Acknowledgement

This is joint work with Hao Zhang(UC Berkeley), Ziyu Shao(CUHK), and Minghua Chen(CUHK).

An Optimized Distributed Video-on-Demand Streaming System: Theory and Design

by Kang Wook Lee

Research Project

Submitted to the Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, in partial satisfaction of the requirements for the degree of **Master of Science, Plan II**.

Approval for the Report and Comprehensive Examination:

Committee:

Kannan Ramchandran
Research Advisor

Date

Abhay Parekh
Second Reader

Date

Abstract

In this report[1], we present a general framework for a distributed Video-on-Demand content distribution problem by formulating an optimization problem yielding a highly distributed implementation that is highly scalable and resilient to changes in demand. Our solution takes into account several individual node resource constraints including disk space, network link bandwidth, and node-I/O degree bound. First, we present a natural formulation that is NP-hard. Next, we design a simple fractional storage architecture based on codes to "fluidify" the content, thereby yielding a convex content placement problem. Third, we use a recently developed Markov approximation technique to approximately solve the NP-hard problem of topology selection under node degree bound, and propose a simple distributed solution. We prove analytically that our algorithm achieves close-to-optimal performance. Based on the distributed algorithm we design, we consider practical settings and propose a practical design of a distributed VoD system. We implement a packet level simulator written in MATLAB. Our simulation results show that the proposed design and algorithms improve significantly over existing approaches including Least-Recently-Used (LRU), Least-Frequently-Used (LFU) and Mixed Integer Programming (MIP) [2]. We establish via simulations that the system is robust to changes in user demand or in network condition and churn. Based on the algorithm derived from theory, we design a practical algorithm considering practical issues.

[1]Joint work with Hao Zhang(UC Berkeley), Ziyu Shao(CUHK), and Minghua Chen(CUHK). Parts of this report were published in [1]. The main contributions relate to Ch. 6 and Ch. 7.

Contents

1 Introduction

The shape of the internet is changing. On the one hand, large internet exchanges and datacenters have made it possible to centralize a lot of compute and storage resources. It is clear that established players such as Netflix, Google and Amazon are more likely to exploit the reliability and economies of scale that come from such architectures, and that is of course what they are doing. On the other hand, the edge of the internet is growing exponentially. Phones, tablets, laptops, ebook readers are growing in power and sophistication to the point that that they resemble the desktop computers of a few years ago. But fully 50% of the traffic of the internet does not originate from data centers and hierarchical CDNs[3]. This traffic consists of applications such as file sharing and P2P, and relies more heavily on edge-devices which take on the role of potentially unreliable servers. As we play out the evolution of content and the internet it seems clear that both kinds of content distribution will co-exist. Videos of police action in an oppressive state are easier to detect and throttle in a centralized architecture than a distributed one, and there will always be situations in which small groups of individuals will want to share content without the "prying eyes" of a media giant.

In this report we present a highly distributed edge-based scheme where the content is streamed video and has quality of service constraints. The demand for these videos is not specified to the system. Each edge device has limited storage but can store content from any video (even ones that the owner of that device is not watching). The devices are assumed to have limited connectivity and the network is allowed to be unreliable. As demand and network connectivity fluctuates so does what is stored at each node. In other words our goal is to test the limits of how unreliable and distributed we can make the infrastructure and still meet the stringent quality of service constraints of streamed video. Central to our architecture is the existence of a single reliable server or Seedbox [4], that can fill in the gaps of service when our distributed algorithms cannot meet QoS constraints and the objective of our distributed algorithms is to ensure that for any set of adverse network conditions, the load on the Seedbox is minimized.

Our approach is the following. First, we formulate the problem as a static convex optimization problem and then show that a highly distributed algorithm converges to the optimal allocation. We then examine, via simulations, the dynamic problem in which network conditions change, and the demand for the underlying content fluctuates significantly. We find that our scheme is highly resilient to changes in network and in demand in addition to being near optimal in a static setting.

More concretely, in formulating the optimization problem we jointly solve the following problems:

1. Content Placement: What content should be stored at each device/node given the storage constraints, network capacity and current demand?
2. Overlay Topology: Given that each node can only support a bounded number of end devices, how should end devices be matched to nodes?
3. Minimal Server Load: When there is no available node that can serve an end device to watch a specific piece of content from, the Seedbox "fills in the gap" by streaming directly to it. We wish to minimize such occurrences.

We illustrate these problems further in the example depicted in Figure 1. The system has two 1 GB videos, A and B, which must be delivered at a streaming rate of 1 Mbps. There are 4 users: two request video A and two request video B. The three cache nodes are constrained by bandwidth, maximum degree (a bound on the number of simultaneously supported streaming connections) and storage. Figure 1(b) shows that under a certain "bad" topology and a "bad" content allocation scheme, demand cannot be satisfied. In Figure 1(c), a "good" content placement strategy is chosen. In Figure 1(d), a "good" topology is also chosen. As we can see, the three problems enumerated earlier are closely related. Further, taken in isolation each problem is hard: there are an exponential number of possible topologies (from which we must select in a distributed manner) and as we will see (although it may be clear to some readers even at this point), the content selection problem for a fixed topology, is also NP-hard.

The contributions of this report are four-fold. To the best of our knowledge, we are the first to jointly optimize topology selection, content placement, and link rate allocation in the design of VoD systems. Second, we design a fractional coding architecture that allows for "fluidification" of the content placement problem, thereby converting a combinatorial VoD content placement problem into a convex problem that admits a simple distributed algorithm. Third, we overcome the difficulty of the combinatorial topology graph selection problem by applying a recently-developed and

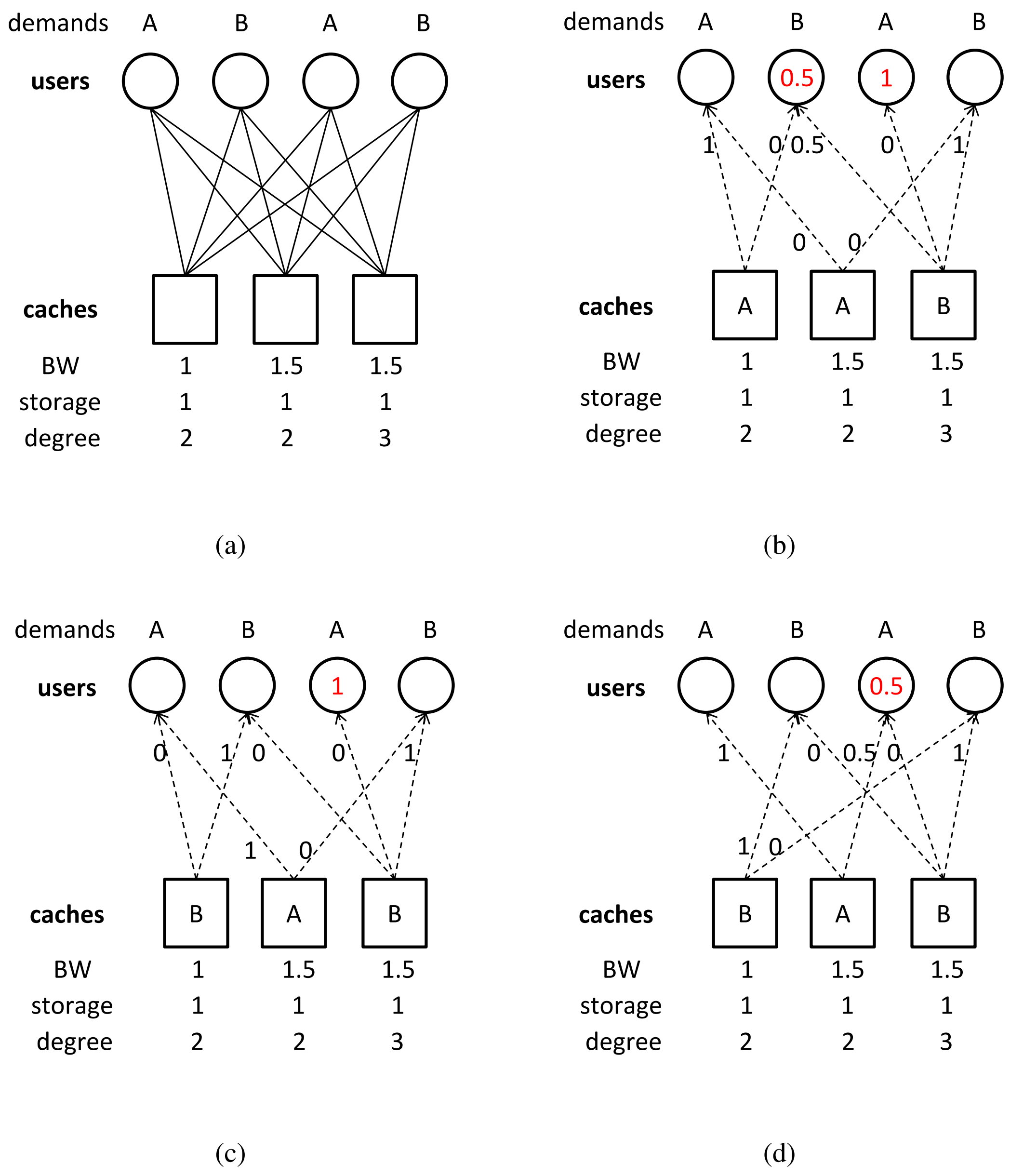


Figure 1: A simple example of VoD caching problem. The system has two videos of size 1 GB and rate 1 Mbps and there are 2 users requesting each video. The system employs 3 cache nodes with constraints on storage, bandwidth and out-degree as shown in (a). The problem is to decide, for each cache, which videos to store, which users to connect to, and how much bandwidth to allocate for each user. These questions are coupled. The connections between the cache nodes and users in (b) form a "bad" topology, and the content placement is non-optimal. The content placement in (c) is "good". In (d), the topology is "good", and with the same content placement strategy in (c), only one user is in deficit of half of a video. In general, finding the "best" storage, bandwidth and topology combination is a combinatorially-hard problem.

novel Markov approximation technique, and design a simple and distributed “soft-worst-neighbor-choking” algorithm. Fourth, we give a theoretical proof that our solution is close-to-optimal and that the optimality gap diminishes when the number of users becomes large. We validate that our algorithm outperforms existing results via extensive real world trace simulations. While our theoretical analysis is based on a “static” setting where the demand and supply resources are fixed, we also show in our experiments that our algorithm works well even for “dynamic” cases. Our strong findings suggest that a functional implementation of our solution would perform well in practice, and this is the focus of our future work.

2 Related Work

The optimization of VoD systems has received wide attention [5, 6, 7, 8, 9, 10, 2]. Almeida et al. [5] studied the delivery cost minimization problem under a fixed topology by optimizing over content replication and routing. Boufkhad et al. [6] investigated the problem of maximizing the number of videos that can be served by a collection of peers. Zhou et al. [7] focused on minimizing the load imbalance of video servers while maximizing the system throughput. Tan and Massoulie [10] studied the problem of optimal content placement in P2P networks. Their goal is to maximize the utilization of peer uplink bandwidth resources. Optimal content placement strategies are identified in a particular scenario of limited content catalog under the framework of loss networks. Their work assumes that the peers' storage capacity grows unboundedly with system size. In contrast, our work does not make any assumption on the storage capacities and also takes into overlay topology. Applegate et al. [2] formulated the problem of content placement into a mixed integer program (MIP) that takes into account constraints such as disk space and link bandwidth. However, they assume the knowledge of content popularity under a fixed topology and a video is either stored in full or not stored. In our work, we use a class of network codes that enables fractional storage and do not assume any prior knowledge of demand is given. We also optimize over the topology graph selections.

With regard to network resource utilization, Borst et al. [11] solved a link bandwidth utilization problem assuming a tree structure with limited depth. An LP is formulated, and under the assumption of symmetric link bandwidth, demand, and cache size, a simple local greedy algorithm is designed to find a close-to-optimal solution. Valancius et al. [12] propose an LP-based heuristic to calculate the number of video copies placed at customer home gateways. The network topology in our work is not constrained to be a tree, and the video request patterns can be arbitrary in different network areas. Zhou and Xu [13] aimed to minimize the load imbalance among servers subject to disk space egress link capacity from servers. In contrast, we consider the link capacity constraints that may exist anywhere in the network.

Topology building is also an important design dimension and has been studied in various works [14, 15, 16]. While most works focus on enforcing locality-awareness and/or improving ISP-friendliness, they make the simple assumption that the graph is fully connected, *i.e.,* no node-degree-bound is taken into consideration. Zhang et al. solves the problem of optimal P2P streaming under node degree constraints [17]. However their topology selection algorithm depends on global statistics which are easily accessible under a live-streaming scenario. In VoD, directly applying their technique requires global statistics of all users' utility functions, which can create an enormous overhead. Our distributed algorithm requires knowledge of only local information of neighboring overlay link rates.

To the best of our knowledge, we are unaware of any other works to jointly optimize topology graph selection, content placement and link rate allocation. Our solution is fully distributed and adapts well to system dynamics as we will show in the simulations.

3 Problem Formulation

In this chapter, we define the VoD optimization problem. Our formulation assumes a static setting: the video catalog, the set of users and subscriptions, and the set of caches are fixed. In chapter 4 and chapter 5, we find a fully distributed algorithm and prove it achieves near-optimal performance. In chapter 6, we apply the proposed algorithm to dynamic setting and show its performance through simulations.

