Software-Defined Multimedia Streaming System Aided By Variable-Length Interval In-Network Caching

Jian Yang, *Senior Member, IEEE*, Zhen Yao, Bowen Yang, Xiaobin Tan, Zilei Wang, *Member, IEEE*, and Quan Zheng

***Abstract*—Explosive growth in video traffic volumes incurs a high percentage of redundancy in today's Internet, following the 80–20 rule. Fortunately, the advanced in-network cache is considered as an effective scheme for eliminating the repetitive traffic by caching the popular content in network nodes. Besides, the emerging software-defined networking (SDN) enables centralized control and management, as well as the collaboration between network devices and upper applications. Moreover, the Network Functions Virtualization is also developed to support for customized network functions, including caching and streaming. This inspires us to design an SDN-assisted multimedia streaming Video-on-Demand system, integrating in-network cache, to improve the quality of service. The designed architecture is capable of reducing the redundant traffic via the reusable duplications. In particular, it can achieve greater performance gains by deploying specific scheduling policy. We further propose a variable-length interval cache strategy for RTP streaming, which can realize the self-adaptive adjustment of the size of cached video segments based on their access patterns. Our goal is to efficiently utilize the limited storage resources and increase the cache hit ratio. We present the theoretical analysis to demonstrate the attainable performance of the proposed algorithm; furthermore, the integrated system design is implemented as a prototype to show its feasibility and applicability. Ultimately, emulation experiments are conducted to evaluate the achievable performance improvement more comprehensively.**

***Index Terms*—Software defined networking, in-network cache, multimedia streaming system, OpenFlow, Video-on-Demand.**

I. Introduction

VIDEO services occupy a substantial percentage of today's network traffic. According to the forecast from Cisco [1], IP video traffic will be 82% of all consumer Internet traffic by 2021, up from 73% in 2016. And every second, nearly a million minutes of video content will cross the network by that time. Such video demand can quickly saturate these networks, and then lead to delay, jitter, as well as packet loss when providing video services, thus making it challenging to guarantee end users acceptable Quality of Service (QoS) and Quality of Experience (QoE).

According to the studies on understanding user behavior of video services [2], [3], the video popularity exhibits skewness, which well follows the Pareto Principle [4], specifically, 20% of the video objects account for 80% of all accesses. This fact implies that the popular video objects have higher frequency of accesses. Therefore, if unicast is adopted to stream the video, considerable redundant traffic will be incurred due to the repetitive accesses to the same videos. Although IP multicast is capable of forwarding data to a group of the interested receivers in a single transmission, the data reception of the multicast members are synchronous, and it does not support personalized service such as Video-on-Demand (VoD). In conclusion, the multicast is not beneficial for reducing video traffic redundancy over the time domain. The root reason for the incompetence of the current network to deal with the video traffic redundancy lies in its end-to-end principle designed, which keeps the core network devices (routers) simplified and optimized to only forward data packets, while moving the intelligence as much as possible to the terminal nodes.

To solve this problem, application-level innovation is introduced to improve the efficiency of the video content delivery. Content Delivery Network (CDN) [5] has been deployed upon traditional network protocol stack, aiming for delivering content to end-users with high availability and high performance. It relies on the content duplication from origin server to the proxy servers distributed across the network, and enables fast and reliable delivery by retrieving content locally to provide nearby service. There are many well-known CDN service providers like Akamai [6] and Limelight [7], devoted to hosting the content of third-party content providers in their servers, which are further replicated on several servers spread over the world, and then transparently redirect the customers' requests to the "best replica" [8]. However, CDN is a logical overlay network that sparsely deploys proxy servers, and it is not beneficial for reducing the redundant video traffic between the end-users and the chosen proxy server. A worse case is that if a CDN server

Manuscript received May 29, 2017; revised October 31, 2018 and January 9, 2018; accepted July 9, 2018. Date of publication August 1, 2018; date of current version January 24, 2019. This work was supported in part by the Equipment Preliminary R&D Project 6141B0801010a, in part by the National Natural Science Foundation of China under Grant 61573329, in part by the State Key Program of National NSF of China under Grant 61233003, and in part by the Youth Innovation Promotion Association CAS. The associate editor coordinating the review of this manuscript and approving it for publication was Prof. Christian Timmerer. *(Corresponding author: Jian Yang.)*

The authors are with the School of Information Science and Technology, University of Science and Technology of China, Hefei 230027, China (e-mail: jianyang@ustc.edu.cn; yaozhen1@mail.ustc.edu.cn; ybw92@mail.ustc.edu.cn; xbtan@ustc.edu.cn; zlwang@ustc.edu.cn; qzheng@ustc.edu.cn).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TMM.2018.2862349

lies outside the Autonomous System (AS), a large number of incoming cross-domain redundant traffic may be induced, since each streaming session corresponding to the end user is treated individually.

Recently, network landscape is dramatically changing so as to support intelligent and dynamic nature of future network and applications. Network Functions Virtualization (NFV) [9] is being widely studied and developed in recent years, which makes it possible to customize and deploy new functions on the core network devices. Different from the conventional routers in IP networks, the network node in the context of NFV supports content cache such as SmartRE [10], which is referred to as in-network cache [11]. Naturally, in-network cache is integrated on network nodes instead of the storage servers such as CDNs, thus capable of caching or delivering content at the network layer, and further bringing the content as close as possible to the users. Consequently, it succeeds in reducing the redundant traffic inside the edge network, and avoids redirecting too many requests to the proxy servers outside the ISP's network.

Although NFV reduces equipment costs and decreases the time to market while attaining scalability, elasticity, and agility, there are numbers of challenges to migrate to NFV over today's static inflexible network architectures. Software defined networking (SDN) [12] provides a scalable, elastic and on-demand framework well suited to the dynamic NFV networking requirements for both virtual and physical networking infrastructure [13]. OpenFlow [14] is the first platform implementing the SDN principles. It provides an interface to enable interaction between the control plane and the data plane. In this paper, we utilize the network knowledge obtained from the Openflow Controller in the context of SDN-enabled NFV, in order to design a software-defined VoD system with the aid of in-network cache for achieving an efficient video content delivery.

A. Contributions

In this paper, we propose a SDN-based multimedia streaming system for in-network cache architecture, achieving video delivery from the network node closest to end users, thus reducing transmission delay and redundant traffic, to improve QoS/QoE. The main contributions are listed as follows:

- Relying on the SDN architecture, we design a software-defined video streaming system framework equipped with in-network cache. A cache strategy module is abstracted for optimizing the content cache of the VoD service. This open architecture enables the service provider to deploy advanced cache strategies flexibly and agilely.
- We propose a variable-length interval cache strategy for the in-network cache architecture, capable of: a) Adaptively adjusting the length of the video segments cached following with the trend of the popularity to increase cache hit ratio; b) Aggregating the repetitive video requests over the time domain to reduce the redundant video traffic; c) Dynamically synchronizing the sliding cache window with playback to release storage space for improving cache utility. Consequently, the storage resource of the global network can be allocated more appropriately and efficiently.
- We present the theoretical analysis and emulation results to demonstrate the achievable performance improvement. Furthermore, a prototype system is built to show the feasibility and applicability of the proposed solution.

II. Related Work

A. NFV Based CDN and In-Network Cache

There are plenty of contributions demonstrating that NFV is beneficial for improving CDN. For instance, the works [15], [16] both conceived respective frameworks for CDN scenario by enabling the virtual nodes or applications based on NFV, to achieve an efficient content distribution with a low overall cost for CDN providers. The authors of [17] also presented a new optimization technique for cache distribution with underlying virtualization tools, and an intelligent migration algorithm for virtual content delivery functions. The work [18] designed an effective multimedia transmission by using OpenStack, so as to construct NFV for the sake of providing the services of high quality to users through CDN. Akamai Technologies and Juniper Networks have also proposed an elastic commercial CDN solution by leveraging SDN and NFV to efficiently utilize available network resources in [19]. Despite the achievements made by these contributions, the redundant traffic between the end-users and the proxy servers still has not been eliminated.

Since in-network cache is considered able to further reduce the repetitive traffic, it has garnered lots of attention from academia. In [20], the concept of resource management was introduced for in-network caching environments. A probabilistic in-network caching method for information-centric networking (ICN) was proposed, for reducing caching redundancy and making efficient utilization of available cache resources along a content delivery path. The work [21] conceived a solution leveraging the in-network cache capacity of Content Centric Network/Named Data Networking (CCN/NDN), to distribute and pre-cache the video segments encapsulated by MPEG-Dynamic Adaptive Streaming over HTTP (MPEG-DASH) according to a specific proactive content caching scheme, thus achieving higher quality and reliability of video delivery without interruption for mobile users. In [22], the optimal content assignment for two in-network caching policies, *i.e.*, Single-Path Caching and Network-Wide Caching was proposed for NDN. These works already show that in-network cache achieves significant performance improvement over conventional cache frameworks based on proxy servers, while it is hard to realize an efficient global management in traditional networks, that is why SDN is introduced in the following.

B. SDN Assisted Video Streaming and Cache Management

Several solutions utilized SDN to construct video streaming systems. The contribution of [23] investigated the realization of the OpenFlow-controlled network orchestration for facilitating efficient scalable video coding (SVC) streaming to heterogeneous clients. An SDN based CDN and ISP collaboration architecture was proposed in [24] for managing the high volume and long living flows such as video distribution. In [25], a distributed

OpenFlow-based QoS architecture was designed for optimizing end-to-end QoS provision over multi-domain SDNs. In our previous work [26], we considered the joint admission control and routing optimization for video streaming service in the context of software-defined networking. However, these solutions hardly concerned about the beneficial cache capability.

Integrating in-network caching with SDN is one of the hot spots in current research. The work [27] presented a solution for integrating the caching functionality in a LTE network based on SDN technology. In [28], an architecture named ContentSDN was presented, which combined ICN with SDN together, and provided transparent in-network caching for content delivery. In [29], an efficient, transparent and highly configurable OpenFlow-assisted in-network caching service named OpenCache was proposed for improving the distribution efficiency. However, the OpenCache Node conceived in [29] is an independent server deployed on the edge of the network, and the request scheduling is based on redirection with the assistance of the OpenFlow controller, which is similar to the mechanism of CDN. By contrast, the in-network cache we present in this paper is integrated on the OpenFlow Switches, so that the transmission of video streams need not follow the end-to-end principle.

C. Cache Replacement Strategies

Many classical content replacement algorithms have been designed for conventional caching systems. For instance, simple static policies such as Least Recently Used (LRU) and Least Frequently Used (LFU) are widely used in practice for their simplicities and ease of implementation [22], [30] and [31]. Besides, the contributions [32] and [33] both proposed online learning cache replacement methods respectively, which relyed on learning the video popularity over time and forecasting its trend. Despite the performance improvement achieved, all the contributions above are based on popularity prediction that may be inaccurate, and insist on caching the entire objects.

Hence, in order to improve the cache utilization, many studies focused on caching video segments. In [34], the author took into account the information of the recent user behavior to predict future segment requests, and further proposed two strategies with a different level of coordination between the cached video segments in the network, to increase the cache hit ratio and reduce bandwidth consumption. The authors of [35] extended the basic interval caching policy [36] for handling mixed workloads consisting of long movies and short interactive clips in a multimedia server system. Motivated by this work, we tailor the variable-length interval cache strategy for software-defined video streaming system with in-network cache, aiming for substantially reducing the redundant video traffic.

