Providing Hierarchical Lookup Service for P2P-VoD Systems

TIEYING ZHANG and XUEQI CHENG, Institute of Computing Technology, Chinese Academy of Sciences
JIANMING LV, South China University of Technology
ZHENHUA LI, Peking University
WEISONG SHI, Wayne State University

Supporting random jump in P2P-VoD systems requires efficient lookup for the "best" suppliers, where "best" means the suppliers should meet two requirements: *content match* and *network quality match*. Most studies use a DHT-based method to provide content lookup; however, these methods are neither able to meet the network quality requirements nor suitable for VoD streaming due to the large overhead. In this paper, we propose Mediacoop, a novel hierarchical lookup scheme combining both content and quality match to provide random jumps for P2P-VoD systems. It exploits the *play position* to efficiently locate the candidate suppliers with required data (content match), and performs refined lookup within the candidates to meet quality match. Theoretical analysis and simulation results show that Mediacoop is able to achieve lower jump latency and control overhead than the typical DHT-based method. Moreover, we implement Mediacoop in a BitTorrent-like P2P-VoD system called CoolFish and make optimizations for such "total cache" applications. The implementation and evaluation in CoolFish show that Mediacoop is able to improve user experiences, especially the jump latency, which verifies the practicability of our design.

Categories and Subject Descriptors: C.2.0 [**Computer-Communication Networks**]: General—*Data communications*; H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval—*Retrieval models; search process*

General Terms: Design, Performance

Additional Key Words and Phrases: Peer-to-peer, video-on-demand, distributed lookup, hierarchical overlay

ACM Reference Format:

Zhang, T., Cheng, X., Lv, J., Li, Z., and Shi, W. 2012. Providing hierarchical lookup service for P2P-VoD systems. ACM Trans. Multimedia Comput. Commun. Appl. 8S, 1, Article 15 (February 2012), 23 pages.
DOI = 10.1145/2089085.2089092 http://doi.acm.org/10.1145/2089085.2089092

1. INTRODUCTION

In recent years, Peer-to-Peer Video-on-Demand (P2P-VoD) systems have attracted enormous attention from both industries and academic institutions. As a popular data intensive Internet application,

This research is supported by the National Basic Research Program of China (Grant No. 2011CB302305), the National High-tech R&D Program of China (Grant No. 2010AA012500) and the National Natural Science Foundation of China (Grant No. 60933005, No. 60873245 and No. 61073015).

Author's address: T. Zhang and X. Cheng, Institute of Computing Technology, Chinese Academy of Sciences, Beijing China; email: zhangtiey@software.ict.ac.cn; J. Lv, South China University of Technology, Guangzhou, China; email: jml@scut.edu.cn; Z. Li, Peking University, Beijing, China; email: lzh@net.pku.edu.cn; W. Shi, Wayne State University, Detroit, MI 48202; email: weisong @wayne.edu.

DOI 10.1145/2089085.2089092 http://doi.acm.org/10.1145/2089085.2089092

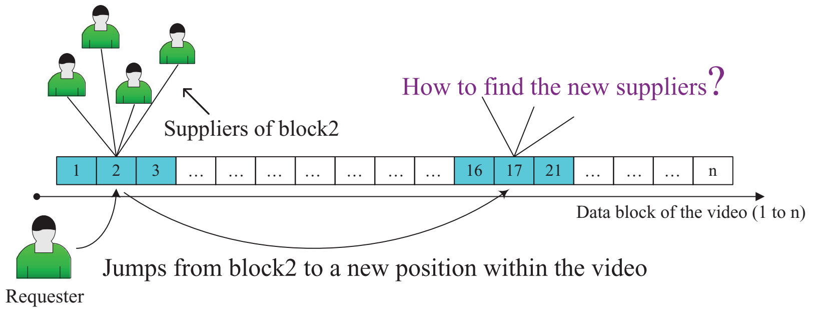

How to find the new suppliers?
Suppliers of block2
1
2
3
16
17
21
n
Data block of the video (1 to n)
Jumps from block2 to a new position within the video
Requester

Fig. 1. An example of jump in P2P-VoD. When a peer jumps to a new position within the video, it should find some suppliers who can provide the new data.