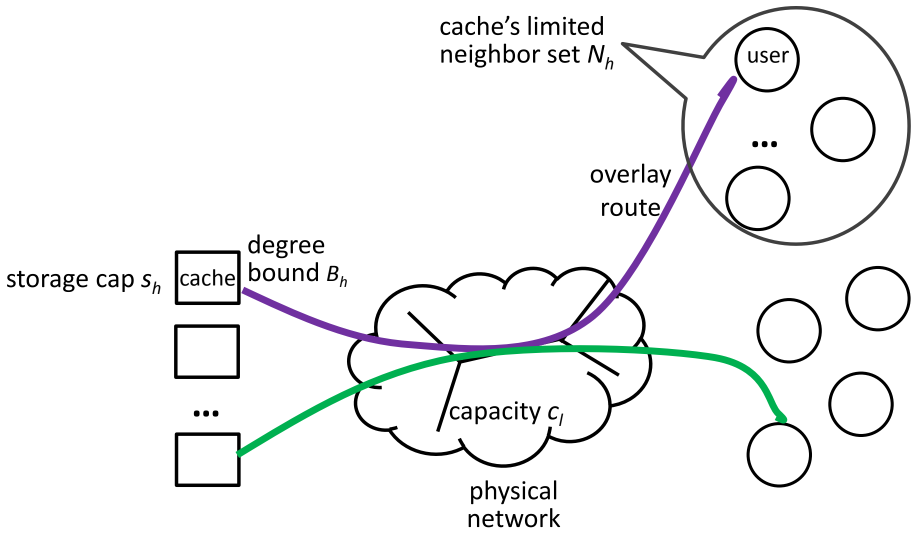


Figure 2: Caches and users connected by a physical network. The link capacity constraints can exist arbitrarily anywhere in the network.

As illustrated in Figure 2, a set of caches(H) and a set of users(U) are connected by a fixed physical network which consists of links with capacity. Caches and users are connected by a overlay graph configuration, g, which is expressed by the set of overlay links R and the corresponding routing matrix A. We denote the set of overlay links and the routing matrix by R^g and A^g under a graph configuration g. Each overlay link $r \in R$, consists of a set of underlay links, $L_r \subset L$ and we say $l \in r$ if $l \in L_r$. An overlay link $r = (h, u)$ enables cache node h to send data to node u in the overlay graph by setting up TCP/UDP connections. The routing matrix $A := (A_{lr}, (l, r) \in L \times R)$ is defined as usual where $A_{lr} = 1$ if $l \in r$ and 0 otherwise, We denote the connected neighbor of node v by N_v^g. Node v cannot connect to more than B_v users, which leads the node-I/O constraints. Let $G = \{\forall g||N_v^g| \leq B_v \;\; \forall v \in H \cup U\}$ be the set of possible overlay graphs. Let link $l \in L$ have a capacity c_l, and let x_r be the rate on the overlay link r. This leads to natural routing constraints as follows: $Ax \leq c$, where x is the column vector of the overlay rates x_r and c is the column vector of link capacity constraints c_l.

The video catalog consists of videos in the set M. Each video m has size β_m and is streamed at a constant rate of γ_m. Since we assume a fixed demand in this model, we denote the set of users watching m as U_m. To model the storage constraints, let s_h be the storage capacity of cache node h, and denote by $s := (s_h, h \in H)$ the column vector of the storage capacities. Let $W := (W_{hm}, h \in H, m \in M)$ be the storage matrix where $W_{hm} \in \{0, 1\}$ indicates if video m is stored on cache node h ($W_{hm} = 1$) or not ($W_{hm} = 0$). Denote by $\beta := (\beta_m, m \in M)$ the vector of the sizes (in MB) of all videos. The storage constraints can then be expressed by $W\beta \leq s$. Availability constraints are also modeled : caches can only serve stored videos. From cache h to user u, the streaming rate can be only 0 if cache does not store the video m which is being viewed by the user. If the cache stores the video, the streaming rate can be anything no greater than γ_m. It can be equivalently expressed as $x_{r:=(h,u)} \leq W_{hm}\gamma_m$.

Let $z_u = \sum_{r=(h,u):h \in N_u^g} x_r$ be the total received rate of user u, and $V^u(z)$ be a concave function that represents the utility of user u when the received rate is z. Table 1 lists all relevant notations.

Now, we have the following optimization problem. The objective is to find a graph g, content placement W, and rate allocation x, which jointly maximize the sum of user utilities under constraints.

$$\begin{aligned} \max_{\substack{g \in G \\ x_r \geq 0, W_{hm} \in \{0,1\}}} \quad & \sum_{u \in U} V^u(z_u) \\ \text{s.t.} \quad & x_{r:=(h,u)} \leq W_{hm}\gamma_m, \\ & A^g x \leq c, \; W\beta \leq s. \end{aligned}$$

The above optimization is highly difficult to solve due to the exponentially large size of the feasible graph set G and integer constraints of storage matrix W. Our approach to solve the problem is following. First, we decompose the problem into two parts: graph selection part and resource allocation which consists of content placement and rate allocation. In chapter 4, we first solve the optimal rate allocation algorithm under a fixed graph. In the following chapter 5, we propose an algorithm to find the near-optimal overlay graph.

Table 1: Key Notations

Parameters	**Definition**
H, U, M	set of caches / users / videos
U_m	set of users watching video m
γ_m, β_m	video m's streaming rate and size
s_h	storage capacity of cache h
G, B_v	set of feasible overlay graphs / I/O constraints
L, c_l	set of underlay links / link capacity
Auxiliary Variables	**Definition**
θ_l, q_r	shadow price of link l / route r
q_r	$q_r = \sum_{l \in r} \theta_l$ is the shadow price of route r
$\lambda_r, \Sigma_{h,m}$	demand index of route r / video m at cache h
ω_h	storage price of cache h
Decision Variables	**Definition**
x_r	route rate of r
W_{hm}	storage of video m on cache h
g, A^g, R^g	overlay graph / routing matrix / overlay links
N_v^g	connected neighborhood of v under g
p_g	probability of each topology graph g

4 Resource Allocation

The sub-optimization problem under a fixed g is still difficult to solve because of the integer constraints of storage matrix W. In order to overcome this challenge, we allow caches to store videos in parts and to serve *fractional streams*.

4.1 Fractional Storage and Streams with Coding

We first define fractional streams.

Definition 1. *Given a data stream S_0 with rate x_0, $S_0(x_0)$, a fractional stream S of rate x, $S(x)$, is defined as a data stream which satisfies the following conditions.*
1. Additivity : $S(x)$ can be generated by $S(x_1)$ and $S(x_2)$ if $x_1 + x_2 \geq x$.
2. Recovery : $S_0(x_0)$ can be generated by any $S(x)$ if $x \geq x_0$.

In our techincal report [18], we propose a novel way of storing videos and generating fractional streams. In a nutshell, a fractional stream can be realized through the use of MDS codes or rateless fountain codes. The minimum storage to serve a fractional stream is found as following.

Proposition 2. *Given a data stream $S_0(x_0)$, a cache can generate a fractional stream $S(x)$ if and only if the cache stores no less than $\frac{x}{x_0}$ fraction of the source file of the stream.*

By using the concept of fractional storage and streams and the proposition 2, the sub-optimization problem can be reformulated. Caches can store videos in parts and upload fractional streams to users. Users receive multiple fractional streams from caches and add them. If they received enough fractional streams, they can recover the original stream. If not, they will request a fractional stream with the deficit rate at user and recover the original stream. The integer constraints are replaced with box constraints and the availability constraints remain as same by the proposition. The reformulated optimization problem is following.

$$\begin{aligned} \max_{x \geq 0, 0 \leq W_{hm} \leq 1} \quad & \sum_{u \in U} V^u(z_u) \\ \text{s.t.} \quad & x_{r:=(h,u)} \leq W_{hm}\gamma_m, \\ & A^g x \leq c, \ W\beta \leq s. \end{aligned} \tag{1}$$

Note that our approach completely differs from the well-known class of relaxations to hard mixed-integer problems in the optimization literature, in which the relaxation is done to convert the problem to a computationally tractable one whose solution is then quantized back to enforce feasibility with the discrete-valued constraints of the original problem. In other words, our approach increases performance because using fractional streams expands the design space. Figure 3 shows an example where resource allocation with fractional streams outperforms one without them.

There is an added complication if we are interested in distributed operation and for the ability to have cache nodes populated in a distributed way with content (e.g. from a distributed server system or from other cache nodes): this requires the notion of network coding which can attain the abstraction of distributed MDS codes. We use a specific class of such codes, dubbed DRESS codes [19], in our system.

4.2 Distributed Solution

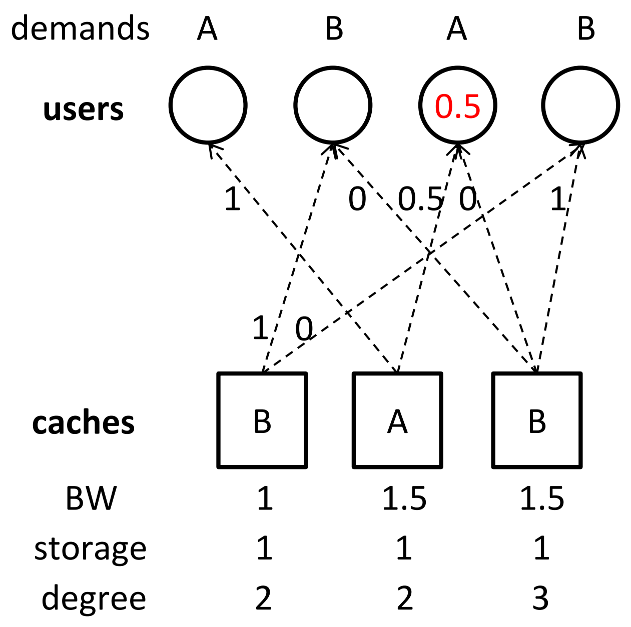
We present the following theorem.

Theorem 3. *The problem* (1) *is convex and can be solved by the following three-step primal dual algorithm that converges to the optimal solution*
1) Step 1: Update the upload rate.

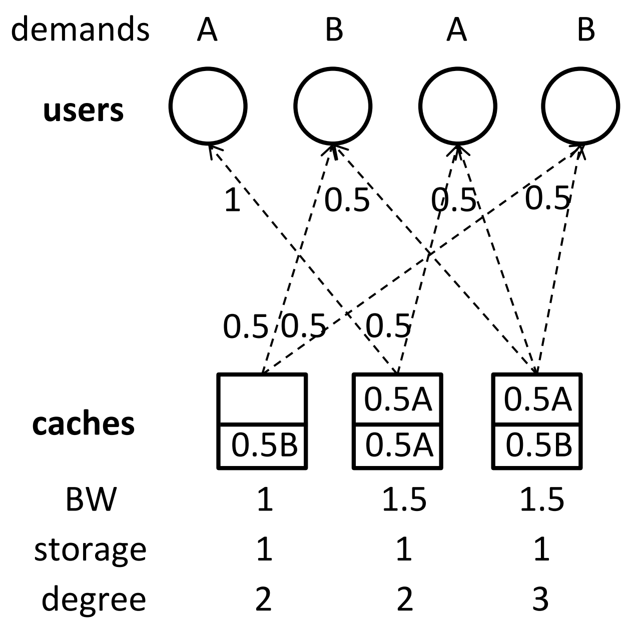
$$\dot{x}_r = [\delta_r(V_{x_r}(z_u) - \lambda_r - q_r)]^+_{x_r} \tag{2}$$

where $V^u_{x_r}(z_u)$ *is the derivative of function* $V^u(z_u)$ *with regard to* x_r. $q_r = \sum_{l:l \in r} \theta_l$ *is the aggregate route price, where* θ_l *is the single link price on link* l *which is updated by:*

$$\dot{\theta}_l = [\eta_l(\sum_{r:l \in r} x_r - c_l)]^+_{\theta_l}. \tag{3}$$



(a)



(b)

Figure 3: A simple example demonstrating the usefulness of fractional storage. The system has the same setup as that in Figure 1. In (a), the videos are stored in full. Even with the optimal topology, the oracle server still needs to fill in the gap of half of a video. In (b), with fractional storage and streams, caches can fully serve users even with less storage.

where $\delta_r, \eta_l > 0$ are adaptation parameters. λ_r is explained in the following.

2) Step 2: Update the demand index. The parameter λ_r is updated via:

$$\dot{\lambda}_r = [\kappa_r(x_r - W_{hm}\gamma_m)]^+_{\lambda_r} \tag{4}$$

where step size κ_r is a positive constant, and m is the video user u watches. This parameter captures the relative demand, i.e., *absolute demand minus supply, of the video delivered on route r, hence its name demand index.*

3) Step 3: Update cache storage

$$\begin{cases} \dot{W}_{hm} = [\iota_{hm}(\Lambda_{hm} - \beta_m\omega_h)]^{[0,1]}_{W_{hm}} \\ \dot{\omega}_h = [\nu_h(\sum_{m\in M} W_{hm}\beta_m - s_h)]^+_{\omega_h} \end{cases} \tag{5}$$

where the Lagrangian variable ω_h is interpreted as the storage price for the storage constraint, and $\Lambda_{hm} = \gamma_m \cdot (\sum_{r=(h,u):u\in U_m, h\in N^g_u} \lambda_r)$ is the aggregate demand index and β_m the size of video m.

The proof is relegated to our technical report [18]. The above algorithms can be implemented in a fully distributed way. The update equations are intuitive. The upload rate increases linearly with the marginal utility function and decreases with the demand index and link shadow price. The demand index increases if the desired link rate exceeds what the stored video can offer, and vice versa. Cache node h increases storage of video m if there is more demand than supply, *i.e.,* when the popularity index is larger than the storage price; and vice versa.

