design and deploy a comprehensive and effective solution. This suggested scenario requires intensive research to fully define and design a secure payment mechanism that also curtails illegal copying, as well as verify and validate the mechanism for business deployment.

VII. Conclusion and Future Work

This research has described a content-based transparent caching architecture in SDN. It provides a highly available, reliable and scalable caching of named content on SDN-based ISP networks, independent of specific underlying applications and middleware protocols. The research has also demonstrated that the caching mechanisms are driven by business policy needs and can be deployed in any networks, using the NFV approach and the microservice-based framework architecture. One notable aspect of the CDCA architecture is its support for the HTTP protocol, which remains the primary protocol for content delivery over the Internet. Rather than replacing HTTP, the system complements it by introducing transparent caching mechanisms that enhance content delivery and improve QoE for users.

The experimental evaluation conducted in the research demonstrates the effectiveness of the CDCA system. It shows improvements in user QoE and various QoS network metrics related to delivery times and scalability. This validation reinforces the benefits of the architecture and its potential to enhance content delivery in real-world network scenarios.

The CDCA architectural system has some important outstanding issues that should be addressed in future research and development work. At present, there is no effort by content providers and related industry players to develop a standardized naming scheme for content, which is crucial for efficient and optimum search and delivery of content, as well as for avoiding duplication of names and content hosted all throughout the provider network. In addition, the new naming scheme should avoid the same content with different names being downloaded multiple times. It is important for content providers and industry players to recognize the significance of a standardized naming scheme and work toward its development and adoption.

In future research work, the intent is to address many of the issues mentioned in the previous section. In addition, further research work is planned to perform analyses of several other video content providers, other than YouTube, to adapt the CDCA solution, if necessary. Another critical issue, which will be investigated, is the *popularity prediction* of UGC, which is a valuable tool for content providers and advertisers for revenue generation.

Finally, another area for future work involves system security. In the CDCA architecture, the ISP acts as a content provider, and it could suffer external and internal DDoS attacks, affecting servers slowing/shutting down the service and frustrating users. Further, multimedia content could be manipulated for illegal cybercrime activities, which should be avoided through the proper implementation of safety measures. Traditional IDPS is largely inefficient for the CDCA environment due to its architecture and virtualization. A new IDPS paradigm should be designed to achieve a high level of security health in the service provider network. In addition, we have highlighted issues related to secure payment and royalty awarding schemes for content that is primarily declared as public but that requires payment to intellectual property owners or their agents. Property rights issues also involve various safety measures. These system security challenges require creative solutions and therefore offer opportunities for further research.

Acknowledgment

I sincerely wish to thank Alex F R Trajano, Ahmed Patel, and Marcial P Fernandez for allowing the use of their experimental data and for their valuable guidance throughout this study and research work, including proofreading this paper.

References

[1] J. S. Turner and D. E. Taylor, "Diversifying the Internet," in *IEEE Global Telecommunications Conference (GLOBECOM 2005)*, vol. 2, IEEE. Institute of Electrical & Electronics Engineers (IEEE), Dec 2005, pp. 760–766.

[2] Cisco, "Cisco visual networking index: Forecast and methodology, 2016-2021 white paper," [Online]. Available: http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/complete-white-paper-c11-481360.html

[3] J. F. Gantz, D. Reinsel, C. Chute, W. Schlichting, J. Mcarthur, S. Minton, I. Xheneti, A. Toncheva, and A. Manfrediz, "The expanding digital universe: A forecast of worldwide information growth through," *Information and Data 2007*, pp. 1–21, 2010.

[4] B. Ahlgren, C. Dannewitz, C. Imbrenda, D. Kutscher, and B. Ohlman, "A survey of information-centric networking," *IEEE Communications Magazine*, vol. 50, no. 7, pp. 26–36, July 2012.

[5] L. Veltri, G. Morabito, S. Salsano, N. Blefari-Melazzi, and A. Detti, "Supporting information-centric functionality in software defined networks," in *IEEE International Conference on Communications (ICC2012)*. IEEE, 2012, pp. 6645–6650.

[6] A. Ooka, S. Ata, T. Koide, H. Shimonishi, and M. Murata, "Openflow-based content-centric networking architecture and router implementation," in *Future Network and Mobile Summit (FutureNetworkSummit 2013)*. IEEE, July 2013, pp. 1–10.

[7] X. N. Nguyen, D. Saucez, and T. Turletti, "Efficient caching in content-centric networks using openflow," in *Proceedings IEEE INFOCOM 2013*, April 2013, pp. 1–2.

