NCTM: A Novel Coded Transmission Mechanism for Short Video Deliveries

Zhenge Xu
Tsinghua University
Shenzhen, China
Peng Cheng Laboratory
Shenzhen, China
xzg22@mails.tsinghua.edu.cn

Qing Li*
Peng Cheng Laboratory
Shenzhen, China
liq@pcl.ac.cn

Wanxin Shi
Fudan University
School of Computer Science
Shanghai, China
shiwx@fudan.edu.cn

Yong Jiang
Tsinghua University
Shenzhen, China
Peng Cheng Laboratory
Shenzhen, China
jiangy@sz.tsinghua.edu.cn

Zhenhui Yuan
University of Warwick
Coventry, UK
zhenhui.yuan@warwick.ac.uk

Peng Zhang
Tencent Cloud
Shenzhen, China
rocalzhang@tencent.com

Gabriel-Miro Muntean
Dublin City University
Dublin, Ireland
gabriel.muntean@dcu.ie

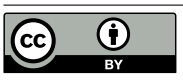
ABSTRACT

With the rapid popularity of short video applications, a large number of short video transmissions occupy the bandwidth, placing a heavy load on the Internet. Due to the extensive number of short videos and the predominant service for mobile users, traditional approaches (e.g., CDN delivery, edge caching) struggle to achieve the expected performance, leading to a significant number of redundant transmissions. In order to reduce the amount of traffic, we design a Novel Coded Transmission Mechanism (NCTM), which transmits XOR-coded data instead of the original video content. NCTM caches the short videos that users have already watched in user devices, and encodes, multicasts, and decodes XOR-coded files separately at the server, edge nodes, and clients, with the assistance of cached content. This approach enables NCTM to deliver more short video data given the limited bandwidth. Our extensive trace-driven simulations show how NCTM reduces network load by 3.02%-14.75%, cuts peak traffic by 23.01%, and decreases rebuffering events by 43%-85% in comparison to a CDN-supported scheme and a naive edge caching scheme. Additionally, NCTM also increases the user's buffered video duration by 1.21x-13.53x, ensuring improved playback smoothness.

CCS CONCEPTS

• **Networks** → **Mobile networks**.

*Qing Li is the corresponding author.



WWW '24, May 13–17, 2024, Singapore, Singapore

ACM ISBN 979-8-4007-0171-9/24/05.
https://doi.org/10.1145/3589334.3645387

KEYWORDS

short video delivery, coded transmission, client-side cache

ACM Reference Format:
Zhenge Xu, Qing Li, Wanxin Shi, Yong Jiang, Zhenhui Yuan, Peng Zhang, and Gabriel-Miro Muntean. 2024. NCTM: A Novel Coded Transmission Mechanism for Short Video Deliveries. In *Proceedings of the ACM Web Conference 2024 (WWW '24), May 13–17, 2024, Singapore, Singapore.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3589334.3645387

1 INTRODUCTION

Short video applications such as Tiktok [1] and YouTube Shorts [2] are rapidly rising in popularity, attracting billions of active users per month [3, 4]. Taking Tiktok as an example, it had 1.4 billion monthly active users in 2022, and it is predicted to reach 1.8 billion by the end of 2023 [5]. Billions of users imply a huge number of video streams. According to TikTok's report [5], globally, 167 million hours of short videos are consumed every minute, putting an enormous pressure on the current Internet infrastructure.

Traditional video transmission solutions maintain videos on the cloud and stream them to users via content delivery networks (CDNs) [6, 7]. A major weakness of CDN is the huge redundant traffic, causing network pressure and affecting the user viewing experience. Edge caching approaches [8–10] can reduce the amount of redundant traffic by caching user-desired contents at edge nodes [11–13]. However, in short video services, users have different preferences determined by their hobbies, culture, etc. Even though videos are rarely watched in groups, they might be popular with others [14]. Therefore, it is difficult to identify which are the most popular videos among various ones. As a result, it is difficult for edge nodes to cache adequate user-desired contents due to limitations in cache capability, leading to low caching efficiency [15].

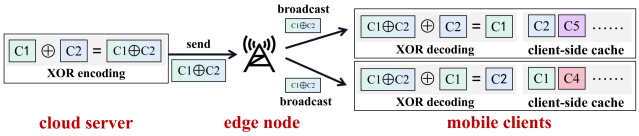


Figure 1: File flows example for NCTM

Recent studies [16, 17] have revealed that using client-side caches within a peer-to-peer (P2P) network can significantly reduce bandwidth pressure caused by redundant video transmissions [18]. P2P is a promising solution to increase the cache hierarchy and can compensate for the limited edge node capability. However, the majority of short video deliveries are from mobile devices, e.g., 97% of TikTok video deliveries [5]. The expensive uploading data bills or the data cap make P2P not a feasible solution. Therefore, we aim to make better use of the client-side cache to compensate for the lack of edge cache capacity, while avoiding the upload traffic costs.

Coded cache [19–22] could provide a solution. It is based on merging two separate video files into one coded file at the server and forward it to the destination. The most commonly encoding approach is XOR-merging (Exclusive OR). After receiving the coded file, the destination device separates the original files by locally cached content prefetched during network idle time. So the coded cache technique requires fewer transmission files to transmit all data. But it also means some contents need to be cached in the destination device before use, and devices need to be active simultaneously to receive the coded files in time. However, in practice, short video applications generally do not allow network usage when they are closed, and the server lacks mechanisms to confirm the apps running status. The above-mentioned problems hinder the practical deployment of the coded cache solution.

In this paper, we propose an innovative **Novel Coded Transmission Mechanism (NCTM)** for efficient short video delivery. With deep insight on the characteristics of short video delivery, NCTM avoids content prefetching or uploading through an effective cooperation between the cloud, edge, and mobile clients. Particularly, NCTM caches videos that the user **has already watched** for subsequent decoding rather than prefetching ones.

As shown in figure 1, the cloud server in NCTM applies XOR operations to combine video files into a single coded file, and sends it to the edge node (coded transmission). Then the edge node multicasts the coded file to specific mobile clients. The mobile clients will decode the coded file by applying XOR operations with the previously watched and cached videos. To make sure that the coded file can be successfully decoded, we should divide the clients into multiple groups, ensuring that any two clients within a group have previously watched and cached the files being requested currently by each other (group divided problem). So we introduce a `User Cache Table (UCT)` and a `Transmission Matching Graph (TMG)` that record the cache status and explore coded transmission opportunities. Recognizing the similarity between the above-mentioned requirement and the mathematical concepts of "cliques", we model the group divided problem as the clique cover problem [23, 24] and propose the novel *Minimum Clique Coverage algorithm*. This algorithm involves multiple linear-time searches to find a sub-optimal solution. Furthermore, we introduce the new *Recommendation Reorder algorithm*, which modifies the playback order by bringing forward videos scheduled for later to create more opportunities for coded transmission. Finally, we propose a *Client-side Cache Update method* based on the video recommended queue, which is informed by the recommendation system in short video services.

To evaluate the performance of NCTM, we utilized the kuaiRec dataset [25] and collected real user interactions with the Kuaishou app on August 12, 2020, totaling 117,977 records. For each client, edge node, and cloud server, we created separate docker containers [26] to simulate video requests and playback behaviors based on their app usage time. The results show that under sufficient bandwidth, NCTM reduces the average network load by 3.02%-14.75%, and cuts the peak traffic by 23.01%. With limited bandwidth, NCTM reduces 43%-85% of rebuffering events and increases video occupancy in the user's buffer by 1.21x-13.53x. Furthermore, we use the NCTM to assist the edge caching, proving that they are compatible. In addition, we demonstrate that NCTM can achieve real-time performance on the server and provide reference values for the hyperparameters in the proposed approach.

In summary, our contributions are as follows:

- We propose the innovative **NCTM** that sends XOR-coded files and utilizes the client-side cache to store watched videos for file decoding. This approach reduces the network load without data prefetching or uploading.
- We design the *Minimum Clique Coverage algorithm* and the *Recommendation Reorder algorithm* to find a relatively optimal solution for the client dividing problem with linear time complexity, which ensures successful decoding for the coded files in coded transmissions. The *Client-side Cache Update method* based on the video recommended queue is also proposed.
- We devise trace-driven experiments to emulate the NCTM and verify its effectiveness in the reduction of bandwidth consumption, buffer variation, and rebuffering frequency.

The remaining of the paper is structured as follows. In section 2, we summarize the NCTM related work, in section 3, we illustrate the overall structure of NCTM and in section 4, NCTM is described in details. In section 5, the experimental results are presented. Finally, we give a brief conclusion of our work in section 6.

2 RELATED WORKS

The popularity of short video applications leads to massive video traffic and introduces a significant load on CDNs.

Edge caching [27, 28] is a key approach to alleviate the CDN load. It utilizes cache-enabled edge servers, such as base stations and smart gateways, to store popular contents, so that these contents can be transmitted directly from the caches instead of from the remote cloud [11]. As these cache edges are closer to the users, there is a reduction in the traffic load on the core network. The cache hit rate is an essential metric to evaluate the performance of edge caching. For short video deliveries, [29] proposes to consider the number of views and likes as the popularity basis to choose the edge cache content. Furthermore, [30] takes user preferences into consideration. To achieve a higher hit rate, [31] introduces a multi-agent deep reinforcement learning algorithm where each edge learns its own best policy.