The rest of this paper is organized as follows. We describe the software-defined multimedia streaming architecture with variable-length in-network cache in Section III. Section IV presents the variable-length interval cache scheme for the proposed software-defined multimedia system. In Section V, we give the theoretical analysis on the performance of the proposed cache strategy. In order to demonstrate the feasibility and applicability, we design and implement a prototype system for the proposed solution in Section VI. Furthermore, emulation experiments are conducted in Section VII to comprehensively investigate the achievable performance of the proposed strategy. Finally, we draw a conclusion in Section VIII. The main notations used in variable-length interval cache strategy can be found at TABLE I.

TABLE I
NOTATIONS USED IN VARIABLE-LENGTH INTERVAL CACHE STRATEGY

Notation	Definition
T	Initial time span of the cache window
$T^{'}$	Updated time span of the cache window
T_{max}	Upper bound of the size of cache window
n	Number of requests have arrived in the cache window
C_i	The i-th video object
N	Number of videos available in Streaming Media Server
$W_{i,j}$	The j-th cache window of the i-th video object
$K_{i,j}$	Total number of the requests in the window $W_{i,j}$
L_i	Total number of cache windows for the video object C_i
$R^k_{i,j}$	The k-th request to the video object i in the window $W_{i,j}$
$T^k_{i,j}$	Arrival time of $R^k_{i,j}$
$R^n_{i,j}$	The last request arrived
$I^k_{i,j}$	The k-th interarrival time between two consecutive request arrivals for video object i in cache window $W_{i,j}$
$I^n_{i,j}$	The interval between the last request arrival and the end of the time span for video object i in cache window $W_{i,j}$
$\overline{I}_{i,j}$	Average inter-arrival time of the cache window $W_{i,j}$

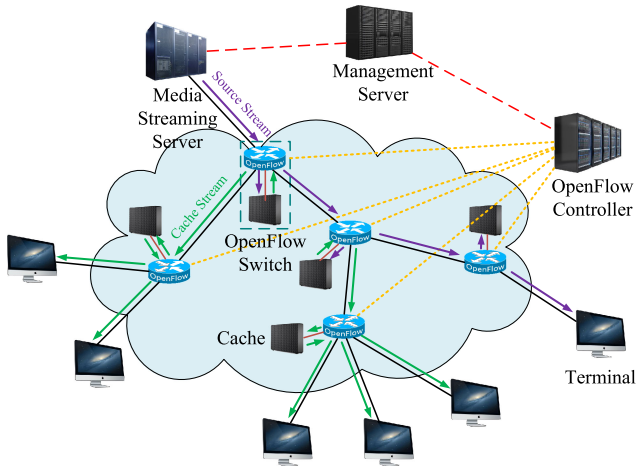

Fig. 1. Software-defined multimedia streaming system with in-network cache.

III. SOFTWARE-DEFINED MULTIMEDIA STREAMING ARCHITECTURE WITH IN-NETWORK CACHE

This section aims for presenting the overview of the software-defined multimedia streaming architecture with variable-length in-network cache.

A. System Architecture

Fig. 1 shows the conceived software-defined multimedia streaming system, which is mainly designed for the VoD scenarios that content providers deploy video streaming services in ISP's small-scale edge/access networks. The system is primarily

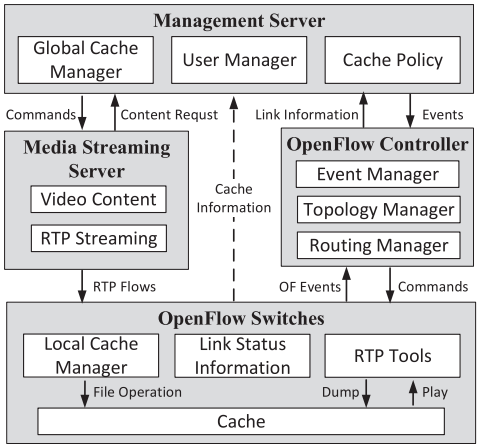

Fig. 2. Architecture of software-defined multimedia streaming system with in-network cache.

composed of OpenFlow Controller, Media Streaming Server, Management Server, OpenFlow Switches, and Terminals. As the provider of the video content, the Media Streaming Server has all available video contents, and a media application is deployed on it to receive video requests and generate RTP video stream for each session. Correspondingly, a media player operates on the terminal to trigger requests, receive and play back the video streams. The Management Server acts as the brain of the system, utilizing the topology, cache distribution and link-state information obtained from the OpenFlow Switches to run specific cache strategies. The OpenFlow Controller is responsible for controlling the behaviors of the OpenFlow Switches according to the intention of the Management Server. The Cache called in-network cache is integrated into the OpenFlow Switch rather than a proxy server, which is different from the traditional network. It means that the OpenFlow Switch is capable of both file management and content delivery. The SDN architecture is leveraged for its centralized control and management, ability of information collection and status monitoring, as well as the collaboration enabled between network nodes and the upper applications.

The system components are depicted in the architecture diagram of Fig. 2. Media Streaming Server has a fundamental function of providing VoD services, including video content management and RTP streaming. Management Server acts as a bridge between data plane and control plane, since that it is capable of instructing the data caching and video streaming according to the specific cache strategy, *i.e.*, the variable-length interval cache strategy described below. Specifically, it contains Global Cache Manager, User Manager, and Cache Policy. The Global Cache Manager maintains the information of content cached in distributed OpenFlow Switches. User Manager has the capability of Authentication, Authorization and Accounting (AAA) for a complete access service, while the abstracted Cache Policy makes decisions based on the distribution information of cached content to improve the utilization of cache. The OpenFlow Controller is made up of several modules including Topology Manager, Routing Manager, and Event Manager. The Topology Manager component assists the collection of network status information, and the other two enable the Management Server to configure the OpenFlow Switches for forwarding, caching, as well as streaming. All the OpenFlow Switches are equipped with storage, thus having the abilities of local cache management, link state monitoring, and RTP data processing. The Local Cache Manager in OpenFlow Switches supports basic file operations. Besides, the information of links (topology and bandwidth), as well as caches (status and distribution) provided by Local Cache Manager are both generated and raised as events periodically to OpenFlow Controller, then submitted to Management Server through TCP socket. RTP Tools [37] operate as a network function of caching RTP data, or reading corresponding output files and sending RTP streams to the Terminals. In order to enable OpenFlow Controller to realize the function of in-network cache management and content delivery, we extend OpenFlow protocol by defining new events related to the storage management and video streaming respectively.

Fig. 1 also depicts the video transmission aggregation relying on the in-network cache for reducing redundant traffic. OpenFlow Switches of the delivery path have a chance to duplicate and cache the corresponding RTP data packets. Thus, any request from other users for the same video can be served by the switches with cache rather than by the Media Streaming Server. Following this principle, the videos may spread all over the network. Hence, an effective cache management dedicated for video streaming over SDN is highly desired for the sake of efficient cache utilization. In this paper, we will adopt interval caching technique to develop in-network cache for streaming video over SDN, rather than the cache techniques for caching entire video objects.

B. Variable-Length Interval Caching Technique

Considering that the RTP Tools in OpenFlow Switches have the capability of identifying, printing, parsing and streaming for fine-grained RTP data, and the attainable timestamp information can also be used to measure the length of cached video segments, which can be arbitrarily specified by assigning corresponding parameters. So it inspires us to implement the proposed variable-length interval caching in a scenario of RTP streaming.

As explained in the above subsection, the cache in OpenFlow switches is controlled by Global Cache Management in Management Server. Once an OpenFlow Switch is instructed by Global Cache Management to cache a specific video stream, a local file is created to host RTP data packets, which contains a basic unit of the cache referred to as variable-length interval window as illustrated in Fig. 3. Specifically, the whole video file is considered as a time axis with fixed length, while the portion cached by a switch is a window of a certain length sliding over the time axis of the video. Since the granularity for window allocation is the Group of Pictures (GOP), of which the length is negligible compared to the window's size in general, so we neglect the influence caused by video encoding, and reasonably assume that the length of the window merely depends on the arrival pattern of the video requests, which is referred to as the probability distribution of their arrival times. In Section-IV, we will discuss the strategy for forming the cache window. Naturally, the video requests in the same window reuse a single stream from the upstream OpenFlow switch, thus aggregating

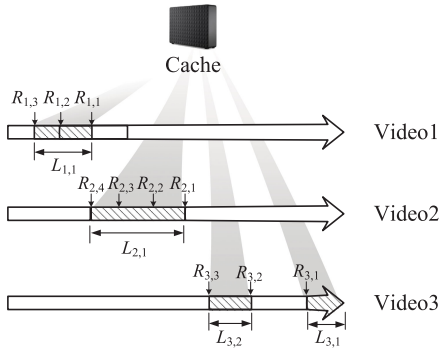

Fig. 3. Illustration of variable-length interval caching.

the redundant video traffic over time domain. It should be noted that the interval cache technique only caches the video traffic within the cache window.

Since Global Cache Manager in Management Server controls the cache in the OpenFlow switches, it dynamically maintains the data structure of the storage organization based on linked lists and STL maps as shown in Fig. 4, which contains the mapping relationships between video objects and windows, as well as windows and requests. The notations in Fig. 4 are given as follows. The i-th video object is denoted by C_i ($i = 1, \ldots, N$), where N is the number of video objects available in the Streaming Media Server. The j-th cache window of the i-th video object is represented by $W_{i,j}$ ($j = 1, 2, \ldots, L_i$), where L_i is the total number of cache windows for the video object C_i. $R^k_{i,j}$ and $T^k_{i,j}$ respectively denote the k-th request to the video object i in the window $W_{i,j}$ and its arrival time. As depicted in Fig. 4, the video object list consists of video objects C_i ($i = 1, \ldots, N$), and a video object C_i is associated with a window list consisting of windows $W_{i,j}$, which may be associated with different switches. A window $W_{i,j}$ contains a request list with requests $R^k_{i,j}$ ($k = 1, 2, \ldots, K_{i,j}$), where $K_{i,j}$ is the total number of the requests in the window $W_{i,j}$. As shown in Fig. 4, each window keep a record of Window ID, Content ID, Switch Datapath ID (DPID), etc. The Switch DPID assigned to the window is to indicate which switch it belongs to. In addition, Start Time, Length, Number of Requests and Average Interarrival Time are the attributes of the window object.

C. Basic Operating Process

The basic operating process of the system depicted in Fig. 1 is described as follows. When a user triggers a request to the Media Streaming Server, the server parses the received message, and extracts user's IP address, video ID, as well as the arrival time, which are further submitted to the Management Server. The User Manager Module in the Management Server authenticates this user based on User ID associated with IP address, and authorize the access to the video services. For the authorized user, Management Server utilize link status provided by OpenFlow Controller as well as the cache distribution provided by Global Cache Manager to determine the content delivery path for the user. A simplest way to determine the delivery path is applying the shortest-path method to achieve the nearest delivery of the video content as discussed in Section IV-C.

Once the delivery path is determined, the Manager Server maps all actions into commands executable in the OpenFlow Switches, which are configured by OpenFlow Controller, and then updates the cache information maintained in Global Cache Manager. The OpenFlow Switch which receives the command of providing the corresponding video content starts a process to transmit the RTP packets to the user through RTP Tools, except for the case of cache miss, where Media Streaming Server will still deliver the video since there is no better cache window available in OpenFlow Switches. If a switch on the delivery path has not cached the video content in its external memory space, it receives a command triggered by the Global Cache Manager to create a new window to host the served request, and caches the RTP packets delivered from the upstream switch also by utilizing RTP Tools.

IV. Variable-Length Interval Window Based In-Network Cache Strategy

In this section, we present a variable-length interval window based in-network cache strategy for the proposed system.