[8] D. Syrivelis, G. Parisis, D. Trossen, P. Flegkas, V. Sourlas, T. Korakis, and L. Tassiulas, "Pursuing a software defined information-centric network," in *European Workshop on Software Defined Networking (EWSDN2012)*. IEEE, Oct 2012, pp. 103–108.

[9] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, Mar 2008.

[10] M. Ciosi *et al.*, "Network functions virtualisation: an introduction, benefits, enablers, challenges and call for action, introductory white paper," in *"SDN and OpenFlow World Congress"*, 2012, pp. 152–160.

[11] A. Chanda and C. Westphal, "A content management layer for software-defined information centric networks," in *Proceedings of the 3rd ACM SIGCOMM workshop on Information-centric networking*, ACM. ACM, 2013, pp. 47–48.

[12] P. Georgopoulos, M. Broadbent, B. Plattner, and N. Race, "Cache as a service: leveraging sdn to efficiently and transparently support video-on-demand on the last mile," in *23rd International Conference on Computer Communication and Networks (ICCCN2014)*. IEEE, Aug 2014, pp. 1–9.

[13] A. F. Trajano and M. P. Fernandez, "Two-phase load balancing of in-memory key-value storages using network functions virtualization

(nfv)," *Journal of Network and Computer Applications*, vol. 69, pp. 1–13, Jul 2016.

[14] D. Levin, A. Wundsam, B. Heller, N. Handigol, and A. Feldmann, "Logically centralized?: State distribution trade-offs in software defined networks," in *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*, ser. HotSDN '12, ACM. New York, NY, USA: ACM, 2012, pp. 1–6.

[15] M. P. Fernandez, "Comparing openflow Controller paradigms scalability: Reactive and proactive," in *IEEE 27th International Conference on Advanced Information Networking and Applications (AINA2013)*. IEEE, March 2013, pp. 1009–1016.

[16] F. Sultan, K. Srinivasan, D. Iyer, and L. Iftode, "Migratory tcp: Connection migration for service continuity in the internet," in *Distributed Computing Systems, 2002. Proceedings. 22nd International Conference on*. IEEE, 2002, pp. 469–470.

[17] S. Koulouzis, A. S. Belloum, M. T. Bubak, Z. Zhao, M. Živković, and C. T. de Laat, "SDN-aware federation of distributed data," *Future Generation Computer Systems*, vol. 56, pp. 64–76, March 2016.

[18] M. Belshe, R. Peon, and M. Thomson, "Hypertext Transfer Protocol Version 2 (HTTP/2)," RFC 7540 (Proposed Standard), Internet Engineering Task Force, May 2015.

[19] S. Wei and V. Swaminathan, "Low latency live video streaming over http 2.0," in *Proceedings of Network and Operating System Support on Digital Audio and Video Workshop*, ser. NOSSDAV '14. New York, NY, USA: ACM, 2014, pp. 37:37–37:42.

[20] S. Alcock and R. Nelson, "Application flow control in YouTube video streams," *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 2, p. 24, apr 2011.

[21] T. Hoßfeld, R. Schatz, and U. R. Krieger, "QoE of Youtube video streaming for current internet transport protocols," in *Measurement, Modelling, and Evaluation of Computing Systems and Dependability and Fault Tolerance*. Bamberg, Germany: Springer International Publishing, 2014, pp. 136–150.

[22] V. K. Adhikari, S. Jain, and Z.-L. Zhang, "Where do you"tube"? Uncovering youtube server selection strategy," in *Proceedings of 20th International Conference on Computer Communications and Networks (ICCCN2011)*. IEEE, 2011, pp. 1–6.

[23] A. Balamash and M. Krunz, "An overview of web caching replacement algorithms," *IEEE Communications Surveys & Tutorials*, vol. 6, no. 2, pp. 44–56, 2004.

[24] J. Wang, "A survey of web caching schemes for the internet," *ACM SIGCOMM Computer Communication Review*, vol. 29, no. 5, pp. 36–46, 1999.

[25] S. Gao, Y. Zeng, H. Luo, and H. Zhang, "Scalable control plane for intra-domain communication in software defined information centric networking," *Future Generation Computer Systems*, vol. 56, pp. 110 – 120, March 2016.

[26] K. Phemius, M. Bouet, and J. Leguay, "Disco: Distributed multi-domain sdn controllers," in *IEEE Network Operations and Management Symposium (NOMS2014)*. IEEE, May 2014, pp. 1–4.