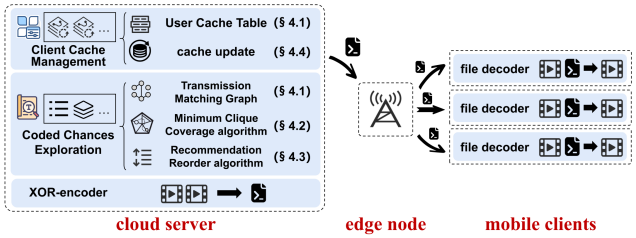


Figure 2: Overall structure of NCTM

P2P-CDNs [32–35] enables content caching on mobile phones by utilizing wireless channels for cache sharing, also known as P2P sharing. With a P2P approach, users can send their stored content to other users, effectively balancing the upstream and downstream transfers, and increasing the cache hierarchy [36]. [37] considers different network environments. High-quality Internet access shares its resources (e.g., bandwidth) with slower or unreliable ones. [38] takes advantage of user social information (friends/ followers), using the friend list to identify additional P2P sharing.

Coded cache [19–22] utilizes network coding techniques, which aggregate (encode), multicast, and separate (decode) data messages in the cloud, edge node, and client devices, respectively. This technique results in lower traffic load for data transmission. For example, if user A caches video file c_1 and user B caches video file c_2, when user A requests video file c_2 and user B requests video file c_1, the server can transmit a single coded file $c_1 \oplus c_2$ (the symbol $\oplus$ represents bitwise XOR) instead of two separate files c_1 and c_2. It can deliver the coded file simultaneously to both users by multicasting at the edge node, and users can decode it based on their local files (e.g., for user A: $c_1 \oplus c_2 \oplus c_1 = c_2$).

The previous works made important contributions in terms of edge caching, P2P-CDNs and coded cache solutions individually, but they have not combined them in an approach that reduces the load for short video deliveries as proposed by NCTM.

3 NCTM ARCHITECTURE AND PRINCIPLE

The overall NCTM architecture is shown in figure 2. The mobile clients cache their watched video files based on their cache capabilities, and the cloud server sends the coded files to the edge node. Then the edge node multicasts the coded file to the mobile clients. These clients decode the coded file based on the cached files to obtain the desired one. To make sure the coded file can be decoded, we need to ensure that all the files involved in the coded file have been cached at the clients, except the desired one. So the challenge lies in finding **how to efficiently and accurately identify the clients with the above cache situation among multiple ones**. We employ the following designs to address this challenge.

The cloud server includes three key modules: **Client Cache Management**, **Coded Chances Exploration**, and **XOR-encoder**. The Client Cache Management informs the cache status of the client's devices. We design the `User Cache Table (UCT)` (section 4.1) to record the cached files in clients. Due to the limited cache capability, we specify the cache update policy (section 4.4). The Coded Chances Exploration module finds the coded transmission opportunities, thus minimizing bandwidth consumption. Additional, we design the `Transmission Matching Graph (TMG)` and reduce the chances exploration problem to the clique cover problem [23, 24] (section 4.1). To solve this NP-Hard problem, we designed the *Minimum Clique Coverage algorithm* to find a sub-optimal solution within linear time complexity (section 4.2). Moreover, based on our observation, switching the playback order of short video can create more coded transmission opportunities, so we further proposed *Recommendation Reorder algorithm* (section 4.3). File encoder merges several origin video files into one coded file by XOR before transmission.

The edge nodes copy the coded file and multicast it to clients. Clients decode the coded file and extract the original files, as desired.

4 CODED TRANSMISSION MECHANISM

4.1 Definitions

We assume that there are N short video files, denoted as $\mathbb{C} = \{C_1, C_2, ..., C_N\}$. Short videos are commonly delivered with an adaptive bitrate paradigm [39, 40]. Now we assume that each short video consists of only one video chunk; this will be generalized to the common case in the next section. Suppose there are K mobile clients, denoted as $\mathbb{U} = \{U_1, U_2, ..., U_K\}$. Each client is assigned an recommendation queue. We denote v_{ij} as the j-th video pushed to the i-th user ($v_{ij} \in \mathbb{C}$ determined by the recommendation system). Therefore, for the user U_i, the recommendation queue can be denoted as $V_i = \{v_{i1}, v_{i2}, ...\}$. At this point, we define the `UCT` as $\mathbb{T} = \{t_{ij}\}$, where $t_{ij} = 1$ indicates that user U_i has **watched and cached** video C_j, and vice versa ($i \in [1..K]$, $j \in [1..N]$). Thus, for the user U_i, all the videos in their client-side cache can be recorded as $T_i = \{C_j | t_{ij} = 1, j \in [1..N]\}$.

For example, here we assume that user U_a has cached video C_j, and user U_b has cached video C_i. It can be denoted as $t_{aj} = 1$ and $t_{bi} = 1$, respectively. When the user U_a needs the video C_i, and user U_b needs video C_j, the coded file $C_i \oplus C_j$ can be transmitted. So the condition for coded transmission is that **they have cached the files needed by each other** (e.g., $t_{aj} = 1$, $t_{bi} = 1$), which is to ensure successful decoding in the user device (e.g., for user U_a, $(C_i \oplus C_j) \oplus C_j = C_i$, where C_j is watched and cached video file of user U_a). This is defined as a binary-coded transmission.

Similarly, we assume that there are three users U_a, U_b, and U_c who need videos C_i, C_j, and C_k, respectively. When $t_{aj} = t_{ak} = 1$, $t_{bi} = t_{bk} = 1$ and $t_{ci} = t_{cj} = 1$, indicating that each user has cached the files needed by other two users, the coded file $C_i \oplus C_j \oplus C_k$ can be transmitted. The client devices can also decode the coded file with their cached video files. (e.g., for user U_a, decoding can be achieved through $(C_i \oplus C_j \oplus C_k) \oplus C_j \oplus C_k = C_i$). This is defined as a ternary-coded transmission. Similar operations can be generalized to multivariate-coded transmission involving x video files ($x \geq 2$).

Clearly, a larger value of x indicates merging more video files into one coded file, resulting in fewer number of transmissions and less bandwidth consumption. Therefore, we prefer the coded transmission that covers more users. To make sure the client devices can successfully decode, multivariate-coded transmission requires that any two users satisfy the condition for binary-coded transmission. This requirement can be associated with the mathematical concept of "cliques". So we design the `TMG` as $\mathbb{G} = (E, D)$ to model the multivariate-coded transmission exploration problem as a graph theory problem. D is the set of vertices representing the active users

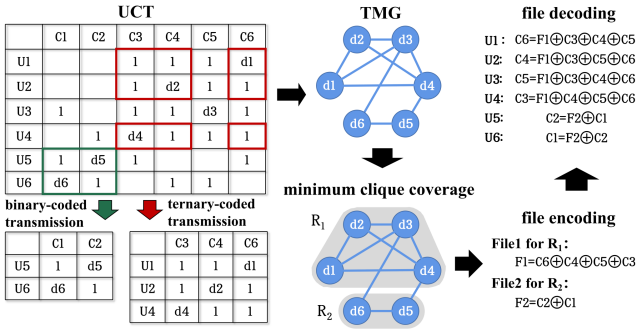


Figure 3: Example of NCTM with 6 users and 6 videos

currently. If $d_i \in D$, it means the user U_i is **watching videos**. E is the set of edges. If $e_{ij} \in E$, it means there is an edge between nodes i and j, indicating that users U_i and U_j currently satisfy the **condition for binary-coded transmission**.

Let us denote the watching video of user U_i as v_i^* ($v_i^* \in V_i$). In this case, for users U_i and U_j, if $t_{iv_j^*} = t_{jv_i^*} = 1$, binary-coded transmission can be achieved, i.e., $e_{ij} \in E$. Following this rule, we can construct the $\mathbb{G}$ with K' nodes, where K' is the number of active users ($K' \leq K$). In TMG, if we can find a clique with x nodes, it means that among these x users, any two users satisfy the condition for binary-coded transmission, thus satisfying the condition for multivariate coded transmission involving x users. On the other hand, all user requests need to be fulfilled, so any node in TMG needs to be covered by a clique (a single node is also considered a clique, we define it as the separate clique). Therefore, the problem can be transformed into finding the **minimum number of cliques** that **cover all nodes** in the TMG. We call this the group divided problem.

Formally, we denote $\mathbf{R} = \{R_1, R_2, ...\}$ as all cliques in TMG, where the i-th clique is represented as $R_i = \{d_x, d_y, ...\}$. We must ensure $R_1 \cup R_2 \cup ... = D$ and minimize $|\mathbf{R}|$. This problem is a classical clique cover problem and has been proven to be NP-hard when the degree of vertices is greater than 6 [23, 24]. In this paper, we propose the *Minimum Clique Coverage algorithm*, which utilizes the previous result and a single traversal to solve this problem with linear time complexity. The details are presented in section 4.2.

Figure 3 illustrates an example containing 6 users and 6 videos, and gives intuitive cases of binary-coded transmission and ternary-coded transmission. In the example, two cliques are used to cover the TMG, with $R_1 = \{d_1, d_2, d_3, d_4\}$ and $R_2 = \{d_5, d_6\}$. Therefore, in the coded transmission, two coded files need to be transmitted. F_1 merging 4 video files C_6, C_4, C_5, C_3 and F_2 merging 2 video files C_2, C_1. They will be decoded in the client devices with cached video files. In practical deployment, we group users with close geographic proximity and similar playback queues together to perform coded transmission. So the scale of the UCT and the TMG will not be very large. In our experiments, there are 1398 users and 10230 videos, reflecting the coded transmission processes of a single group of users. In this situation, the storage capacity for the UCT is 202MB, which does not impose a significant load on the server.