A. State Machine for Managing Cache Window

For a cache window with the initial time span T as shown in Fig. 5 (in *Building State*), assume that n requests have arrived, and $R^n_{i,j}$ denotes the last one. The k-th interarrival time between two consecutive request arrivals for the video object i in the cache window $W_{i,j}$ is denoted by $I^k_{i,j}$, which is defined as

$$I^k_{i,j} = \begin{cases} \infty, & \text{if } k = n = 1 \\ T^{k+1}_{i,j} - T^k_{i,j}, & \text{if } 1 < k < n \\ T + T^1_{i,j} - T^n_{i,j}, & \text{if } k = n > 1. \end{cases} \quad (1)$$

Note that if $n = 1$, the interval $I^1_{i,j}$ is assigned with ∞ because there is no interval. The interval $I^n_{i,j}$ between the last request arrival and the end of the time span is referred to as waiting time, since it waits for the next request. In addition, we define $\overline{I}_{i,j}$ as the average inter-arrival time of the cache window $W_{i,j}$, which is calculated as below:

$$\overline{I}_{i,j} = \begin{cases} \infty, & \text{if } n = 1 \\ \sum_{k=1}^{n-1} I^k_{i,j}/(n-1), & \text{if } n > 1. \end{cases} \quad (2)$$

If there is only one request arrival within time span T, *i.e.*, $n = 1$, average inter-arrival time does not exist, and thus we set it as ∞. Naturally, smaller average inter-arrival time of a video object implies more frequent access to this video. Therefore, inter-arrival time series or average inter-arrival time characterize the popularity of video objects and the user request pattern at current time.

In the proposed strategy, state machine pattern is used to describe and manage the lifetime of a cache window. As shown in Fig. 5, we define three cache window states, *i.e.*,

1) *Building State:* Upon arrival of a request, if there are no cache windows containing the requested video segment, which means the request has no suitable window to join, then an initial cache window with initial length of T is created. Its size may be adjusted to accommodate less or more requests before it is frozen. Here, "frozen" implies

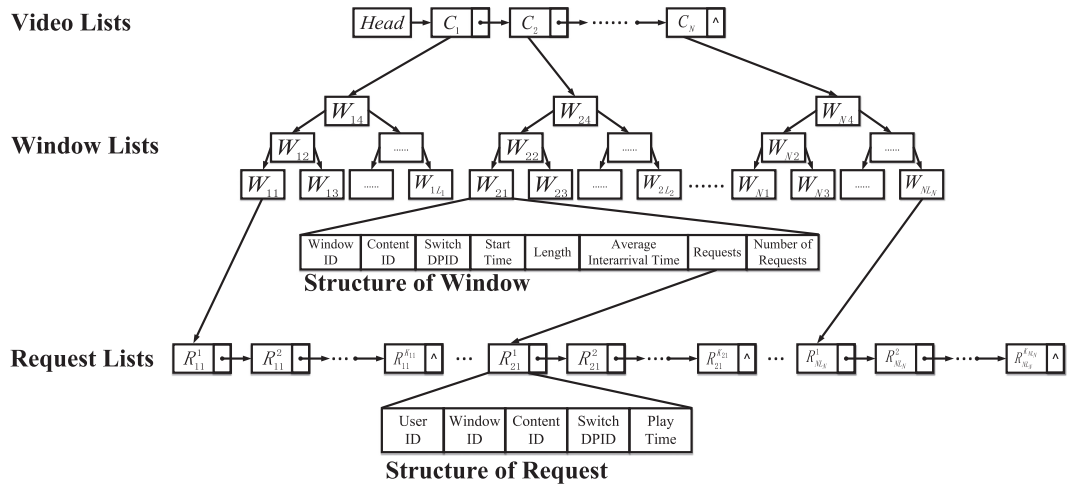

Fig. 4. Data structure of the storage organization.

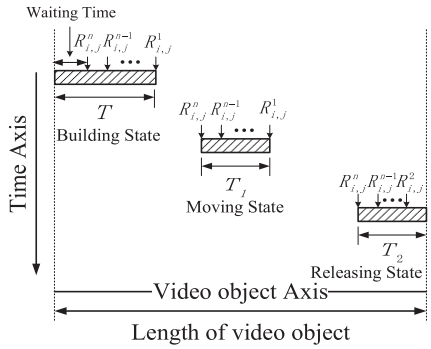

Fig. 5. Illustration of cache window $W_{i,j}$ in three states.

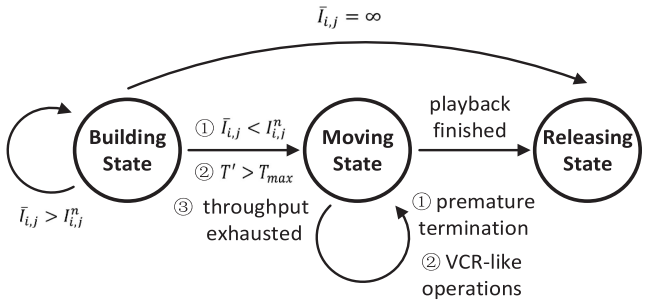

Fig. 6. State transition of cache window and corresponding conditions.

no more new arrival requests of the video object corresponding to the cache window will be accepted. Before the cache window is frozen, the state of the cache window is defined as *Building State*. If no cache window exists for a requested object, a first cache window with a subscript $j = 1$ is allocated. Subsequent cache window creations use monotonically increasing j.

2) *Moving State:* When the cache window is frozen, it enters *Moving State*. During this state, the cache window moves along the time axis of the video object with the streaming sessions corresponding to their requests. It should be noted that the size of cache window in *Moving State* may be adjusted only when premature termination or VCR-like operation (pause, jump forward, jump backward) [38] happens (see Section IV-B).
3) *Releasing State:* When the video session corresponding to the first request within the cache window reaches the end of the video stream, the cache window enters *Releasing State*.

B. Variable-Length Interval In-Network Cache Strategy

All the requests within the cache window are dispatched to the same OpenFlow Switch for reuse a same data block from cache, thus reducing the redundant the network traffic. When a user request of the object i is arriving, the proposed cache strategy operates as follows:

1) If the cache windows which contain the requested video segment are in *Building State* and the OpenFlow Switches corresponding to them have available bandwidth for output the video traffic, then the request is scheduled to the optimal OpenFlow switch among them to achieve the shortest delivery path. A new cache window with a initial size T is created for each OpenFlow Switch on the delivery path, which has no cache window.
2) Otherwise, the request will be scheduled to the Media Streaming Server to start a new RTP session. Furthermore, a new cache window of length T is created for each OpenFlow Switch on the delivery path.

Below, we discuss on adjusting the size of a cache window. When a cache window is created, it is assigned with an initial size of T, and begins to go through a three-state life-cycle. In these states, the cache window adjusts its size according to the rules below, which is illustrated by a state transition diagram Fig. 6.

- *Building State:* When the cache window $W_{i,j}$ is in *Building State*, then at the end of its current time span, a decision is made to adjust its size as:
  1) If $\bar{I}_{i,j} = \infty$, which implies only one request arrival for the video object i within the cache window $W_{i,j}$. The cache window is released soon and formed with a size of 0.
  2) If the waiting time $I^n_{i,j} > \bar{I}_{i,j}$ which implies that the subsequent request for the video object i becomes less

frequent, the initial cache window is shrunk to the arrival time of the most recent request at the end of the time span. Then, the cache window is frozen and enters *Moving State*.

3) If $I_{i,j}^n \leq \bar{I}_{i,j}$, the cache window keeps the *Building State*, and its size grows to the length $T^{'}$, where $T^{'} = T - I_{i,j}^n + \bar{I}_{i,j}$, expecting that a new request arrives very soon. At the end of the time span $T^{'}$, the average inter-arrival time $\bar{I}_{i,j}$ and the waiting time $I_{i,j}^n$ are recalculated and the above steps are repeated. Finally, the cache window will enter *Moving state* if one of the situations below happens: a) The request for the object becomes less frequent as 2); b) The size of cache window reaches the upper bound T_{max}; c) The throughput of the OpenFlow switch is exhausted. Applying such scheme, the requests for most frequently accessed objects can be served by a same block from the cache within the switch.

- *Moving State:* Once a cache window enters *Moving State*, the subsequent requests cannot be accommodated by this cache window. In this case, a new cache window with an initial size T is created when a new request arrives, and goes through its life-cycle starting from *Building State*. The only two special cases, *i.e.*, premature termination and VCR-like operations, may lead to the size adjustment of the cache window in *Moving State*. The rules for such adjustments will be described in this section.
- *Releasing State:* When a cache window enters *Releasing State* and a request reaches its end, the cache window is shrunk to the ending request's subsequence and be destroyed if all streaming sessions corresponding to the window are ended.

To prevent an extremely large cache window size, the cache window size is bounded by an upper bound T_{max}.

The session terminations also possibly result in the adjustment of the cache window size. There exist two different types of session terminations, *i.e.*, complete session termination and premature session termination. The complete session termination, in which the session is released at the end of the video stream, can be handled according to *Releasing State* described above. If the premature session termination happens, a later request may terminate before the earlier requests. In this situation, the size of the cache window should be carefully adjusted. Without loss of generality, we only describe the rule of size adjustment for a cache window in *Moving State* since similar processing can be easily derived for the other two states. Consider a cache window $W_{i,j}$ in *Moving State* with consecutive requests $R_{i,j}^1$ to $R_{i,j}^n$. If the session for the request $R_{i,j}^k$ $(1 \leq k \leq n)$ terminates before anyone else, the size of the cache window is adjusted according to the rules as follows:

- If $R_{i,j}^k$ is in the middle of its cache window, *i.e.*, $1 < k < n$, it has preceding and succeeding requests in the same cache window, and the cache window maintain its current state and $R_{i,j}^k$ is removed from the cache window.
- If $R_{i,j}^k$ is the head of its cache window, *i.e.*, $k = 1$, $R_{i,j}^1$ is removed from the cache window. If this request is the only one in the cache window, the cache window is destroyed after deleting the request. Otherwise, the head of the cache window is shrunk to $R_{i,j}^2$. Thus, the window span between $R_{i,j}^1$ and $R_{i,j}^2$ is removed.
- If $R_{i,j}^k$ is at the tail of the cache window, *i.e.*, $k = n$, $R_{i,j}^n$ is deleted from the request list, and the end of the cache window is shrunk to the extent that $R_{i,j}^{n-1}$ is the last request in the cache window.

After above steps, the parameters of the cache window like the average inter-arrival time should be updated instantly.

As for the case when VCR-like operations are involved, it will become more complicated, since the relative access position of the request may be changed, and therefore influences the states of the corresponding cache windows. Consequently, the primary cache strategy described above needs to be extended for supporting the VCR-like operations, such as pause and resume. To be specific, the pause operation is handled similar to the case of premature termination introduced, while the resume and jump operations are treated as new requests received, but requesting to play the previous video from certain timestamps after 0 second. As a result, once again, the variant request will be reallocated to an appropriate cache window which may be different from the original one, or rescheduled to the media server in the worst case. After finishing the VCR-like operations, it should be noted that the states of all relevant cache windows have to be updated accordingly. However, due to space limitations, we won't go into detail here.

C. Shortest Path Based Video Delivery Strategy

In addition to the cache strategy in the OpenFlow switches as discussed above, we have to choose the best switch for delivering the video content. Here, we apply the shortest path based video delivery strategy for the proposed software-defined video streaming system, which is summarized in **Algorithm 1**. Specifically, we take the state of cache window, capacity of the OpenFlow Switch as well as network topology into consideration, to make a decision on choosing the optimal OpenFlow Switch as the video source, thus achieving proximity principle based service. The codes from Line 1 to 8 is to search the potential OpenFlow switches, while those from Lines 12 to 17 is to search OpenFlow switch having shortest path to the user as well as lowest number of sessions for workload balance. Once the optimal switch or origin server (in case cache miss occurs), as well as the delivery path are determined by **Algorithm 1**, the switches on the delivery path should create new window and cache the video traffic as described in Section III-C.

