Computers, Materials & Continua

Tech Science Press

Doi:10.32604/cmc.2025.059682

ARTICLE

A Collaborative Broadcast Content Recording System Using Distributed Personal Video Recorders

Eunsam Kim[1] **and Choonhwa Lee**[2,*]

[1]Deparment of Computer Engineering, Hongik University, Seoul, 04066, Republic of Korea
[2]Deparment of Computer Science, Hanyang University, Seoul, 04763, Republic of Korea

*Corresponding Author: Choonhwa Lee. Email: lee@hanyang.ac.kr

Received: 30 October 2024; Accepted: 06 January 2025; Published: 17 February 2025

ABSTRACT: Personal video recorders (PVRs) have altered the way users consume television (TV) content by allowing users to record programs and watch them at their convenience, overcoming the constraints of live broadcasting. However, standalone PVRs are limited by their individual storage capacities, restricting the number of programs they can store. While online catch-up TV services such as Hulu and Netflix mitigate this limitation by offering on-demand access to broadcast programs shortly after their initial broadcast, they require substantial storage and network resources, leading to significant infrastructural costs for service providers. To address these challenges, we propose a collaborative TV content recording system that leverages distributed PVRs, combining their storage into a virtual shared pool without additional costs. Our system aims to support all concurrent playback requests without service interruption while ensuring program availability comparable to that of local devices. The main contributions of our proposed system are fourfold. First, by sharing storage and upload bandwidth among PVRs, our system significantly expands the overall recording capacity and enables simultaneous recording of multiple programs without the physical constraints of standalone devices. Second, by utilizing erasure coding efficiently, our system reduces the storage space required for each program, allowing more programs to be recorded compared to traditional replication. Third, we propose an adaptive redundancy scheme to control the degree of redundancy of each program based on its evolving playback demand, ensuring high-quality playback by providing sufficient bandwidth for popular programs. Finally, we introduce a contribution-based incentive policy that encourages PVRs to actively participate by contributing resources, while discouraging excessive consumption of the combined storage pool. Through extensive experiments, we demonstrate the effectiveness of our proposed collaborative TV program recording system in terms of storage efficiency and performance.

KEYWORDS: Collaborative recording; TV content; personal video recorder; erasure coding; replication

1 Introduction

Recent advances in storage, network, and recording technologies have transformed the way users consume TV content [1–3]. Traditionally, television was a one-directional medium with limited programming choices and rigid schedules. The introduction of personal video recorders (PVRs) dramatically altered this paradigm by enabling viewers to record programs in local storage and watch them whenever they want, thereby overcoming the constraints of live broadcasting [4,5]. This capability is especially beneficial for catching programs that might otherwise be missed due to scheduling conflicts or international events with time zone differences, such as the Olympic Games. Despite these advantages, standalone PVRs

have significant limitations. Since each device operates independently with restricted storage capacity for recording programs, it can store only a limited number of programs.

In contrast, cloud-based streaming services like Netflix and Hulu have expanded on-demand viewing options by providing catch-up TV services that make broadcast programs available shortly after their initial broadcast [6–11]. Although these services alleviate the limitations of individual PVRs, they introduce new challenges. Streaming services require massive storage to keep a huge number of programs and substantial network bandwidth to support simultaneous streaming requests across numerous devices. To mitigate these issues, service providers need to deploy large-scale Content Delivery Networks (CDNs). However, it is evident that as the number of users increases, service providers require substantial infrastructural expansions in servers, network bandwidth, and CDN deployment to maintain service quality.

An alternative solution to these challenges can be found in a collaborative peer-to-peer (P2P) system, which can provide high scalability without incurring additional costs [12–16]. By pooling a portion of the storage space of many PVRs into a combined virtual storage, each PVR can access data from this shared storage over the Internet [17–20]. This approach also efficiently reduces storage duplication, unlike standalone PVRs that independently store identical programs. When multiple PVRs record the same program independently, each PVR must allocate storage for the same program. This results in unnecessary data duplication and wasted resources. It is evident that the more popular the program, the greater the amount of storage waste that occurs. However, since a P2P approach allows for shared storage and access, it is not necessary for each device to store an identical program. Thus, if PVRs collaborate to record and play back programs, only a subset of them would need to store the program, significantly increasing storage efficiency.

For this collaborative P2P system to be effective, it must maintain the same level of data availability as standalone PVRs. However, since PVRs often join and leave the system, the high churn rate of PVRs makes it essential to store programs redundantly among multiple PVRs. This ensures that the original program can be obtained even if some PVRs storing the program are not available. Erasure coding [21–23] and replication [24,25] have been commonly used in distributed systems to ensure data availability. In addition to guaranteeing program availability, it is also crucial to ensure that the recording and playback processes remain seamless without quality interruption, even though the number of requests increases. Therefore, the key technical challenge in TV program recording using P2P systems is to support all concurrent playback requests in real time while guaranteeing program availability at a level comparable to that provided by local devices. To address this challenge, we propose a collaborative broadcast program recording system that utilizes distributed PVRs, improving both storage efficiency and performance.

The primary contributions of our proposed system are fourfold: First, by collaborating with other PVRs to share resources, including storage space and upload bandwidth, our system achieves significant advantages over standalone PVRs. Specifically, by using the combined storage pool efficiently, our system significantly expands the overall recording capacity, enabling a substantially larger number of programs to be recorded. Furthermore, this collaborative approach ensures that recording is performed independently, unaffected by the user's current activity or device status, even when a different channel is being watched or when the devices are turned off. Additionally, since there are no physical constraints, such as the number of tuners, each PVR is capable of recording multiple programs simultaneously.

Second, by efficiently utilizing erasure coding, our system requires less storage space to achieve the same program availability compared to traditional replication, which simply duplicates the original data. As a result, our system can record more programs within the same storage capacity. Additionally, our system further enhances storage utilization by distributing the fragments of recorded programs as evenly as possible across participating PVRs. This is accomplished by prioritizing the PVRs with the most available storage space for storing the encoded fragments. Without this strategy, some PVRs may exhaust storage capacity,

while others still have sufficient space available. This imbalance leads to a situation where the system's overall capacity is exhausted despite resources still being available.

Third, by adapting the degree of redundancy of each recorded program according to its changing playback demands over time, our system accommodates as many playback requests as possible. Our system guarantees the minimum degree of redundancy required to ensure the target availability for each program. However, this degree of redundancy does not necessarily guarantee that all playback requests for popular programs can be supported without quality degradation. To address this, as a program's popularity increases, we increase the number of fragments encoded through erasure coding based on our established criteria. Conversely, as the popularity of the program decreases, we reduce the number of fragments accordingly. This is feasible because our system stores each fragment on a distinct PVR. As the redundancy degree of a program increases, the number of PVRs storing them also increases, thus increasing the aggregated upload bandwidth available for playback. Therefore, this scheme significantly reduces the chances of having to reject requests for popular programs due to insufficient upload bandwidth.

Fourth, by implementing a contribution-based incentive policy, our system encourages PVRs to actively participate by contributing their resources, while discouraging excessive use of the combined storage pool. The storage space allocated to each PVR for recording programs is determined by not only the amount of storage space it donates but also its overall contribution to the system. This contribution is measured by the amount of upload bandwidth provided for sharing and the efficiency with which the PVR minimizes storage usage for its own recordings. PVRs that donate more storage space and make larger contributions receive higher priority in storage allocation, allowing them to record more programs within the combined storage pool.

Through extensive experiments, we demonstrate the effectiveness of our proposed collaborative TV program recording system in terms of storage efficiency and performance. First, we show that our system can significantly reduce the redundancy factor for each program, compared to using replication, by utilizing erasure coding, which requires considerably less storage space to achieve the same level of data availability. Second, we illustrate that our system achieves significant improvement in storage efficiency compared to standalone PVRs, in terms of storage requirement per program and the total number of stored programs in the system, achieved through resource sharing and collaboration among PVRs. Finally, we reveal that our adaptive redundancy scheme outperforms a static redundancy scheme in terms of the ratios of continuous playback sessions relative to all requests by controlling the degree of redundancy of each program according to its current playback demand.

This paper is organized as follows. Section 2 discusses the work related to TV content recording systems. Section 3 details the structure and functionalities of our proposed system, which provides TV recording services through collaboration among PVRs. Section 4 presents the experimental results of our proposed system in comparison to standalone PVRs and a static redundancy scheme. Finally, Section 5 concludes the paper.

2 Related Work

In recent years, it is becoming difficult to assume that many people watch TV content at the time that it is broadcast [26,27]. This shift in viewing habits is largely due to the emergence of new distribution channels such as streaming through broadband networks and an increasingly diverse range of devices including TV sets, set-top boxes, PCs, mobile phones, tablets, and game consoles where broadcast programs are being consumed. These evolving consumption trends of broadcast TV content can be summarized as accessing any content anytime, anywhere, and on any device. The rapid evolution of broadcast technologies has transformed content consumption, with advanced systems like ATSC 3.0 that enable hybrid

broadcast-broadband delivery [28], innovative caching mechanisms that improve multimedia access [29], and intelligent recommendation systems that personalize viewer experiences [30]. These technological developments have expanded content accessibility across multiple platforms. The advent of PVRs marked the beginning of a significant shift, enabling users to overcome the live broadcasting constraints by allowing them to record and watch TV programs at their convenience [4,5]. However, they operate independently and have restricted storage capacities, which limits the number of programs that can be recorded on a single device. Subsequently, online TV streaming services have gained popularity, providing over-the-top(OTT) services that distribute TV programs via the public Internet without requiring a dedicated network [6–11]. These platforms effectively address the limitations of standalone PVRs by offering catch-up TV services, allowing viewers to access previously broadcast programs on demand. Several studies have focused on developing efficient caching and delivery algorithms for catch-up and time-shifting services, with the aim of optimizing content distribution and user experience [31–39]. However, to support a large number of concurrent playback requests without compromising quality, servers must provide considerable network bandwidth and extensive storage for broadcast programs.

To address the challenges of the growing demands of network traffic and storage demands driven by the increasing popularity of OTT services, Content Delivery Networks (CDNs) have been employed to place proxy servers close to user devices, providing a robust, globally distributed infrastructure essential for reliable video delivery. While recent studies have made progress in improving performance [40,41], they still rely heavily on centralized infrastructures. As the demand for OTT services increases, continuous investment in infrastructure expansion is required. From a cost perspective, it is essential to alleviate the burden caused by these substantial investments in service expansion. Addressing this issue remains crucial for achieving cost-effective, scalable services on a large scale. As a result, several systems based on P2P structures have been proposed as alternatives to centralized infrastructures [17–20]. In these systems, PVRs are equipped with broadband interfaces, enabling them to exchange data directly with each other instead of relying on central servers. This allows users to access desired TV programs from neighboring PVRs. Some studies have designed PVR systems specifically designed for mobile devices [42] or for vehicles [43]. However, these studies have primarily concentrated on developing frameworks appropriate for sharing data among PVRs.

On the other hand, despite the inherent nature of PVRs joining and leaving the network freely, P2P systems must still ensure data availability comparable to that of streaming servers. To address the challenges caused by high device churn rates, it is essential that programs are stored redundantly across multiple PVRs. This redundancy ensures that the original data can be reconstructed from the PVRs currently turned on, even if some storing the data are unavailable at a given time. Given the vast amount of programs to be stored, considering storage efficiency is crucial. Erasure coding [21–23] is preferred over replication [24,25] as it requires significantly less storage space to store the same amount of data. However, previous systems have not adequately addressed the storage efficiency of PVRs when recording broadcast programs.