4.2 Minimum Clique Coverage algorithm

To ensure smooth video playback, short video transmission uses the adaptive bitrate paradigm. We define a video C_i as composed

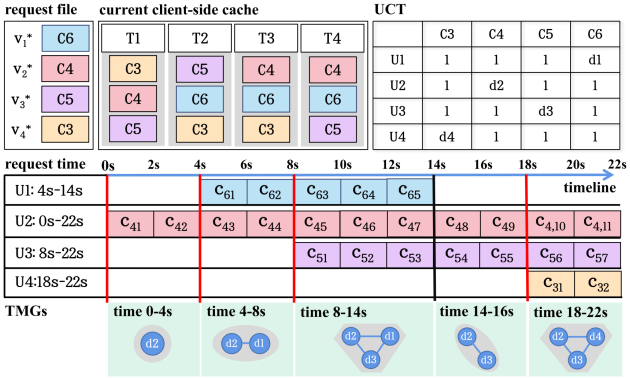


Figure 4: An example of asynchronous user requests

of multiple video chunks, denoted as $C_i = \{c_{i1}, c_{i2}, ...\}$. Note that the UCT and TMG are dynamically maintained as the videos play.

As shown in figure 4, users $U_1 - U_4$ have already watched and cached parts of the videos. For example, the user U_1 cached the videos C_3, C_4, C_5, i.e., $T_1 = \{C_3, C_4, C_5\}$. Now let us consider that user U_1, U_2 is watching video C_6, C_4, i.e., $v_1^* = C_6$, $v_2^* = C_4$, and so on. User U_1, U_2 watches video C_6, C_4 within the time range of 4-14 and 0-22 seconds, and so on. Therefore, the TMG also changes within different time intervals. The figure depicts the TMGs corresponding to the current 5 intervals. At this point, encoding operations are performed between different video chunks rather than the entire video (for example, at the 8th second, $c_{63} \oplus c_{45} \oplus c_{51}$ is encoded for transmission). Now the TMG changes only when a user either completes a video playback (e.g., 14th second) or starts watching a new video (e.g., 0th,4th,8th,18th second). During each time interval, the TMG remains unchanged. Note that the example above illustrates a simplified scenario where the video bitrates are the same. We discuss the more general case in the appendix C.

After completing a video playback, the user will leave the current TMG. Since the subgraph of a clique remains a clique, the removal of a node will not break the current clique. In the given example, at the 14th second, user U_1 (node d_1) leaves the TMG, but users U_2 (node d_2) and U_3 (node d_3) can still form a clique. When a user starts playing a new video, a new node is added to TMG, but the cliques formed by all the existing nodes remain unchanged. So we can decide whether the new node can be added to an existing clique (the number of cliques unchanged) or the new node forms a separate clique (number of cliques increase by 1). For example, at the 8th second, user U_3 joins the TMG as a new node d_3. According to the rules, node d_3 has edges with nodes d_1 and d_2, so it can be added to this clique, forming a clique with three nodes and achieving ternary-coded transmission.

Therefore, whenever a new request arrives, we need to add a new node to the TMG and try to incorporate this node into existing cliques. We can judge whether the incorporation condition is met by iterating through all existing cliques and checking whether all nodes can connect to the new node. If a suitable clique is found, the new node will be added to this clique, maintaining the number of cliques and the coded files to be transmitted unchanged. Otherwise, the new node forms a separate new clique, leading to an increase in both the number of cliques and the coded files to be transmitted.

However, the aforementioned approach often yields poor performance, as shown in figure 5. As new requests from user U_5 and

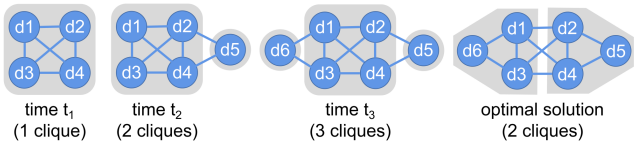


Figure 5: Naive solutions fail to achieve optimal solutions

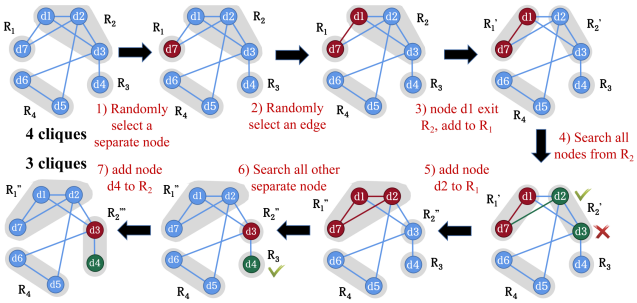


Figure 6: Example of Minimum Cliques Coverage algorithm

U_6 arrive at time t_2 and t_3, new nodes d_5 and d_6 are added to TMG. Since they do not have edges connecting to all nodes in the existing clique, they will form new cliques. Clearly, the optimal solution involves two cliques, but the result at time t_3 contains three. To address this issue, we propose a search algorithm with linear time complexity, the *Minimum Cliques Coverage algorithm*, to find a sub-optimal solution based on the previous results. The algorithm includes T iterations, where T is a hyperparameter that controls the computational cost. In each iteration, following steps are executed:

(1) Randomly select a node d_i among all separate cliques. We assume that d_i comes from the clique R_x.
(2) Randomly select an edge e_{ij} from all the edges connected the node d_i. Obviously e_{ij} connects to node d_j. We assume that d_j comes from the clique R_y.
(3) Let d_j exits original clique R_y and joins clique R_x where d_i belongs to. The processed cliques are denoted as $R_y^{'}$ and $R_x^{'}$.
(4) Iterate through all the nodes (d_k) from $R_y^{'}$. Check if there exist nodes that have edge connected to d_i ($e_{ik} \in E$).
(5) Make these nodes (d_k) exit original clique $R_y^{'}$ and join clique $R_x^{'}$. The processed cliques are denoted as $R_y^{''}$ and $R_x^{''}$.
(6) Iterate through all remaining separate cliques (d_k). Check if one can be added to clique $R_y^{''}$ after excluding some nodes.
(7) If such a separate clique exists, node d_k joins the clique $R_y^{''}$. The processed clique is denoted as $R_y^{'''}$. Now a solution is found and the algorithm terminates. Otherwise, all adjustments are retained and the next iteration starts.

The pseudocode is shown in appendix A, and we prove this algorithm in appendix B. During the iteration, if such a separate clique is found in Step 7, it means that we merge two separate cliques (R_x, R_z) and one non-separate clique (R_y) into two non-separate cliques ($R_x^{''}, R_y^{'''}$). The number of cliques and the coded files to be transmitted reduces by one, and the algorithm terminates. Otherwise, we still preserve the above operations so that the next iteration can discover more matching opportunities. Figure 6 illustrates an example of the *Minimum Cliques Coverage algorithm*. Here the separate cliques $R_3(d_4)$ and $R_1(d_7)$ are merged with the clique R_2 containing three nodes $\{d_1, d_2, d_3\}$ to form the new clique $R_1^{''}$ with three nodes $\{d_1, d_2, d_7\}$ and the new clique $R_2^{'''}$ with two nodes $\{d_3, d_4\}$. At this point, the clique R_3 disappears.

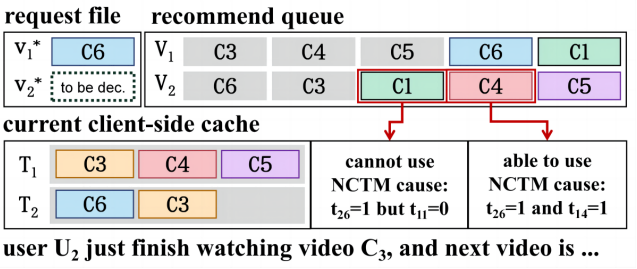


Figure 7: Example of Recommendation Reorder Algorithm

This algorithm is executed whenever a new request arrives and the newly added node in the TMG cannot join an existing clique (i.e., forming a separate clique). In each round, all nodes are traversed at most three times, resulting in a computational complexity of $O(KT)$, where K is the number of users in the current TMG, and T is a hyperparameter that controls the number of iterations. We will further explore the time complexity of this algorithm and the impact of T on the success rate of the search in our experiments.

The major benefit of the above algorithm is that we do not rely on any prior knowledge (such as video popularity or predictions of user viewing behavior). Instead, we dynamically adjust the cliques through the UCT and TMG. This allows us to determine which users currently satisfy the coded transmission conditions. Furthermore, with the *Minimum Clique Coverage algorithm*, we utilize the historical results to explore a better solution for the clique cover problem in linear time complexity. This allows us to use fewer cliques, which means fewer coded files to be transmitted, thereby reducing bandwidth consumption.

4.3 Recommendation Reorder algorithm

Based on our observations, in some cases, playing a video located further in the recommended queue may create more chances for coded transmission than playing the very next video. A typical example is presented in figure 7.

The example contains two users, U_1 and U_2. User U_1 has a recommendation queue $V_1 = \{C_3, C_4, C_5, C_6, C_1, \ldots\}$, and videos C_3, C_4, C_5 have been watched and cached in the client device, i.e., $T_1 = \{C_3, C_4, C_5\}$. User U_1 is watching the video C_6, i.e., $v_1^* = C_6$. Similarly, $V_2 = \{C_6, C_3, C_1, C_4, C_5, \ldots\}$, and $T_2 = \{C_6, C_3\}$. Since the previous video has just ended, no video is being played by user U_2. Now we find that if video C_1 is played next based on the recommendation queue, since $t_{11} = 0$, NCTM cannot be used between users U_1 and U_2. However, if video C_4 is chosen (also in the recommendation queue, but not the very next one), we have $t_{26} = t_{14} = 1$, and NCTM can be used. Specifically, if the user U_2 chooses video C_1 as the next one, there will be no edges connecting to node d_2 in the TMG, and d_2 will inevitably form a separate clique. However, if U_2 chooses video C_4, in the TMG, the node d_2 will have at least one edge. Even if it forms a separate clique, with the *Minimum Clique Coverage algorithm*, it also offers the potential for coded transmission.