V. Performance Analysis and Discussion

In this section, we discuss the performance of the proposed cache strategy. As shown in Fig. 4, the requests and cache windows for each video are individually organized, and thus we can investigate the performance of the proposed strategy by presenting the analysis for an individual video. It seems difficult to give rigorous theoretic results for its performance, since the cache window is variable which makes it complex. Therefore, we eval-

Algorithm 1: Shortest Path Based Video Delivery Strategy

Input:
C_i : the requested video;
$W_{i,j}$: the j-th cache window of C_i;
$N_{i,j}$: ID of the OpenFlow switch in which $W_{i,j}$ locates;
$h_{i,j}$: number of hops between the user and the switch $N_{i,j}$;
$m_{i,j}$: number of concurrent sessions served by the switch $N_{i,j}$;
h_0: the number of hops from Media Streaming Server to the user;
M: the capacity upper bound of the switches;
$W(i)$: the set of available cache windows;
Output: N^*: the optimal node to deliver the video;
$W(i) = \phi, h_{min} = h_0, m_{min} = M$
for all $W_{i,j} \in C_i$ **do**
  if $W_{i,j}$ is in Building State **and** $m_{i,j} < M$ **then**
    $W(i) = W(i) \bigcup \{W_{i,j}\}$
    if $h_{i,j} < h_{min}$ **then**
      $h_{min} = h_{i,j}$
    end if
  end if
end for
if $W(i) == \phi$ **or** $h_{min} == h_0$ **then**
  N^* = Media Streaming Server
else
  for all $W_{i,j} \in W(i)$ **do**
    if $h_{i,j} == h_{min}$ **and** $m_{i,j} < m_{min}$ **then**
      $m_{min} = m_{i,j}$
      $N^* = N_{i,j}$
    end if
  end for
  $m_{i,j} = m_{i,j} + 1$
end if
return N^*

uate its performance by analyzing fixed and equal cache window case that is a degenerated case of our proposed cache algorithm. Assume the size of all cache windows is T. The requests within a same cache window except the first request can be served from cache. The number of concurrent cached streams, N, can be found by counting the number of arriving requests that fall within a cache window. Without loss of generality, we assume the duration of the video object is D. Then, D can be divided into $K = D/T$ non-overlapping T-length cache windows. Let the arrivals of requests for this video follow a Poisson Process distribution with parameter λ. Larger value of λ implies the video is more popular. The probability of m arrivals for video object within a T-length cache window is $P_m = \frac{(\lambda T)^m}{m!} e^{-\lambda T}$. Thus, the expected number of requests falling in the cache windows that have exactly m arrivals during D, can be expressed using Bernoulli distribution [39] as follows:

$$\mathbb{E}[N_m] = (m-1)\sum_{k=1}^{\lceil K \rceil} k \binom{\lceil K \rceil}{k} (P_m)^k \cdot (1-P_m)^{\lceil K \rceil - k}, \quad (3)$$

where $m-1$ is the number of the cached requests in each of those T-length cache windows except the first one. The equation is precise when D is a multiple of T. The expected total number of the arriving requests for the given video object served from cache is $\mathbb{E}[N] = \sum_{m=2}^{\infty} \mathbb{E}[N_m]$.

Considering the actual cache window may be expanded, and the additional length may accommodate extra requests serviced from cache, so above result is the low-bound performance of our proposed algorithm. The expected number of requests serviced from the upstream OpenFlow switch can be expressed as $N_{upstream} = \lambda D - \mathbb{E}[N]$.

This means that the video data for $N_{upstream}$ requests is retrieved from the upstream switch, which causes the traffic on the link between the current switch and its upstream switch.

Since the arrival requests follow Poisson Process and the number of requests is given as m ($m \geq 2$) in the cache window with a determined length T, the joint distribution function of the n arrival times is the same as that of n order statistics of random variables which are independent and identically distributed (*i.i.d*) [40], specifically, following the uniform distribution with the probability density $1/T$. Then, the actual length of the cache window, denoted by T_m, is the time distance between the first request and the last one. For the cache window having m requests, the arrival times of the first request and the last one are respectively denoted by t_{first} and t_{last}, and the arrival times of the other $m-2$ requests are defined as t_i $(i = 2, \ldots, m-1)$. The expected actual cache window length can be calculated by $\mathbb{E}[T_m] = \mathbb{E}[t_{last} - t_{first}]$.

In order to calculate $\mathbb{E}[T_m]$, we first derive the joint probability density of t_{first} and t_{last}. For any $0 < y_1 < y_2$ and sufficient small $h > 0$ which satisfies $0 < y_1 < y_1 + h < y_2 < y_2 + h$, we have the joint probability distribution of t_{first} and t_{last}, *i.e.*,

$$\begin{aligned} &P(y_1 < t_{first} < y_1 + h, y_2 < t_{last} < y_2 + h, \\ &\quad y_1 < t_2, \ldots, t_{m-1} < y_2) \\ &\quad = P(y_1 < t_{first} < y_1 + h) P(y_2 < t_{last} < y_2 + h) \cdot \\ &\prod_{i=2}^{m-1} P(y_1 < t_i < y_2) \\ &\quad = m(m-1) \left(\frac{h}{T}\right)^2 \left(\frac{y_2 - y_1}{T}\right)^{m-2}. \end{aligned} \quad (4)$$

Then, the joint probability density of t_{first} and t_{last} can be expressed as

$$\begin{aligned} f_{t_{first}, t_{last}}(y_1, y_2) &= \lim_{h \to 0} \frac{m(m-1)(\frac{h}{T})^2 (\frac{y_2 - y_1}{T})^{m-2}}{h^2} \\ &= m(m-1) \frac{(y_2 - y_1)^{m-2}}{T^m}. \end{aligned} \quad (5)$$

Hence, the expected actual cache window length can be derived as

$$\begin{aligned} \mathbb{E}[T_m] &= \int_0^T \int_{t_{first}}^T m(m-1)(t_{last} - t_{first})^{m-1} \cdot \frac{1}{T^m} dt_{last} dt_{first} \\ &= \frac{m-1}{m+1} T. \end{aligned} \quad (6)$$

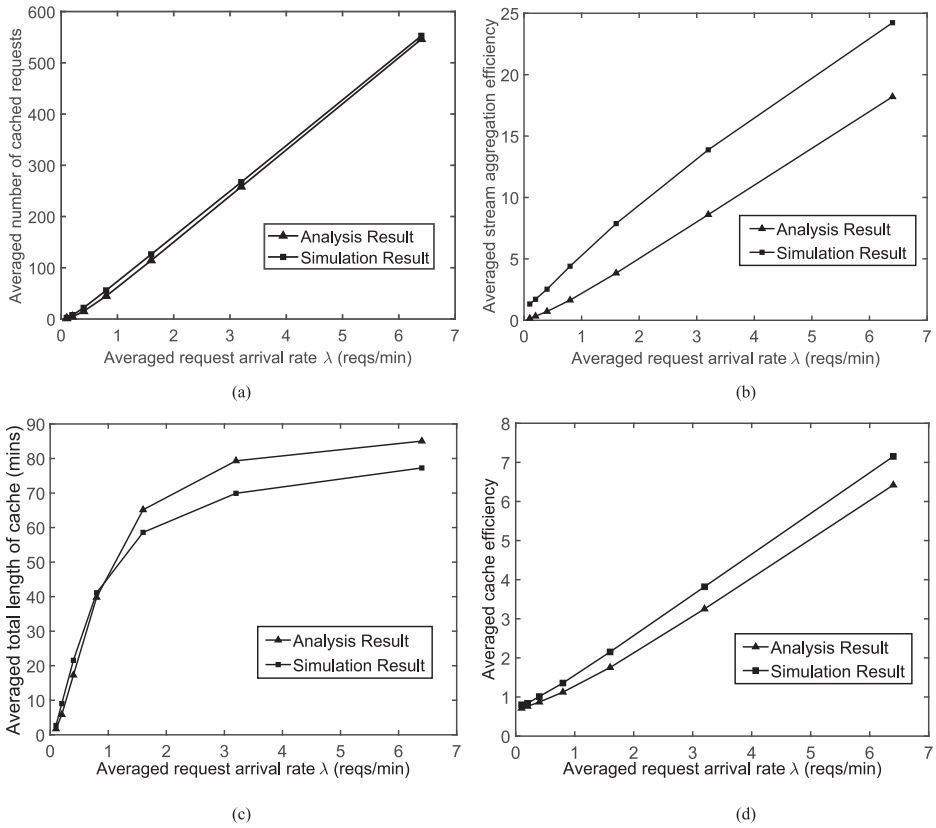

Fig. 7. Comparison between simulation and analysis results of the VLCW strategy. (a) Number of requests serviced by all cache nodes. (b) Aggregation efficiency of a single stream. (c) Total length of the cache windows. (d) Reuse efficiency of the unit cache.

Thus, the expected total length of the cache windows that have exactly m arrivals during D can be found as follows:

$$\mathbb{E}\left[L_m\right] = \frac{m-1}{m+1} T \sum_{k=1}^{\lceil K \rceil} k \binom{\lceil K \rceil}{k} (P_m)^k (1-P_m)^{\lceil K \rceil - k}. \quad (7)$$

Consequently, the total expected actual cache window length corresponding to the given video object is $\mathbb{E}\left[L\right] = \sum_{m=2}^{\infty} \mathbb{E}\left[L_m\right]$, which can approximately characterize the memory requirement of a video object with a given arrival rate.

Then, the utility of the streams from the upstream switch, denoted by $r_{upstream}$, and that of the cache, denoted by r_{cache}, can be defined to measure the performance, which are given, respectively, by:

$$r_{upstream} = 1 + \frac{\mathbb{E}\left[N\right]}{\mathbb{E}\left[N_{upstream}\right]}, \quad (8)$$

$$r_{cache} = \frac{\mathbb{E}\left[N\right]}{\mathbb{E}\left[L\right]}. \quad (9)$$

By developing a numerical simulator for the proposed variable-length cache window (VLCW) strategy, we present the comparison between its practical performances and the analysis results. Specifically, as for the simulation procedure, we generated the sequences of requests following the Poisson process with different arrival rates by leveraging a request generator, which were then sent to the request scheduling module, to be handled orderly based on the VLCW strategy and the real-time state of cache windows. Meanwhile, the status of all cache windows was also maintained and updated depending on the proposed strategy by an independent management module. During the process, each request received will be allocated to an optimal cache window or cause a cache miss, while each cache window will go through the three-state in its life-cycle, as described in Section IV-B. With respect to the parameter settings of the simulator, we only considered the accesses to a single video object, of which the length D was fixed to 90 mins. The initial size for cache window $T = 3$ mins, and its max size $T_{max} = 10$ mins. The fixed cache window size for analytical model was set to $T = 3$ mins as well, and the request process lasted 200 mins. Experiments with different arrival rates $\lambda \in \{0.1, 0.2, 0.4, 0.8, 1.6, 3.2, 6.4\}$ reqs/min for request sequences were conducted, which were repeated for 20 times to average the results.