[27] A. Dixit, F. Hao, S. Mukherjee, T. Lakshman, and R. Kompella, "Towards an elastic distributed sdn Controller," *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 4, pp. 7–12, 2013.

[28] P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz, B. O'Connor, P. Radoslavov, W. Snow *et al.*, "Onos: towards an open, distributed sdn os," in *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*, ser. HotSDN '14, ACM. New York, NY, USA: ACM, 2014, pp. 1–6.

[29] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," *SIGCOMM Comput. Commun. Rev.*, vol. 31, no. 4, pp. 149–160, Aug. 2001.

[30] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: rapid prototyping for software-defined networks," in *Proceedings of the Ninth ACM SIGCOMM Workshop on Hot Topics in Networks*, ser. Hotnets '10. New York, NY, USA: ACM, 2010, pp. 19:1–19:6.

[31] D. Erickson, "Floodlight Java based OpenFlow Controller," [Online]. Available: http://floodlight.openflowhub.org/

[32] B. Augustin and A. Mellouk, "On Traffic Patterns of HTTP Applications," in *IEEE Global Telecommunications Conference (GLOBECOM 2011)*, Dec 2011, pp. 1–6.

[33] N. L. M. Van Adrichem, C. Doerr, and F. Kuipers, "Opennetmon: Network monitoring in OpenFlow Software-Defined Networks," in *IEEE Network Operations and Management Symposium (NOMS'2014)*. IEEE,May 2014, pp. 1–8.

[34] J. Famaey, F. Iterbeke, T. Wauters, and F. De Turck, "Towards a predictive Cache replacement strategy for multimedia content," *Journal of Network and Computer Applications*, vol. 36, no. 1, pp. 219–227, 2013.

[35] N. Talpur, S. J. Abdulkadir, H. Alhussian, M. H. Hasan, N. Aziz, A. Bamhdi. A comprehensive review of deep neuro-fuzzy system architectures and their optimization methods. Neural Comput & Applic 34, 1837–1875 (2022). https://doi.org/10.1007/s00521-021-06807-9.

[36] S. Dumais and H. Chen, "Hierarchical classification of web content," in *Proceedings of the 23rd annual international ACM SIGIR conference on Research and development in information retrieval*, ser. SIGIR '00, ACM. New York, NY, USA: ACM, 2000, pp. 256–263.

[37] F. Figueiredo, J. M. Almeida, M. A. Gonçalves, and F. Benevenuto, "TrendLearner: Early prediction of popularity trends of user generated content," *Information Sciences*, vol. 349-350, pp. 172–187, July 2016.

[38] Tiago Linhares, Ahmed Patel, Ana Luiza Barros and Marcial Fernandez. 2023. SDNTruth: Innovative DDoS Detection Scheme for Software-Defined Networks (SDN). Journal of Network and Systems Management, (2023) 31:55, (online) https://doi.org/10.1007/s10922-023-09741-4

[39] A. Patel, M. Taghavi, K. Bakhtiyari, and J. C. Júnior, "An intrusion detection and prevention system in cloud computing: A systematic review," *Journal of Network and Computer Applications*, vol. 36, no. 1, pp. 25–41, Jan 2013.

[40] N. Talpur, S. J. Abdulkadir, H. Alhussian, M. H. Hasan, N. Aziz, A. Bamhdi. "Deep Neuro-Fuzzy System application trends, challenges, and future perspectives: a systematic survey." Artificial Intelligence Review. 13:1-49. (2023). https://doi.org/10.1007/s10462-022-10188-3.

[41] A. Patel, H. Alhussian, J. M. Pedersen, B. Bounabat, J. C. Júnior, and S. Katsikas, "A nifty collaborative intrusion detection and prevention architecture for smart grid ecosystems," *Computers & Security*, vol. 64, pp. 92–109, January 2017

[42] A. M. Bamhdi. FLORA: Fuzzy Logic - Objective Risk Analysis for Intrusion Detection and Prevention IJCSNS International Journal of Computer Science and Network Security, VOL.23 No.5, pp.179-192 May 2023. https://doi.org/10.22937/IJCSNS.2023.23.5.20

[43] Leyva-Mayorga, Israel, et al. "Network-coded cooperation and multi-connectivity for massive content delivery." IEEE Access 8 (2020): pp 15656-15672.

[44] Cisco Annual Internet Report (2018–2023) White Paper. Available: https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html

[45] Hong, Dohy, Danny De Vleeschauwer, and Francois Baccelli. "A chunk-based caching algorithm for streaming video." NET-COOP 2010-4th Workshop on Network Control and Optimization. 2010.