As a special case, if our goal is to minimize the server load, the following utility function $V^u(z) = \min(\gamma_m, z)$ can be used, where m is s.t. $u \in U_m$. Maximizing such an objective function is equivalent to minimizing the server load. Thus, with the Theorem 3, we now have a fully distributed algorithm to find the optimal resource allocation which minimizes the server load under a fixed topology.

5 Topology Building

Given the bounded degree of each cache node, the topology selection problem is NP-hard. We apply a Markov approximation technique to establish that it is indeed a close-to-optimal solution of the general problem.

5.1 Problem Formulation

Let Φ_g be the optimal objective value to (1) under a fixed graph g, which we can solve using the algorithms shown in chapter 4. Our goal is to maximize the overall system utility by choosing the best graph in the feasible set of graphs.

$$\max_{g \in G} \quad \Phi_g \tag{6}$$

Directly solving for (6) is challenging. Even for the small example in Figure 1(a), the total number of topology graphs is $6 \times 6 \times 4 = 144$. In fact, we can show that in general, it is NP-complete and APX-hard (no effective centralized polynomial-time approximate solution) [18]. To convert the problem into a solvable one with distributed solutions, we design a Markov approximation technique introduced in [20].

The topology building problem can be re-written in the following way:

$$\max_{\substack{p_g \geq 0 \\ \sum p_g = 1}} \sum_{g \in G} p_g \Phi_g . \tag{7}$$

where p_g is the probability associated with topology graph $g \in G$. Its optimal solution is $\max_{g \in G} \Phi_g$, and is obtained by setting the probability corresponding to (one of) the "best" topology graphs to be 1 and the rest probabilities to be 0.

To mitigate the problem of exponentially large number of graphs, consider using

$$\frac{1}{\mu} \log(\sum_{g \in G} \exp(\mu \Phi_g)) \tag{8}$$

to approximate $\max_{g\in G}\Phi_g$. The reason is that while solving for $\max_{g\in G}$ is difficult, solving for (8) can be made much easier as we will explain later on. We present the following theorem adapted from [20].

Theorem 4. (8) *is the optimal value to the following convex optimization problem:*

$$\max_{\substack{p_g\geq 0\\ \sum p_g=1}} \quad \sum_{g\in G} p_g\Phi_g - \frac{1}{\mu}\sum_{g\in G} p_g \log p_g. \tag{9}$$

where μ is a positive constant. The optimal solution is given by:

$$p_g^* = \frac{\exp(\mu\Phi_g)}{\sum_{g'\in G}\exp(\mu\Phi_{g'})},\ \forall g\in G. \tag{10}$$

Moreover, the gap between its optimal value (8) *and the optimal value of the original problem in* (7) *is given by:*

$$0 \leq \frac{1}{\mu}\log(\sum_{g\in G}\exp(\mu\Phi_g)) - \max_{g\in G}\Phi_g \leq \frac{1}{\mu}\log|G|, \tag{11}$$

where $|G|$ is the size of G.

Note that $\sum_{g\in G} p_g\log p_g$ is also the entropy $H(p)$ of the distribution $p := (p_g, g\in G)$, and we call this is an “entropy-approximated” formulation. The only difference between the objective function in (9) and that in (7) is the addition of the weighted entropy term. As $\mu\to+\infty$, $p^*_{g^*}\to 1$ where $g^* = \arg\max_{g\in G}\Phi_g$ and $p^*_g\to 0$ otherwise. Therefore, the optimal value in the entropy-approximated formulation approaches that of the original problem as μ becomes large.

Intuitively, one can see that $\frac{1}{\mu}\log(\sum_{g\in G}\exp(\mu\Phi_g))$ is a good approximation of $\max_{g\in G}\Phi_g$ when μ is large. This is because the term $\exp(\mu\Phi_{g^*})$ will dominate the sum of exponentials $\sum_{g\in G}\exp(\mu\Phi_g)$. When this happens, we have:

$$\frac{1}{\mu}\log(\sum_{g\in G}\exp(\mu\Phi_g)) \to \frac{1}{\mu}\log(\exp(\mu\Phi_{g^*})) = \Phi_{g^*} \tag{12}$$

Recall that our goal is to solve for problem (7), and we know that one approximation is given by (9), but why is this approximation useful? Given the vast number of possibilities, how would we design a distributed algorithm to achieve the optimal topology graph? The key idea is to construct a Markov chain (MC) on the topology graphs, and design its transition rates that can be implemented in a distributed way. Equation (10) is of a product form, and can be the stationary distribution of some time-reversible MC with state space the set of all the topology graphs G. When the MC is in its stationary status, the topology graphs $g \in G$ are time-shared according to the distribution p_g^* in (10). As μ becomes large, the system spends most of the time in the optimal topology graph and the gap between the approximated solution and the optimal solution approaches zero. It was shown in [20, 17] that it is possible to design such Markov chain to guide distributed algorithm designs in various domains, including wireless scheduling, channel assignment and etc. However, in our problem, directly applying these known design options in [20, 17] does not result in Markov chains that are implementable in a distributed fashion.

5.2 Markov Chain Design

To design the Markov chain for our topology selection, we focus on the design of transition rates $q_{g,g'}$ between states g and g'. With slight abuse of notation, we use g and the set of the corresponding overlay routes R interchangeably. We begin by constructing a Markov chain that solves (9) exactly. This Markov chain does not lead to a fully distributed solution, but serves as a seed to our extended discussions in later parts of the report.

To simplify the design, we allow non-zero transition rates from topology graph g to graph g' if and only if they satisfy the following set of conditions, which we name as the "direct-transition condition". Specifically for any topology graphs (or set of overlay routes) g and g', $q_{g,g'}$ is non-zero if and only if for the union of their routes $\tilde{g} = g \cup g'$:

- $|\tilde{g} \setminus g| = 1$ and $|\tilde{g} \setminus g'| = 1$;
- the only overlay route in $\tilde{g} \setminus g$ and that in $\tilde{g} \setminus g'$ should originate from the *same* node denoted by $v(g, g')$.

In other words, we allow transitions to happen only when *a single node* adds a not-in-use neighbor and then drops one active neighbor. $\tilde{g} = g \cup g'$ is a *transient* state where a node has *just* added a new neighbor before choking one of the old neighbors [2]. We have the following

[2]The node can temporarily violate the node degree bound, but the process happens instantaneously and the transient

proposition.

Proposition 5. *The following transition rates ensures that the MC over the topology graph will reach the stationary distribution in* (10).

$$q_{g,g'} = \tau^{-1} \cdot \frac{\exp(\mu(\Phi_{g'} - \Phi_{\tilde{g}}))}{1 + \sum_{g'' \in \mathcal{A}_{v(g,g'),\tilde{g}}} \exp(\mu(\Phi_{g''} - \Phi_{\tilde{g}}))}, \tag{13}$$

if g and g' satisfy the direct-transition condition, and $q_{g,g'} = 0$ otherwise, where $\tilde{g} = g \cup g'$ is the transient state, $\tau > 0$ is a constant, $\mathcal{A}_{v,\tilde{g}}$ is the set of topology graphs derived by node v dropping one of its active neighbors under graph $\tilde{g}$, i.e.,

$$\mathcal{A}_{v,\tilde{g}} \quad = \quad \left\{\hat{g} \in G \mid \hat{g} = \tilde{g} \backslash \{(v,u)\}, \forall u \in N_v^{\tilde{g}}\right\}, \tag{14}$$

where (v,u) is a directed link from node v to node u, and $v(g,g')$ is the node defined in direct-transition condition.

We call this MC an *exact* MC. The above transition rates achieve the optimal value of (9). The proof and an exemplar implementation are given in our technical report [18]. The algorithm however, requires every node v to know global statistics $\Phi_{g'} - \Phi_{\tilde{g}}$ for all $g' \in \mathcal{A}_{v,\tilde{g}}$, which makes distributed implementation difficult.

In the following, we modify the exact MC to a *perturbed* MC which yields a distributed "soft-worst-neighbor-choking" algorithm.

5.3 Soft Worst Neighbor Choking

Let g denote the current topology graph. Each cache node $v \in H$ (and similarly for user node $u \in U$) implements the soft-worst-neighbor-choking algorithm as follows:

- **Initialization:** It randomly selects and builds connections with B_v (which is its maximum degree) number of neighbors from its neighbor list Q_v. Denote by N_v^g the connected neighbors.

state is almost non-existent. We use it only to simplify our theoretical discussions.

- **Step 1:** It counts down to zero from a timer $T = \tau/(|Q_v| - |N_v^g|)$, where $\tau > 0$ is a constant[3].
- **Step 2:** When the count-down expires, it randomly chooses a new inactive neighbor w from $Q_v \setminus N_v^g$, and requests to connect to it. If the node degree bound of neither node w nor v is violated, the connection is established; otherwise, no new connection is made. The system transits to a temporary topology graph $\tilde{g}$.
- **Step 3**: Node v measures the overlay link rate $\bar{x}_{(v,u)}$ from every neighbor $u \in N_v^{\tilde{g}}$ (including the newly added neighbor w if there is any), and then chokes an in-use neighbor u with probability
$$\frac{\exp(-\frac{1}{2}\mu\bar{x}_{\tilde{g}\backslash g'})}{1 + \sum_{g'' \in \mathcal{A}_{v,\tilde{g}}} \exp(-\frac{1}{2}\mu\bar{x}_{\tilde{g}\backslash g''})}, \quad g' = \tilde{g}\backslash \{(v,u)\}. \tag{15}$$
Afterwards, node v repeats **Step 1**.

The above algorithm is fully distributed – each node uses the overlay link rates from his neighbors as the only metric to perform the topology selection. Note that the worst link is dropped with high probability, hence the term “soft-worst” choking. This algorithm results in the following perturbed MC.

Proposition 6. *The soft-worst-neighbor-choking algorithm induces the following transition rates for the perturbed Markov chain: for any two topology graphs $g, g' \in G$ satisfying direct-transition condition:*
$$q_{g,g'} = \tau^{-1} \cdot \frac{\exp(-\frac{1}{2}\mu\bar{x}_{\tilde{g}\backslash g'})}{1 + \sum_{g'' \in \mathcal{A}_{v(g,g'),\tilde{g}}} \exp(-\frac{1}{2}\mu\bar{x}_{\tilde{g}\backslash g''})}, \tag{16}$$
where $\bar{x}_{\tilde{g}\backslash g'}$ is the rate of the only overlay link in $\tilde{g}\backslash g'$ under topology graph $\tilde{g}$; and $q_{g,g'} = 0$ otherwise.

Comparing (16) with (13), we can see that the global quantity $\Phi_{g'} - \Phi_{\tilde{g}}$, which is the overall utility difference between ‘after’ and ‘before’ the node drops the overlay link $\tilde{g}\backslash g'$, is replaced by $-\frac{1}{2}\bar{x}_{\tilde{g}\backslash g'}$ which is a locally measurable quantity. Fortunately, we can show that under some reasonable assumptions, the perturbed MC can still achieve close-to-optimal system performance, as we show as follows.

[3] τ is a tuning parameter which affects the count-down time and the algorithm’s mixing time. Details will be given in the subsequent section. One example is to set τ such that the count down time is one minute, which BitTorrent uses [21].

5.4 Performance Guarantee

Theorem 7. *Denote by $\Phi^O = \max_{g \in G} \Phi_g$ the optimal system utility, $\Phi^E = \sum_{g \in G} p_g^* \cdot \Phi_g$ the expected system utility of the exact Markov chain, and $\Phi^P = \sum_{g \in G} p_g^{'} \cdot \Phi_g$ the expected system utility of perturbed Markov chain, where p_g^* and $p_g^{'}$ are the stationary distributions of the exact and perturbed MC respectively. Let $\bar{\Phi}^O = \frac{1}{|U|}\Phi^O$, $\bar{\Phi}^E = \frac{1}{|U|}\Phi^E$ and $\bar{\Phi}^P = \frac{1}{|U|}\Phi^P$ be the corresponding system utilities averaged among the users. When the users' utility function $V(z_u) = \min(\gamma_m, \sum x_r), u \in U_m$, the optimality gaps with and without perturbation errors are shown as follows:*

$$0 \leq \bar{\Phi}^O - \bar{\Phi}^E \leq \tfrac{1}{\mu} B_{\max} \log N_{\max} \tag{17}$$

$$0 \leq \bar{\Phi}^O - \bar{\Phi}^P \leq \tfrac{1}{\mu} B_{\max} \log N_{\max} + \tfrac{1}{2|U|} c_{\max} \tag{18}$$

where $B_{\max}$ is the maximum degree bound over all users, $N_{\max}$ is the max neighbor size over all users, $c_{\max}$ is the maximum underlay link capacity, and $|U|$ is the total number of users.

The proof is given in our technical report [18]. We make the following observations:

- The upper bound on the optimality gap per user of the perturbed Markov chain is $\frac{c_{\max}}{2|U|}$ away from that of the exact Markov chain, which we call "the price of local perturbation".
- When we formulated the topology selection algorithm, we made the assumption that the underlying content placement and link rate allocation algorithms have fully converged. When this is not the case however, we obtain inaccurate values of the link rates x_{uv} and therefore $\Phi_g, g \in G$. We can treat this inaccuracy as a one-dimension perturbation error to exact system utilities $\Phi_g, g \in G$. Following the same method to the proof of Theorem 7, we can still obtain bounds on utility gap similar to those in (17)-(18).
- While larger μ reduces the optimality gap, it may also increase the mixing time of the Markov chain and consequently the convergence rate of the worst-neighbor-choking algorithm. We omit the relationship between μ and the mixing time, and refer interested readers to our technical report [18].