In erasure coding, a file is divided into multiple blocks, with each block split into k fragments. An additional $(n - k)$ fragments are then encoded. These fragments are distributed across the n nodes in the network. Even if some fragments are lost due to network issues, the original block can be reconstructed through erasure coding as long as at least k out of the n fragments are received. This resilience has led to the widespread adoption of erasure coding in the development of distributed storage systems, particularly in network environments where nodes frequently become unavailable or packet loss is common [44–49]. Although existing distributed storage systems have focused on providing reliable storage by ensuring data availability, they have not been designed to support streaming services for TV programs. While fragments divided and encoded by erasure coding are stored across multiple nodes, which could potentially increase latency and complexity compared to simple replication, recent research has demonstrated the practical

viability of erasure coding in video streaming systems, particularly in challenging environments such as vehicular networks [50] and lossy networks [51]. Building on these proven approaches, our system efficiently distributes fragments across PVRs to ensure seamless content delivery while preserving the storage efficiency benefits of erasure coding. Additionally, it is challenging to dynamically adjust the system's capacity in response to the skewed and evolving popularity of programs over time. To the best of our knowledge, there have been no systems that ensure playback quality for all playback requests by adapting to changes in the popularity of recorded programs while ensuring program availability.

3 Proposed Collaborative TV Program Recording System

In this section, we introduce a novel TV program recording system that improves storage efficiency and performance through collaborative PVRs. We detail the following: 1) the overall architecture of the proposed system, 2) methods for recording broadcast programs, 3) procedures for playing back recorded programs, 4) strategies for adaptively controlling degree of redundancy to ensure the quality of recorded programs, and 5) a contribution-based incentive policy. Table 1 shows the symbols used in this paper.

Table 1: Important symbols

Symbol	Definition
k	Number of original fragments per block.
n	Total number of fragments per block, including both original and encoded fragments.
r	Redundancy factor representing the number of times each program should be duplicated.
a_{pv}	Average availability of PVRs in the system.
a_{pg}	Availability of a TV program when using erasure coding.
a_{pg}^{rc}	Availability of a TV program when using replication.
$n_{(k,a_{pv},a_{pg})}$	Minimum number of fragments required to ensure TV program availability of at least a_{pg} %, given k and a_{pv}.
pv_x	PVR with index x.
pv^*	PVR chosen to manage the erasure coding task.
P_{selected}	Set of PVRs selected to store the encoded fragments.
C_j	Set of all candidate PVRs currently tuned to the channel j
e	Evaluation period for PVR availability.
$ap(pv_x,\ e)$	Proportion of time pv_x was available during the specified e.
$as(x)$	Amount of available storage space of pv_x.
pg_i	Program with index i.
$AL_i(t)$	List of available PVRs storing fragments of pg_i at time t.
$u_i^a(t)$	Actual aggregated upload bandwidth available for pg_i at time t.
$u_x^d(t)$	Upload bandwidth donated by pv_x at time t.
$u_x^u(t)$	Upload bandwidth used by pv_x at time t.
$z_i(t)$	Probability that pg_i is viewed at time t.
$\lambda(t)$	Average playback request rate per second in the system at time t.
d_i	Duration of pg_i in seconds.
$r_i^t(t)$	Total playback rate of expected requests for pg_i at time t.
r_i^{pg}	Playback rate of pg_i.

(Continued)

Table 1 (continued)

Symbol	Definition
m	Margin value to prevent unnecessary frequent addition and deletion of fragments.
$n_i(t)$	Total number of fragments per block for pg_i at time t.
$n_i^a(t)$	Number of additional fragments to be generated for pg_i at time t.
$n_i^d(t)$	Number of fragments to be deleted for pg_i at time t.
$Sd_x(t)$	Amount of storage space donated by pv_x.
$Psd_x(t)$	Portion proportional to the amount of storage space donated by pv_x at time t.
$Pcs_x(t)$	Portion proportional to the degree of contribution to the system by pv_x at time t.
$u_x^c(t)$	Amount of upload bandwidth contributed by pv_x at time t.
$Cru_x(t)$	Contribution ratio of pv_x based on upload bandwidth at time t.
$su_x(t)$	Amount of storage currently used by pv_x at time t.
$rus_x(t)$	Ratio of unused storage for pv_x relative to its donated storage space at time t.
$Crs_x(t)$	Contribution ratio of pv_x based on its unused storage space at time t.
$Cr_x(t)$	Total contribution ratio of pv_x to the system at time t.
$SA_x(t)$	Amount of storage space allocated to pv_x for recording programs at time t.
$ts(t)$	Total amount of storage space for recording programs within the system at time t.
α	Weight value adjusted for the balance between $Psd_x(t)$ and $Pcs_x(t)$.

3.1 System Architecture

Our TV program recording system consists of PVRs, a combined storage pool donated by PVRs for program recording, and a recording manager, as shown in Fig. 1. We assume that PVRs are equipped with tuners, storage devices, and broadband network interfaces. Thus, they can store broadcast TV programs by receiving them directly via tuners and share recorded TV programs with each other over the Internet after donating part of their storage space for recording broadcast programs. When each program to be recorded starts to broadcast, PVRs store fragments assigned by the recording manager in their storage space reserved for program recording. Users can schedule specific programs for recording, play them back, and delete them at any time. When users start to playback a recorded program, the corresponding PVR receives a certain number of fragments for each block from other PVRs redundantly and decodes them. PVRs can join and leave the system at any time.

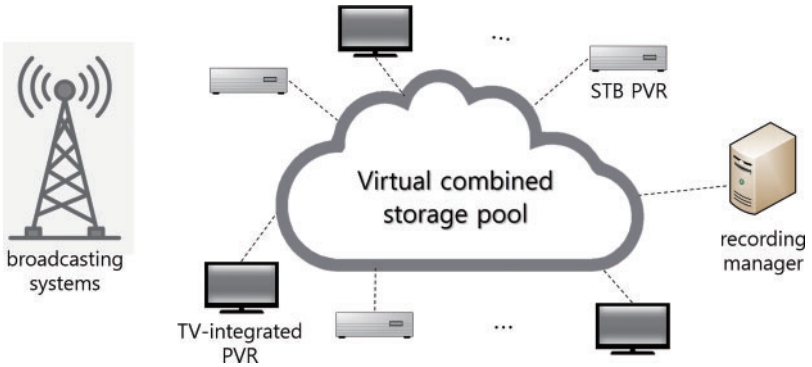

Figure 1: The overall architecture of our proposed collaborative TV program recording system

The recording manager coordinates the program recording process. It determines which PVR will perform the erasure coding task of each requested program, assigns the encoded fragments of the program among PVRs, and facilitates the sharing of these fragments among PVRs. To do so, it maintains all necessary information for each PVR, such as a list of currently connected PVRs, available storage capacity for program recording, and available upload bandwidth. It also keeps track of the information related to each program, such as the current degree of program redundancy, the program length, the storage locations for each fragment, and the number of playback requests over a specific period. The recording manager periodically collects this information from all PVRs.

The combined storage pool is virtual storage space contributed by all participating PVRs for collaborative recording. To ensure efficient sharing of programs stored in the storage pool, PVRs are interconnected based on mesh-based overlay networks. This network structure allows efficient data transmission and redundancy control by distributing storage and playback loads across multiple PVRs, enhancing performance and scalability. When users request program recording or deletion, the recording manager coordinates the storage of new fragments and the removal of existing ones from the shared storage pool.

It is worth noting that our collaborative system is inherently designed to be resilient to failures and data loss incidents, considering the dynamic nature of distributed PVR participation. Since PVRs can frequently join and leave the system, our system fundamentally assumes and handles partial failures of PVRs storing required data. This resilience is achieved through two primary mechanisms. First, our distributed architecture eliminates single points of failure by storing data across multiple PVRs, ensuring system reliability even when individual PVRs become unavailable. Second, our erasure coding implementation provides built-in data recovery capabilities: when a data block is encoded into n fragments, the system can fully recover the original block even if up to $(n - k)$ fragments are lost. This robustness is further enhanced by our adaptive redundancy scheme, which dynamically adjusts the number of fragments based on program popularity over time. This ensures sufficient redundancy is maintained even during periods of high PVR churn or multiple failures. Accordingly, our system can maintain both data integrity and content availability despite the dynamic nature of PVR participation.

3.2 Recording of Broadcast TV Programs

3.2.1 Utilizing Erasure Coding for Enhanced Storage Efficiency

To support all concurrent program playback requests without quality degradation and ensure data availability comparable to that of local storage devices while using less storage space, our system stores all recorded programs by encoding them through erasure coding. As shown in Fig. 2, a recorded program is divided into multiple fixed-length blocks, denoted as B_1, B_2, B_3, …, B_c, where c indicates the total number of blocks belonging to each recorded program. Each block is then divided into k fragments, with additional $(n - k)$ fragments encoded through erasure coding. For instance, B_1 is divided into k original fragments, labeled OF_1 to OF_k, and $(n - k)$ fragments are additionally encoded, labeled CF_{k+1} to CF_n. Consequently, a total of n fragments per block are distributed among PVRs.

Assuming that the redundancy factor, r, represents the number of times each program should be duplicated within the system, and considering that each block of a specific program is divided into k fragments, $r \times k$ fragments are generated for each block and distributed among PVRs. In this context, $r \times k$ thus represents the total number of fragments required for each block, n. Erasure coding uses the dependencies between fragments to enhance availability. These $r \times k$ fragments are dependent on each other and any k fragments out of the $r \times k$ are sufficient to reconstruct the original block. Assuming that one fragment per block is placed on each PVR, at least k PVRs out of the $r \times k$ PVRs that store the block's fragments must be available. This requirement allows us to determine the probability that at least k PVRs are

available, which is calculated by summing the probabilities that exactly k, $k+1$, $\cdots$, or rk PVRs are available. Since the number of available PVRs follows a binomial distribution, the availability of a TV program when using erasure coding (a_{pg}) is calculated as Eq. (1):

$$\begin{aligned} a_{pg} &= \binom{rk}{k} a_{pv}^{k} (1-a_{pv})^{rk-k} + \binom{rk}{k+1} a_{pv}^{k+1} (1-a_{pv})^{rk-k-1} + \cdots + \binom{rk}{rk} a_{pv}^{rk} (1-a_{pv})^{rk-rk} \\ &= \sum_{i=k}^{rk} \binom{rk}{i} a_{pv}^{i} (1-a_{pv})^{rk-i} \end{aligned} \tag{1}$$