Fig. 7 depicts the comparison results from four aspects: the number of requests serviced by all cache windows (*i.e.*, $\mathbb{E}\left[N\right]$), the aggregation efficiency of a single stream (*i.e.*, $r_{upstream}$), the total length of cache windows generated during the whole process (*i.e.*, $\mathbb{E}\left[L\right]$) and the reuse efficiency of the unit cache (*i.e.*, r_{cache}). We can observe that the simulation performance of the proposed algorithm exceeds the analytical results, especially for large arrival rate. Fig. 7(a) shows that the number of requests serviced by cache in simulation is approximately the same as the theoretical value which is referred to as the lower-bound, for the reason that it is degraded by disabling the size

adjustment of cache window in above theoretical derivation. From Fig. 7(b), it is seen that larger arrival rate can achieve more efficient utilization of streams. The reason is that the more intensively the requests arrive, the smaller the interval between two consecutive requests, and the more requests a cache window can accommodate, which means a single stream can be reused by more requests, in turn reducing the redundant video traffic. Fig. 7(c) indicates that the total length of cache windows, which is roughly proportional to their amount, increases with the request arrival rate. Specifically, the cache window in simulation can accommodate more requests compared to the analysis result, therefore, the number of which may be smaller than expect, thus resulting in the deviation between these two results as the arrival rate increases. Besides, a larger arrival rate leads to more frequent aggregation operations for video streams, thus achieving higher reuse efficiency of the unit cache as shown in Fig. 7(d).

As for the computation cost involved, we consider that the computation overhead is concentrated on the maintenance and update for the information of network status and cache distribution, as well as searching for the optimal cache window, rather than spent in calculating the window size or routing path for video delivery. Since we just need to recalculate the averaged interarrival time and the waiting time when a request arrives, which is based on the simple linear operations without complicated iterations, we can neglect the time cost for calculating the window size. Besides, the optimal routing path is determined based on Floyd-Warshall algorithm [41] once the optimal cache window is selected, of which the time complexity is $O(n^3)$, while due to that the proposed system is implemented in small-scale edge network where n is relatively small, thus the total time cost is acceptable. To the contrary, considering that the data structures were realized based on linked lists and STL maps as illustrated in Fig. 4, and there are two non-nested for loops to iterate through the cache windows relevant to a specific video as shown in **Algorithm 1**, so the time complexity is $O(n_v) + O(n_w)$ for finding the optimal cache window, where n_v is the number of video objects, and n_w denotes the number of cache windows traversed. Moreover, the time complexity for adding, deleting, modifying and querying operations can be estimated as $O(n_v) + O(logn_w) + O(n_r)$, where n_r is the number of requests in the selected cache window. We can further adopt the HashTable or HashMap to reduce the time cost in large-scale network scenario.

VI. Prototype Implementation

In order to verify the success of our design, we built a SDN testbed by ourselves to implement a prototype system for the proposed software-defined multimedia streaming system having variable-length in-network cache, as illustrated in Fig. 1. The implementation details of the key components were described as follows.

- The OpenFlow Switch was implemented as NFV hardware. We chose the commercial NETGEAR WNDR3800 wireless routers as the underlying hardwares, and rebuilt them by using OpenWrt [42] to obtain economical OpenFlow switches that support OpenFlow version 1.3. We chose Kingston's DataTraveler SE9G2 with 32GB memory space as the external USB flash disk in OpenFlow Switch for caching videos. Moreover, each OpenFlow Switch installed RTP tools, including RTPDump for parsing/caching RTP packets and RTPPlay for sending RTP data cached by RTPDump.
- We chose POX [43] as the OpenFlow Controller for simplicity, and ran it on a Linux server having three Network Interface Cards (NICs). The first NIC was connected to a standard switch TL-SG1024DT with 24 ports, which was further connected to the WAN ports of all OpenFlow Switches, in order to construct an out-of-band control-plane network. The second NIC was connected to the Management Server to share the information of network and cache, as well as to receive control messages. The corresponding modules of OpenFlow Controller in Fig. 2 were also implemented in POX. The network topology of the prototype was organized as depicted in Fig. 8.
- We also implemented the Management Server in Fig. 2 based on a Linux server having two NICs. One of the NICs was connected to the OpenFlow Controller for constructing the control plane, while the other one was connected to the Media Streaming Server.
- The Media Streaming Server was implemented based on a Linux server which installed the VLC media player 2.0.8 as the RTP Streaming module to encapsulate the video data into RTP/UDP. Multiple UDP sockets with different port number were created for streaming the video data to users, so that OpenFlow switches were able to identify the flow corresponding to a specific request.
- The terminal was implemented on a Linux client. It has a VLC media player installed to decode the received RTP streams and to play back the video.

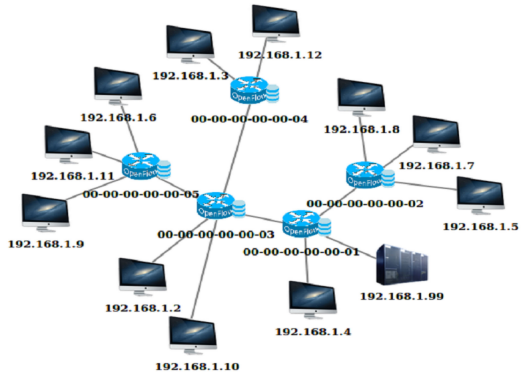

Fig. 8. Network topology of the implemented prototype.

We investigated the performance of our proposed solution based on the above prototype system, and compared it with the other widely adopted scheduling policies, namely the LRU strategy, which was considered suitable for CDN and ICN networks. Two different forms of LRU were implemented,*i.e.*, the one ICN suggested [44], named NNC+LRU, and its simpler variant, Rdm+LRU strategy [45]. As for NNC+LRU, all nodes along the content delivery path will remember the data packets as long as

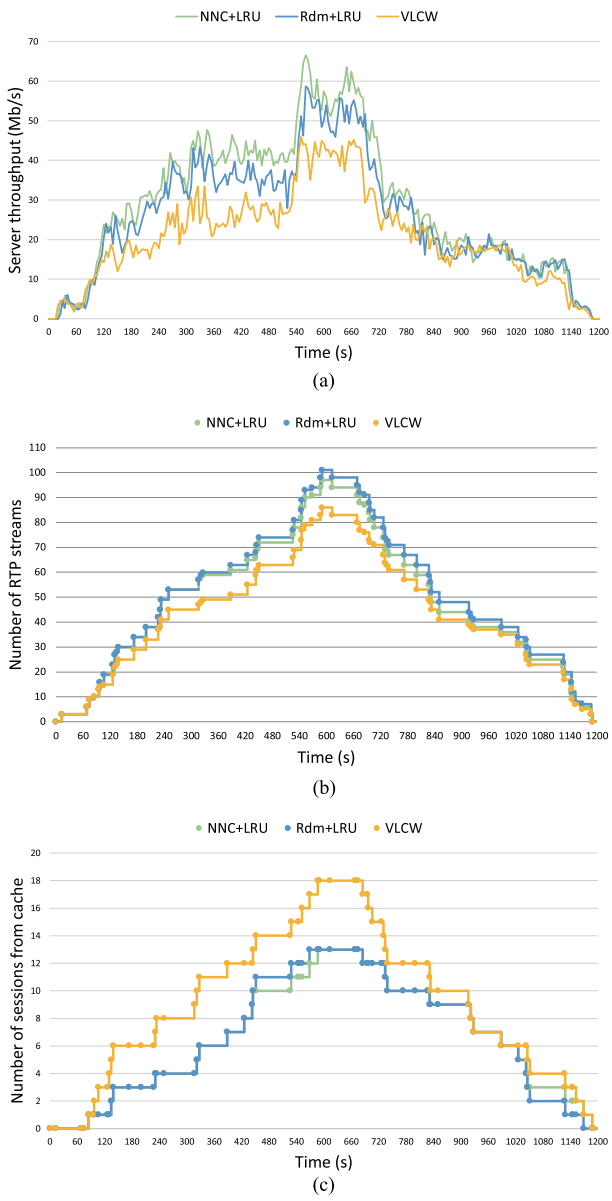

Fig. 9. Experimental results obtained from the prototype. (a) Real-time throughput of the server. (b) Real-time number of concurrent RTP streams in whole network. (c) Real-time number of RTP sessions delivered from all cache nodes.

possible according to LRU strategy. In contrast, only one node that has enough space or replaceable objects will be selected from the ones mentioned above randomly in Rdm+LRU.

In the comparison experiment, the request access rate λ was set to 2 req/min due to the limited bandwidth and hardware performance of the prototype system, and the duration of the video clip was set to 10 mins. At the beginning, no video was cached in the OpenFlow switches. We investigate the performance of the implemented prototype system in terms of the real-time throughput of the server, the real-time number of concurrent RTP streams in whole network, as well as the real-time number of RTP sessions delivered from all cache nodes. Fig. 9 presents the comparison results among the three strategies under the same conditions, such as the request sequence, storage space, and so on. As we can see from Fig. 9(a), VLCW strategy achieves the lowest throughput in most of the time, which means the minimum number of requests serviced by the server. Here, we treat the flows in different physical links while belonging to the same RTP session as different streams to quantify the bandwidth consumption of the whole network. Obviously, VLCW also achieves the least number of streams as illustrated in Fig. 9(b), thus consuming the least bandwidth. The reason for this is that VLCW strategy is able to remember a larger amount of fragments belonging to different videos with limited storage space, while there are much fewer entire videos cached for the other two strategies in the same circumstance. Since the requests are uniformly distributed, the videos desired have more chance to be obtained from the cache nodes for VLCW strategy, that is to say, it has a higher cache hit ratio as can be verified by Fig. 9(c).

Our above experiments based on the prototype system aim for demonstrating the feasibility of the proposed solution, which is not a comprehensive performance evaluation. Hence, Section VII is devoted to comprehensively investigating the performance of the proposed solution via emulations.

VII. Performance Evaluation

In this section, we comprehensively evaluate the performance of the proposed solution through emulations.

A. Performance Metrics

In order to evaluate the proposed strategy, we use the following metrics:

- Hop Count ($\bar{h}$): The hop count is defined as the averaged number of hops from the client to the selected node or the source server for each request.
- Transmission Delay ($\bar{t}$): The transmission delay is defined as the averaged time spent in establishing a RTP session for each request, from the client issues the request to it receives the first RTP packet from the selected node or the media server.
- Cache Load ($\bar{\rho}$): The cache load is defined as the time-averaged number of overall concurrent sessions delivered from the nodes with in-network cache.
- Server Load ($\bar{\varphi}$): The server load is defined as the time-averaged number of concurrent sessions delivered from the server.
- Efficiency Factor (η): The efficiency factor is defined as the ratio between Cache Load and Server Load. Larger η implies a higher efficiency of the cache.
- Server Throughput ($\bar{b}$): The throughput of the server is defined as the time-averaged total bandwidth occupied for data delivery.
- Total Storage Length ($\bar{L}$): The total storage length is defined as the time-averaged total length of all cache windows on all cache nodes in the cache system.
- Stream Efficiency (r_{server}): The stream efficiency is derived from the definition of $r_{upstream}$ in V, and is defined as the aggregation efficiency of a single stream delivered from the server.

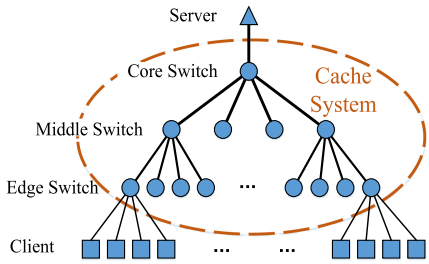

Fig. 10. Network topology of the emulation experiments.