6 Simulation Results

6.1 MATLAB Simulator

The theory, described in Ch.3- Ch.5, explains that the proposed algorithm achieves near-optimal performance for a static case(for fixed system parameters). However, the current theory does not address the algorithm's performance under dynamic conditions such as user and cache churn, varying content popularity and network conditions. However, formal modeling of the dynamics of the Video-on-Demand system and its optimization need a sophisticated theoretical analysis. Moreover, it is not clear which system dynamics need to be targeted by the new theory.

As a stepping stone to bridge theory and practice and the theory, we have implemented a large-scale MATLAB-based simulator, and equipped it with full functionality. The simulator allows us to test the robustness of the system against various dynamics by varying system environments such as number of users and caches, video popularity, and network conditions. Due to the distributed nature of the system, it is also possible to run a highly large scale simulation in parallel using multi-core processors.

We first run a small scale simulation to validate the simulator. The simulation setup is as following. There are 100 users in the system and each is watching one of 20 videos. The video's popularity follows the Zipf's Law, which is a heavy-tail distribution. There are 50 caches with zero cost functions. Each cache can store upto only 2 videos and has tight upload bandwidth which is twice of the video streaming rate. The server is connected to all users. Figure 4 show how the system running algorithms evolves. After 2000 iteration steps of algorithms, the system evolves such that the caches provide 95.4% of the overall traffic and the server needs to provide only 4.6% of the overall traffic. These results imply that the proposed system can be a prominent way of distributed load balancing across the caches and the server and hence reducing operation costs.

We also attach the visualized results in Figure 4. 100 circles on the bottom of each figure represent users. A color associated with a user visualizes the index of a video. 50 circles on the top represent caches. Above each cache circle, a box represents its disk. A small colored box within each big box represents stored amount of the corresponding video. A cache can fill up the storage up to the height of the box by caching several videos. An overlay topology between users and caches are represented as a bipartite graph between them. A red bar represents the server load. Via the visualized outputs, we were also able to check the validity of the algorithm and moreover find the interesting observations. The following sections will cover extensive simulation results.

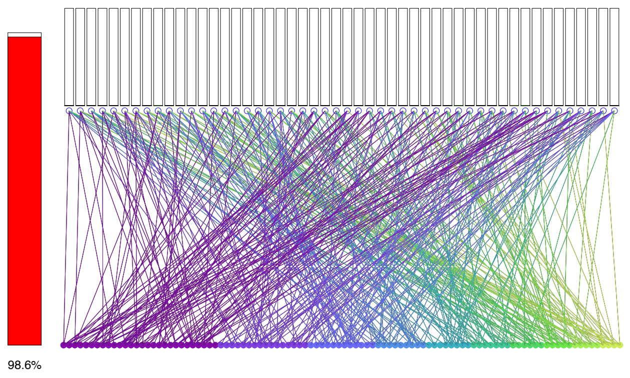
6.2 A toy example

Figure 5 shows a simple VoD system with three caches and six users. Let us first consider the resource allocation algorithm under a fixed overlay topology depicted in Figure 5. Caches have to maximize the the joint upload rates of them so as to minimize the server load. The optimal caching and rate allocation can be easily found by finding the optimal solution via any optimization solver. The optimal solution turns out to be as following and depicted in 5.

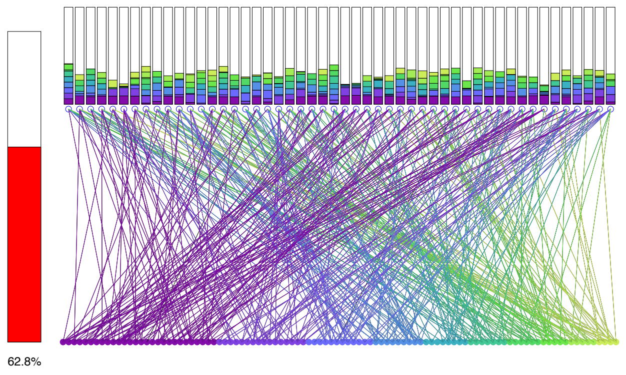
1. Cache 1 : Store video A. Upload the full stream to user 1 and user 2.
2. Cache 2 : Store half of video A and half of video B. Upload the fractional stream of video A with rate half to user 3. Upload the fractional stream of video B with rate half to user 4 and 5.
3. Cache 3 : Store video B. Upload the fractional stream of video B to users 4 and 5. Upload the full stream of video B to user 6.
4. Server : Upload the fractonal stream of video A with rate half to user 3.

Figure 6 shows the convergence of non-cache traffic if we run only the distributed resource algorithm under the same configuration. It is observed that the distributed algorithm also converges to the same optimal point where the non-cache traffic is 0.5Mbps.

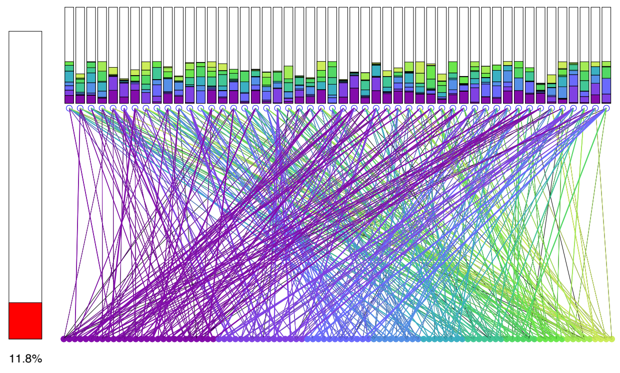
Now, what is the optimal topology selection and resource allocation? Assume that degree bound of of each cache is $3, 4$, and 3 respectively. By considering all possible $\binom{6}{3} \times \binom{6}{4} \times \binom{6}{3} = 6000$ overlay topologies, one can find out the optimal topology and resource allocation scheme as following and it is depicted in figure 7.



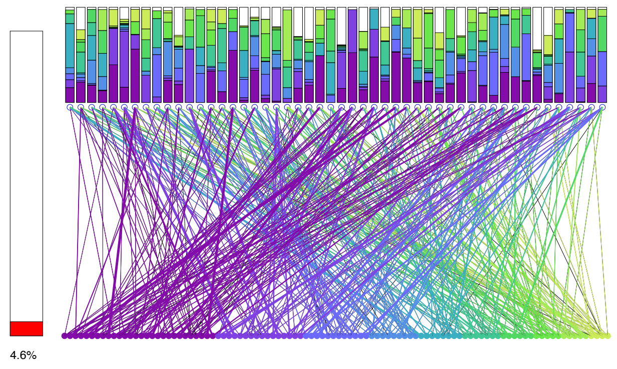
(a) Visualized result. After 10 iterations.



(b) Visualized result. After 50 iterations.



(c) Visualized result. After 100 iterations.



(d) Visualized result. After 1000 iterations.

Figure 4: Visualized simulation results.

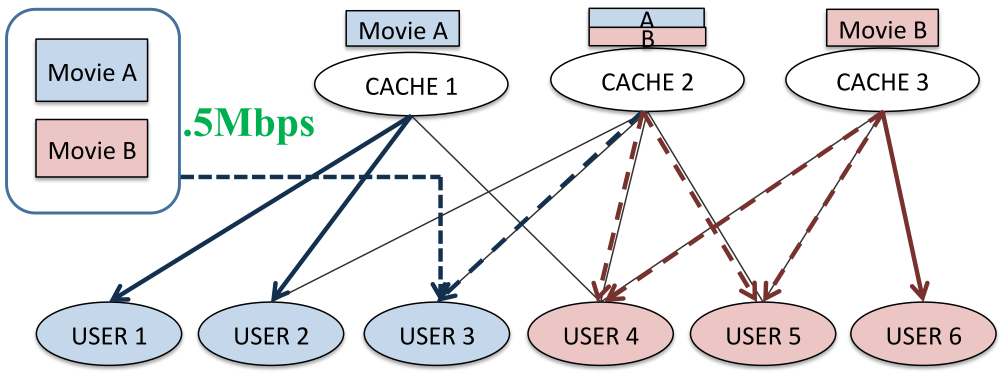


Figure 5: A simple example with 3 caches and 6 users. Three users demand video A and the others demand video B. Each video has a streaming rate of 1 Mbps. The overlay topology is given. The optimal caching and rate allocation is depicted in the figure. The dotted line represents a fractional stream with rate of 0.5 Mbps and each color represents each video.

1. Cache 1 : Connect to User 1, User 2, and User 3. Store video A. Upload the full stream to user 1 and the half stream to user 2 and user 3.
2. Cache 2 : Connect to User 2, User 3, User 4, and User 5. Store half of video A and half of video B. Upload the half stream of video A to user 2 and user 3. Upload the half stream of video B to user 4 and user 5.
3. Cache 3 : Connect to User 4, User 5, and User 6. Store video B. Upload the half stream of video B to users 4 and 5. Upload the full stream of video B to user 6.
4. Server : *Do nothing.*

Here, the server traffic is 0Mbps.

Figure 8 shows the convergence of non-cache traffic if we run the distributed resource allocation algorithm and topology update algorithm together. It is observed that the distributed algorithm also converges to the optimal point where the non-cache traffic is 0Mbps quickly.

This confirms that the simulated distributed algorithm can indeed converge to the theoretical optimal operation point quickly. In the following sections, we now show results from extensive experiments at large scales having interesting consequences.

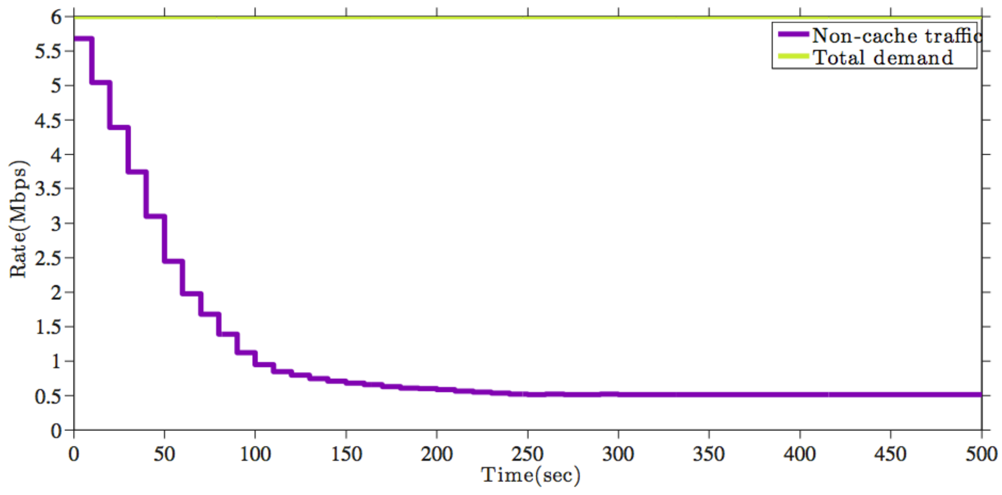


Figure 6: Server traffic in the system.

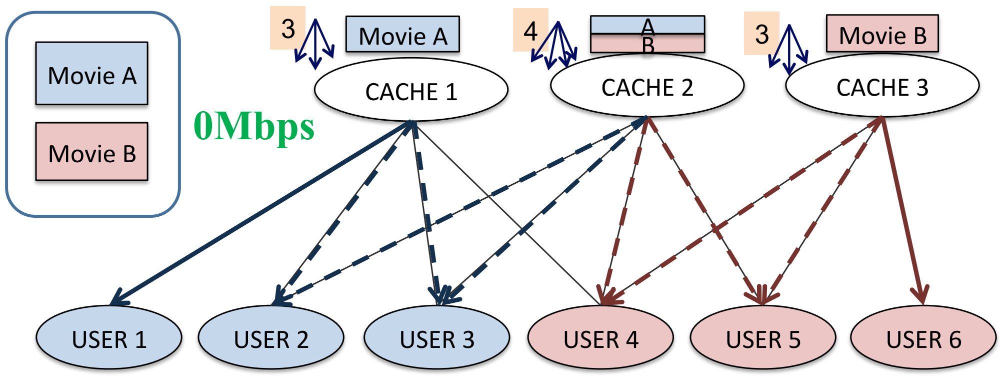


Figure 7: The optimal caching, rate allocation, and overlay topology selection is depicted in the figure. The dotted line represents a fractional stream with rate of 0.5 Mbps and each color represents each video.

6.3 Setup

In this section and the following sections, we evaluate the overall system performance under various realistic settings of interest. By allowing users to join and leave the system and to change their subscription, we also empirically show that our system works well under system dynamics. We show that our algorithm implicitly learns the video demand, and yields a high bandwidth utilization.