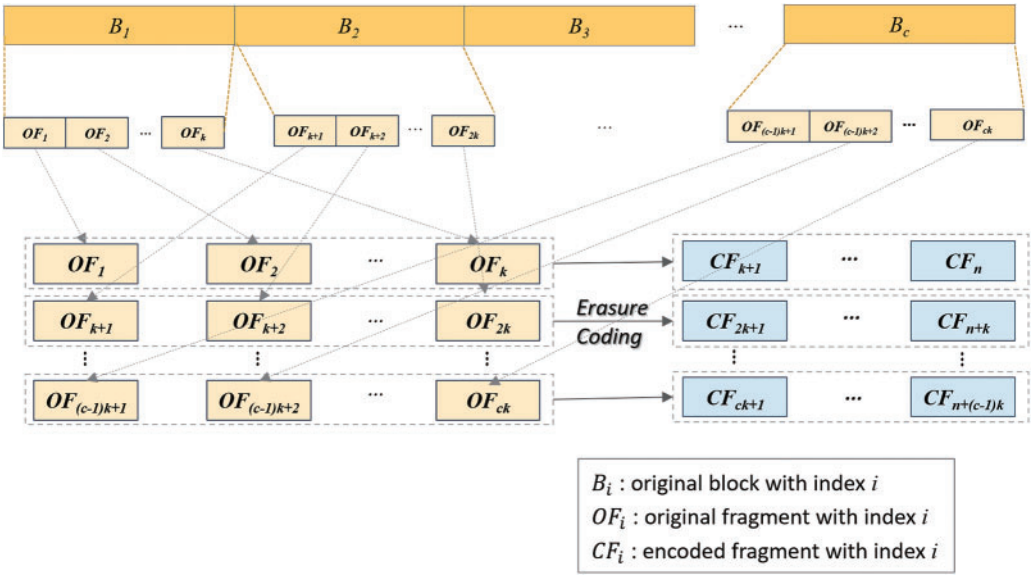

Figure 2: Erasure coding process for storing and distributing a recorded program

To further refine our model, we can redefine n based on the specific requirements of our system. Given that each block is divided into k fragments and PVRs maintain an average availability of a_{pv}, we introduce $n_{(k,a_{pv},a_{pg})}$ as the minimum number of fragments required to ensure a TV program availability of at least a_{pg} %. This value corresponds to rk in Eq. (1). Thus, by calculating rk to meet the requirements of Eq. (1), we can determine $n_{(k,a_{pv},a_{pg})}$ based on the values of k, a_{pv}, and a_{pg}. Notably, a_{pg} is a predefined system parameter, and k is a fixed value chosen based on the system's erasure coding configuration. Thus, the primary parameter requiring accurate estimation is a_{pv}. The value of a_{pv} is calculated as the average availability of all PVRs within a specified observation period. Because the recording manager regularly exchanges messages with PVRs to track their status, the system ensures accurate measurements of a_{pv}. Additionally, the system incorporates a feedback loop to monitor real-time playback requests and dynamically adjust redundancy degree, as described in Section 3.4. This mechanism further mitigates inaccuracies in a_{pv} estimations.

In contrast to erasure coding, when replicating the original block, at least one out of the r replicated blocks is needed to obtain the block. This means that at least one PVR must be available from the r PVRs storing the replicated blocks. Therefore, the program availability when using replication (a_{pg}^{rc}) is calculated as Eq. (2):

$$a_{pg}^{rc} = \binom{r}{1} a_{pv}^{1} (1-a_{pv})^{r-1} + \binom{r}{2} a_{pv}^{2} (1-a_{pv})^{r-2} + \cdots + \binom{r}{r} a_{pv}^{r} (1-a_{pv})^{r-r}$$

$$= \sum_{i=1}^{r} \binom{r}{i} a_{pv}^{i} (1 - a_{pv})^{r-i} \tag{2}$$

We will demonstrate through experiments in Section 4.1 that the redundancy degree required for a_{pg} is lower than that for a_{pg}^{rc}, given that the values of a_{pg} and a_{pg}^{rc} are identical and the same values of a_{pv} and k are applied to both schemes. This implies that erasure coding requires less storage space than replication to achieve the same program availability. Given the vast number of recorded programs that need to be stored, storage efficiency becomes crucial. Therefore, our system utilizes erasure coding, which takes advantage of the dependencies between fragments to significantly decrease the required storage space compared to replication.

On the other hand, as the value of k increases, the overhead of data transmission and computation required to reconstruct each block also increases. This is due to the need for more PVRs to participate in receiving and combining fragments to retrieve each block. However, a higher value of k can improve the load balancing between PVRs, particularly when the system handles numerous programs because fragments are more likely to be evenly distributed between PVRs. Therefore, the value of k should be determined by considering the trade-off between the overhead of erasure coding and the degree of load balance required in specific system environments.

3.2.2 Selecting PVRs for Erasure Coding and Fragment Storage

In our system, users can request to record desired programs. The recording manager evaluates the number of scheduled recording requests for a specific program before the program starts. If the number of those requests is less than the redundancy factor r, which is calculated as $n_{(k,a_{pv},a_{pg})}/k$, each PVR individually stores the entire program in its own storage space to improve storage efficiency. Conversely, if the number exceeds this threshold, the recording manager uses erasure coding to generate multiple fragments for each block and distributes them across PVRs for collaborative recording within the combined storage pool. For example, if r is five, but only three recording requests are made, the three requesting PVRs individually store the program in their respective storage spaces. However, if r is seven, the recording manager employs erasure coding and distributes the encoded fragments among seven designated PVRs.

Once the recording manager decides to use erasure coding to record a program, it must identify a PVR to manage the erasure coding task for the program, and $n_{(k,a_{pv},a_{pg})}$ PVRs to store the encoded fragments in their storage spaces. The PVRs currently tuned to the relevant channel are considered potential candidates for managing the erasure coding task. When selecting a suitable PVR from these candidates, the recording manager should consider the risk that the chosen PVR might turn off or switch channels, which would require handing over the erasure coding task to another PVR. To mitigate the risk of losing data during such transitions, the recording manager prioritizes selecting a PVR that is expected to remain most available throughout the broadcast duration of the program. This availability-based selection approach is particularly crucial in our system because any data loss during live recording would permanently affect playback quality or require significant recovery overhead. In P2P systems, availability-based node selection for task allocation is widely recognized as an effective method for managing peer churn and maintaining system stability, especially in dynamic distributed computing environments where node reliability is inherently unpredictable [52,53].

This expected availability is calculated based on the proportion of time each PVR has been available during a specific preceding period. Assuming that the program to be recorded is scheduled to be broadcast on channel j, the mathematical representation for selecting the suitable PVR is given by Eq. (3):

$$pv^* = \arg\max_{pv_x \in C_j} ap(pv_x,\ e) \tag{3}$$

where pv_x, pv^*, C_j, and $ap(pv_x, e)$ represent the PVR with the index i, the PVR chosen to manage the erasure coding task, the set of all candidate PVRs currently tuned to the channel j, and the proportion of time pv_x was available during the specified evaluation period e, respectively. By selecting the PVR with the highest expected availability, the system can minimize potential disruptions in the encoding process and ensure a more robust recording process. However, if the selected PVR becomes unavailable due to being turned off, the recording manager initiates the selection process again.

The recording manager also decides which PVRs will store the encoded fragments. Unlike the selection process for the PVR responsible for the erasure coding task, which is based on PVR availability, the recording manager chooses the $n_{(k,a_{pv},a_{pg})}$ PVRs with the most available storage space among those currently on. If more fragments are allocated on PVRs with higher availability, these PVRs might quickly become overloaded, potentially saturating their bandwidth. To avoid this, our system strives to distribute fragments as evenly as possible among all PVRs, ensuring balanced storage utilization throughout the system. Otherwise, some PVRs may exhaust storage space, while others still have unused space. This imbalance can result in our system's overall capacity being prematurely exhausted, even though sufficient resources remain available within the system. The set of $n_{(k,a_{pv},a_{pg})}$ PVRs with the most available storage space is defined in Eq. (4):

$$P_{\text{selected}} = \{x \in P_{\text{on}} : |P_{\text{selected}}| = n_{(k,a_{pv},a_{pg})} \text{ and } \forall y \in P_{\text{on}} \backslash P_{\text{selected}},\ as(x) \geq as(y)\} \tag{4}$$

where $P_{selected}$ is the set of selected PVRs with a cardinality of $n_{(k,a_{pv},a_{pg})}$, P_{on} denotes the set of PVRs currently turned on, and $as(x)$ is a function that returns the amount of available storage space of pv_x. The condition $\forall y \in P_{on} \backslash P_{selected}, as(x) \geq as(y)$ ensures that every PVR in the selected set has greater or equal available storage space compared to any PVR not in the selected set. This guarantees that the top $n_{(k,a_{pv},a_{pg})}$ PVRs with the most available storage space are selected from those that are turned on. The encoded fragments are then distributed among these selected PVRs.

3.3 Playback of Recorded Programs

To playback a recorded program, a requesting PVR must retrieve the original data by reconstructing each block through erasure coding. This process requires the PVR to receive at least k fragments out of the n fragments, which have been distributed to n distinct PVRs. Our system is designed to handle the dynamic nature of P2P networks, where PVRs join or leave the system at any time. Consequently, our system facilitates video streaming among PVRs through a mesh-pull structure based on P2P networks. Even if one of the source PVRs leaves the system during playback, the PVR can continue to receive fragments from its remaining source PVRs. In the meantime, the system searches for another available PVR to replace the one that left the system, ensuring that the fragment availability remains sufficient to reconstruct the program.

When a PVR initiates a request to playback a recorded program, the recording manager provides it with a list of source PVRs. The requesting PVR then sends requests to these source PVRs and receives the necessary fragments simultaneously. This concurrent retrieval process significantly reduces the delay in reconstructing the original data. As a result, our system ensures seamless playback of recorded programs in the face of PVR unavailability or network fluctuations.

3.4 Adaptive Redundancy Control to Ensure Playback Quality

There are typically significant differences in the number of playback requests among recorded programs based on their popularity. The playback quality of a popular program may not be guaranteed if the number

of PVRs that store its fragments is not sufficient to accommodate all playback requests. While the previously described value of $n_{(k,a_{pg},a_{pv})}$ determines the minimum number of fragments required to ensure a_{pg} % program availability given k and a_{pv}, this number of PVRs storing these fragments does not necessarily guarantee sufficient bandwidth to ensure playback quality without disruption. To address this, our system implements an adaptive redundancy scheme to accommodate as many playback requests as possible for each recorded program, regardless of its popularity. This scheme aims to minimize request rejections due to insufficient upload bandwidth.

To maximize the utilization of aggregated upload bandwidth, our system dynamically adapts the degree of redundancy of each program to meet changing playback demands over time. This is achieved by adding new fragments or deleting exsiting fragments for each block as needed. Each fragment is stored on a distinct PVR, ensuring that the number of PVRs used corresponds exactly to the total number of fragments. Consequently, as the redundancy degree of a program increases (i.e., as the number of fragments per block increases), the number of PVRs storing them also increases, thus increasing the aggregated upload bandwidth available for playback. Specifically, if the total playback rate of the expected number of requests for the program during a subsequent period exceeds the aggregated upload bandwidth provided by the currently available PVRs among the $n_{(k,p,a)}$ PVRs, our system must create additional fragments for the program and distribute them to more PVRs to increase the aggregated upload bandwidth. Conversely, if the expected demand playback rate is lower than the current aggregated upload bandwidth for the program, some fragments need to be deleted from our system to prevent the wastage of storage space that could otherwise be used for other programs.

The **Algorithm 1** for our adaptive redundancy scheme outlines how our system adjusts the redundancy degree of a program based on changing playback demands, either by adding or deleting fragments as necessary. First, our system measures the actual aggregated upload bandwidth available in the system for each program at time t, denoted as $u_i^a(t)$, defined by Eq. (5):

$$u_i^a(t) = \sum_{pv_x \in AL_i(t)} (u_x^d(t) - u_x^u(t)) \tag{5}$$

where pg_i is the program with index i, $AL_i(t)$ represents the list of available PVRs storing fragments of pg_i at time t, $u_x^d(t)$ is the upload bandwidth donated by pv_x at time t, and $u_x^u(t)$ represents the upload bandwidth currently used by pv_x at time t.