For push-playback short videos, there are no content correlations between two consecutive videos. Therefore, changing the playback order of videos without altering their content will not significantly

affect the user experience. The above example illustrates that in certain situations, reordering the recommendation queue can create more opportunities for coded transmission. Given that the users might end, skip, or re-watch videos at any time, this makes it difficult to predict users' viewing behaviors. Therefore, we propose the *Recommendation Reorder algorithm*, which focuses on short-term benefits. Whenever a user switches videos, this algorithm will filter the videos that can trigger **NCTM** based on the current UCT and select one based on the recommendation queue. This method does not change the video contents, but changes the video playback order by prioritizing the ones that enable coded transmission to create more chances for coded transmission. The pseudocode is shown in appendix D. The algorithm considers the next G videos in the recommendation queue of user U_i. For each video, it iterates through all users to determine if the binary-coded transmission can be used. The algorithm can be described as the following process:

(1) Traverse the next G videos in V_i, denote as C_j.
(2) For each C_j, traverse all other active users, denote as U_k. Check if one satisfies the requirement of binary-coded transmission, i.e., $t_{kj} = t_{iv_k^*} = 1$.
(3) If such a user exists, let $v_i^* = C_j$. That is, the next video to be played is C_j. Otherwise, still play the first video in V_i.

Here G is a hyperparameter that decides the maximum number of look-forward videos as well as limits the computational cost. This algorithm involves two iterations. So the time complexity is referred to $O(GK)$, where K represents the number of users watching videos. To further prune the search path, we can avoid the case where $t_{iv_j^*} = 0$ during the first traversal, i.e., excluding the users who are watching the video not cached by user U_i. That is because the user U_i can not perform coded transmission whatever the next video is (it is impossible to change others playing videos).

Currently, most short video applications pre-cache the first video chunk of several subsequent videos in the recommendation queue, to alleviate the stalling and rebuffering issues caused by rapid video switching. For example, TikTok pre-caches the first video chunk of the next five videos [41]. Assuming the application pre-caches the first video chunk of P subsequent videos, it should ensure $G \leq P$ to mitigate the risk of increased stalling and rebuffering events.

4.4 Client-side cache update method

As discussed in Section 1, both edge caching and client-side cache have limited storage capabilities. Compared to edge caching, user devices (such as smartphones) may be more restricted. In this section, we will discuss how to perform cache placement/replacement to better cooperate with the *Minimum Clique Coverage algorithm* and *Recommendation Reorder algorithm*.

We define the maximum storage capacity for user U_i as M_i, thus we need to ensure $\sum_{C_j \in T_i} |C_j| \leq M_i$, where T_i is the cached videos of user U_i and $|C_j|$ represents the file size of video C_j. Suppose that after watching video C_k, $(\sum_{C_j \in T_i} |C_j|) + |C_k| > M_i$, indicating that the user U_i cannot store these videos on user device. Therefore, it is necessary to determine which content should be replaced.

For video C_k, we cache video files so we can decode the coded files with the assistance of them in the future. Hence, cached videos C_k are only valuable if they are watched by other users later. Using the "push playback" style, a recommendation system can keep track of the video playback list. Therefore, we can estimate the earliest possible time when the video C_k will be used for file decoding. As NCTM mainly focuses on short-term benefits, we prefer the files that can create coded transmission opportunities in the short term and replace those that would take longer to be used.

Formally, we define $F(C_k)$ as the earliest occurrence time of video file C_k in the recommendation queue. That is, $F(C_k) = \min(j)\ s.t.\ v_{ij} = C_k\ for\ all\ i \in [1..K]$, where we find the smallest j among all K users' recommendation queues such that $v_{ij} = C_k$. This is the earliest possible time when the current video could be used for decoding. For the user U_i, we can sort the videos in T_i by $F(C_k)$ and replace the videos that would take longer to be used.

Note that from the perspective of the *Recommendation Reorder algorithm*, it can be regarded as $F^*(C_k) = \max\{1, F(C_k) - G\}$, where G is the hyperparameter defined in the algorithm description. This is because the *Recommendation Reorder algorithm* can look ahead G videos based on the current recommendation queue, thus this video could potentially be played ahead by up to G videos. Additionally, when we consider edge cache, if video C_k is cached in the edge cache, it means that requests involving video c_k will not use NCTM and can be regarded as $F^*(C_k) = +\infty$.

4.5 Further considerations

Besides the mentioned points above, there are many other aspects that need to be taken into consideration. For the uncompleted watching events, we will consider weighted *TMG* in our subsequent work to explore more stable coded transmission opportunities. Furthermore, short video playback on mobile devices also exhibits a significant degree of randomness. For instance, users may slide the progress bar to skip some uninteresting scenes, network conditions can affect the video bitrate, and user movements can impact the connection status of edge nodes. We have taken these issues into consideration and discussed them in detail in appendix C.

5 EVALUATION

5.1 Methodology

We used the **traditional CDN delivery** and **edge cache** (size of 500MB with a LRU update method) approaches for comparison to evaluate the performance of NCTM. To replicate the users video-watching behavior, we utilized the kuaiRec dataset [25] and collected 117,977 real request records from 1,398 users on the Kuaishou app on August 6, 2020, involving 10,230 short videos. To simulate the network conditions, we used the MAWI [42] and FCC18 [43] network traces from August 12, 2020. In the experiment, we created separate Docker [26] containers for the server, the edge node, and each user device. Users sent requests to servers via edge nodes according to the real request records mentioned above. Additionally, we used the Mahimahi network tool [44] to replay the network trace, aiming to closely reproduce the network conditions at that time. In the experiments, we considered two fundamental scenarios: the sufficient bandwidth network and the limited bandwidth network. For the former one, we mainly considered the network bandwidth utilization. For the latter one, we mainly focused on buffer variation and rebuffer events. More detailed settings are shown in appendix E.

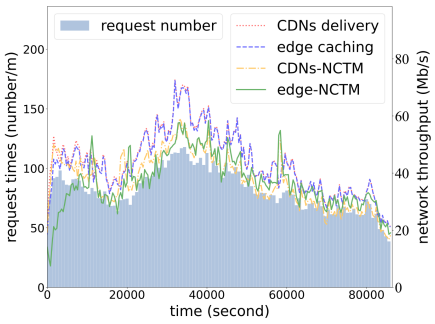


Figure 8: The network throughput variation under sufficient bandwidth

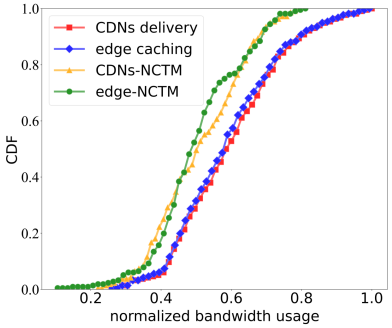


Figure 9: Normalized bandwidth usage under sufficient bandwidth

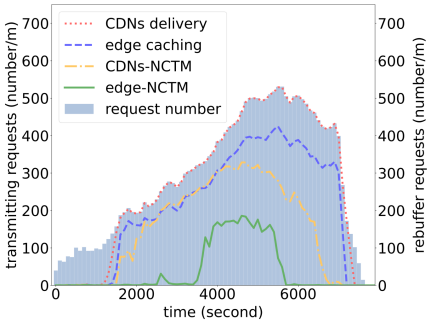


Figure 10: The distribution of rebuffer events under limited bandwidth

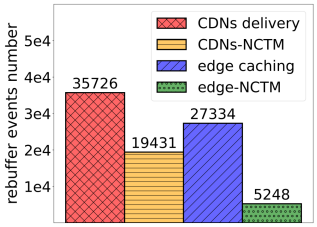


Figure 11: Statistics of rebuffer events under limited bandwidth

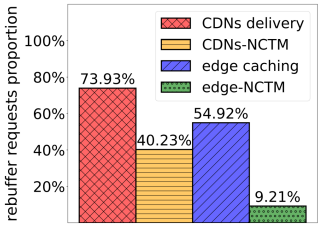


Figure 12: Proportion of rebuffer request under limited bandwidth

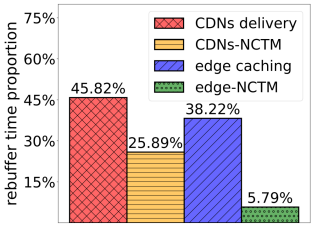


Figure 13: Average proportion of rebuffer time under limited bandwidth

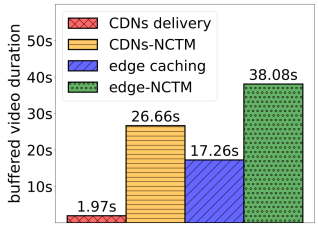


Figure 14: Average buffered video duration under limited bandwidth

5.2 Performance

Under sufficient bandwidth: Figure 8 presents the network load variations under high-speed connections (200Mbps) across CDN and edge caching approaches with and without NCTM, as well as a histogram of short video requests distribution. The network load variations are closely related to the request frequency. The peak value of requests occurs at around the 29700th second. During this time, there are 118 requests within one minute (1398 users), and the network load also reached its peak value. In CDNs delivery mode, the real-time throughput was 69.94 Mb/s and 53.84 Mb/s without and with NCTM, cutting peak traffic by 23.01%. Similar results are observed in the edge caching mode, indicating that NCTM can effectively alleviate peak throughput pressure. Figure 9 presents the cumulative distribution function (CDF) of the normalized bandwidth usage. It can be observed that NCTM can significant reduce bandwidth usage. In CDNs delivery mode/edge caching mode (edge cache size of 500MB), the average network load decreased by 14.06%/13.30% with NCTM versus no NCTM. As a result of the change in the order of video playback, there may be a short period of time when NCTM's throughput exceeds that of baselines. From a global perspective, the NCTM is clearly superior to the baseline.