We assume the existence of a media server which hosts 2000 videos. Each video is 20 minutes long, and has a streaming rate of 2 Mbps. Users join the system with varying arrival rate, (where

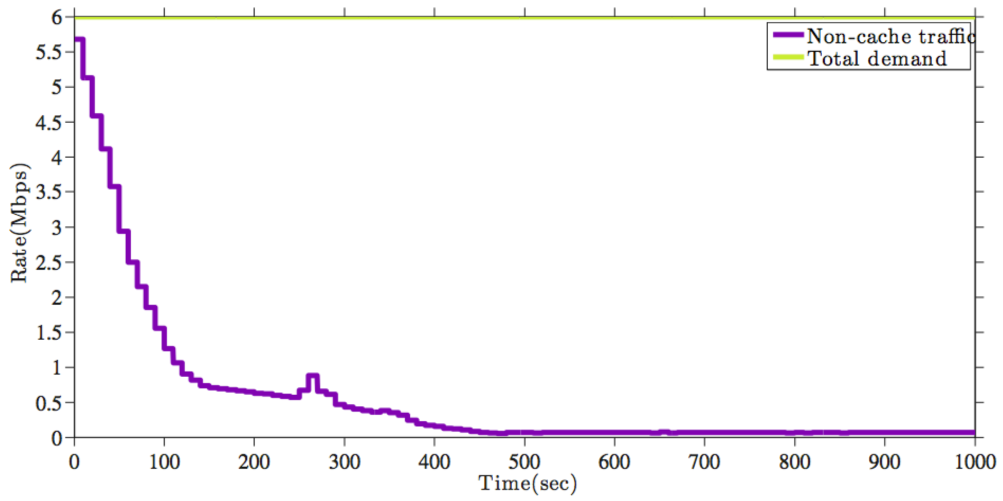
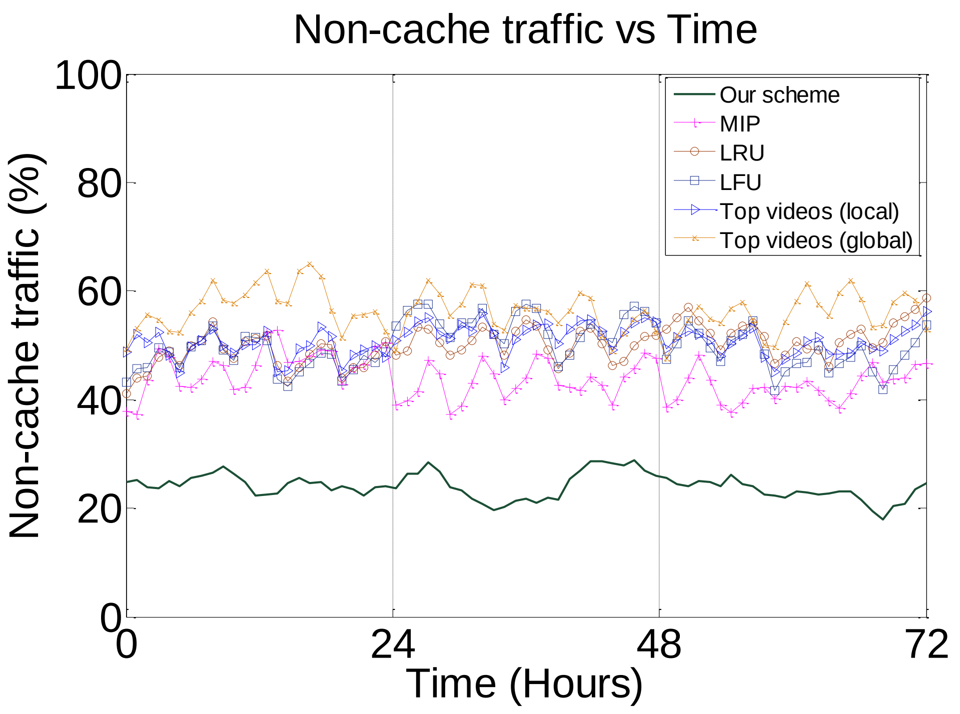


Figure 8: Server traffic in the system.

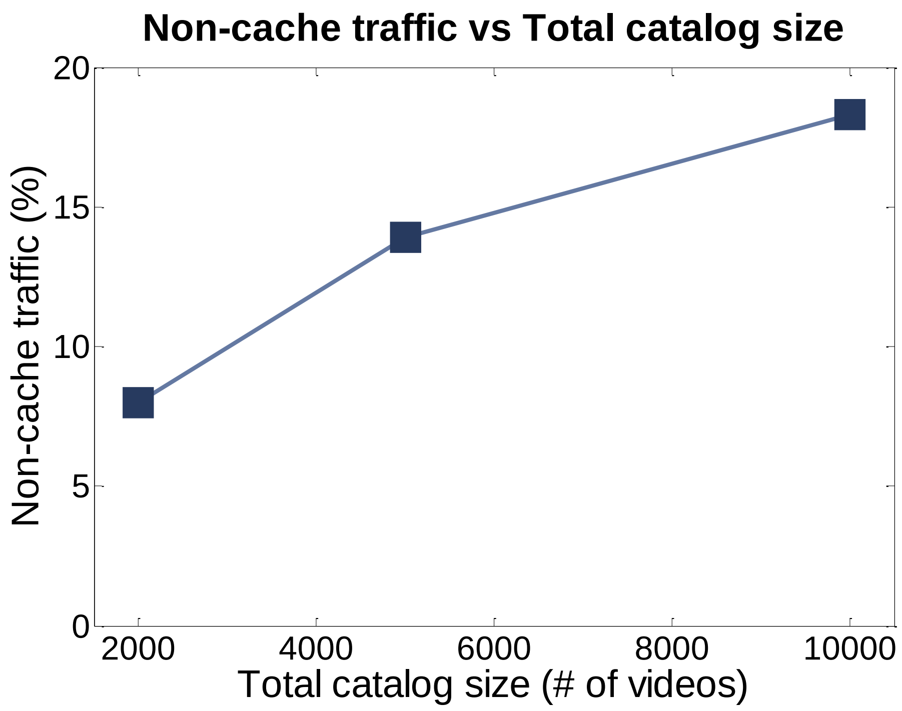
the maximum number of users is 40000) and leave the system after they finish watching the video. When a user joins, it randomly selects a video from the content catalog following a heavy-tailed distribution. There are 50 caches with uniform storage capacity, with a total storage capacity of 2.5 times the entire video catalog. The bandwidth constraint of the cache nodes follows a bi-modal distribution having means of 1.2Gbps and 2Gbps respectively. The aggregate bandwidth among all caches is just enough to cover all users. The node degree bound on each cache is 3200, *i.e.,* each user can be potentially served by 4 caches. We compare our results with Least-Recently-Used (LRU), Least-Frequently-Used (LFU), top videos (local), top videos (global) and Mixed Integer Programming (MIP). The methods and parameters used in the comparing schemes are:

- MIP: on each cache, the fractional storage allocation algorithm [2] is used, and then the entire video with the largest converged fraction of storage is stored, followed by the video with the second largest converged value, and so on and so forth, until the storage capacity is reached.
- LRU(LFU): every cache replaces least recently(frequently) used videos by the most popular videos.
- Top videos (local, global): cache nodes store the most locally(globally) popular videos of the period.

Figure 9(a) shows the percentage non-cache traffic, i.e., the fraction of the total realtime de-



(a) Performance comparison of our scheme with LRU, LFU, top videos (local), top videos (global) and MIP.



(b) Average non-cache traffic varying catalog size.

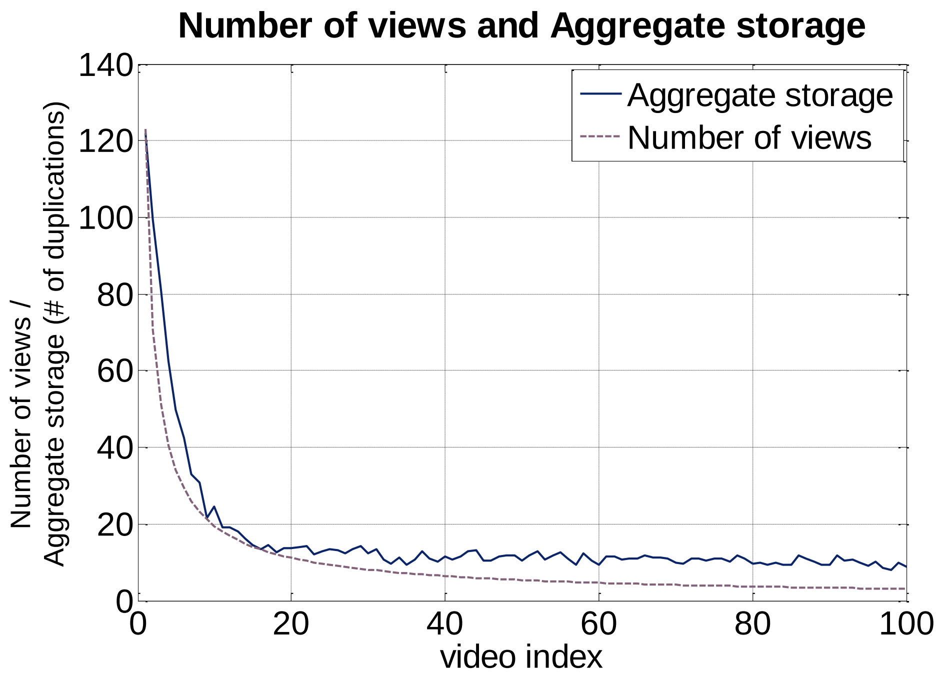
Figure 9: Simulation results

mand that is not covered by the caches, for each method and the total demand versus time. We observe that for the same amount of system resources, our scheme allows the maximum support of the caches, thanks to our theoretical guarantee of optimal performance. Specifically during peak hours where the arrival rate of users is the highest, the corresponding non-cache traffic for our scheme was 24.1% while it is 44.0%, 50.2%, 50.0%, 50.9% and 56.4% for MIP, LRU, LFU, top videos (local) and top videos (global). It is worth noting that our scheme does not achieve 0% of non-cache traffic although the total aggregate bandwidth is enough to cover the demand in a static setup. This happens because there will always be new users joining the system who need to be fed directly by the server. From the results, we can see that storing the globally popular videos, as is done in most practical cases, performs the worst. This is because the video demand distribution is so heavy-tailed that caching of a vast number of unpopular videos becomes unavoidable. MIP performs the closest in performance to our scheme but is still significantly inferior. This corroborates our previous observations in the toy examples that fractional storage provides much-needed flexibility in caching.

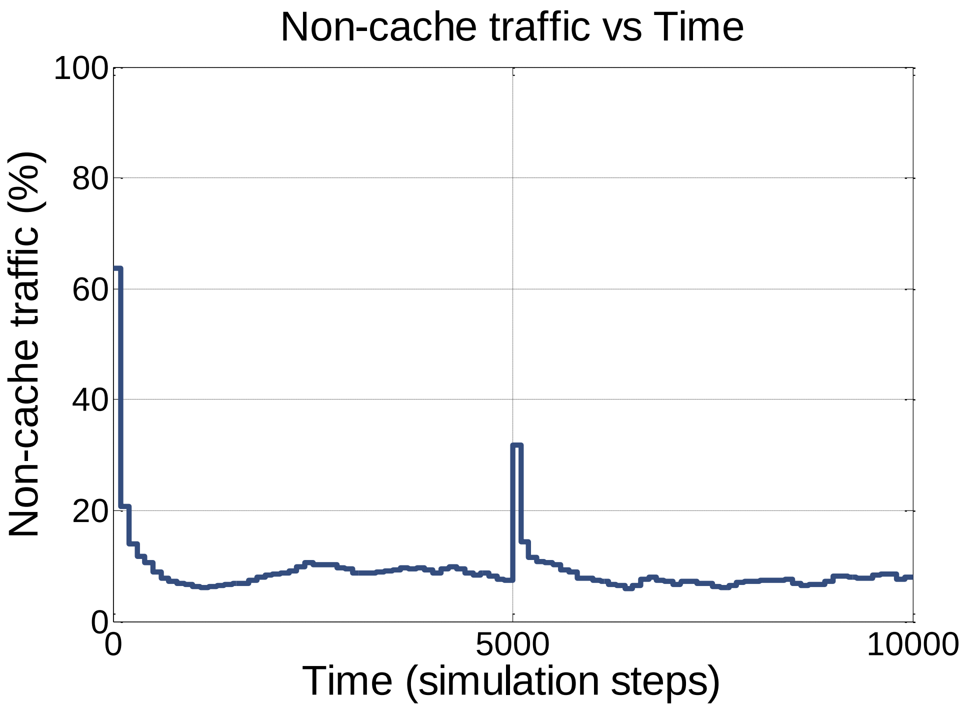
In order to better understand the performance of our algorithm, we also find the utility efficiency for video m on each cache node h as $\frac{\sum_{u \in U_{m,h}} x_{hm}}{W_{hm}\gamma_m |U_{m,h}|}$ which is the ratio between the total upload rate to users watching video m and the total available rate given the storage of video m. It quantifies the utility of any stored faction of a video on each cache. The efficiency factor for each video is averaged across all the caches which store that video. We observe that with our scheme, most of the efficiency factors are greater than 80% uniformly across the videos. On the other hand, the efficiency factor is around 50% with other schemes.

6.4 Scalability, Robustness, and Learning

In this section, we validate empirically that the proposed system is scalable, robust and able to learn the users' demand distribution. In the first experiment, we increase the video catalog size and investigate how the system responds. In Figure 9(b), we plot the average server traffic for different catalog sizes. It can be seen that although the catalog sizes vary from 2000 to 10000, the non-cache traffic increases only slightly. The robustness of the performance to the changes in the catalog size is especially useful when a sudden flash crowd of new videos appear and the system does not have



(a) Total storage of videos in the cache nodes and the appropriately scaled actual demand distributions.



(b) New popular videos are added and1 unpopular videos are removed in the middle of the simulation.

Figure 10: Simulation results

ample time to call for more resources.