Algorithm 1: Adaptive redundancy control

```
Compute u_i^a(t) using Eq. (5);
Compute r_i^t(t) using Eq. (6);
if (u_i^a(t) < r_i^t(t) − m) {
    Calculate n_i^a(t) using Eq. (7);
    Update n_i(t) as n_i(t) = n_i(t_p) + n_i^a(t);
    Select one PVR to perform erasure coding and n_i^a(t) PVRs to store fragments
      based on criteria in Section 3.2;
    Generate n_i^a(t) additional fragments through erasure coding;
    Distribute them among selected PVRs;
}
```

(Continued)

Algorithm 1 (continued)

```
10    else if (u_i^a(t) > r_i^t(t) + m) {
11        Calculate n_i^d(t) using Eq. (9);
12        if (n_(k,a_pg,a_pv) ≤ n_i(t_p) − n_i^d(t)) {
13            Update n_i(t) as n_i(t) = n_i(t_p) − n_i^d(t);
14        }
15        else {
16            Update n_i(t) as n_i(t) = n_(k,a_pg,a_pv);
17            Adjust n_i^d(t) as n_i^d(t) = n_i(t_p) − n_(k,a_pg,a_pv);
18        }
19        Select n_i^d(t) PVRs with least available storage space using Eq. (11);
20        Delete n_i^d(t) fragments from selected PVRs;
21    }
```

Next, our system calculates the total playback rate of the expected requests for pg_i during the subsequent duration of pg_i at the time t, denoted by $r_i^t(t)$, by multiplying $z_i(t)$, $\lambda(t)$, d_i, and r_i^{pg} as specified in Eq. (6):

$$r_i^t(t) = z_i(t) \times \lambda(t) \times d_i \times r_i^{pg} \tag{6}$$

where $z_i(t)$ is the probability that pg_i is viewed at time t, $\lambda(t)$ is the average playback request rate per second in the system at time t, d_i represents the duration of pg_i in seconds, and r_i^{pg} is the playback rate of pg_i. Note that the term $z_i(t) \times \lambda(t)$ represents the average playback request rate of pg_i at time t.

When the number of requests for pg_i increases, the system must secure additional upload bandwidth to accommodate all incoming playback requests for pg_i over the next d_i seconds. Specifically, if $u_i^a(t) < r_i^t(t) - m$, where m represents a margin value to prevent the unnecessary frequent addition and deletion of fragments, the system initiates the generation of new fragments to increase the aggregated upload bandwidth for pg_i. The number of additional fragments to be generated, denoted as $n_i^a(t)$, is calculated using Eq. (7):

$$n_i^a(t) = \frac{r_i^t(t) - u_i^a(t)}{u_i^a(t)/|AL_i(t)|} \tag{7}$$

Here, $r_i^t(t) - u_i^a(t)$ indicates the amount of deficiency in the aggregated upload bandwidth for pg_i at the current time t required to support all expected requests over the subsequent d_i seconds, while $u_i^a(t)/|AL_i(t)|$ represents the average available upload bandwidth per PVR in the set $AL_i(t)$ at time t. Therefore, by dividing $r_i^t(t) - u_i^a(t)$ by $u_i^a(t)/|AL_i(t)|$, our system determines the number of PVRs required to compensate for the deficient upload bandwidth for pg_i. The total number of fragments per block for pg_i at time t, represented as $n_i(t)$, is updated by the sum of $n_i^a(t)$ and $n_i(t_p)$, where t_p denotes the previous update time, as shown in Eq. (8):

$$n_i(t) = n_i(t_p) + n_i^a(t) \tag{8}$$

The recording manager is also responsible for selecting the PVR to perform erasure coding to generate these additional fragments for pg_i, as well as selecting $n_i^a(t)$ PVRs to store them. The selection process follows the criteria outlined in Section 3.2. The system generates $n_i^a(t)$ new fragments and distributes them among an equivalent number of selected PVRs.

Conversely, if $u_i^a(t) > r_i^t(t) + m$, some fragments for pg_i should be deleted to conserve the allocated storage space. In this case, the number of fragments per block for pg_i that are to be deleted at time t, denoted as $n_i^d(t)$, is determined by Eq. (9):

$$n_i^d(t) = \frac{u_i^a(t) - r_i^t(t)}{u_i^a(t)/|AL_i(t)|} \tag{9}$$

It is noted that $u_i^a(t) - r_i^t(t)$ represents the excess in the aggregated upload bandwidth available for pg_i at the current time t, which exceeds the amount necessary to support all expected requests over the subsequent d_i seconds. Therefore, by dividing $u_i^a(t) - r_i^t(t)$ by $u_i^a(t)/|AL_i(t)|$, our system determines the number of PVRs that can delete their stored fragments of pg_i due to the excess in the aggregated upload bandwidth for pg_i. Note that, to ensure that the minimum program availability for pg_i is maintained at any time, the total number of fragments per block for pg_i must not be below $n_{(k,a_{pg},a_{pv})}$. Thus, the value of $n_i(t)$ is calculated as shown in Eq. (10):

$$n_i(t) = \max(n_{(k,a_{pg},a_{pv})}, n_i(t_p) - n_i^d(t)) \tag{10}$$

In other words, if $n_{(k,a_{pg},a_{pv})} \leq n_i(t_p) - n_i^d(t)$, then $n_i(t)$ is set to $n_i(t_p) - n_i^d(t)$, and $n_i^d(t)$ remains unchanged because $n_i(t)$ still exceeds $n_{(k,a_{pg},a_{pv})}$. However, if $n_{(k,a_{pg},a_{pv})} > n_i(t_p) - n_i^d(t)$, then $n_i(t)$ is set to $n_{(k,a_{pg},a_{pv})}$, and $n_i^d(t)$ is adjusted to $n_i(t_p) - n_{(k,a_{pg},a_{pv})}$, ensuring that $n_i(t)$ does not fall below $n_{(k,a_{pg},a_{pv})}$. Note that, contrary to the criteria for choosing PVRs to store additional fragments, where PVRs with the most available storage space are chosen as shown in Eq. (4), fragments are deleted from the $n_i^d(t)$ PVRs with the least available storage space to keep maintaining balance in available storage among PVRs, as represented by Eq. (11):

$$P_{\text{selected}} = \{x \in P_{\text{on}} : |P_{\text{selected}}| = \frac{u_i^a(t) - r_i^t(t)}{u_i^a(t)/|AL_i(t)|} \text{ and } \forall y \in P_{\text{on}} \backslash P_{\text{selected}},\ as(x) \leq as(y)\} \tag{11}$$

The $n_i^d(t)$ existing fragments are then deleted from selected PVRs.

Our system incorporates a feedback mechanism to dynamically determine the degree of redundancy of each program based on changing playback demands over time. First, the recording manager periodically gathers playback request data at predefined intervals, assessing the popularity of each program. Second, if the playback request rate for a program exceeds predefined thresholds, the system proactively increases its redundancy by generating additional fragments, which are then distributed across more PVRs. Third, as the spike for a program subsides, the system reduces its redundancy degree to reclaim storage space for other programs. While this feedback mechanism effectively addresses most changes in program popularity, rare cases of sudden, significant demand spikes may lead to temporary playback interruptions while the system recalibrates its resources. To address such scenarios, integrating the system with OTT streaming servers could provide additional scalability and reliability.

3.5 Contribution-Based Incentive Policy

To encourage active participation of PVRs in terms of sharing their resources while discouraging excessive storage use that could rapidly deplete the combined storage space, our system employs a contribution-based incentive policy. The amount of storage space allocated to each PVR for recording programs is determined by two factors: one proportional to the amount of storage space it donated and the other proportional to its degree of contribution to the system.

The portion proportional to the amount of storage space donated by pv_x at time t, denoted as $Psd_x(t)$, is calculated simply as a function of the amount of storage space donated by pv_x, represented by $Sd_x(t)$, relative to the total amount of storage space donated by all PVRs in the system as Eq. (12):

$$Psd_x(t) = \frac{Sd_x(t)}{\sum_{k=1}^{l} Sd_k(t)} \tag{12}$$

where l indicates the total number of participating PVRs.

In addition, the proportional portion based on the contribution to the system by pv_x at time t, denoted as $Pcs_x(t)$, is determined by two types of contributions: upload bandwidth and reduced storage usage. First, PVRs are rewarded based on the amount of upload bandwidth they contribute to the system. To manage this, the recording manager keeps track of the amount of upload bandwidth provided by each PVR. The contribution ratio of pv_x based on upload bandwidth at time t, represented by $Cru_x(t)$, is calculated by dividing the amount of upload bandwidth contributed by pv_x ($u_x^c(t)$) by the total upload bandwidth contributed by all PVRs as $Cru_x(t) = \frac{u_x^c(t)}{\sum_{k=1}^{l} u_k^c(t)}$. This indicates that the higher the upload bandwidth provided by pv_x, the higher its contribution ratio.

To completely remove a specific program from the system, it must be deleted from all PVRs that requested its recording. Thus, the system needs to encourage users to delete the recorded programs immediately after viewing them. Second, to free up storage space for other users by deleting their existing recordings and prevent individual PVRs from holding onto recorded programs for too long, the system also rewards PVRs based on the amount of storage they do not use relative to Sd_x. The ratio of unused storage for pv_x relative to $Sd_x(t)$ at time t, represented by $rus_x(t)$, is calculated as $rus_x(t) = 1 - \frac{su_x(t)}{Sd_x(t)}$, where $su_x(t)$ indicates the amount of storage currently used by pv_x at time t. This implies that the less storage pv_x uses relative to the amount of its donated storage, $Sd_x(t)$, the greater its contribution reward $rus_x(t)$. Similarly, the contribution ratio of pv_x based on the amount of its unused storage relative to $Sd_x(t)$ at time t, represented by $Crs_x(t)$, is calculated as $Crs_x(t) = \frac{rus_x(t)}{\sum_{k=1}^{l} rus_k(t)}$.

Consequently, the total contribution ratio of pv_x to the system at time t, denoted as $Cr_x(t)$, is determined by the sum of $Cru_x(t)$ (the contribution based on upload bandwidth) and $Crs_x(t)$ (the contribution based on unused storage), calculated as $Cr_x(t) = \frac{Cru_x(t)+Crs_x(t)}{2}$. The reason for dividing $Cru_x(t) + Crs_x(t)$ by 2 is to ensure that the total sum of $Cr_x(t)$ values for all PVRs is equal to 1, even though it merges two different ratios. This normalization allows $Sd_x(t)$ and $Cr_x(t)$ to serve as factors with equal weight when determining the amount of storage space allocated to each PVR. The proportional portion based on the contribution of pv_x to the system at time t ($Pcs_x(t)$), is then calculated by determining the contribution portion of pv_x relative to the total contributions of all PVRs as Eq. (13):

$$Pcs_x(t) = \frac{Cr_x(t)}{\sum_{k=1}^{l} Cr_k(t)} \tag{13}$$

Finally, the amount of storage space allocated to each PVR for recording programs at time t ($SA_x(t)$) can be calculated as Eq. (14):

$$SA_x(t) = (\alpha \times Psd_x(t) + (1-\alpha) \times Pcs_x(t)) \times ts(t) \tag{14}$$

where α is a weight value ranging from 0 to 1, and $ts(t)$ represents the total amount of storage space for recording programs within the system. The α value can be adjusted according to the current condition of the system. If the overall system storage capacity is insufficient, it may be necessary to increase the α value to

give more weight to $Psd_x(t)$. Conversely, if PVRs lack participation in improving system performance and efficient storage space utilization, it may be necessary to decrease the α value to increase the proportion of $Pcs_x(t)$. PVRs with larger SA_x values receive proportionally higher priority in storage allocation within the combined storage pool, allowing them to record more programs. If a recording request from pv_x exceeds this allocated space, SA_x, it will be denied.