Limited bandwidth scenario: Figure 10 shows the distribution of rebuffer events using NCTM in both CDNs delivery mode and the edge caching mode in a limited bandwidth scenario (60Mbps). Figure 11 and 12 show the statistics of rebuffer events and the proportion of rebuffered requests. It can be observed that due to the limited bandwidth, in the CDN delivery mode, 73.93% of user requests suffer from rebuffer events. During peak request times, almost all users experience rebuffer events, resulting in 35,726 rebuffer instances. NCTM relieves some of the bandwidth pressure, resulting in a 40.23% reduction in rebuffer events for user requests. This represents a decrease of 33.7% compared to the baseline. Total rebuffer instances decreased by 45.6% to 19,431. Similarly, in the edge caching mode, with the efficient cooperation of NCTM and edge caching, the number of user requests experiencing rebuffer events has been reduced to only 9.21%, and the total count of rebuffer events has decreased by 80.8% compared to the baseline.

Figure 13 shows the average proportion of rebuffer time across CDN and edge caching approaches with and without NCTM. Figure 14 shows the average buffered video duration during video playback in client devices. Due to frequent rebuffer events in the CDN delivery mode, the average buffered video duration is only 1.97 seconds. About 45.82% time of video watching is waiting for rebuffering. With NCTM, the average buffered video duration increases by 13.53x to 26.66 seconds, mitigating the effects of fluctuations in network performance. Similarly, in the edge caching mode, the average buffered video duration increases by 1.21x, and the proportion of buffer time decreases to only 5.79%. These significantly improve user experience within limited network bandwidth conditions.

Dynamic bandwidth scenario: Figure 15 illustrates the average bandwidth usage in different delivery modes in a scenario where the network bandwidth ranges from 60Mbps to 100Mbps. When the bandwidth is relatively abundant (100Mbps 80Mbps), NCTM reduces the bandwidth usage rate by 3.02%-14.75%. Under the constrained bandwidth (70Mbps), NCTM significantly alleviates the bandwidth pressure, reducing the time of full bandwidth occupancy by 15.28%, substantially reducing rebuffering events.

NCTM assisting edge caching: To further explore the cooperation between NCTM and edge caching, we analyzed the number of requests in different transmission stages, including the number

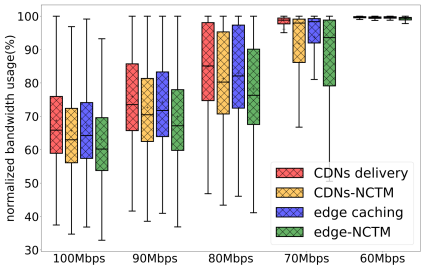


Figure 15: Average bandwidth usage under 60Mbps-100Mbps

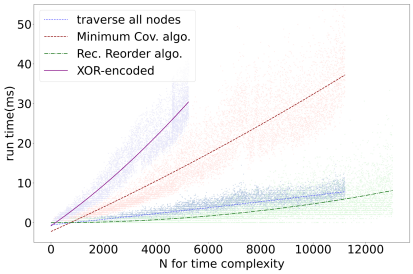


Figure 16: Time complexity of NCTM-related algorithm

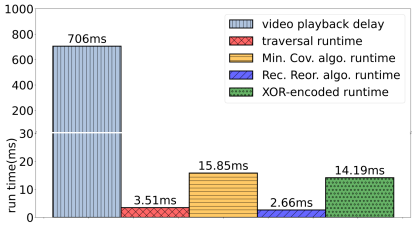


Figure 17: Average execution time of related algorithms and request delay

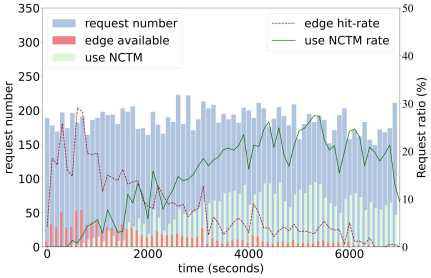


Figure 18: Cooperation between NCTM and edge caching

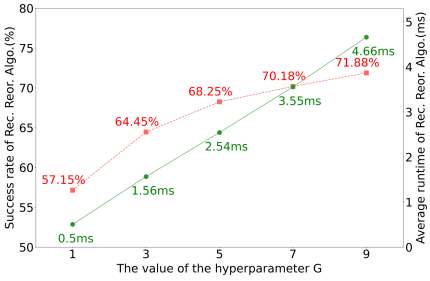


Figure 19: Average execution time and success rate in different T

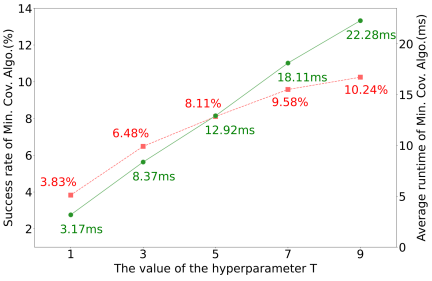


Figure 20: Average execution time and success rate in different G

of hit-edge-cache requests and the number of NCTM requests, as shown in Figure 18. In the early stages, when the response workload of the edge cache is low, LRU can satisfy over 20% of the requests. However, with the video files increasing in the later stages, the edge cache becomes overwhelmed. On the contrary, NCTM does not occur frequently in the early stages due to limited cache content. But in the later stages, as user caches accumulate, many requests (over 25%) can use NCTM. Therefore, edge caching and NCTM are not only compatible, but they are complementary and further reduce the network load by cooperation.

5.3 Time complexity and hyperparameters

Time complexity: We design an experiment to compute the time complexity of NCTM in order to assess NCTM's capability to process the received requests in real-time. Figure 16 illustrates the time complexity of NCTM. Figure 17 illustrates the relationship between the average execution time of NCTM and short video playback latency. The execution time of NCTM primarily includes four distinct components., i.e., cliques traversal, two algorithms, and the XOR encoding. Figure 16 shows four fitting curves, illustrating the actual execution time of the algorithms with different entity counts (N for time complexity), such as the number of current nodes or cliques. It can be concluded that both the two algorithms show nearly linear growth, consistent with our analysis. Besides, the average execution time from opening the APP to the start of the first video playback is 706ms (tested with the Chrome browser for Tiktok). Considering that NCTM's execution time is much shorter than the video playback delay, we can conclude that NCTM can process requests in real time for short videos.

Hyperparameter analysis: In the above section, we set hyperparameters T and G to control the computational cost. To determine the values of T and G, we design experiments with values $\{1, 3, 5, 7, 9\}$ and calculated the success rate for finding solutions and execution time under each parameter setting. Results in Figure 19 and figure 20 reveal that both T and G exhibit a nearly linear increase in average execution time as their values increase. When $T = 5$, the probability of reducing the number of cliques in the *Minimum Clique Coverage algorithm* is 8.11%, with an average execution time of 12.92ms; when $G = 5$, the probability of finding videos in the *Recommendation Reorder algorithm* is 68.25%, with an average execution time of 2.54ms. As T and G further increase, the success rate of finding solutions approaches a plateau, indicating that $T = 5$ and $G = 5$ are relatively optimal values.

6 CONCLUSIONS

As short videos become increasingly common, they put a significant strain on network resources due to the massive amount of data they transmit. Due to the time- and space-intensive nature of short videos, CDNs are faced with considerable bandwidth challenges. Moreover, edge caching and other solutions rely too heavily on predicting popular files. In this paper, we propose NCTM, which employs XOR-encoded files to make effective use of user-side caches, reducing bandwidth consumption and improving transmission efficiency. Trace-driven experiments show that compared to CDNs and edge caching, NCTM reduces bandwidth consumption, rebuffering events, and the reliance on caching popular files, ultimately improving the user experience when watching short videos.

7 ACKNOWLEDGEMENTS

This work is supported in part by the National Key R&D Program of China under grant No. 2022YFB3105000, the Major Key Project of PCL under grant No. PCL2023A06, and the Shenzhen Key Lab of Software Defined Networking under grant No. ZDSYS20140509172-959989. G.-M. Muntean thanks Science Foundation Ireland for grants 12/RC/2289_P2 (Insight) and 21/FFP-P/10244 (FRADIS).