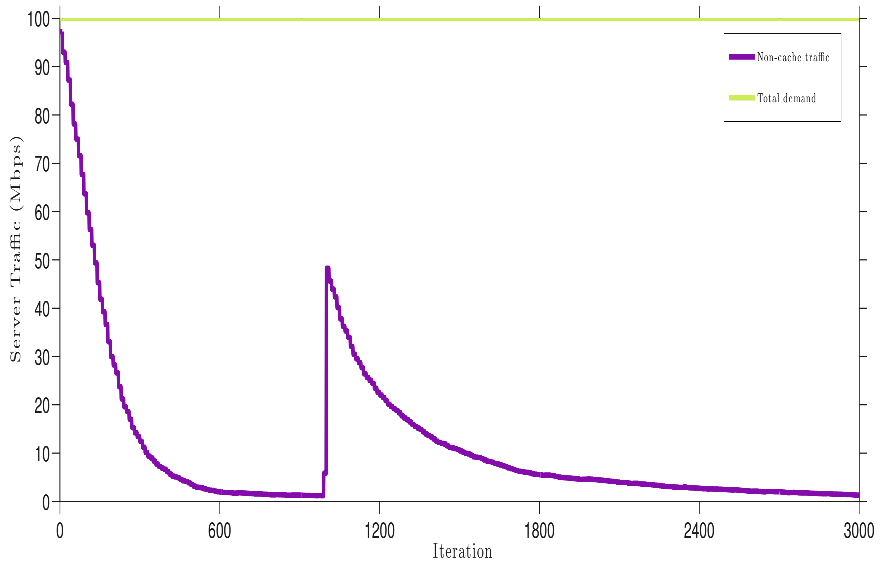
In Figure 10(b), we break up the total storage of entire caches by the videos and compare it with the demand distribution. We can see that despite running in a fully distributed manner, the system is able to implicitly learn the demand distribution, with the aggregate storage distribution similar to that of the demand. This is a useful property of the algorithm, because separately estimating the demand in practice can induce large overhead. Our scheme is thus demand-agnostic, *i.e.,* it does not require any prior knowledge of the demand, automatically learning the demand distribution in a distributed way.

We also test the robustness of the system. In Figure 10(a), the server load is shown when new popular videos are suddenly added in the middle of the simulation. More specifically, we inject 10% of new most popular videos and remove 10% of least popular videos. The other 90% of old videos become slightly less popular than before. The server load experiences a negligible jump when new videos are added, but quickly goes back to normal. In practice, we expect the changes in demand to be much slower, and see that the system easily adapts to such changes.

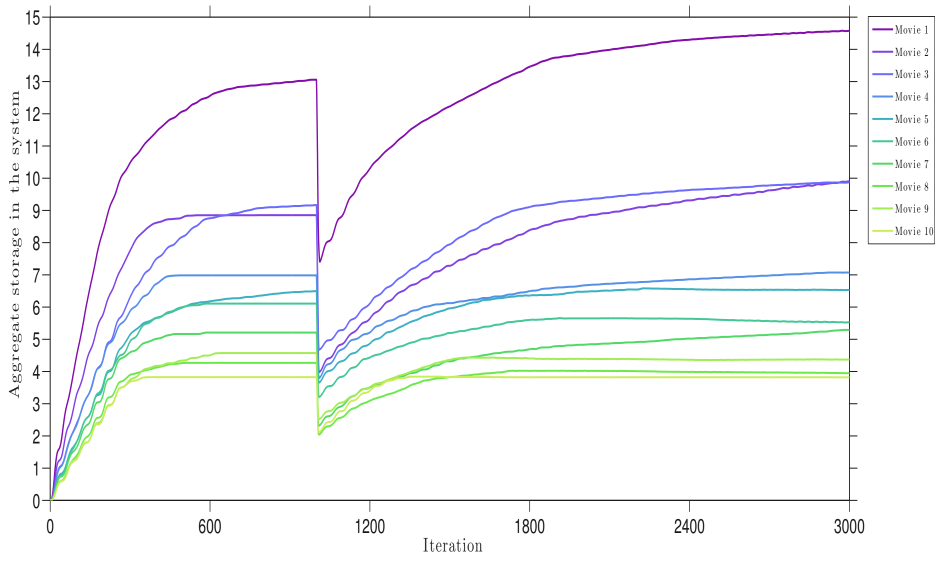
We also test the robustness of the system due to cache collapse, by removing a half of the caches from the system suddenly after 1000 iterations. Figure 11(a) shows the simulation result. The server load increases upto 50% suddenly. Surprisingly, the other caches remaining in the system are able to recover the system by storing more content and serving more users than they did before the collapse. Figure 11(b) shows how the remaining caches can recover the system by storing roughly twice more than before other caches were removed.

6.5 Tradeoff between Storage, Bandwidth and Node Degree

To understand the resource utilization further, we conduct experiments to demonstrate the tradeoff between the three types of cache resources: storage, bandwidth and node degree. We consider a system with 200 users and 100 caches. In Figure 12(a) and (b), we show the average server load versus the total bandwidth resources under different degree and storage capacity. In Figure 12(a), we fix the storage to be 0.7 video's size and vary the degree bound to plot the different curves. We can see that when the degree resource is too scarce, e.g., degree is less than or equal to 5, the server

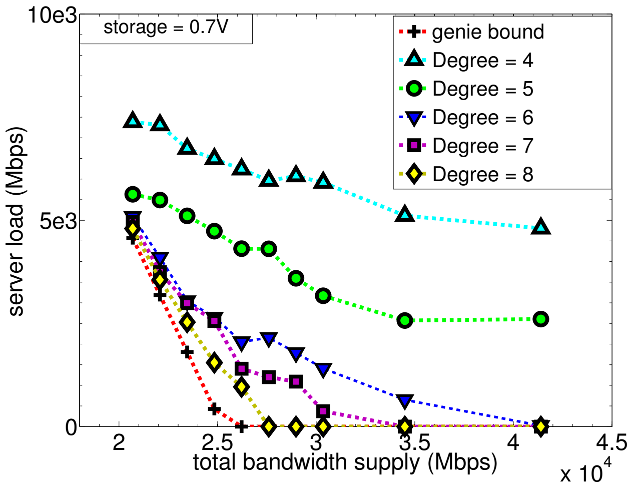


(a) Server traffic.

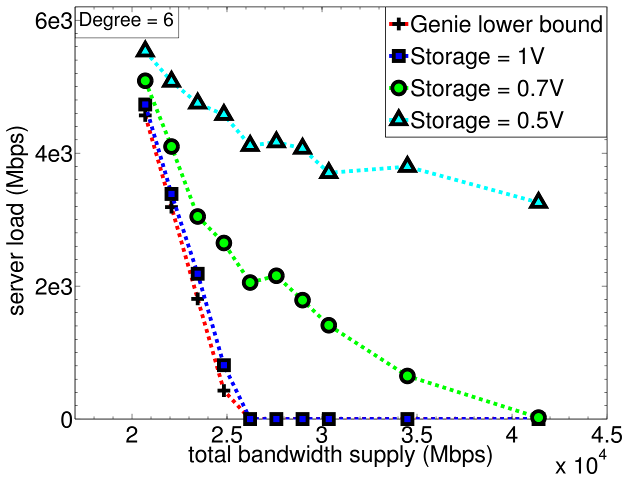


(b) Aggregate amount of used storage across caches per video

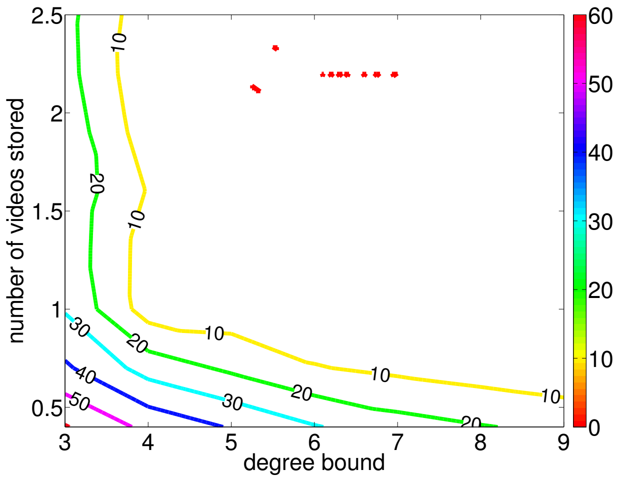
Figure 11: Simulation results



(a) Server traffic.



(b) Aggregate amount of used storage across caches per video



(c) Aggregate amount of used storage across caches per video

Figure 12: Trade-offs between storage, bandwidth and node degree bound.

load cannot achieve the lower bound regardless of the bandwidth sources given. When degree resource is greater than 5, two difference cases occur: (1) when the system is in either extreme bandwidth deficit or extreme bandwidth surplus mode, the server load can easily achieve the lower bound regardless of the storage resources. This can be explained as follows. When the bandwidth is in extreme deficit mode, almost any storage placement works because the resource is scarce. On the other hand, when the bandwidth is in extreme surplus mode, the caches have so much resource such that they can easily bring down the server load to zero. In between these two cases, the more degree resources, the closer the server load is to the genie lower bound. This suggests that there exists some intrinsic minimal degree needed given some fixed bandwidth and storage resources. In Figure 12(b), similar results are shown when the degree is fixed to be 6 while the storage capacity is varied. Similar conclusions can be drawn.

To understand the interplay better, we show a contour plot in Figure 12(c). The x-axis is the degree bound and the y-axis is the storage per cache node. For each contour, the total bandwidth supply is equal to the average total demand. On the contours shows the optimality gap, e.g, 10%, 20% and 40% etc. The optimality gap is defined as the ratio between the performance gap (difference between actual server load and genie bound) and the demand gap (difference between total demand and genie bound). The smaller the optimality gap, the closer the server load is to the genie bound. Clearly, the sweet spots happen near the diagonal area. Outside that area, one needs significantly more degree bound (storage capacity) resources to compensate even a small decrease in storage capacity (degree bound) to achieve the same optimality gap. This figure helps understand the fundamental minimal resources needed to satisfy users' need when designing a VoD system.

We currently lack of the theoretical understandings of the tradeoff we observed through the experiments. We are actively studying the resource tradeoff theoretically with the faith that the results can be used to provide appropriate resource provisionings.

7 System Design

In this chapter, we shift our focus from the ideal setting studied earlier to more practical settings in order to address possible gaps between theory and practice. There are several assumptions we have relied on while developing the theory and the algorithms, some of which do not hold in practice. For example, because the fluid assumption (See Ch.4) cannot be satisfied in a practical system, we have to quantify the loss of the performance of practical codes and suggest a solution. In this chapter, we enumerate such practical issues, and design a practical VoD system based on the theory. The following list enumerates three main assumptions underlying the theory of Ch.4-Ch.5.

1. Fractional streams are always orthogonal.
2. The server has infinite upload bandwidth so that any user which has missing content can fetch it from the server instantaneously.
3. The traffic from the server to the caches for content caching is negligible compared to the "static" traffic to users.

The first assumption does not hold when we use practical codes in the system. In practice, *the duplicate problem* occurs, and the user may not be able to decode the received fractional streams, and recover a full stream without downloading the missing parts from the server. By storing slightly more than that required by the algorithm or by connecting to a few additional caches, this problem can be solved easily in a practical system. These overheads are quantified in the following section.

The second assumption may not hold in practice. The centralized server can usually have high enough upload rate, but it is not infinitely high. Thus, in order to not experience buffering delays, a user should request missing packets a few seconds before the video chunk the user is watching finishes. This assumption can be relaxed by modifying the distributed algorithm to give the server enough time to upload missing packets to the user.

The last assumption does not hold under a dynamic setting. If caches keep updating their storages for new incoming users, the traffic from the server to caches might be even larger than the traffic to users. The period of storage allocation update can be set infrequent enough to alleviate this problem, i.e., even though the caches run the distributed algorithm and update their primal and dual variables at each iteration, they update their storage allocation only at the end of each carefully designed period. For example, if one sets up the period of storage update as 10 iterations, each cache will only update their storage allocation every 10 iterations. These ideas will be reflected in our design of a distributed algorithm for the practical system, which we will describe later.

7.1 Design of Practical Fractional Stream

There can be several design choices to implement the fractional stream which meet the requirements of our theoretical abstraction. We use a "horizontal" fractional stream concept which was introduced in our earlier work [18].

The definition of a fractional stream is stated in Theorem 1. As briefly stated in the Ch. 4, the fractional stream can be approximated by using a Maximum Distance Separable (MDS) code [23]. We have assumed a fluid model, where two fractional streams are added without losing rate, i.e., $S(x_1+x_2)$ is generated by decoding $S(x_1)$ and $S(x_2)$. In practice, two fractional streams from two different caches are not always orthogonal due to possibly duplicate packets. Thus, in general a receiver will experience a loss in effective rate after jointly decoding two fractional streams, though the probability of duplicate packets is relatively low. The probability of the event's occurrence can be reduced via centralized coordination, but it is impossible to be avoided completely.

To understand the duplicate packets problem in-depth, assume that the system uses fractional streams generated by MDS(n,k) codes. With such codes, the rate of fractional streams is now discretized, i.e., $x \in \{\frac{x_0}{k}, \frac{2x_0}{k}, ..., \frac{(k-1)x_0}{k}, x_0\}$. Each cache is also able to store only a discretized fractional stream under this practical setting. Running the distributed algorithm, a cache will keep updating its storage variable f for each video, and quantize the storage variable, i.e., find the nearest number $\bar{f}$ in the set $f^D = \{0, \frac{1}{k}, \frac{2}{k}, ..., \frac{k-1}{k}, 1\}$. Then it downloads and caches a fractional stream with rate of $\bar{f}r$. Further assume that user u is connected to $\mathbf{C_u} = \{c_1, c_2, ..., c_d\}$ and each cache $c \in \mathbf{C_u}$ is able to upload a fractional stream $S(x_c)$, respectively. Define $k_c = \frac{kx_c}{r_c}$ to denote the number of coded packets each cache stores to store a fractional stream. Unfortunately, even though $\sum_{c \in \mathbf{C_u}} k_c \geq k$, the user might not be able to recover the full stream due to the duplication problem.