The incentive policy is implemented through a systematic exchange of information between the recording manager and PVRs. When a PVR joins the system, it registers by sending its donated storage capacity. Subsequently, at predefined intervals, each PVR sends a status message including current upload bandwidth utilization and storage space usage. The recording manager maintains a record of these contributions and calculates storage allocations using Eqs. (12)–(14). When pv_x requests to record a program, the recording manager evaluates the request against pv_x's current storage allocation (SA_x). The policy's effectiveness stems from its clear correlation between contributions and benefits: users who contribute more resources gain proportionally more recording privileges. This direct relationship provides strong incentives for active participation and responsible resource usage, similar to successful incentive mechanisms observed in other P2P systems. While detailed analysis of user behavior patterns requires real-world deployment data, our policy framework establishes clear motivations for cooperative behavior.

4 Experimental Results

To evaluate the effectiveness of our proposed collaborative TV content recording system, we conducted extensive simulations with different parameter configurations using the PeerSim P2P simulator. PeerSim is a highly scalable, modular, and event-driven simulation framework specifically designed for peer-to-peer protocols. We utilized the event-based engine of PeerSim to accurately model network dynamics and peer interactions. PeerSim's modular architecture allows for the implementation of custom protocols and network configurations through its APIs. These features make PeerSim particularly suitable for evaluating large-scale distributed systems like our proposed collaborative TV content recording system. The default simulation parameters used throughout this section are listed in Table 2 unless otherwise indicated.

Table 2: Simulation parameters

Parameter	Default value
Number of simulation runs	10
Duration of each simulation run	10 h
Number of participating PVRs	3000
Amount of upload bandwidth each PVR donated	5 Mbps
Amount of storage space each PVR donated	5 GB
Number of channels	50
Program duration	1 h
Playback rate of each program	3 Mbps
Block size	375 KB (1 s)
Number of original fragments per block (k)	8
Margin value to prevent frequent addition and deletion of fragments (m)	3 Mbps
Target program availability	95%
Duration of each evaluation period	1000 s
Periodicity of PVR information gathering	3 s

(Continued)

Table 2 (continued)

Parameter	Default value
Zipf distribution parameter	0.5
Average PVR availability	0.4
Average inter-arrival rate of playback requests	0.3 requests/s

We performed ten simulation runs, with each simulation lasting 10 h to ensure the consistency and reliability of our results. The number of participating PVRs was set to 3000. Each PVR donated 5 Mbps of upload bandwidth, reflecting typical residential broadband capabilities, and 5 GB of storage space. We simulated 50 channels, with each program lasting 1 h. The playback rate of each program was set at 3 Mbps to match typical HD video streaming requirements, with a block size of 375 KB, corresponding to 1 s of playback. The number of original fragments per block (k) was set to 8. The margin value to prevent frequent addition and deletion of fragments (m) was set to 3 Mbps, equivalent to the bandwidth required for the playback of one program. The target program availability is set at 95%. Each evaluation period to estimate the availability of each PVR lasted 1000 s, and the recording manager gathered status information from each PVR every 3 s.

Since precise popularity distributions for TV programs are not well established in the literature, we assumed that the popularity of each program follows a Zipf distribution with a parameter of 0.5. We set the average inter-arrival rate for all recording and playback requests at 0.3 requests/s. The request rate for each program at the start of its broadcast was determined by multiplying the average inter-arrival rate of all requests by its proportion among total playback requests, based on the Zipf distribution during each hour. The temporal distribution of users' playback requests was modeled based on observed TV viewing patterns, notably peaking one hour after broadcast time, accounting for 20% of total requests, followed by a gradual 10% hourly decline. The inter-arrival and inter-departure times for each PVR followed a Poisson distribution with means of 400 and 600 s, respectively, which implies an average PVR availability of 0.4. These parameters were carefully calibrated to represent realistic user session patterns. To incorporate realistic viewing behaviors, we also assumed that users might switch channels randomly after viewing part of a program.

In each simulation, we varied one or two parameters while keeping the others fixed to clearly assess the impact of specific parameters by minimizing interference from other factors during performance evaluation. In the next subsection, to demonstrate the impact of employing erasure coding on the redundancy degree for each program compared to using traditional replication, we analyze the redundancy factors required by both methods to achieve the same target program availabilities. This comparison was conducted while varying PVR availability from 0.2 to 0.8. To illustrate the effectiveness of our proposed collaborative recording system from the perspective of storage efficiency, we also compare our system with standalone PVRs in terms of the amount of storage space required per program. In these experiments, we also varied the number of concurrent playback requests per program during a peak time from 10 to 50. We then examine the impact of our collaborative recording scheme on the total number of stored programs within the system compared to standalone PVRs. To observe the effect of our adaptive redundancy scheme in adapting to the changing playback demand of each program, we also compare it with a static redundancy scheme while varying the Zipf parameter for different a_{pv} values. To investigate the effect of our adaptive redundancy scheme on playback performance, we examine the ratios of continuous playback sessions relative to all requests for our scheme and a static redundancy scheme, while varying Zipf distribution parameters from 0.3 to 0.9. Finally, we conduct a sensitivity analysis to examine how variations in parameters a_{pv} and a_{pg} affect the results.

4.1 Impact of Utilizing Erasure Coding on Redundancy Factors

Fig. 3 shows the redundancy factors (r) required by erasure coding (denoted by EC) and replication (RC) to achieve target program availabilities (a_{pg}) of 95% and 99% while varying the PVR availability (a_{pv}) between 0.2 and 0.8. These experimental results confirm that EC requires significantly less storage space than RC to achieve the same data availability. The r values required by EC were on average 60.4% and 107.2% less than those required by RC for the target a_{pg} of 95% and 99%, respectively. In particular, when a_{pv} was 0.2 and the target a_{pg} was 99%, the difference in r values was 121.5%. This is because EC can manage data redundancy more efficiently than RC, since EC divides the program into smaller fragments and can restore the program with only a portion of these fragments, whereas RC requires copying the entire program.

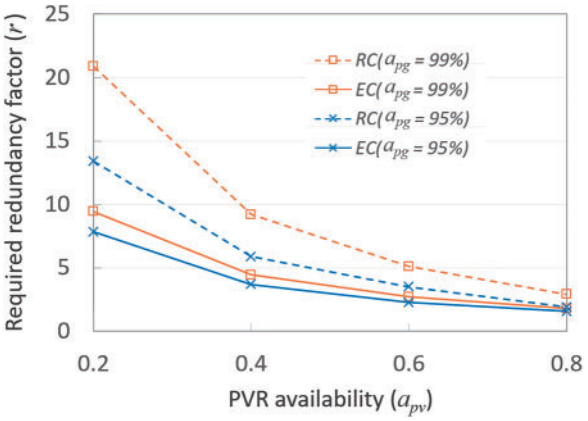

Figure 3: Redundancy factors required by erasure coding (EC) and replication (RC) according to PVR availability (a_{pv}), given target program availabilities (a_{pg})

The results also demonstrate that the superiority of EC over RC becomes increasingly significant as a_{pv} decreases or the target a_{pg} increases. The r value required by RC was significantly higher than that by EC as a_{pv} decreased. For example, with an a_{pv} of 0.8 and a target a_{pg} of 99%, the difference in r values between EC and RC was 1.1. However, with an a_{pv} of 0.2, this difference increased markedly to 11.5. This trend highlights the efficiency of EC in environments with lower a_{pv}. Furthermore, EC makes the system more storage efficient than RC as the target a_{pv} increases. In the case where the target a_{pv} increased from 95% to 99% with an a_{pv} of 0.2, the r value required by EC increased from 7.8 to 9.4. In contrast, the r value required by RC increased from 13.4 to 20.9. Consequently, the increase in the r value for RC, relative to EC, was approximately 4.7 times higher.

The results indicate that EC becomes increasingly advantageous as a_{pv} decreases or as a higher target a_{pg} is required, thereby reducing the need for excessive redundancy compared to RC. This is particularly relevant in real-world scenarios where churn rates are high and uninterrupted playback is crucial, making EC a more favorable choice for efficient storage utilization in the system.

4.2 Impact of Collaborative Recording on Storage Requirements Per Program

Figs. 4 and 5 illustrate the significant improvement in storage efficiency achieved by our collaborative recording system compared to standalone PVRs. The linear increase in storage requirements observed in standalone PVRs, as shown in Fig. 4, highlights a fundamental limitation of independent recording systems. In such systems, each PVR independently stores duplicates of each program, leading to significant storage redundancy across PVRs. In contrast, Fig. 5 demonstrates how our collaborative system efficiently overcomes

this limitation by sharing resources and managing them efficiently. Consequently, our system requires significantly less storage space to support all playback requests without quality disruption across all values of PVR availability (a_{pv}), while varying the number of concurrent playback requests during peak times from 10 to 50. Specifically, when the number of concurrent requests is 10, 20, 30, 40, and 50, our system requires on average only 8.3%, 4.1%, 3.1%, 3.0%, and 2.9% of the storage space needed by standalone PVRs, respectively. Notably, when there are 50 concurrent requests and a_{pv} is 0.8, our system uses only 1.7% of the storage space compared to the standalone PVRs. This efficiency is achieved because our system avoids unnecessary data duplication by enabling PVRs to collaborate in sharing resources. In contrast, standalone PVRs must store each program on their own storage devices, duplicating it as many times as there are individual PVRs that recorded it because the redundancy degree is not adjusted. Furthermore, as the number of concurrent requests increases, the redundancy degree among standalone PVRs gets larger, leading to greater inefficiency. This explains the widening gap in storage requirements between our system and standalone PVRs as the number of concurrent requests increases. For instance, the difference in the redundancy degree between our system and standalone PVRs is 2.7 times higher when the number of concurrent requests is 50 compared to when it is 10.

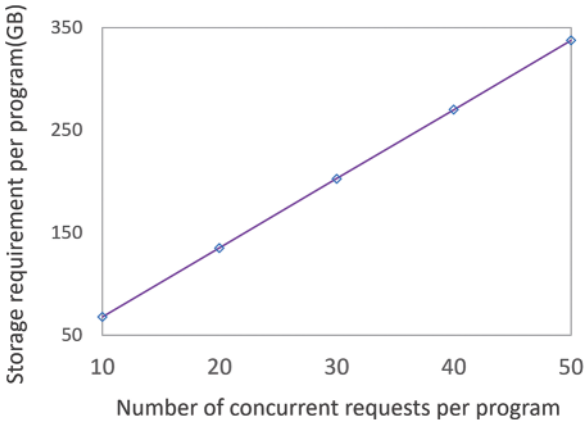

Figure 4: Amount of storage space required per program in standalnoe PVRs

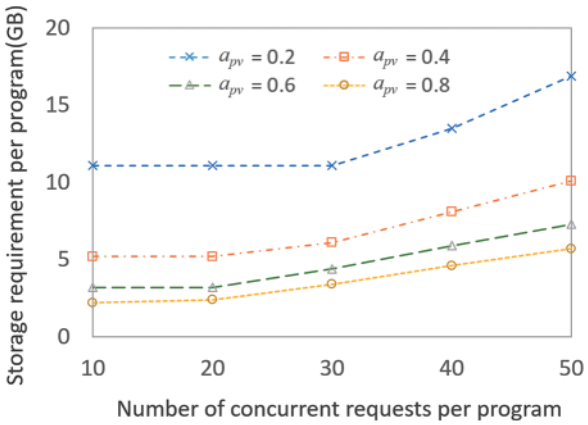

Figure 5: Amount of storage space required per program according to the number of peak-time requests per program for different PVR availabilities (a_{pv})

We can also see from Fig. 5 that in our system, the storage space required per program increases as the number of concurrent playback requests gets larger. The average storage space for all values of a_{pv} is 5.5 GB when there are 10 concurrent playback requests, but this increases to 10.1 GB when there are 50 concurrent playback requests. The value of $n_{(k,a_{pv},a_{pg})}$ represents the minimum number of fragments required to ensure the program availability of a_{pg}, given k and a_{pv}, and these fragments are distributed across an equivalent number of PVRs. When the number of concurrent requests is small, the aggregated upload bandwidth of the PVRs storing $n_{(k,a_{pv},a_{pg})}$ fragments is sufficient to maintain playback quality while ensuring availability. Therefore, with an a_{pv} of 0.2, the original $n_{(k,a_{pv},a_{pg})}$ value was adequate to support all playback requests up to 30 concurrent playbacks, and for an a_{pv} of 0.8, up to 10 concurrent playbacks. However, as the number of concurrent requests increases, the aggregated upload bandwidth of $n_{(k,a_{pv},a_{pg})}$ PVRs becomes insufficient. Consequently, to support all playback requests, the system must adaptively determine the number of fragments, generate additional fragments, and store them across multiple PVRs, thereby increasing storage requirements. For example, with an a_{pv} of 0.2, the required storage capacity was 11.1 GB for 30 requests but increased to 16.9 GB for 50 requests.