TABLE II
LINK BANDWIDTH AND DELAY OF THE EMULATION NETWORK

Link	Bandwidth	Delay
Server ⇔ Core Switch	1000 Mbps	20 ms
Core Switch ⇔ Middle Switch	1000 Mbps	15 ms
Middle Switch ⇔ Edge Switch	500 Mbps	10 ms
Edge Switch ⇔ Client	100 Mbps	5 ms

- Cache Efficiency (r_{cache}): As defined in V, r_{cache} is the reuse efficiency of the unit length video segment in the cache system.

In the following emulation experiments, we will use above metrics together to systematically evaluate the performance of the proposed solution.

B. Emulation Environment

Some key settings are described here, including the network topology and the experiment procedure:

- Network Topology Setting: We use Mininet 2.2.1 [46] to create a realistic virtual SDN enabled network, *i.e.*, running real kernel, switch and application code on a single machine. Since the k-ary tree topology is considered common and representative for both classical and peer-to-peer VoD system [34], [47], [48], we determine to construct a realistic scenario with it. Specifically, we choose a complete 4-ary tree with the depth of 4 (except the server), to scale up the size of the emulation system, as shown in Fig. 10, which is composed of 1 server, 21 switches, and 64 clients. Furthermore, we define the storage capability of each cache node respectively by software method according to the settings described in VII-C. The transmission speed of virtual network interface cards (NICs) is set to 100 Mbps for hosts, and 1000 Mbps for the server and switches respectively. Moreover, the maximal access capability of each switch is set to 50 video streams. And the configuration information for different links are given in TABLE II. Specifically, we install the RTP tools (RTPDump and RTPPlay) on the Linux host of Mininet as well, to enable the caching and streaming capabilities of RTP data for all virtual OpenFlow switch nodes, as well as the server for simplicity.
- Procedure Setting: We assume that the arrivals of the requests to the server follow a Poisson Process with an average arrival rate λ. Assuming that accesses to video objects follow a Zipf-like distribution [4]:

$$p_i = \frac{\frac{1}{i^\theta}}{\Sigma_{i=1}^{N} \frac{1}{i^\theta}}, 0 < \theta \leq 1, \tag{10}$$

where N is the total number of available videos and θ is a skew factor for the popularity. For simplicity, we set $\theta = 1$ here. We use different video clips having duration of 10 minutes, 1280 × 720 resolution, and the bit rates ranging from 2Mbps to 4Mbps. For the benchmark algorithms NNC+LRU and Rdm+LRU, we assume that the entire videos are transferred to the in-network cache along with the streams delivered for the first request of them, while consuming no extra backhaul bandwidth. Besides, the videos cached stay available all along unless being removed, even though they have not been generated completely, which is similar to the scene for VLCW strategy. Furthermore, we also consider an advanced cache algorithm of better performance between the two proposed by [34], namely, the Election-based strategy, which is designed for caching video segments in network nodes as well. Its basic idea is to select the optimal node along the transmission path with highest potential gain of caching the segments locally, which are expected to be requested soon based on the information of the recent user behavior. Specifically, we set the fixed length of all video segments to 10 seconds in this emulation.

C. Result Analysis

1) Experiment 1: This experiment was carried out to investigate the performance of our proposed strategy for different request access rates λ (req/min), which range from 5 req/min to 25 req/min. We use M to denote the storage capacity of a single node, which is an integer that denotes the multiples of the entire video file. Considering that the size of all available videos is much larger than that can be cached in practice, we set M to 2, that is to say, there are at most 20 mins video data cached in each switch. The total number of available videos denoted by N is set to 10, and the default initial size of the cache window denoted by T is 3 mins. The experimental results were plotted in Fig. 11.

From Fig. 11(a), we can observe that the server throughput $\bar{b}$ increases as raising λ for NNC+LRU and Rdm+LRU strategies, followed by Election-based strategy, while it almost remains instant for VLCW. The reason is that the first two strategies operate the cache at the granularity of entire videos, and fewer number of cached videos leads to a lower cache hit ratio, which involves more cache misses especially for the large λ, thus increasing the workload of the origin server. By contrast, the other two run the cache at the granularity of video segments, to cache larger amounts of video fragments for aggregating the video requests, so that resulting in a much lower $\bar{b}$ which is also insensitivity to λ. Specifically, VLCW outperforms Election-based since that the size of its video segment cached is variable following with the access pattern, namely, fitting to the video popularity dynamically, thus the cache hit ratio is further increased. Fig. 11(b) characterizes the cache load $\bar{\rho}$, which also indicates that VLCW indeed has a higher cache hit ratio. From Fig. 11(c), we can observe that VLCW occupies much less cache space than other strategies as well. This is because VLCW only caches the segments that will certainly be reused by subsequent requests, and removes the useless por-

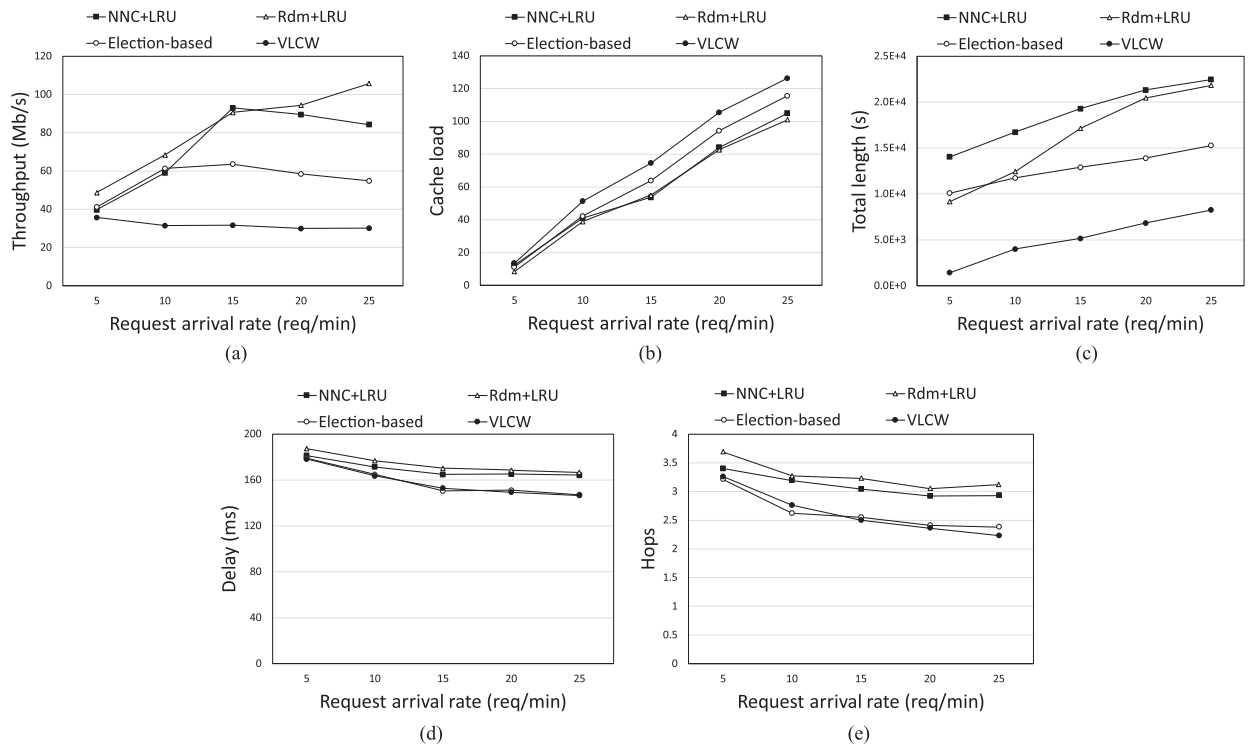

Fig. 11. Performance comparison of VLCW, Election-based, NNC+LRU, and Rdm+LRU for different arrival rates. (a) Average throughput of the server, $\bar{b}$. (b) Average load of all cache nodes, $\bar{\rho}$. (c) Average total length of the cached videos, $\bar{L}$. (d) Average delay for each request, $\bar{t}$. (e) Average hops for each request,$\bar{h}$.

TABLE III
EFFICIENCY COMPARISON WITH DIFFERENT ARRIVAL RATES

Arrival Rate		5	10	15	20	25
NNC+LRU	r_{cache}	0.0005	0.0012	0.0013	0.0023	0.0022
	r_{server}	2.30	3.72	3.04	4.15	5.18
	η	0.91	1.97	1.70	2.68	3.32
Rdm+LRU	r_{cache}	0.0007	0.0016	0.0017	0.0026	0.0026
	r_{server}	1.77	3.61	3.43	3.86	5.76
	η	0.48	1.69	1.83	2.52	2.82
Election-based	r_{cache}	0.0014	0.0019	0.0040	0.0053	0.0061
	r_{server}	2.12	4.10	4.74	7.00	8.10
	η	0.77	2.16	2.99	4.42	5.49
VLCW	r_{cache}	0.0116	0.0143	0.0154	0.0163	0.0162
	r_{server}	2.94	8.50	12.38	20.36	21.58
	η	1.11	4.84	7.04	10.39	12.20

tions to release space. While the other strategies are all based on cache replacement policy (LRU), where lots of entire videos or segments cached will not be accessed until driven out, thus wasting the storage resource. Fig. 11(d) and Fig. 11(e) illustrate that $\bar{t}$ and $\bar{h}$ change identically, and are similar for VLCW and Election-based strategies, while both are smaller than those of NNC+LRU and Rdm+LRU. The reason lies on that the former two provide higher likelihood to serve the request at the edge nodes, which are benefited from the diversity of more video segments cached.

From TABLE III, we can observe that VLCW has a much higher r_{cache}. This means more requests are handled by the cache of unit length for VLCW, which is consistent with the results characterized in Fig. 11(b) and Fig. 11(c). Similarly, VLCW achieves a higher r_{server} benefiting from its efficiency in aggregating the repeated traffic in the time domain. According to the definition of η, it is determined by both $\bar{\rho}$ and $\bar{\varphi}$. Hence, VLCW achieves a higher efficiency factor as well, due to its higher cache load and lower server load as indicated by Fig. 11(a) and Fig. 11(b). The experimental results demonstrate that VLCW outperforms the other three strategies, especially for the scenario of the large request arrival rate.

2) Experiment 2: This experiment was conducted to investigate the performance of our proposed strategy for different numbers of videos N. The corresponding results indirectly characterize the performance influence of the video popularity distribution. The values of M and T remained the same as those in *Experiment 1*, while λ was fixed to 20 req/min, and the range of N was from 5 to 25. The experimental results are characterized in Fig. 12.