This problem can be understood as an instance of a *generalized coupon collector problem* where a coupon collector draws multiple coupons at a time, and wants to collect all the coupons. For simplicity, assume that $k_c = a$ for all d connected caches and $ad = k$. The duplication problem happens unless there is no shared packet between any pair of d fractional streams stored at every cache. However, because each cache is randomly choosing its fractional stream, it happens rarely. Each fractional stream of rate $\frac{ax_0}{k}$ corresponds to drawing a balls d times from a bag of n balls with replacement. The following theorem on the generalized coupon collecting problem states how many packets a user will receive from multiple fractional streams.

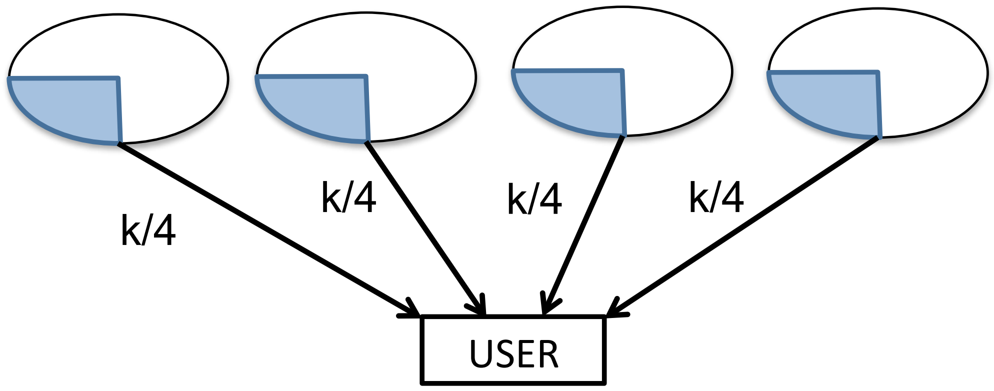
Theorem 8. *Consider a generalized coupon collector problem where a collector draws a balls simutaneously out of n balls at each draw. The expected number of collected coupons after d drawings is $\mathbb{E}[N] = n(1-(1-a/n)^d)$.*

Note that $\mathbb{E}[N]$ is strictly less than k, the number of packets a user would have get if fractional streams were all orthogonal. Expanding the RHS of Theorem 8 gives the following lemma.

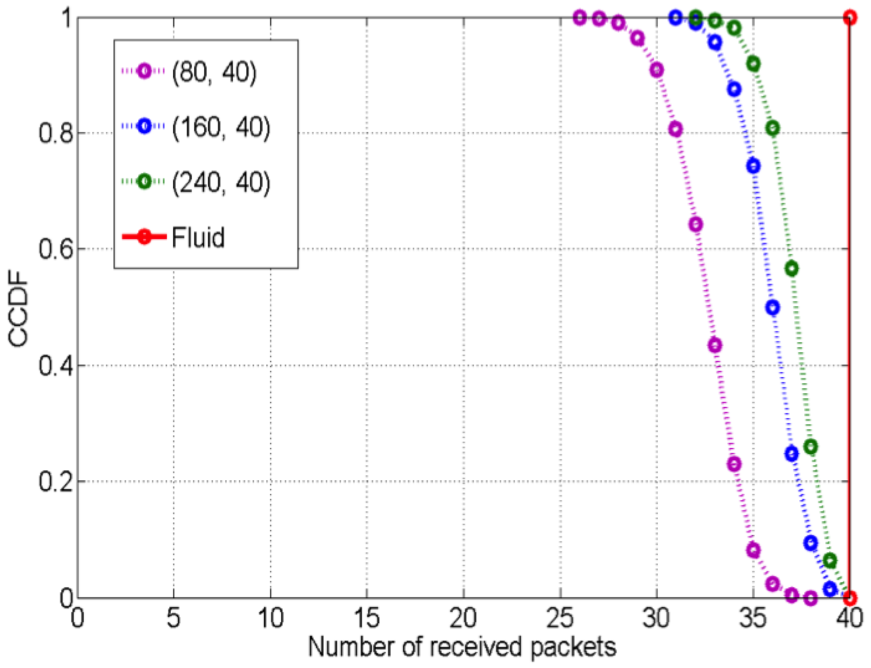
Lemma 9. *The average number of missing packet is $\mathcal{O}(n^{-1})$.*

Proofs are given in the appendix. Thus, in a practical system, users will request these missing packets from the central server and this will incur a higher server cost. The first claim of the Theorem 8 implies that it is unlikely for a user to recover a full stream even though the sum of the rates of fractional streams is matched to the full streaming rate by the algorithms. The implication of Lemma 9 is that using a higher n will induce fewer missing packets. However, using higher n value has drawbacks: high encoding and decoding complexity, and difficulty of populating caches in a distributed way.

Consider a small example depicted in Figure 13(a). A user is connected to 4 caches, $d = 4$, and each cache is storing a fractional stream with rate one quarter of the streaming rate, i.e., $\bar{f} = a/k = 1/4$. We choose k equal to 40 and hence $a = 10$. Then, we vary n from 80 to 240 and observe number of packets the user receive for each value. Figure 13(b) shows CCDFs of number of received packets after $d = 4$ drawings. When the fluid model, $n = \infty$, is used, as in the theory there is no packet loss. However, results of finite $n = 80, 160, 240$ cases show the rate loss is not negligible as expected in Lemma 9.

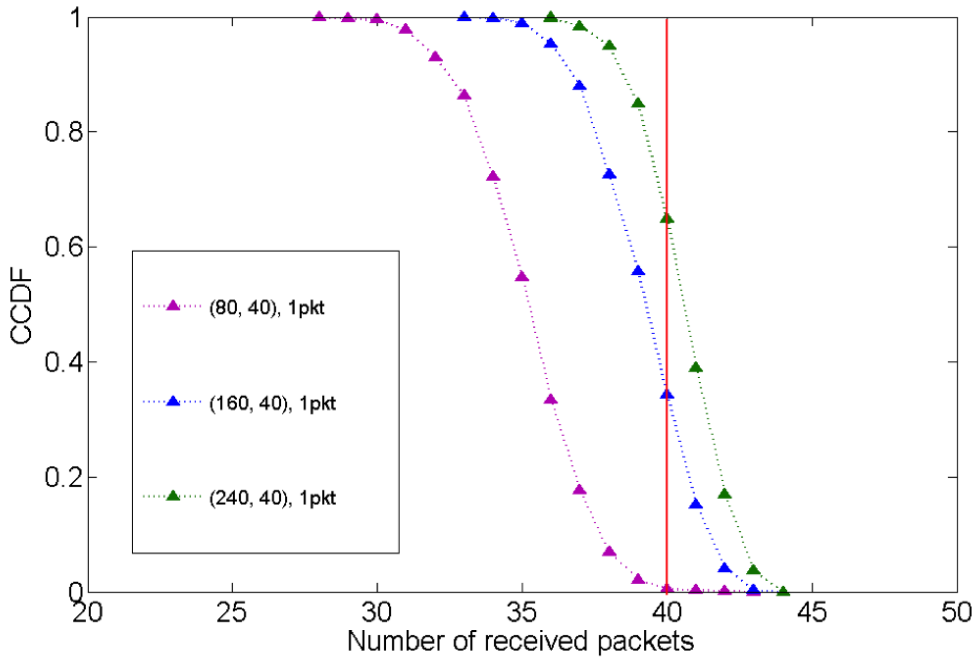


(a) A small VoD system. One user is connected to four caches and each caches stores a fractional stream with rate one quarter of the streaming rate.

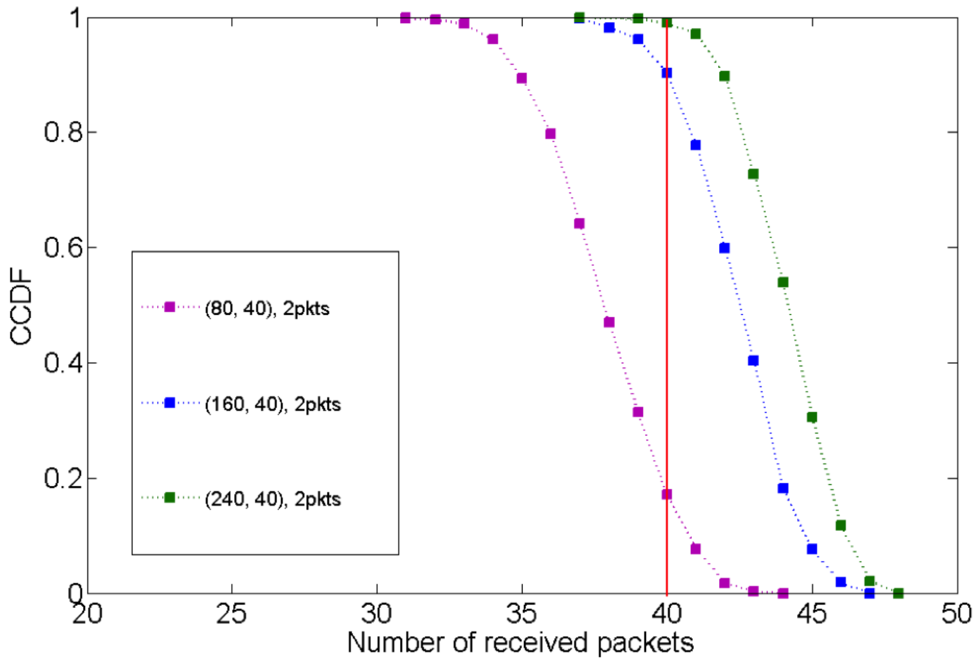


(b) CCDF of Number of received packets for different n.

Figure 13



(a)



(b)

Figure 14: CCDF of Number of received packets for different n when each cache stores 1 more packet, (b) CCDF of Number of received packets for different n when each cache stores 2 more packets.

How can we solve the problem? If the caches' cost is relatively cheaper than the server's cost, the user can connect to more caches and request missing packets or connected caches can store more packets, i.e., increase the rate of the cached fractional stream. Figure 14 shows how many packets the user will receive if the caches store 1 or 2 more packets each in the previous example. Storing 1 or 2 more packets significantly reduces the probability of getting less than k packets. The following theorem quantify the storage overhead of each cache to not incur additional server cost.

Theorem 10. *If a user is connected to d caches and assuming every cache stores a equal fraction of a content for the user, the number of packets each cache to store, a^*, to guarantee that the expected total number of received packets of user exceeds k is as following.*

$$a^* = \frac{n}{(d-1)}\left(1 - \sqrt{1 - \frac{2(d-1)k}{nd}}\right)$$

Table 2 provides the storage overhead in number of packets for various n for the example. If

	Fluid	$(n,k) = (240, 40)$	$(n,k) = (160, 40)$	$(n,k) = (80, 40)$
a^*	10 pkts	10.718 pkts	11.169 pkts	13.333 pkts

Table 2: Storage overhead

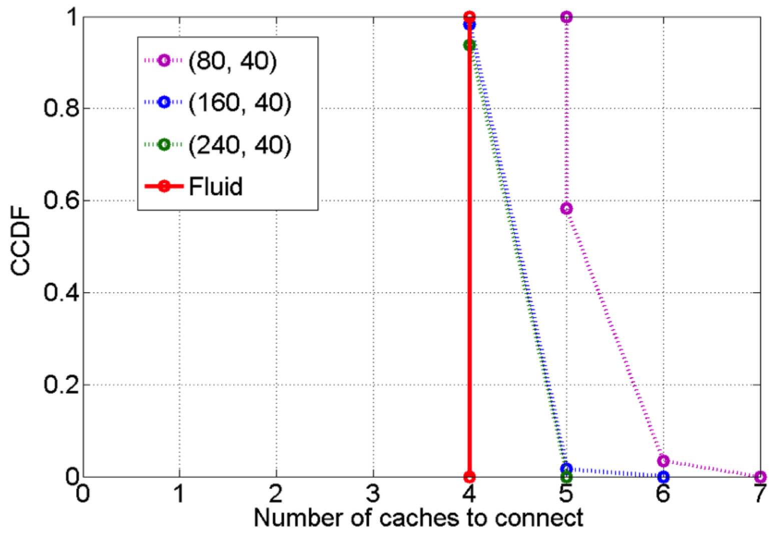


Figure 15: CCDF of minimum number of caches required to recover the full stream

the cost of degree and bandwidth is relatively cheaper than the cost of storage, user can connect to more caches and retrieve the full stream. Figure 15 shows how many cache the user should connect to in order to receive if caches store $a = k/d = 10$ packets each in the previous example. The following theorem quantify the degree overhead of each cache to not incur additional server cost.

Theorem 11. *Assuming every cache stores a packet, the number of connection a user needs to connect to, d^*, to guarantee that the expected total number of received packets of user exceeds k is as following.*

$$d^* = 0.5 + \frac{n}{a} - \sqrt{\left(0.5 + \frac{n}{a}\right)^2 - \frac{2nk}{a^2}}$$

Table 3 provides the storage overhead in number of packets for various n for the example.

	Fluid	$(n,k) = (240,40)$	$(n,k) = (160,40)$	$(n,k) = (80,40)$
d^*	4	4.294	4.489	5.627

Table 3: Degree overhead

Over-caching We chose to exploit Theorem 10 for the practical algorithm, i.e., caches store slightly more packets than that required by the algorithm. By *over-caching* packets at the caches, users will probably be able to collect enough number of packets even with the existence of the duplication problem.