As expected, we have observed that lower PVR availability (a_{pv}) requires a higher redundancy factor (r). This is because more redundancy is required to maintain data availability when individual PVRs are less reliable. For example, when the number of concurrent playback requests is 50, 16.9 GB of storage is required when a_{pv} is 0.2, while only 5.7 GB is needed when a_{pv} is 0.8.

4.3 Effect of Our System on Total Number of Stored Programs

Fig. 6 illustrates the total number of stored programs for different a_{pv} values under varying Zipf parameter s in our adaptive redundancy scheme (denoted as *ARS*), compared to a static redundancy scheme (*SRS*). The SRS maintains the initial redundancy degree determined at the time of recording to ensures basic availability, regardless of changes in program popularity over time. This static approach serves as a baseline to evaluate the effectiveness of our proposed ARS, which dynamically adjusts redundancy degrees based on program popularity. First, we can observe from Fig. 6 how our collaborative recording system affects the total number of stored programs compared to standalone PVRs. In this experiment, a standalone PVR can store only 3.7 programs because it uses 5 GB of space for recording on its storage device. In contrast, our system can store between 780 and 6280 programs depending on PVR availability (a_{pv}) and Zipf distribution parameter (denoted as s), which represents the degree of popularity bias among the programs. This means that users can record 211 to 1700 times more programs in our system compared to standalone PVRs through resource sharing and PVR collaboration.

Furthermore, to examine the impact of *ARS* on the total number of stored programs, we conducted experiments with a_{pv} values of 0.2, 0.4, 0.6, and 0.8, while varying s values from 0.3 to 0.9. As seen in Fig. 7, which displays the ratios of playback requests to total requests for 100 programs ranked by popularity, lower s values correspond to smaller differences in program popularity, resulting in a relatively more uniform distribution among programs. Conversely, higher s values make these differences more significant, leading to a more skewed popularity distribution. For example, when an s value is 0.9, the most popular program accounts for 15.6% of all playback requests and maintains a high request rate for a longer period even as requests decrease by 10% per hour after the peak period.

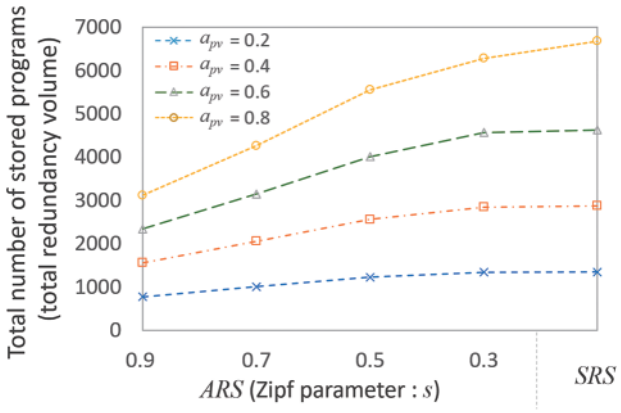

Figure 6: Total number of stored programs for different a_{pv} values under varying Zipf parameter s in our adaptive redundancy scheme (*ARS*), compared to a static redundancy scheme (*SRS*)

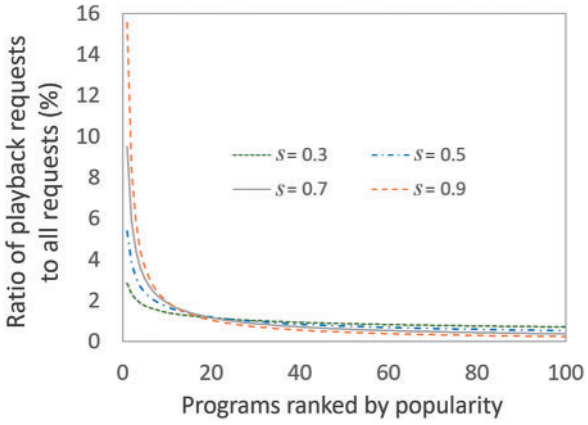

Figure 7: Ratio of playback requests to total requests for 100 programs ranked by popularity

We observed that as the s value increases, the number of stored programs decreases. Specifically, the average number of stored programs across all values of a_{pv} was 3762 when s was 0.3, while it drops to just 1953 when s increased to 0.9. This is because higher s values require more redundancy to maintain playback quality, which increases the amount of storage required per program and thus reduces the total number of stored programs. In our system, the total redundancy volume required for storing each program comprises a base redundancy volume, which is necessary to guarantee availability (as determined by the $n_{(k,a_{pv},a_{pg})}$ value), and an additional redundancy volume, which is essential to ensure playback quality. Figs. 8 and 9 illustrate the composition of the base and the additional volume of redundancy for each case depicted in Fig. 6. As shown in Fig. 8, while the base redundancy volume remains consistent for the same a_{pv} regardless of the s values, the additional redundancy volume increases significantly for more popular programs as s increases. For example, when a_{pv} = 0.2, the additional redundancy volume was equivalent to 31.4 programs at a s of 0.3, but increased to 4815 programs at 0.9. Consequently, an increase in the additional redundancy volume leads to a decrease in the total number of stored programs.

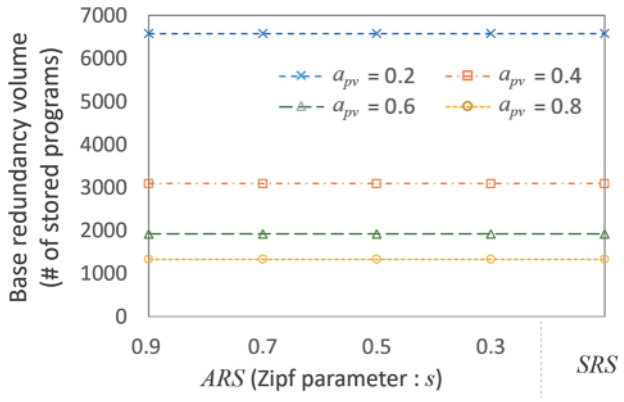

Figure 8: Base redundancy volume for different a_{pv} values under varying Zipf parameter s in our adaptive redundancy scheme (*ARS*), compared to a static redundancy scheme (*SRS*)

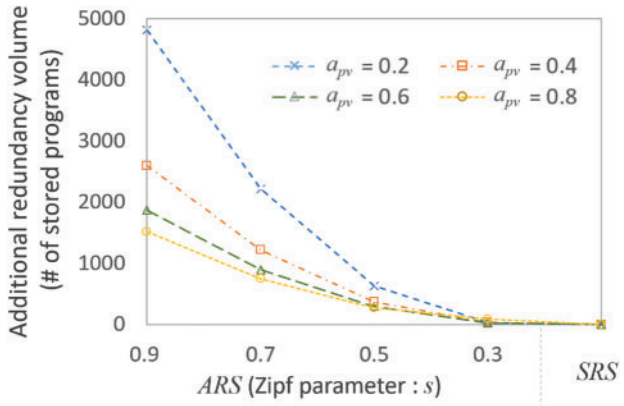

Figure 9: Additional redundancy volume for different a_{pv} values under varying Zipf parameter s in our adaptive redundancy scheme (*ARS*), compared to a static redundancy scheme (*SRS*)

We can also see that when s is 0.3, which indicates the smallest difference in popularity among programs, the increase in additional redundancy volume is only slight, making it nearly identical to that of *SRS*. *SRS* does not consider fluctuations in popularity over time, instead fixing the redundancy factor (r) based on a given a_{pv} and determining r values only to ensure minimum availability. For example, with an s value of 0.3, *ARS* increased the additional redundancy volume by only 6.4% compared to *SRS*. Consequently, *SRS* allows for storing more programs compared to *ARS*. However, it may fail to guarantee playback quality as the number of playback requests exceeds a certain threshold, as discussed in Section 4.4.

As expected, when a_{pv} increases, the $n_{(k,a_{pv},a_{pg})}$ value required to ensure the target program availability (a_{pg}) decreases, thereby reducing the redundancy factor (r) needed to store a program. Consequently, more programs can be stored in the system because each program requires less storage space. For example, with an s value of 0.7, the system could store 4269 programs when a_{pv} was 0.8, but only 1009 programs when a_{pv} was 0.2.

4.4 Impact of Adaptive Redundancy Control on Playback Continuity

Fig. 10a–d illustrates the ratios of continuous playback sessions–defined as those maintaining over 95% playback continuity–relative to all requests. These figures compare the performance of *ARS* and *SRS* across different Zipf distribution parameters *s* of 0.3, 0.5, 0.7, and 0.9, respectively. The total number of playback requests per hour for each program varies with *s* values. It is assumed that after the initial peak hour following a broadcast, the number of playback requests for each program decreases by 10% per hour. Consequently, programs with higher popularity maintain a larger number of playback requests over a longer period, which leads to a higher total number of playback requests. To clearly demonstrate the differences between the two schemes, we analyzed the performance around the points at which the ratios of continuous playback sessions reach saturation while varying the number of playback requests for each *s* value. As the number of playback requests per hour increases beyond the capacity of the aggregated upload bandwidth provided by the PVRs, both schemes show a decline in the ratios of continuous playback sessions. Specifically, the ratios of continuous playback sessions began to decrease noticeably when the number of playback requests per hour reached 1070, 1580, 2470, and 3540 for *s* values of 0.3, 0.5, 0.7, and 0.9, respectively.