REFERENCES

[1] The official website of tiktok. (2023, Jul 26).
[2] The official website of YouTube Shorts. (2023, Jul 26).
[3] TikTok: Thanks a billion! (2023, Jun 24).
[4] YouTube Shorts Video-Making App Now Receiving 3.5 Billion Daily Views. (2023, Jun 24).
[5] TikTok Revenue and Usage Statistics (2022). (2023, Jun 24).
[6] Gang Peng. Cdn: Content distribution network. *arXiv preprint cs/0411069*, 2004.
[7] Fahao Chen, Peng Li, Deze Zeng, and Song Guo. Edge-assisted short video sharing with guaranteed quality-of-experience. *IEEE Transactions on Cloud Computing*, 2021.
[8] Sem C. Borst, Varun Gupta, and Anwar Walid. Distributed caching algorithms for content distribution networks. In *Conference on Information Communications*, 2010.
[9] Hanling Wang, Qing Li, Heyang Sun, Zuozhou Chen, Yingqian Hao, Junkun Peng, Zhenhui Yuan, Junsheng Fu, and Yong Jiang. Vabus: Edge-cloud real-time video analytics via background understanding and subtraction. *IEEE Journal on Selected Areas in Communications*, 41(1):90–106, 2022.
[10] Ying Chen, Qing Li, Aoyang Zhang, Longhao Zou, Yong Jiang, Zhimin Xu, Junlin Li, and Zhenhui Yuan. Higher quality live streaming under lower uplink bandwidth: an approach of super-resolution based video coding. In *Proceedings of the 31st ACM Workshop on Network and Operating Systems Support for Digital Audio and Video*, pages 74–81, 2021.
[11] Ju Ren, Deyu Zhang, Shiwen He, Yaoxue Zhang, and Tao Li. A survey on end-edge-cloud orchestrated network computing paradigms: Transparent computing, mobile edge computing, fog computing, and cloudlet. *ACM Computing Surveys (CSUR)*, 52(6):1–36, 2019.
[12] Yushan Siriwardhana, Pawani Porambage, Madhusanka Liyanage, and Mika Ylianttila. A survey on mobile augmented reality with 5g mobile edge computing: Architectures, applications, and technical aspects. *IEEE Communications Surveys & Tutorials*, 23(2):1160–1192, 2021.
[13] Xiaojie Wang, Jiameng Li, Zhaolong Ning, Qingyang Song, Lei Guo, Song Guo, and Mohammad S Obaidat. Wireless powered mobile edge computing networks: A survey. *ACM Computing Surveys*, 2023.
[14] Muhammad Yasir, Sardar Khaliq uz Zaman, Tahir Maqsood, Faisal Rehman, and Saad Mustafa. Copup: Content popularity and user preferences aware content caching framework in mobile edge computing. *Cluster Computing*, 26(1):267–281, 2023.
[15] Xiaobo Zhou, Zhihui Ke, and Tie Qiu. Recommendation-driven multi-cell cooperative caching: A multi-agent reinforcement learning approach. *IEEE Transactions on Mobile Computing*, 2023.
[16] Nhu-Ngoc Dao, Anh-Tien Tran, Ngo Hoang Tu, Tran Thien Thanh, Vo Nguyen Quoc Bao, and Sungrae Cho. A contemporary survey on live video streaming from a computation-driven perspective. *ACM Computing Surveys*, 54(10s):1–38, 2022.
[17] Shilpa Budhkar and Venkatesh Tamarapalli. An overlay management strategy to improve qos in cdn-p2p live streaming systems. *Peer-to-peer networking and applications*, 13:190–206, 2020.
[18] Reza Farahani, Abdelhak Bentaleb, Ekrem Çetinkaya, Christian Timmerer, Roger Zimmermann, and Hermann Hellwagner. Hybrid p2p-cdn architecture for live video streaming: An online learning approach. In *GLOBECOM 2022-2022 IEEE Global Communications Conference*, pages 1911–1917. IEEE, 2022.
[19] Mohammad Ali Maddah-Ali and Urs Niesen. Fundamental limits of caching. *IEEE Transactions on information theory*, 60(5):2856–2867, 2014.
[20] Mohammad Ali Maddah-Ali and Urs Niesen. Decentralized coded caching attains order-optimal memory-rate tradeoff. *IEEE/ACM Transactions On Networking*, 23(4):1029–1040, 2014.
[21] Urs Niesen and Mohammad Ali Maddah-Ali. Coded caching with nonuniform demands. *IEEE Transactions on Information Theory*, 63(2):1146–1158, 2016.
[22] Ramtin Pedarsani, Mohammad Ali Maddah-Ali, and Urs Niesen. Online coded caching. *IEEE/ACM Transactions on Networking*, 24(2):836–845, 2015.
[23] Jens Gramm, Jiong Guo, Falk Hüffner, and Rolf Niedermeier. Data reduction and exact algorithms for clique cover. *ACM J. Exp. Algorithmics*, 13, feb 2009.
[24] Marek Cygan, Marcin Pilipczuk, and Michał Pilipczuk. Known algorithms for edge clique cover are probably optimal. *SIAM Journal on Computing*, 45(1):67–83, 2016.
[25] Chongming Gao, Shijun Li, Wenqiang Lei, Jiawei Chen, Biao Li, Peng Jiang, Xiangnan He, Jiaxin Mao, and Tat-Seng Chua. Kuairec: A fully-observed dataset and insights for evaluating recommender systems. In *Proceedings of the 31st ACM International Conference on Information & Knowledge Management*, pages 540–550, 2022.
[26] Babak Bashari Rad, Harrison John Bhatti, and Mohammad Ahmadi. An introduction to docker and analysis of its performance. *International Journal of Computer Science and Network Security (IJCSNS)*, 17(3):228, 2017.
[27] Zheng Chang, Yunan Gu, Zhu Han, Xianfu Chen, and Tapani Ristaniemi. Context-aware data caching for 5g heterogeneous small cells networks. In *2016 IEEE International Conference on Communications (ICC)*, pages 1–6. IEEE, 2016.
[28] N Golzerai, K Shanmugam, AG Dimakis, AF Molisch, and G Caire. Femtocaching: wireless video content delivery through distributed caching helpers. In *Proc. IEEE INFO COM*, 2012.
[29] Zhuang Chen, Qian He, Zhifei Mao, Hwei-Ming Chung, and Sabita Maharjan. A study on the characteristics of douyin short videos and implications for edge caching. In *Proceedings of the ACM Turing Celebration Conference - China*, ACM TURC '19, New York, NY, USA, 2019. Association for Computing Machinery.
[30] Yu Guan, Xinggong Zhang, and Zongming Guo. Prefcache: Edge cache admission with user preference learning for video content distribution. *IEEE Transactions on Circuits and Systems for Video Technology*, 31(4):1618–1631, 2021.
[31] Fangxin Wang, Feng Wang, Jiangchuan Liu, Ryan Shea, and Lifeng Sun. Intelligent video caching at network edge: A multi-agent deep reinforcement learning approach. In *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications*, pages 2499–2508, 2020.
[32] Mingyue Ji, Giuseppe Caire, and Andreas F Molisch. Fundamental limits of distributed caching in d2d wireless networks. In *2013 IEEE Information Theory Workshop (ITW)*, pages 1–5. IEEE, 2013.
[33] Mingyue Ji, Giuseppe Caire, and Andreas F Molisch. Wireless device-to-device caching networks: Basic principles and system performance. *IEEE Journal on Selected Areas in Communications*, 34(1):176–189, 2015.
[34] Sepandar D Kamvar, Mario T Schlosser, and Hector Garcia-Molina. The eigentrust algorithm for reputation management in p2p networks. In *Proceedings of the 12th international conference on World Wide Web*, pages 640–651, 2003.
[35] Alexander Pyattaev, Olga Galinina, Sergey Andreev, Marcos Katz, and Yevgeni Koucheryavy. Understanding practical limitations of network coding for assisted proximate communication. *IEEE Journal on Selected Areas in Communications*, 33(2):156–170, 2014.
[36] Qi Liu. A brief discussion on p2p network file transfer. *Science, education and literature*, (3):182–185, 2016.
[37] Tang, Ming, Pang, Haitian, Huang, Jianwei, Sun, Lifeng, Gao, and Lin. Optimizations and economics of crowdsourced mobile streaming. *IEEE Communications Magazine: Articles, News, and Events of Interest to Communications Engineers*, 55(5):21–27, 2017.
[38] Tz-Heng Hsu and Yao-Min Tung. A social-aware p2p video transmission strategy for multimedia iot devices. *IEEE Access*, 8:95574–95584, 2020.
[39] Thomas Stockhammer. Dynamic adaptive streaming over http– standards and design principles. In *Proceedings of the second annual ACM conference on Multimedia systems*, pages 133–144, 2011.
[40] Dilip Kumar Krishnappa, Divyashri Bhat, and Michael Zink. Dashing youtube: An analysis of using dash in youtube video service. In *38th Annual IEEE Conference on Local Computer Networks*, pages 407–415. IEEE, 2013.
[41] Zhuqi Li, Yaxiong Xie, Ravi Netravali, and Kyle Jamieson. Dashlet: Taming swipe uncertainty for robust short video streaming. 2022.
[42] MAWI Working Group Traffic Archive. (2023, Jul 26).
[43] Download link for the FCC18 dataset. (2023, July 7).
[44] Ravi Netravali, Anirudh Sivaraman, Somak Das, Ameesh Goyal, Keith Winstein, James Mickens, and Hari Balakrishnan. Mahimahi: accurate {Record-and-Replay} for {HTTP}. In *2015 USENIX Annual Technical Conference (USENIX ATC 15)*, pages 417–429, 2015.
[45] Alberto Caprara, Paolo Toth, and Matteo Fischetti. Algorithms for the set covering problem. *Annals of Operations Research*, 98(1-4):353–371, 2000.
[46] Harry R Lewis. Michael r. πgarey and david s. johnson. computers and intractability. a guide to the theory of np-completeness. wh freeman and company, san francisco1979, x+ 338 pp. *The Journal of Symbolic Logic*, 48(2):498–500, 1983.
[47] Mallesham Dasari, Kumara Kahatapitiya, Samir R Das, Aruna Balasubramanian, and Dimitris Samaras. Swift: Adaptive video streaming with layered neural codecs. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, pages 103–118, 2022.
[48] Yunzhuo Liu, Bo Jiang, Tian Guo, Ramesh K. Sitaraman, Don Towsley, and Xinbing Wang. Grad: Learning for overhead-aware adaptive video streaming with scalable video coding. In *Proceedings of the 28th ACM International Conference on Multimedia*, MM '20, page 349–357, New York, NY, USA, 2020. Association for Computing Machinery.
[49] Hongzi Mao, Ravi Netravali, and Mohammad Alizadeh. Neural adaptive video streaming with pensieve. In *the Conference of the ACM Special Interest Group*, 2017.
[50] Xiaoqi Yin, Abhishek Jindal, Vyas Sekar, and Bruno Sinopoli. A control-theoretic approach for dynamic adaptive video streaming over http. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, pages 325–338, 2015.