7.2 Design of Practical Distributed Algorithm

In this section, we describe the design of distributed algorithm that performs well under practical settings. As we discussed at the beginning of this chapter, several practical issues can be tackled by judicious modification of the distributed algorithm.

Emergency Threshold The server does not have infinite upload bandwidth in practice. We propose to use an *Emergency Threshold* as a threshold for a user to request the server to upload the missing packets to the user. We assume that each user knows the measured upload bandwidth from the server to the user u and denote it as $c_{s,u}$. We also denote the time remaining to finish watching the current chunk as t'. If the number of packets received for the next chunk is m, the following condition must be satisfied to avoid the buffering,

$$\frac{k-m}{t'} \leq c_{s,u}.$$

This condition can be used to detect the need of the server, i.e., request the server to upload all missing packets if the number of packets received for the next chunk is less than the emergency threshold.

$$m < k - t'c_{s,u} = \text{Emergency Threshold}$$

Cache storage update frequency The traffic from the server to caches can be reduced by having infrequent updates for caching. Every second, caches update all variables other than the variables related to caching. We choose D as a design parameter which denote the time period between two consecutive updates of caching variables, W and w. By adopting a higher D, the system achieves worse performance because of its slower caching adaptation but the traffic from the server to caches reduces and vice versa. Considering the practical objectives of the system, one can design the D parameter to strike a balance between performance and the cost of additional traffic from the server to the caches.

We present the following protocols for both the users and the caches.

Algorithm 1 Cache Protocol

```
1: Set $t_1 = 30$ and $t_2 = D$ iterate:
2: Retrieve from each user node $u$ their marginal utility $V_{x_r}(z_u)$ and measure the round-trip delay $q_r$ for each route $r$.
3: Update the rate $x_r$ for each route using $\Delta x_r = [\delta_r(V_{x_r}(z_u) - \lambda_r - q_r)]^+_{x_r}$.
4: Update the availability price for each route $r$ by $\Delta\lambda_r = [\kappa_r(x_r - W_{hm}\gamma_m)]^+_{\lambda_r}$.
5: if $t_2 = 0$ then
6:     Change the storage fraction $W_{hm}$ for each video by $\Delta W_{hm} = [\iota_{hm}(\Lambda_{hm} - \beta_m\omega_h)]^{[0,1]}_{W_{hm}}$. If $\Delta W_{hm}$ is negative, it removes video $m$ by $W_{hm}$ amount; otherwise, it chooses randomly $\lfloor nW_{hm} \rfloor$ number of packets exclusive of what it already has of every chunk of video $m$, and requests to download from neighboring caches (or from the backup server if not successful). Finally, update the storage price by $\Delta\omega_h = [\nu_h(\sum_{m\in M} W_{hm}\beta_m - s_h)]^+_{\omega_h}$. Reset $t_2 = D$.
7: end if
8: For each neighbor user $u$, allocate number of packets proportional to $x_{hu}$.
9: if $t_1 = 0$ then State Randomly choose and connect to a new neighbor from the neighbor list. Then drop a connected user $u$ with probability $\frac{\exp(-\frac{1}{2}\mu\bar{x}_{\tilde{g}\backslash g'})}{1+\sum\limits_{g''\in\mathcal{A}_{h,\tilde{g}}}\exp(-\frac{1}{2}\mu\bar{x}_{\tilde{g}\backslash g''})}$. Reset $t = 30$.
10: end if
11: $t_1 \leftarrow t_1 - 1$, $t_2 \leftarrow t_2 - 1$.
```

Algorithm 2 User Protocol

```
1: Set $t = 30$ and iterate:
2: Compute the average received rate from each connected cache node since the last update. Derive its marginal utility value and send it to each connected cache node.
3: if buffer length is less than an emergency threshold then
4:     Download from the server all the missing packets to fill up the buffer.
5: end if
6: if has received $k$ packets of the next chunk then
7:     Decode the chunk and flush the data to the media player.
8: end if
9: if $t = 0$ then
10:    Randomly choose and connect to a new neighbor from the neighbor list. Then drop a connected cache node $h$ with probability $\frac{\exp(-\frac{1}{2}\mu\bar{x}_{\tilde{g}\backslash g'})}{1+\sum_{g''\in\mathcal{A}_{h,\tilde{g}}}\exp(-\frac{1}{2}\mu\bar{x}_{\tilde{g}\backslash g''})}$. Reset $t = 30$.
11: end if
12: $t \leftarrow t - 1$.
```

8 Conclusion

In this report, we formulate a general framework for a distributed VoD streaming system considering storage, bandwidth, and node degree bound. We propose a fully-distributed solution and prove that it achieves close-to-optimal performance. We present detailed simulation results and show its performance, scalability and robustness. The resource tradeoff between storage, bandwidth, and node degree is investigated via the experiments. We show that our proposed scheme maximizes the video traffic supported by cache and minimizes the server load. We modify the algorithm derived from the theory to fit the practical settings. We propose concepts of over-caching, emergency threshold, and cache storage update frequency to bridge the gap between theory and practice.

We briefly introduce the various future directions we are actively working on based on this work.

Serverless model : In the current framework, we assume the existence of a central main server which stores all content and which provides any demand whenever caches are not enough to match the demand. Technically, the existence of a main server alleviates difficulty of the problem formulation and affects the solution. However, the assumption of the server's existence may not always hold in many cases. If we can remove the assumption and propose new algorithms, it will be useful not only for cases where the central server does not exist, but also for cases where fully distributed systems are desired in order to populate and disseminate video content under oppressive authorities, for example in Syria or North Korea. Removing the server from the formulation results in several issues which we have not considered in the current framework.

Energy-aware model : Caching can be considered as a way of sprinkling a huge amount of data, which usually sits in a 'server warehouse', to a lot of small caches which locate in different places. We conjecture that our framework can significantly save the cost of energy needed to manage and cool down servers, which leads a greener solution of storage.

Incentive mechanism : For the system with peer caches, each selfish individual cache has his/her own objective and tends to be reluctant to act altruistically by providing its resource for caching and distribution. A well designed incentive system can take an important role to incentive such entities so that peers can actively participate the system and achieves a better system performance.

Evaluation of the practical system : Evaluation of the practical system's performance is another direction of future work. We are actively building the system and deploying it on Amazon EC2 to further validate its usefulness and efficiency at large scales. Especially, we plan to deploy the system and use it as content distribution system for free online education platform. We believe that this solution can support the free online education system by providing scalable content distributions.

Appendices

A Proof of Theorems

Theorem 8. *Consider a generalized coupon collector problem where a collector draws a balls simultaneously out of n balls at each draw. The expected number of collected coupons after d drawings is $\mathbb{E}[N] = n(1-(1-a/n)^d)$.*

Proof. We can expressing N as the sum of indicators as following.

$$N = \sum_{1 \le i \le n} \mathbf{1}[i\text{'th ball is chosen after d drawings}]$$

Then, taking expectation gives,

$$\begin{aligned}
\mathbb{E}[N] &= \sum_{1 \le i \le n} \mathbb{P}(i\text{'th ball is chosen during d drawings}) \\
&= \sum_{1 \le i \le n} 1 - \mathbb{P}(i\text{'th ball is not chosen during d drawings}) \\
&= \sum_{1 \le i \le n} 1 - (1-a/n)^d \\
&= n(1-(1-a/n)^d).
\end{aligned}$$

□

Lemma 9. *The average number of missing packet is $\mathcal{O}(n^{-1})$.*

Proof. Using the second-order Taylor expansion, from Theorem 8 we get the following approximation.

$$\begin{aligned}
ad - \mathbb{E}[N] &= ad - n(1-(1-a/n)^d) \\
&= ad - n(1-(1-ad/n + d(d-1)a^2/2n^2 + \mathcal{O}(n^{-3}))) \\
&= ad - n(ad/n - d(d-1)a^2/2n^2 + \mathcal{O}(n^{-3})) \\
&= ad - n(ad/n - d(d-1)a^2/2n^2 + \mathcal{O}(n^{-3})) \\
&= d(d-1)a^2/2n + \mathcal{O}(n^{-2})
\end{aligned}$$

□

Theorem 10. *If a user is connected to d caches and assuming every cache stores a equal fraction of a content for the user, the number of packets each cache to store, a^*, to guarantee that the expected total number of received packets of user exceeds k is as following.*

$$a^* = \frac{n}{(d-1)}\left(1 - \sqrt{1 - \frac{2(d-1)k}{nd}}\right)$$

Proof. By equating the expected number of packets to k and solve the quadratic equation of a, we have

$$\begin{aligned} \mathbb{E}[N] &\approx ad - \frac{d(d-1)a^2}{2n} = k \\ \Rightarrow a &= \frac{nd - \sqrt{n^2d^2 - 2d(d-1)kn}}{d(d-1)} \\ &= \frac{n}{(d-1)}\left(1 - \sqrt{1 - \frac{2(d-1)k}{nd}}\right). \end{aligned}$$

□

Theorem 11. *Assuming every cache stores a packet, the number of connection a user needs to connect to, d^*, to guarantee that the expected total number of received packets of user exceeds k is as following.*

$$d^* = 0.5 + \frac{n}{a} - \sqrt{\left(0.5 + \frac{n}{a}\right)^2 - \frac{2nk}{a^2}}$$

Proof. Similarly, solving the quadratic equation of d proves the claim. □

References

[1] K. Lee, H. Zhang, Z. Shao, M. Chen, A. Parekh, and K. Ramchandran. An optimized distributed video-on-demand streaming system: Theory and design. In *Communication, Control, and Computing (Allerton), 2012 50th Annual Allerton Conference on*. IEEE, 2010.

[2] D. Applegate, A. Archer, V. Gopalakrishnan, S. Lee, and KK Ramakrishnan. Optimal content placement for a large-scale vod system. In *Proceedings of the 6th International Conference*, page 4. ACM, 2010.

[3] Craig Labovitz. The Other 50% of Internet Traffic. In *Proc. of North American Network Operators Group Meeting, NANOG 54*, 2012.

[4] Wikipedia. Seedbox — Wikipedia, the free encyclopedia, 2012. [Online; accessed 7-October-2012].

[5] J.M. Almeida, D.L. Eager, M.K. Vernon, and S.J. Wright. Minimizing delivery cost in scalable streaming content distribution systems. *IEEE Transactions on Multimedia*, 6(2):356–365, 2004.

[6] Y. Boufkhad, F. Mathieu, F. de Montgolfier, D. Perino, and L. Viennot. Achievable catalog size in peer-to-peer video-on-demand systems. In *Proc. of IPTPS*, 2008.

[7] X. Zhou and C.Z. Xu. Efficient algorithms of video replication and placement on a cluster of streaming servers. *Journal of Network and Computer Applications*, 30(2):515–540, 2007.

[8] N. Laoutaris, V. Zissimopoulos, and I. Stavrakakis. On the optimization of storage capacity allocation for content distribution. *Computer Networks*, 47(3):409–428, 2005.

[9] J. Wu and B. Li. Keep cache replacement simple in peer-assisted vod systems. In *Proc. of IEEE INFOCOM*, 2009.

[10] B. Tan and L. Massoulié. Brief announcement: adaptive content placement for peer-to-peer video-on-demand systems. In *Proc. of ACM PODC*, 2010.

[11] S. Borst, V. Gupta, and A. Walid. Distributed caching algorithms for content distribution networks. In *INFOCOM, 2010 Proceedings IEEE*, pages 1–9. IEEE, 2010.

[12] V. Valancius, N. Laoutaris, L. Massoulié, C. Diot, and P. Rodriguez. Greening the internet with nano data centers. In *Proceedings of the 5th international conference on Emerging networking experiments and technologies*, pages 37–48. ACM, 2009.

[13] X. Zhou and C.Z. Xu. Optimal video replication and placement on a cluster of video-on-demand servers. In *Parallel Processing, 2002. Proceedings. International Conference on*, pages 547–555. IEEE, 2002.

[14] Y. Liu, X. Liu, L. Xiao, L.M. Ni, and X. Zhang. Location-aware topology matching in p2p systems. In *proc. of IEEE INFOCOM*, 2004.

[15] N. Laoutaris, D. Carra, and P. Michiardi. Uplink allocation beyond choke/unchoke. In *Proc. of ACM CoNEXT*, 2008.

[16] V. Aggarwal, O. Akonjang, and A. Feldmann. Improving user and isp experience through isp-aided p2p locality. In *proc.of IEEE INFOCOM*, 2008.

[17] Shaoquan Zhang, Ziyu Shao, and Minghua Chen. Optimal distributed p2p streaming under node degree bounds. In *Proc. of IEEE ICNP*, 2010.

[18] H. Zhang, M. Chen, A. Parekh, and K. Ramchandran. An Adaptive Multi-channel P2P Video-on-Demand System using Plug-and-Play Helpers. In *UC Berkeley Technical Report*, 2010.

[19] S. Pawar, S. Rouayheb, H. Zhang, K. Lee, and K. Ramchandran. Codes for a distributed caching based video-on-demand system. In *Asilomar Conference on Signals, Systems, and Computers*, 2011.

[20] M. Chen, S.C. Liew, Z. Shao, and C. Kai. Markov approximation for combinatorial network optimization. In *Proc. of IEEE INFOCOM*, 2010.

[21] B. Cohen. Incentives build robustness in BitTorrent. In *Proc. of IPTPS*, 2003.

[22] J. S. Plank, J. Luo, C. D. Schuman, L. Xu, and Z. Wilcox-O'Hearn. A performance evaluation and examination of open-source erasure coding libraries for storage. In *FAST-2009: 7th Usenix Conference on File and Storage Technologies*, February 2009.

[23] Shu Lin and Daniel J. Costello. *Error Control Coding, Second Edition*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 2004.