Fig. 12(a) shows that the server throughput $\bar{b}$ corresponding to all strategies grows as increasing the number of video objects, N. The reason lies in that according to (10), the larger value of N reduces the popularity of each video, which is not beneficial for improving the cache hit ratio, and the number of the video streams from the streaming server are increased consequently. Fortunately, VLCW uses the limited cache space in the manner of caching popular video segments of variable size, which enable it to maintain a relatively high level of cache hit ratio, and to reduce the growth of $\bar{b}$. Fig. 12(b) also indicates that VLCW achieves the slowest reduction of cache hit ratio when increasing N. Since VLCW caches the video portion within the sliding window, it achieves the lowest cache space occupation which is kept insensitive to N, as shown in Fig. 12(c). The average delays $\bar{t}$ illustrated in Fig. 12(d) , as well as the average hops $\bar{h}$ shown by Fig. 12(e) both increase as raising N for all strategies. This

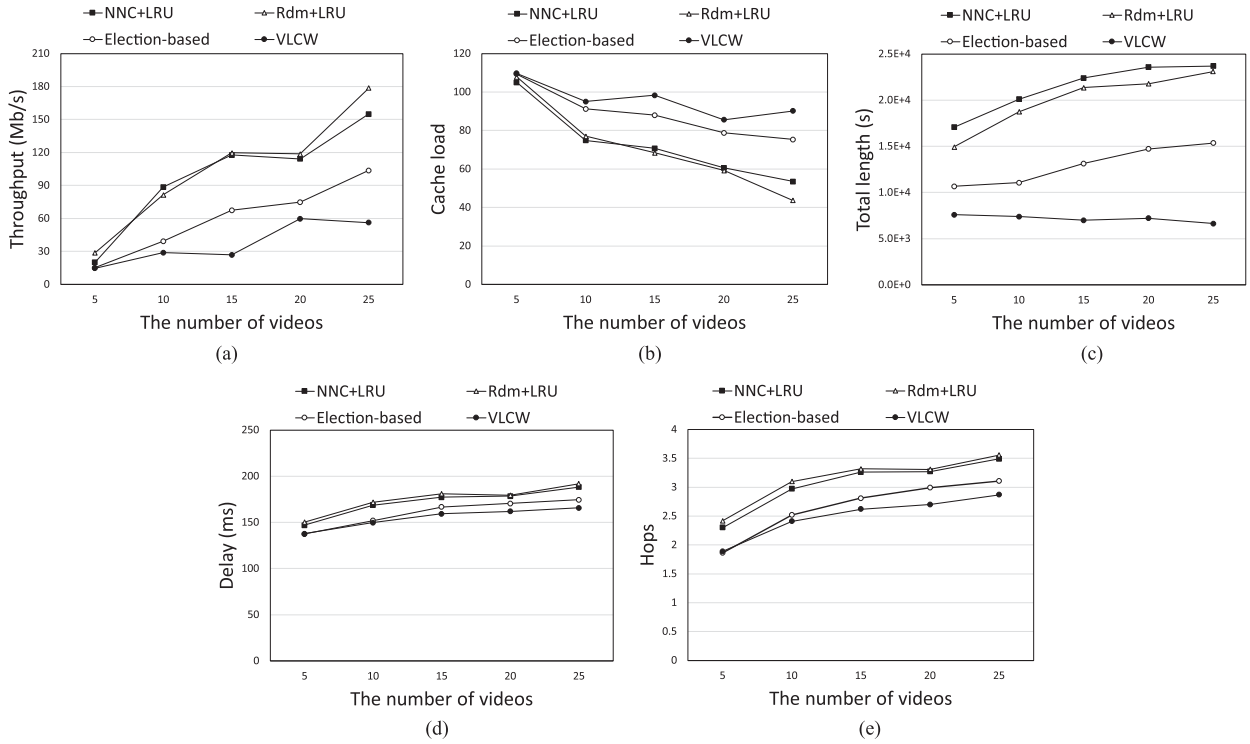

Fig. 12. Performance comparison of VLCW, Election-based, NNC+LRU, and Rdm+LRU for different number of videos. (a) Average throughput of the server, $\bar{b}$. (b) Average load of all cache nodes, $\bar{\rho}$. (c) Average total length of the cached videos, $\bar{L}$. (d) Average delay for each request, $\bar{t}$. (e) Average hops for each request, $\bar{h}$.

TABLE IV
EFFICIENCY COMPARISON FOR DIFFERENT NUMBER OF VIDEOS

Videos Number		5	10	15	20	25
NNC+LRU	r_{cache}	0.0037	0.0019	0.0012	0.0011	0.0008
	r_{server}	30.86	4.18	3.19	2.51	1.86
	η	15.99	2.51	1.62	1.34	0.83
Rdm+LRU	r_{cache}	0.0040	0.0023	0.0016	0.0014	0.0008
	r_{server}	21.60	5.69	3.01	2.42	1.65
	η	10.89	2.79	1.50	1.28	0.59
Election-based	r_{cache}	0.0057	0.0049	0.0029	0.0024	0.0021
	r_{server}	43.20	11.39	5.86	5.02	3.28
	η	20.36	6.76	3.35	2.92	1.77
VLCW	r_{cache}	0.0150	0.0140	0.0148	0.0132	0.0145
	r_{server}	43.20	20.50	10.85	7.92	5.12
	η	22.17	9.85	6.14	4.26	3.25

may be caused by the fact that due to the reduced popularity for the increased N, more video requests are served by the video streaming server, which induces more hops. But VLCW has the minimum $\bar{h}$ and $\bar{t}$ benefited from the relatively higher cache hit ratio than those of the other three strategies.

TABLE IV shows efficiency comparisons for different number of videos, N. Since VLCW achieves an almost unchanging value for $\bar{L}$ and a slightly decreased $\bar{\rho}$, which are indicated by the results in Fig. 12(c) and Fig. 12(b), thus it still outperforms much better than other methods in cache utilization r_{cache}. As for r_{server} and η, the other three strategies also perform well when N is very small, because the video accesses concentrate on a small set of available videos. However, they are incurred with much faster performance degradation than VLCW.

3) Experiment 3: This experiment was conducted to illustrate the impact imposed by each switch's storage capacity M on the performance of the proposed solution. The cache capacity ranges from 1 to 5 times of the size of a video object. The value of λ and T were both kept the same as *Experiment 2*, and N was set to 20. The experimental results were characterized in Fig. 13.

We can observe from Fig. 13(a) that $\bar{b}$ reduces as increasing the cache capacity, M, for NNC+LRU, Rdm+LRU and Election-based strategies, which benefits from more entire or segmented videos cached in switch nodes as shown in Fig. 13(c). By contrast, it is kept almost constant for VLCW. The reason is that VLCW only caches the video segments necessary for aggregating the video requests, which has the lowest requirement of cache capacity as shown in Fig. 13(c). Fig. 13(b), Fig. 13(d) as well as Fig. 13(e) further confirm that once the minimum cache capacity is satisfied, increasing the cache space is not beneficial for improving the performance of the proposed VLCW solution. In other words, VLCW is able to ensure a high aggregation efficiency for the repetitive video streams even though constrained by the limited cache space in network nodes.

The further results are shown in TABLE V in terms of cache efficiency, stream efficiency and efficiency factor. It can be seen that all NNC+LRU, Rdm+LRU and Election-based gain significant benefits in both r_{server} and η as M increases, resulted from their increase in $\bar{\rho}$ and the corresponding decrease in $\bar{\varphi}$ compared to VLCW. However, their cache efficiencies are still kept a low increase except for Election-based strategy, of which the total length of videos cached increases relatively slow, due to the segment-based caching mechanism. From TABLE V, we can observe that all three metrics for VLCW maintain almost unchanged level. This also implies that our strategy, VLCW, can achieve better performance in reducing redundant traffic while consuming less storage resource.

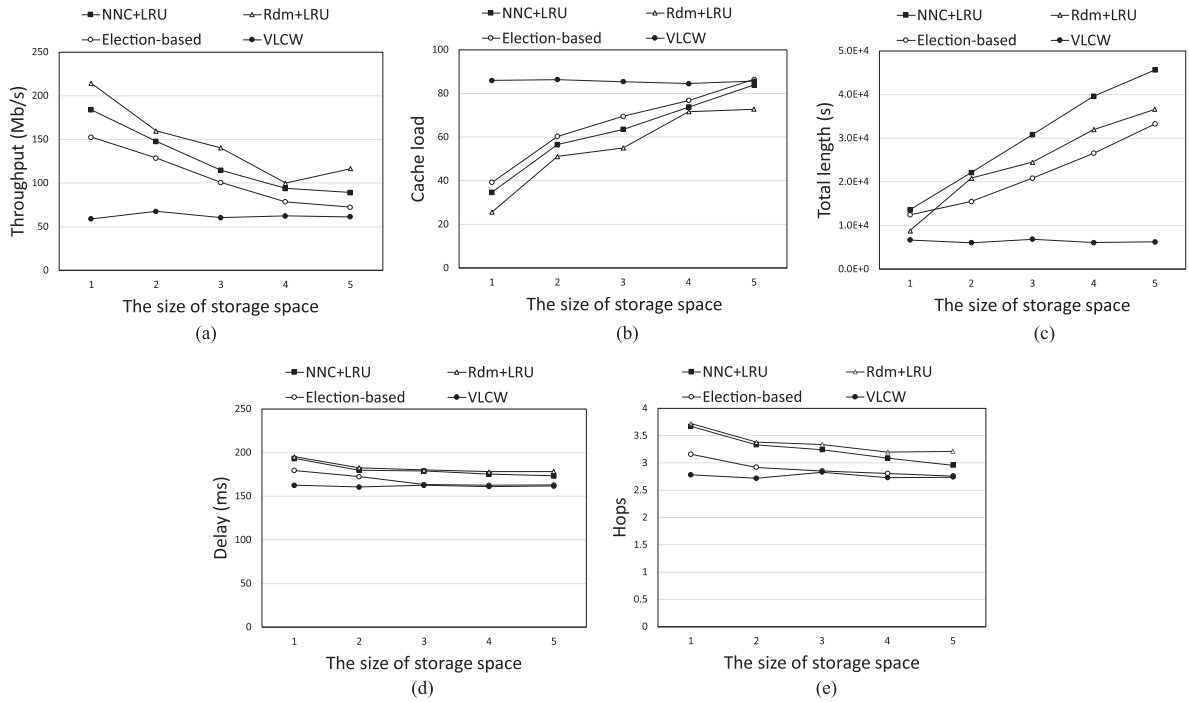

Fig. 13. Performance comparison of VLCW, Election-based, NNC+LRU and Rdm+LRU for different sizes of storage space. (a) Average throughput of the server, $\bar{b}$. (b) Average load of all cache nodes, $\bar{\rho}$. (c) Average total length of the cached videos, $\bar{L}$. (d) Average delay for each request, $\bar{t}$. (e) Average hops for each request, $\bar{h}$.

TABLE V
EFFICIENCY COMPARISON FOR DIFFERENT SIZES OF STORAGE SPACE

Storage Size		1	2	3	4	5
NNC+LRU	r_{cache}	0.0008	0.0009	0.0008	0.0009	0.0010
	r_{server}	1.52	2.20	2.78	3.98	4.76
	η	0.48	1.07	1.49	2.24	2.67
Rdm+LRU	r_{cache}	0.0008	0.0013	0.0012	0.0014	0.0014
	r_{server}	1.41	2.15	2.47	3.98	3.63
	η	0.32	0.87	1.08	2.05	1.70
Election-based	r_{cache}	0.0016	0.0029	0.0034	0.0040	0.0038
	r_{server}	1.62	2.43	3.39	4.41	7.64
	η	0.58	1.22	1.89	2.58	4.13
VLCW	r_{cache}	0.0146	0.0148	0.0141	0.0156	0.0157
	r_{server}	7.50	6.59	7.41	5.97	7.13
	η	4.07	3.69	4.09	3.82	4.33

VIII. CONCLUSION

In order to eliminate the redundant traffic incurred by VoD service, we designed a software-defined multimedia streaming VoD system with the aid of in-network cache. A variable-length interval cache strategy was proposed for improving the utilization of storage resource and the aggregation efficiency of streams. We presented the theoretic analysis to demonstrate the attainable performance of the proposed strategy. Furthermore, a prototype system was developed to verify the feasibility and applicability of the proposed solution. In order to comprehensively evaluate the achievable performance improvement, we defined the performance metrics in terms of hop count, transmission delay, server throughput, cache efficiency *etc.*, and conducted comparative experiments with three benchmark algorithms, namely NNC+LRU, Rdm+LRU and Election-based by emulation. The experimental results indicate that the performance improvement is up to 50% reduction in server throughput, 25% decrease in the hops of the delivery path, 3 to 5 times increase in the utilization of the storage resource.