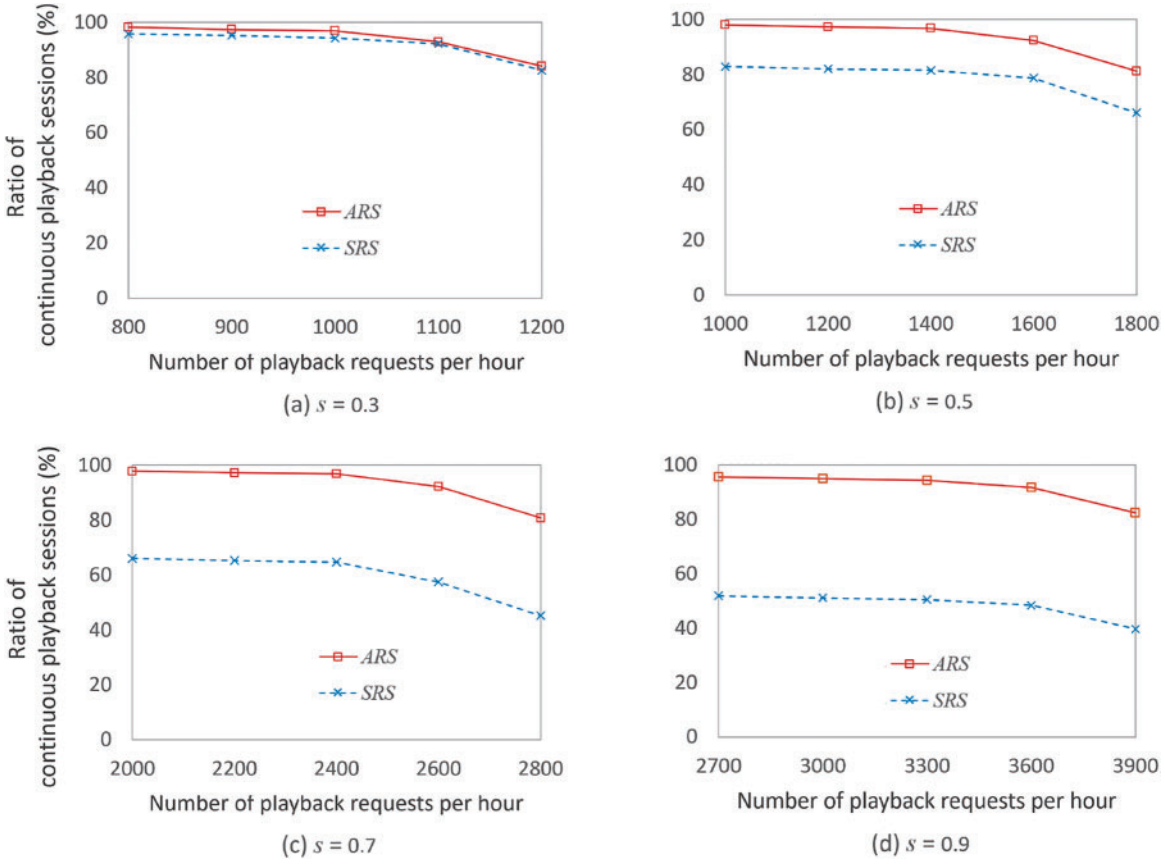

Figure 10: Ratios of continuous playback sessions relative to all requests in *ARS* and *SRS* for different Zipf parameters *s*: (a) $s = 0.3$, (b) $s = 0.5$, (c) $s = 0.7$, (d) $s = 0.9$

In Fig. 10a–d, *ARS* outperformed *SRS* by an average of 1.8%, 14.9%, 33.3% and 43. 5% in terms of the ratio of continuous playback sessions *s* of 0.3, 0.5, 0.7, and 0.9, respectively. *SRS* focuses only on ensuring program availability without considering program popularity, which limits its ability to provide sufficient

upload bandwidth as the number of requests increases. Consequently, as popularity bias among programs increases, *SRS* struggles to support playback requests without quality disruption, especially for more popular programs. This limitation leads to a marked decrease in the overall ratio of continuous playback sessions. Specifically, the ratio was on average 43.7% lower for $s = 0.9$ compared to $s = 0.3$. In contrast, by adaptively controlling the degree of redundancy of each program based on its popularity, *ARS* not only ensures program availability but also provides sufficient upload bandwidth for high-quality playback of popular programs. *ARS* maintained average ratios of continuous playback sessions at 96.6%, 96.8%, 96.9%, and 94.3% before reaching the saturation points for each respective s value. This performance indicates that irrespective of the s value, *ARS* effectively adjusts the total number of fragments to match the playback demand of each program. As a result, it efficiently utilizes the system's aggregated upload bandwidth for each program until it is exhausted. Therefore, *ARS* performs more advantageously compared to *SRS*, particularly in practical situations where disparities in program popularity frequently occur.

4.5 Effect of Varying a_{pv} and a_{pg} on Storage Requirements and Performance

We conducted a sensitivity analysis to examine how variations in a_{pv} and a_{pg} affect the storage requirements and system performance. By altering one parameter at a time while keeping the other constant, we were able to isolate and understand the individual effects of each parameter on the system. First, we analyzed the effects of varying a_{pv} while keeping a_{pg} fixed at 95%. Fig. 11 illustrates the impact of changing a_{pv} from 0.2 to 0.8 in increments of 0.2. As a_{pv} increases, the amount of required storage space per program decreases. Specifically, at $a_{pv} = 0.8$, the storage requirement is 79.2% lower compared to $a_{pv} = 0$. This reduction occurs because higher a_{pv} values indicate that a PVR is more likely to be available, allowing the system to achieve the target a_{pg} with fewer fragments. The number of supported concurrent playbacks increases as a_{pv} increases. At $a_{pv} = 0.8$, the system supports 3.8 times more playbacks than at a_{pv}. This is due to the increased availability of PVRs, which provides greater aggregate upload bandwidth. These results demonstrate that a_{pv} significantly influences both storage requirements and system performance, with higher a_{pv} values leading to better resource utilization and playback capacity.

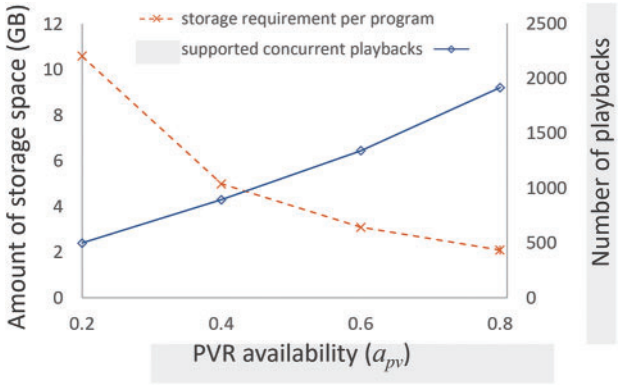

Figure 11: Storage requirement per program and the number of supported concurrent playbacks according to varying a_{pv} while fixing $a_{pg} = 95\%$

We also examined the impact of varying a_{pg} from 91% to 99% in increments of 2% while keeping a_{pv} fixed at 0.4. Fig. 12 shows the effects on storage requirements and concurrent playbacks. As a_{pg} increases, the required storage space per program also rises. At $a_{pg} = 99\%$, the storage requirement is 28.3% higher than at $a_{pg} = 91\%$. This is expected as achieving higher program availability requires storing more fragments

to account for potential PVR failures. The number of supported concurrent playbacks remains largely unaffected by changes in a_{pg}. This is because playback capacity is determined by the system's available upload bandwidth, which is independent of a_{pg}. These findings suggest that while a_{pg} primarily impacts storage requirements, its effect on playback performance is minimal compared to variations in a_{pv}. Consequently, this sensitivity analysis provides insights into the system's ability to maintain consistent performance under different conditions.

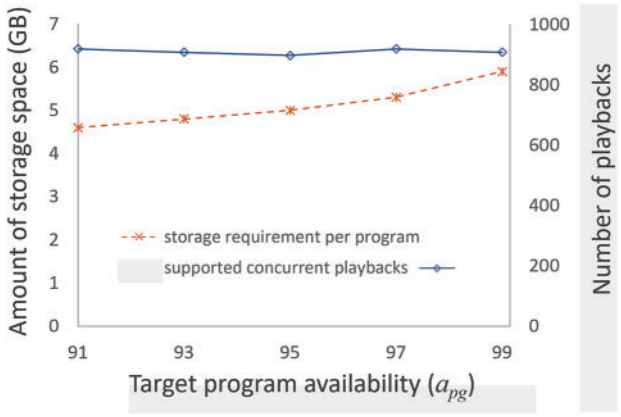

Figure 12: Storage requirement per program and the number of supported concurrent playbacks according to varying a_{pg} while fixing $a_{pv} = 0.4$

5 Conclusions

In this paper, we proposed a collaborative TV content recording system that addresses the limitations of standalone PVRs and on-line catch-up TV by leveraging the combined resources such as storage and upload bandwidth of distributed PVRs without additional costs. By pooling these resources into a virtual shared storage pool, our system increases storage efficiency for program recording and accommodates as many playback requests as possible while maintaining high-quality streaming performance.

Our system considerably expands recording capacity through efficient resource sharing among PVRs, enabling the recording of a much larger number of programs. This collaboration also allows for simultaneous recording of multiple programs regardless of the user's current activity or device status. By employing erasure coding, our system minimizes the required storage space per program while maintaining high data availability. Additionally, we introduced an adaptive redundancy scheme that dynamically controls the degree of redundancy based on each program's playback demand, ensuring high-quality playback for popular programs. We also implemented a contribution-based incentive policy, which rewards PVRs that actively donate resources and promote fair use of the shared storage pool.

Our experiments demonstrated that our system not only surpasses standalone PVRs in terms of storage efficiency and capacity but also outperforms static redundancy schemes in terms of ratios of continuous playback sessions. As online streaming and catch-up TV services continue to increase in popularity, our proposed collaborative recording system has the potential to play an increasingly important role in supporting scalable high-quality TV content recording and playback.

As future work, we plan to improve the selection criteria for PVRs that manage erasure coding by developing a multi-criteria evaluation framework that incorporates additional factors in addition to

availability. Additionally, we plan to refine our incentive policy on user behavior by monitoring actual user responses in real-world system deployments.

Acknowledgement: The authors gratefully acknowledge the editor, reviewers, and all contributors for their valuable suggestions and efforts that improved this paper.

Funding Statement: This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (Nos. 2019R1A2C1002221 and RS-2023-00252186) and Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (Nos. 2021-0-00590, RS-2021-II210590, Decentralized High Performance Consensus for Large-Scale Blockchains).

Author Contributions: The authors confirm contribution to the paper as follows: study conception and design: Eunsam Kim, Choonhwa Lee; data collection: Eunsam Kim; analysis and interpretation of results: Eunsam Kim, Choonhwa Lee; draft manuscript preparation: Eunsam Kim. All authors reviewed the results and approved the final version of the manuscript.

Availability of Data and Materials: The authors confirm that the data supporting the findings of this study are available within the article.

Ethics Approval: Not applicable.

Conflicts of Interest: The authors declare no conflicts of interest to report regarding the present study.