APPENDICES

A PSEUDOCODE FOR MINIMUM CLIQUE COVERAGE ALGORITHM

Algorithm 1 Minimum Clique Coverage algorithm

Input: $\mathbb{G} = (E, D)$, $\mathbb{R}$, T
Output: $\mathbb{R}^{'}$

function MAIN
  $\mathbb{R}^{'} = \mathbb{R}$
  repeat
    randomly select $d_i \in D$, s.t. $d_i \in R_x$, $|R_x| = 1$
    randomly select $e_{ij} \in E$, s.t. $d_j \in R_y$, $|R_y| \neq 1$
    $R_x^{'} = R_x \cup \{d_j\}$
    $R_y^{'} = R_y - \{d_j\}$
    for $d_k \in R_y^{'}$ **do**
      if $e_{ik} \in E$ **then**
        $R_x^{''} = R_x^{'} \cup \{d_k\}$
        $R_y^{''} = R_y^{'} - \{d_k\}$
      end if
    end for
    for d_k s.t. $d_k \in R_z$, $|R_z| = 1$ **do**
      if $\nexists\, d_l$ s.t. $d_l \in R_y^{''}, e_{kl} \notin E$ **then**
        $R_y^{'''} = R_y^{''} \cup \{d_k\}$
        $\mathbb{R}^{'} = \mathbb{R}^{'} - \{R_x, R_y, R_z\}$
        $\mathbb{R}^{'} = \mathbb{R}^{'} \cup \{R_x^{''}, R_y^{'''}\}$
        return $\mathbb{R}^{'}$
      end if
    end for
    $\mathbb{R}^{'} = \mathbb{R}^{'} - \{R_x, R_y\}$
    $\mathbb{R}^{'} = \mathbb{R}^{'} \cup \{R_x^{''}, R_y^{''}\}$
    $T = T - 1$
  until $T = 0$
end function

B PROOF OF THE MINIMUM CLIQUE COVERAGE ALGORITHM

In the *Minimum Clique Coverage algorithm*, we randomly select a separate clique in each iteration and search for a solution to reduce the clique number. It's worth noting that although there are situations where the clique number remains unchanged, we still retain all adjustments made during this iteration. Figure 21 explains why we should retain the states. With simple TMG adjustments, larger cliques can be split into smaller ones, creating more opportunities for successful matches. In this example, the clique $R_1 = \{d_1, d_2, d_3, d_4\}$ is split into smaller clique $R_1^{'} = \{d_2, d_3, d_4\}$ in the first iteration, leading to a successful match in the next iteration. More generally, if we can find several **separate nodes** whose connected edges completely **cover all nodes** of a **non-separate clique**, we can then divide this non-separate clique into several parts putting in the separate cliques.

Formally, we define P_i as the set of pointed nodes for all edges connected to node d_i. For example, in the first subgraph of figure 21, $P_1 = \{d_2, d_3, d_4, d_6, d_8\}$, which means node d_1 is connected to nodes d_2, d_3, d_4, d_6, and d_8. We also define Q_i as the set of nodes belonging to the same clique as node d_i. For example, in the first subgraph of figure 21, we have $Q_1 = R_1 = \{d_1, d_2, d_3, d_4\}$, $Q_2 = R_1 = \{d_1, d_2, d_3, d_4\}$, and $Q_8 = R_5 = d_8$. If we can find several separate cliques (nodes) $\{d_i\}, \{d_j\}, \{d_k\}$, ... and a non-separate clique Q_x such that $Q_x \subseteq P_i \cup P_j \cup P_k \cup ...$, it means the clique Q_x can be divided. For example, in the first subgraph of figure 21, we have $P_5 = \{d_3, d_4\}$, $P_7 = \{d_2\}$, $P_8 = \{d_1\}$, and thus $P_5 \cup P_7 \cup P_8 = \{d_1, d_2, d_3, d_4\}$. At the same time, we have $Q_1 = \{d_1, d_2, d_3, d_4\}$, so we have $Q_1 \subseteq P_5 \cup P_7 \cup P_8$. In this case, we can reconstruct the cliques $Q_1(R_1), Q_5(R_2), Q_7(R_4), Q_8(R_5)$ to form cliques $Q_5(R_2^{'}) = \{d_3, d_4, d_5\}$, $Q_7(R_4^{'}) = \{d_2, d_7\}$, $Q_8(R_5^{'}) = \{d_1, d_8\}$ in the third subgraph. It means we form three non-separate cliques from three separate cliques and one non-separate clique.

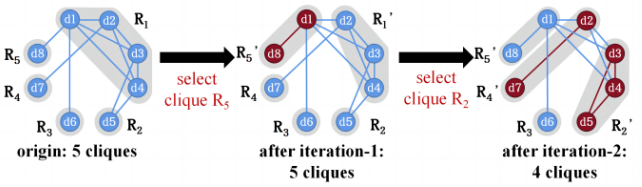


Figure 21: An example of finding the solution through multi-step iteration search

This problem is a classic Set Cover Problem (SCP) that involves selecting specific sets and taking their union to cover all elements of a given set [45]. SCP is NP-hard in the strong sense, proven by Garey and Johnson[46]. In the *Minimum Clique Cover algorithm*, we utilize the characteristics of separate cliques. Each time, we select a separate clique and a non-separate clique, removing **all** nodes that are connected to the separate clique in the non-separate clique. Formally, it can be proven that if $Q_x \subseteq P_i \cup P_j \cup P_k \cup ...$, then for any set P', we have $(Q_x - P') \subseteq ((P_i \cup P_j \cup P_k \cup ...) - P')$. Assuming it is for separate node d_i, we have $(Q_x - P_i) \subseteq ((P_i \cup P_j \cup P_k \cup ...) - P_i) \subseteq P_j \cup P_k \cup ....$ In other words, in the *Minimum Clique Coverage algorithm*, the SCP problem is broken into multiple subproblems which can be solved linearly. In each search, we use a separate clique to grab parts of a non-separate clique. This process continues until a suitable solution is found or the iteration limit has been reached.

Through the above example, we demonstrate that the *Minimum Clique Coverage algorithm* helps us reach the sub-optimal solution without deteriorating the current situation. Therefore, we should retain all adjustments made during each iteration.

C FURTHER CONSIDERATIONS IN DETAILS

Uncompleted watching events: An uncompleted watching event occurs when the user switches to the next video before finishing the last one. This is very common in short video playbacks. Obviously, the video chunks that are not downloaded before switches will not be saved in the client-side cache. Therefore, in the NCTM, if these video chunks are involved in the coded file, the client can not decode it successfully. We can make the node exit the current clique in TMG, forming a separate clique, this will evidently impair the performance of NCTM (the number of clique increase). A compromise solution approach is to assign a weight s_{ij} for each edge in the TMG, representing how long the edge between node d_i and d_j can be maintained (how long the coded transmission can be

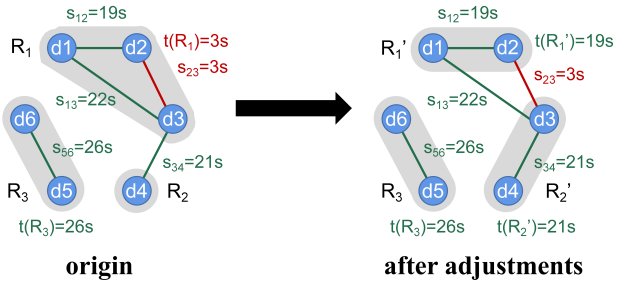


Figure 22: Minimum Clique Coverage algorithm with weight edges in TMG

sustained). Next, we denote $t(R_i)$ to represent how long the clique R_i can be sustained. In the *Minimum Clique Coverage algorithm*, the problem is transformed into a weighted search problem. Greedy algorithms or heuristic search methods can be used to determine whether to retain the adjustments in each iteration. Figure 22 illustrates the process of the weighted algorithm. This iteration does not result in a decrease in clique number, but it extends their duration significantly. This implies a longer duration for coded transmission.

Sliding the progress bar: Sliding the progress bar indicates that the user may skip some uninterested content within one video. This is often accompanied by uncompleted watching events. In the NCTM, when a user slides the video, the edges of the corresponding node in the TMG need to be recalculated (sliding may cause some edges to disappear due to uncompleted watching). The node exits the current clique and then re-runs the *Minimum Clique Cover algorithm*. Furthermore, since NCTM is designed around continuous video playback, if there are missing video chunks in a video, it's recommended to only store the continuous video chunks from the beginning up to the missing portion in the client-side cache. The subsequent video chunks will be rarely used in file decoding.