REFERENCES

[1] "Cisco visual networking index: Forecast and methodology, 2016-2021," Cisco Syst., San Jose, CA, USA, Tech. Rep. 1465272001663118., Sep. 2017. [Online]. Available: https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/complete-white-paper-c11-481360.html

[2] I. Ullah, G. Doyen, G. Bonnet, and D. Gaiti, "A survey and synthesis of user behavior measurements in P2P streaming systems," *IEEE Commun. Surv. Tut.*, vol. 14, no. 3, pp. 734–749, Jul–Sep. 2012.

[3] A. Balachandran, "Large scale data analytics of user behavior for improving content delivery," Ph.D. dissertation, Carnegie Mellon Univ., Pittsburgh, PA, USA, Dec. 2014.

[4] S. Mitra *et al.*, "Characterizing web-based video sharing workloads," *ACM Trans. Web*, vol. 5, no. 2, pp. 1–27, May 2011.

[5] G. Peng, "CDN: Content distribution network," *Research Proficiency Exam Report*, pp. 1–6, Nov. 2004, *arXiv:cs/0411069*.

[6] Akamai website. 2017. [Online]. Available: https://www.akamai.com/cn/zh/

[7] Limelight website. 2017. [Online]. Available: https://www.limelight.com/

[8] N. Bartolini, E. Casalicchio, and S. Tucci, "A walk through content delivery networks," in *Proc. Int. Workshop Model., Anal., Simul. Comput. Telecommun. Syst.* (Lecture Notes in Computer Science, vol. 2965), Apr. 2004, pp. 1–25.

[9] Chiosi, *et al.*, "Network functions virtualisation: An introduction, benefits, enablers, challenges and call for action," in *Proc. SDN Open-Flow World Congr.*, Oct. 2012, pp. 22–24.

[10] A. Anand, V. Sekar, and A. Akella, "SmartRE: An architecture for coordinated network-wide redundancy elimination," in *Proc. ACM SIGCOMM Conf. Data Commun.*, Aug. 2009, pp. 87–98.

[11] N. Eisley, L.-S. Peh, and L. Shang, "In-network cache coherence," in *Proc. 39th Annu. IEEE/ACM Int. Symp. Microarchitecture*, Dec. 2006, pp. 321–332.

[12] S. Sezer *et al.*, "Are we ready for SDN? Implementation challenges for software-defined networks," *IEEE Commun. Mag.*, vol. 51, no. 7, pp. 36–43, Jul. 2013.

[13] ONF Solution Brief, *OpenFlow-enabled SDN and Network Functions Virtualization*. Menlo Park, CA, USA: Open Netw. Found., 2014.

[14] N. McKeown *et al.*, "OpenFlow: Enabling innovation in campus networks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, Apr. 2008.
[15] M. Mangili, F. Martignon, and A. Capone, "Stochastic planning for content delivery: Unveiling the benefits of network functions virtualization," in *Proc. IEEE 22nd Int. Conf. Netw. Protocols*, Oct. 2014, pp. 344–349.
[16] N. Herbaut, D. Negru, D. Magoni, and P. A. Frangoudis, "Deploying a content delivery service function chain on an SDN-NFV operator infrastructure," in *Proc. IEEE Int. Conf. Telecommun. Multimedia*, Jul. 2016, pp. 1–7.
[17] H. Ibn-Khedher, E. Abd-Elrahman, and H. Afifi, "OMAC: Optimal migration algorithm for virtual CDN," in *Proc. IEEE 23rd Int. Conf. Telecommun.*, May 2016, pp. 1–6.
[18] J. Kim *et al.*, "An efficient multimedia transmission control methodology based on NFV," in *Proc. IEEE 5th Int. Conf. IT Convergence Secur.*, Aug. 2015, pp. 1–4.
[19] "The elastic CDN solution: Dynamic content delivery network built on NFV and SDN from Akamai and Juniper," Juniper Netw., Sunnyvale, CA, USA, Solution Brief 3510532-001-EN, Dec. 2014, [Online]. Available: https://www.juniper.net/assets/kr/kr/local/pdf/solutionbriefs/3510532-en.pdf
[20] I. Psaras, W. K. Chai, and G. Pavlou, "In-network cache management and resource allocation for information-centric networks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 11, pp. 2920–2931, Nov. 2014.
[21] K. Kanai *et al.*, "Proactive content caching for mobile video utilizing transportation systems and evaluation through field experiments," *IEEE J. Sel. Areas Commun.*, vol. 34, no. 8, pp. 2102–2114, Aug. 2016.
[22] Y. Kim and I. Yeom, "Performance analysis of in-network caching for content-centric networking," *Comput. Netw.*, vol. 57, no. 13, pp. 2465–2482, Sep. 2013.
[23] N. Xue *et al.*, "Demonstration of OpenFlow-controlled network orchestration for adaptive SVC video manycast," *IEEE Trans. Multimedia*, vol. 17, no. 9, pp. 1617–1629, Sep. 2015.
[24] M. Wichtlhuber, R. Reinecke, and D. Hausheer, "An SDN-based CDN/ISP collaboration architecture for managing high-volume flows," *IEEE Trans. Netw. Service Manage.*, vol. 12, no. 1, pp. 48–60, Mar. 2015.
[25] H. E. Egilmez and A. M. Tekalp, "Distributed QoS architectures for multimedia streaming over software defined networks," *IEEE Trans. Multimedia*, vol. 16, no. 6, pp. 1597–1609, Oct. 2014.
[26] J. Yang, K. Zhu, Y. Ran, W. Cai, and E. Yang, "Joint admission control and routing via approximate dynamic programming for streaming video over software-defined networking," *IEEE Trans. Multimedia*, vol. 19, no. 3, pp. 619–631, Mar. 2017.
[27] M. Kimmerlin, J. Costa-Requena, and J. Manner, "Caching using software-defined networking in LTE networks," in *Proc. IEEE Int. Conf. Adv. Netw. Telecommun. Syst.*, Dec. 2014, pp. 1–6.
[28] A. F. Trajano and M. P. Fernandez, "ContentSDN: A content-based transparent proxy architecture in software-defined networking," in *Proc. IEEE 30th Int. Conf. Adv. Inf. Netw. Appl.*, Mar. 2016, pp. 532–539.
[29] P. Georgopoulos, M. Broadbent, A. Farshad, B. Plattner, and N. Race, "Using software defined networking to enhance the delivery of video-on-demand," *Comput. Commun.*, vol. 69, pp. 79–87, Sep. 2015.
[30] Z. Li, G. Simon, and A. Gravey, "Caching policies for in-network caching," in *Proc. IEEE 21st Int. Conf. Comput. Commun. Netw.*, Jul. 2012, pp. 1–7.
[31] S. Traverso *et al.*, "Unravelling the impact of temporal and geographical locality in content caching systems," *IEEE Trans. Multimedia*, vol. 17, no. 10, pp. 1839–1854, Oct. 2015.
[32] S. Li, J. Xu, M. van der Schaar, and W. Li, "Trend-aware video caching through online learning," *IEEE Trans. Multimedia*, vol. 18, no. 12, pp. 2503–2516, Dec. 2016.
[33] Y. Zhou, L. Chen, C. Yang, and D. M. Chiu, "Video popularity dynamics and its implication for replication," *IEEE Trans. Multimedia*, vol. 17, no. 8, pp. 1273–1285, Aug. 2015.
[34] M. Claeys *et al.*, "Cooperative announcement-based caching for video-on-demand streaming," *IEEE Trans. Netw. Serv. Manage.*, vol. 13, no. 2, pp. 308–321, Mar. 2016.
[35] A. Dan and D. Sitaram, "Generalized interval caching policy for mixed interactive and long video workloads," in *Proc. SPIE*, Mar. 1996, vol. 2667, pp. 344–351.
[36] A. Dan and D. Sitaram, "Buffer management policy for an on-demand video server," IBM, Yorktown Heights, NY, USA, Tech. Rep. RC 19347, 1994.
[37] H. Schulzrinne, RTP Tools. 2016. [Online]. Available: http://www.cs.columbia.edu/irt/software/rtptools/
[38] H.-Y. Chen *et al.*, "A novel multimedia synchronization model and its applications in multimedia systems," *IEEE Trans. Consum. Electron.*, vol. 41, no. 1, pp. 12–22, Feb. 1995.
[39] A. W. Marshall and I. Olkin, "A family of bivariate distributions generated by the bivariate Bernoulli distribution," *Amer. Statist. Assoc.*, vol. 80, no. 390, pp. 332–338, 1985.
[40] L. Devroye, "Laws of the iterated logarithm for order statistics of uniform spacings," *Ann. Probab.*, vol. 9, no. 5, pp. 860–867, Oct. 1981.
[41] S. Pallottino, "Shortest-path methods: Complexity, interrelations and new propositions," *Networks*, vol. 14, no. 2, pp. 257–267, 1984.
[42] OpenWrt Project. 2015. [Online]. Available: https://www.openwrt.org/
[43] Open Networking Lab, POX Wiki. 2015. [Online]. Available: https://openflow.stanford.edu/display/ONL/POX+Wiki
[44] V. Jacobson *et al.*, "Networking named content," in *Proc. ACM 5th Int. Conf. Emerg. Netw. Exp. Technol.*, Dec. 2009, pp. 1–12.
[45] W. K. Chai, D. He, I. Psaras, and G. Pavlou, "Cache "less for more" in information-centric networks," in *Proc. 11th Int. IFIP TC 6 Conf. Netw.*, May 2012, pp. 27–40.
[46] Mininet Team, Mininet Project. 2017. [Online]. Available: http://mininet.org//
[47] Y. Cui, B. Li, and K. Nahrstedt, "oStream: Asynchronous streaming multicast in application-layer overlay networks," *IEEE J. Sel. Areas Commun.*, vol. 22, no. 1, pp. 91–106, Jan. 2004.
[48] Y. Liu, Y. Guo, and C. Liang, "A survey on peer-to-peer video streaming systems," *Peer-to-Peer Netw. Appl.*, vol. 1, no. 1, pp. 18–28, Mar. 2008.

Jian Yang (SM'16) received the B.S. and Ph.D. degrees from the University of Science and Technology of China (USTC), Hefei, China, in 2001 and 2006, respectively. From 2006 to 2008, he was a Postdoctoral Scholar with the Department of Electronic Engineering and Information Science, USTC. Since 2008, he has been an Associate Professor with the Department of Automation, USTC.

He is currently a Professor with the School of Information Science and Technology, USTC. His research interests include future network, distributed system design, modeling and optimization, and multimedia over wired and wireless and stochastic optimization. He was the recipient of the Lu Jia-Xi Young Talent Award from Chinese Academy of Sciences in 2009.

Zhen Yao received the B.S. degree in 2015 from the University of Science and Technology of China, Hefei, China, where he is currently working toward the Ph.D. degree with the School of Information Science and Technology.

His research interests include software-defined networking, multimedia transmissions in future networks, and in-network intelligence assisted by machine learning.

Bowen Yang received the B.S. degree in 2014 from the University of Science and Technology of China, Hefei, China, where he is currently working toward the Ph.D. degree with the School of Information Science and Technology.

His research interests include multimedia communications, wireless network, and stochastic optimization of multimedia transmissions in future networks.

Xiaobin Tan, photograph and biography not available at the time of publication.

Zilei Wang, photograph and biography not available at the time of publication.

Quan Zheng, photograph and biography not available at the time of publication.