References

1. Leiner DJ, Neuendorf NL. Does streaming TV change our concept of television? J Broadcast Electron Med. 2022;66(1):153–75. doi:10.1080/08838151.2021.2013221.
2. Vanattenhoven J, Geerts D. Broadcast, video-on-demand, and other ways to watch television content: a household perspective. In: Proceedings of ACM International Conference on Interactive Experiences for TV and Online Video; 2015; Brussels, Belgium. p. 73–82.
3. Stevens T, Appleby S. Video delivery and challenges: TV, broadcast and over the top. MediaSync: Handbook on Multimedia Synchronization: Springer International Publishing; 2018. p. 547–64.
4. Bae J, Kim D, Kang HI. An efficient personal video recorder system. In: Proceedings of International Conference on Intelligent Computation Technology and Automation; 2010; Changsha, China. p. 501–4.
5. Divakaran A, Otsuka I. A video-browsing-enhanced personal video recorder. In: Proceedings of International Conference of Image Analysis and Processing; 2007; Modena, Italy. p. 137–42.
6. Khanna P, Sehgal R, Gupta A, Dubey AM, Srivastava R. Over-the-top (OTT) platforms: a review, synthesis and research directions. Mark Intell Plan. 2024. doi:10.1108/MIP-03-2023-0122.
7. Al-Abbasi AO, Aggarwal V, Ra MR. Multi-tier caching analysis in CDN-based over-the-top video streaming systems. IEEE/ACM Trans Netw. 2019;27(2):835–47. doi:10.1109/TNET.2019.2900434.
8. Kumar T, Sharma P, Tanwar J, Alsghier H, Bhushan S, Alhumyani H, et al. Cloud-based video streaming services: trends, challenges, and opportunities. CAAI Trans Intell Technol. 2024;9(2):265–85. doi:10.1049/cit2.12299.
9. Timmerer C, Begen AC. Over-the-top content delivery: state of the art and challenges ahead. In: Proceedings of ACM International Conference on Multimedia; 2014; Orlando, FL, USA. p. 1231–2.
10. Nogueira J, Guardalben L, Cardoso B, Sargento S. Catch-up TV forecasting: enabling next-generation over-the-top multimedia TV services. Multimed Tools Appl. 2018;77:14527–55. doi:10.1007/s11042-017-5043-9.
11. Abreu J, Nogueira J, Becker V, Cardoso B. Survey of catch-up TV and other time-shift services: a comprehensive analysis and taxonomy of linear and nonlinear television. Telecommun Syst. 2017;64:57–74. doi:10.1007/s11235-016-0157-3.
12. Yang H, Liu M, Li B, Dong Z. A P2P network framework for interactive streaming media. Proc Int Conf Intell Human-Mach Syst Cybern. 2019;2:288–92.

13. Miguel EC, Silva CM, Coelho FC, Cunha IFS, Campos SVA. Construction and maintenance of P2P overlays for live streaming. Multimed Tools Appl. 2021;80(13):20255–82. doi:10.1007/s11042-021-10604-w.
14. Ali M, Asghar R, Ullah I, Ahmed A, Noor W, Baber J. Towards intelligent P2P IPTV overlay management through classification of peers. Peer Peer Netw Appl. 2022;15:827–38.
15. Iqbal MJ, Ullah I, Ali M, Ahmed A, Noor W, Basit A. Machine learning-based stable P2P IPTV overlay. Comput, Mater Contin. 2022;71(3):5381–97. doi:10.32604/cmc.2022.024116.
16. Zare S. A program-driven approach joint with pre-buffering and popularity to reduce latency during channel surfing periods in IPTV networks. Multimed Tools Appl. 2018;77:32093–105. doi:10.1007/s11042-018-6235-7.
17. Kim E, Lee C. An on-demand TV service architecture for networked home appliances. IEEE Commun Mag. 2008;46:56–63. doi:10.1109/MCOM.2008.4689208.
18. Hecht F, Bocek T, Clegg R, Landa R, Hausheer D, Stiller B. Liveshift: mesh-pull live and time-shifted P2P video streaming. In: Proceedings of IEEE Conference on Local Computer Networks; 2011; Bonn, Germany. p. 315–23.
19. Gallo D, Miers C, Coroama V, Carvalho T, Souza V, Karlsson P. A multimedia delivery architecture for IPTV with P2P-based time-shift support. In: Proceedings of IEEE Consumer Communications and Networking Conference; 2009; Las Vegas, NV, USA. p. 1–2.
20. Kim E, Kim J, Shin H. Cooperative recording to increase storage efficiency in networked home appliances. IEICE Trans Inf Syst. 2022;105(3):727–31. doi:10.1587/transinf.2021EDL8077.
21. Li JY, Li B. Demand-aware erasure coding for distributed storage systems. IEEE Trans Cloud Comput. 2018;9(2):532–45. doi:10.1109/TCC.2018.2885306.
22. Qiao Y, Zhang M, Zhou Y, Kong X, Zhang H, Xu M, et al. NetEC: accelerating erasure coding reconstruction with in-network aggregation. IEEE Trans Parallel Distrib Syst. 2022;33(10):2571–83. doi:10.1109/TPDS.2022.3145836.
23. Zhou T, Tian C. Fast erasure coding for data storage: a comprehensive study of the acceleration techniques. ACM Trans Storage. 2020;16(1):1–24. doi:10.1145/337555.
24. Panda S, Naik S. An efficient data replication algorithm for distributed systems. Int J Cloud Appl Comput. 2018;8(3):60–77. doi:10.4018/978-1-7998-5339-8.ch065.
25. Huang Z, Yuan Y, Peng Y. Storage allocation for redundancy scheme in reliability-aware cloud systems. In: Proceedings of IEEE International Conference on Communication Software and Networks; 2011; Xi'an, China. p. 275–9.
26. Westerlund O. A Concept on the Internet-based Television [M.S. thesis]. KTH Royal Institute of Technology; 2014 [cited 2024 Dec 25]. Available from: https://www.diva-portal.org/smash/get/diva2:839253/FULLTEXT01.pdf.
27. Qi J, Neumann P, Reimers U. Dynamic broadcast. In: Proceedings of ITG Conference on Electronic Media Technology; 2011; Dortmund, Germany. p. 1–6.
28. Melzer J, Melzer N. Standardizing web TV: adapting ATSC 3.0 for use on any IP network. In: IEEE International Symposium on Broadband Multimedia Systems and Broadcasting; 2024; Toronto, ON, Canada. p. 1–5.
29. Petkovic P, Valeev T, Basicevic I. Enhancing caching efficiency of DSM-CC data carousel BIOP messages for android TV broadcast stack virtual file system. In: Zooming Innovation in Consumer Technologies Conference; 2024. Novi Sad, Serbia. p. 209–12.
30. Zhu Y, Li Z, Sun L, Gao L. Lightweight multi-role recommendation system in TV live-streaming. In: IEEE International Symposium on Broadband Multimedia Systems and Broadcasting; 2023; Beijing, China. p. 1–6.
31. Konwar KM, Prakash N, Lynch N, Medard M. A layered architecture for erasure-coded consistent distributed storage. In: Proceedings of ACM Symposium on Principles of Distributed Computing; 2017; Washington, DC, USA. p. 63–72.
32. Okada H, Shiroma T, Wu C, Yoshinaga T. A color-based cooperative caching strategy for time-shifted live video streaming. In: Proceedings of International Symposium on Computing and Networking Workshops; 2018; Takayama, Japan. p. 119–24.
33. Nogueira J, Gonzalez D, Guardalben L, Sargento S. Over-the-top catch-up TV content-aware caching. In: Proceedings of IEEE Symposium on Computers and Communication; 2016; Messina, Italy. p. 1012–7.
34. Neumann P, Reimers U. Live and time-shifted content delivery for dynamic broadcast: terminal aspects. IEEE Trans Consum Electron. 2012;58(1):53–9. doi:10.1109/TCE.2012.6170055.

35. Avramova Z, Vleeschauwer D, Wittevrongel S, Bruneel H. Performance analysis of a caching algorithm for a catch-up television service. Multimed Syst. 2011;17:5–18. doi:10.1007/s00530-010-0201-1.
36. Hwang IS, Tesi C, Pakpahan AF, Ab-Rahman MS, Liem AT, Rianto A. Software-defined time-shifted IPTV architecture for locality-awareness TWDM-PON. Optik. 2020;207:164179. doi:10.1016/j.ijleo.2020.164179.
37. Lall S, Sivakumar R. Will they or won't they?: toward effective prediction of watch behavior for time-shifted edge-caching of netflix series videos. In: Proceedings of ACM/IEEE Symposium on Egde Computing; 2021; San Jose, CA, USA. p. 257–70.
38. Li Z, Simon G. Time-shifted TV in content centric networks: the case for cooperative in-network caching. In: Proceedings of the IEEE Conference on Communications; 2011; Kyoto, Japan. p. 1–6.
39. Lall S. Time-shifted prefetching and edge-caching of video content: insights, algorithms, and solutions [PhD dissertation]. Georgia Institute of Technology; 2022 [cited 2024 Dec 25]. Available from: https://repository.gatech.edu/entities/publication/d045fae8-1cfb-4e1c-adae-04cf6904f399.
40. Al-Abbasi AO, Aggarwal V, Lan T, Xiang Y, Ra MR, Chen YF. FastTrack: minimizing stalls for CDN-based over-the-top video streaming systems. IEEE Trans Cloud Comput. 2021;9(4):1453–66. doi:10.1109/TCC.2019.2920979.
41. Pal R, Sastry N, Obiodu E, Prabhu S, Psounis K. EdgeMart: a sustainable networked OTT economy on the wireless edge for saving multimedia IP bandwidth. ACM Trans Auton Adapt Syst. 2023;18(4). doi:10.1145/3605552.
42. Dai J, Xia K, Hua Z, Kou Z. Personal video recorder function based on android platform set-top box. Telkomnika Indonesian J Electr Eng. 2014;12(1):550–57. doi:10.11591/telkomnika.v12i1.3231.
43. Kim Y, Shin D. Improving file system performance and reliability of car digital video recorders. IEEE Trans Consum Electron. 2015;61(2):222–9. doi:10.1109/TCE.2015.7150597.
44. Li X, Li R, Lee PPC, Hu Y. OpenEC: toward unified and configurable erasure coding management in distributed storage systems. In: Proceedings of USENIX Conference on File and Storage Technologies; 2019; Boston, MA, USA. p. 331–44.
45. Liu C, Wang Q, Chu X, Leung YW, Liu H. ESetStore: an erasure-coded storage system with fast data recovery. IEEE Trans Parallel Distrib Syst. 2020;31(9):2001–16. doi:10.1109/TPDS.2020.2983411.
46. Xu L, Lv M, Li Z, Li C, Xu Y. PDL: a data layout towards fast failure recovery for erasure-coded distributed storage systems. In: Proceedings of IEEE INFOCOM-IEEE Conference on Computer Communications; 2020; Toronto, ON, Canada. p. 736–45.
47. Deng M, Liu F, Zhao M, Chen Z, Xiao N. GFCache: a greedy failure cache considering failure recency and failure frequency for an erasure-coded storage system. Comput Mater Contin. 2019;58(1):153–67. doi:10.32604/cmc.2019.03585.
48. Abebe M, Daudjee K, Glasbergen B, Tian Y. EC-store: bridging the gap between storage and latency in distributed erasure coded systems. In: Proceedings of the IEEE International Conference on Distributed Computing Systems; 2018. Vienna, Austria. p. 255–66.
49. Xu L, Lyu M, Li Z, Li Y, Xu Y. Deterministic data distribution for efficient recovery in erasure-coded storage systems. IEEE Trans Parallel Distrib Syst. 2020;31(10):2248–62.
50. Mammeri A, Boukerche A, Fang Z. Video streaming over vehicular ad hoc networks using erasure coding. IEEE Syst J. 2016;10(2):785–96. doi:10.1109/JSYST.2015.2455813.
51. Zhang XJ, Peng XH. A testbed of erasure coding on video streaming system over lossy networks. In: International Symposium on Communications and Information Technologies; 2007; Sydney, Australia. p. 535–40.
52. Toporkov V, Yemelyanov D. Availability-based resources allocation algorithms in distributed computing. In: Supercomputing. Moscow, Russia: Springer; 2020. p. 551–62.
53. Leontiadis I, Marzolla M, Datta A, Rieche S, Cilia M. Choosing partners based on availability in P2P networks. In: Proceedings of ACM Conference on Distributed Computing and Networking; 2012; Hong Kong, China. p. 9–24.