Video chunks with different bitrates: Short videos are usually encoded as video chunks with different bitrates using the adaptive bitrate paradigm. In the NCTM, we treat video chunks with different bitrates as distinct videos. Fortunately, most short video platforms encode videos in only a limited range of bitrates. For instance, TikTok has only three bitrate options [41]. Furthermore, recently researched layered coding schemes [47, 48] are more compatible with NCTM. In these schemes, video files of different bit rates are compatible. High-bitrate files can also be used for decoding the coded files involving low-bit rate files.

Coded file involving different bitrate chunks: As described in Section 4.2, XOR-encoding occurs between video chunks. In addition, there is sufficient evidence indicating that in short video services (e.g., TikTok, a popular short video service provider), videos are split into size-based chunks (size of 1MB for TikTok) rather than duration-based chunks [41]. Therefore, the file sizes of high-bitrate and low-bitrate video chunks are the same, but the durations of the decoded videos are different. When a coded file involving different bitrates arrives at the edge node, it will be stored in the memory. The higher bitrate clients imply shorter video durations for each chunk, leading to this coded file being more promptly transmitted to the client's device. In contrast, the lower bitrate clients must transmit the former coded file before this one, which will be stored in the memory for a while before it is transmitted. This usually indicates that the user suffers from a lower bandwidth on the link from the edge node to the device, which becomes the bottleneck link, resulting in requests for videos with lower bitrates.

User movement: The user rapid movement implies frequent changes to the connecting base stations, while the user's cache remains the same. In NCTM, due to the involvement of edge nodes (base stations), changing base stations means that the existing coded transmission conditions cease to be satisfied. Any change can be regarded as exiting the original TMG and joining the new TMG with a new identity to explore coded transmission opportunities. Indeed, NCTM is not suitable for situations with frequent handovers between base stations. However, it's important to note that the purpose of NCTM is to reduce the throughput of the shared links. When other users (who do not frequently move) trigger coded transmission, it means that the high-speed moving users have the opportunity to be allocated more bandwidth, leading to a better video viewing experience. Therefore, NCTM takes into consideration the interests of all users, not just a specific category of users. So we still believe that NCTM has strong potential applications.

D PSEUDOCODE FOR RECOMMENDATION REORDER ALGORITHM

Algorithm 2 Recommendation Reorder algorithm

Input: V_i , $\mathbb{T}$, G
Output: v_i^*

```
function MAIN
    flag = FALSE
    for C_j in V_i next G videos do
        for U_k ∈ 𝕌 do
            if t_kj = 1 and t_{i v_k^*} = 1 then
                v_i^* = C_j
                flag = TRUE
            end if
        end for
        if flag = TRUE then
            break
        end if
    end for
    if flag = FALSE then
        v_i^* = V_i next video
    end if
    return v_i^*
end function
```

E OVERALL SETTING OF THE EXPERIMENT

Comparison baselines: We will compare NCTM with the following approaches. 1) *Traditional CDN delivery*: CDNs maintain all videos on the cloud servers. Mobile clients send requests to CDNs through base stations and the cloud servers will transmit the desired content to the mobile clients through the same path. 2) *Edge caching*: The edge nodes (e.g., base stations) cache popular files and use the Least Recently Used (LRU) method to update the content. When a client requests a file cached at the edge node, the edge node will prioritize providing it.

Entity settings: The video distribution over mobile networks generally involves three entities, i.e., content distribution servers

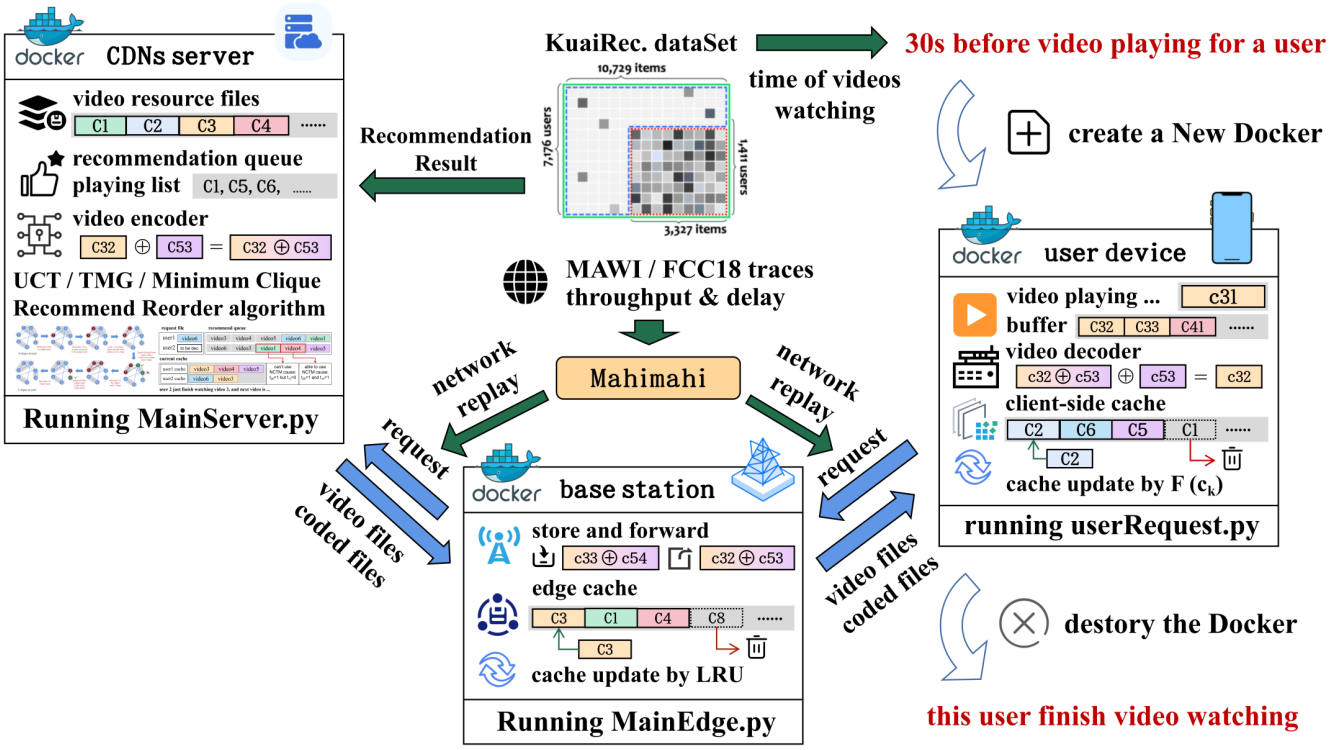


Figure 23: Experimental Overall Structure Design

(cloud servers), base stations (edge nodes), and client devices. In the evaluation, we use Docker [26] to simulate the above transmission process. At the beginning of the experiment, we create one Docker for the cloud server and another Docker for the edge node, separately running codes implemented with Python, to serve the clients. For each client, a new Docker is created during the first 30 seconds before the user starts watching videos, and then simulating the behaviors of requesting video content as well as video playbacks. Once the client completes the session, the corresponding Docker will be deleted. The overall system is presented in figure 23.

Viewing behaviors: The experiment uses the kuaiRec dataset [25], which is a publicly available dataset released by Kuaishou. All data is collected from real user interaction records on the Kuaishou app, a popular short video app, between July 5 and September 5, 2020. We use the user interaction records from August 6, 2020, as the dataset for the experiment. It includes 117,977 video viewing records from 1,398 users requesting 10,230 video files, during 24 hours. In the experiment, we use the order of watched videos in the dataset as the original recommendation queue. However, due to the *Recommendation Reorder algorithm*, the order and timing of video views by users may not follow the original data, still ensuring video playback completion.

Network conditions: The experiment consists of two end-to-end network scenarios, i.e., from the cloud server to the base station (cable network) and from the base station to the end client devices (wireless network). We used the network traces provided by the MAWI Working Group [42] for the wide backbone network to simulate network throughput variations. Specifically, we use the network traces from August 12, 2020, from the main IX link of WIDE to DIX-IE. The network fluctuations ranged from 60Mbps to 200Mbps in mean value, constructing different bandwidth scenarios (abundant bandwidth or limited bandwidth). We also used the FCC18 network traces to simulate network throughput fluctuations from the base station to the end client devices. The FCC18 dataset has been commonly used in previous works [49, 50]. To reflect the differences among clients, the network fluctuations were adjusted to differentiate bandwidth conditions including mean values of 2Mbps, 4Mbps, 6Mbps, and 12Mbps, each with a random variation of up to 5%.

Video representations: We use videos downloaded from the TikTok app and choose the corresponding video with a matching duration for each video in the dataset. Following the existing method [41], we divided each video into chunks of 1MB in size (the last chunk may be smaller than 1MB) and stored them on the cloud server. During the transmission process, if NCTM is triggered, these 1MB-sized chunks are XOR-encoded into coded files and transmitted to the clients over the network.

Evaluation metrics: NCTM was evaluated in terms of network bandwidth utilization, buffer variation, and rebuffer events. Network bandwidth utilization refers to the amount of bandwidth used between a cloud server and a base station. As a video is played, buffer variation refers to changes in the duration of the buffered content. Rebuffer events occur when video playback is interrupted due to low throughput. Since we consider the limited bandwidth struggles to meet the demands of all users, the coded transmission implemented by NCTM can transmit more data content within the limited bandwidth, thus reducing rebuffering events.