P2P-VoD desperately demands capturing and accessing data effectively and fast so as to support free user interactivities, especially random jump operations. However, in a P2P-VoD system, a requester fetches data directly from the streaming buffer of suppliers, so the random jump operations often make the current suppliers useless. As shown in Figure 1, when a peer jumps to a new position within the video, it should find some suppliers who can provide the new data. Hence, a key goal when designing P2P-VoD systems is to efficiently find the "best" suppliers whose buffer stores the required data (*content match*) with sound network quality (*quality match*).

The goal involves two aspects. One is *content match*, which means the suppliers we found can provide the required data block. This content match requirement is the baseline for the lookup results. The other is *quality match*, which means the suppliers should be of *good* network properties, such as high upload bandwidth and low end-to-end delay to the requester. Quality match is not only an optimization but also a critical factor for on-demand streaming services, as suppliers are often unable to provide enough data in time to satisfy the requester due to their network properties [Huang et al. 2008; Pucha et al. 2007; Hefeeda et al. 2003]. Two important metrics for measuring network properties are delay and bandwidth [Zhou et al. 2008]. In this article, we use delay as the quality performance for supplier lookup, while bandwidth is used for supplier "select" rather than lookup ("select" means to choose good peers after connection establishment, and "lookup" means to find good peers before connection establishment). The details will be given in Section 2 and 4.1.

Until recently, most existing methods mainly focus on the content lookup for P2P-VoD system. A typical approach is using distributed hash table (DHT) based network (e.g., PROP [Guo et al. 2004], PROMISE [Hefeeda et al. 2003] and VMesh [Yiu et al. 2007]), periodically publishing the information of streaming buffer to DHT. However, the content of the buffer changes constantly as playing, and peers have to continuously update the DHT accordingly, which results in large state update overhead. Although VMesh considers the network properties and put locality information into the DHT search keys, how to compute the distance between the multidimensional keys is challenging. The method in PROMISE also explores peers' network properties, but it does not aim at random jump, and can not handle the interactive operations. OBN [Liao et al. 2006] and RINDY [Cheng et al. 2007] propose non-DHT methods to avoid publishing overhead. However, they maintain a loose relationship between peers and can not locate the target accurately. Further more, they do not exploit the network properties of peers either.

In this paper, we propose Mediacoop, a hierarchical lookup method combining both content and quality match to provide random jump service for P2P-VoD systems. The lookup process in Mediacoop is divided into two stages. In the first stage, we use *unchanged playpoint distance* to locate the candidate suppliers with the required data block. Here *unchanged* means the viewers of the same video are expected to playback continuously and thus their playpoint distances do not change. If one viewer performs VCR operations (e.g., jump, stop, and pause) or network churn happens, new playpoint updates will be triggered. In the second stage, we index the candidates into a novel treelike sub-overlay. Refined search is performed within the sub-overlay to efficiently find the "best" suppliers.

Our contributions can be summarized as follows.

(1) We propose an efficient hierarchical lookup scheme combining both content and quality match function to provide random jump service. Our scheme is a general solution, independent of cache volume and data replacement.

(2) For content match, unlike previous DHT-based methods, our approach does not publish content sharing messages to avoid large overhead. To meet quality match, we design a tree-like structure to index candidate suppliers as a sub-overlay, which provides efficient quality match lookup.

(3) Theoretical analysis and simulation results show that compared with traditional methods, Mediacoop can significantly reduce jump latency while improving playback continuity and startup time with less overhead.

(4) We have implemented Mediacoop scheme into a BT-like P2P-VoD system called CoolFish [CoolFish 2011] and make improvements for such "total-cache" system. The results demonstrate that Mediacoop can achieve 40% reduction of jump latency than the old version of CoolFish.

The rest of this article is organized as follows. Section 2 reviews the related work. Section 3 gives a system model for Mediacoop, followed by design details in Section 4. In Section 5, we evaluate the performance of Mediacoop through theoretical analysis and simulations. Section 6 presents our empirical study of Mediacoop in a real-world system. Finally, this article is concluded in Section 7.

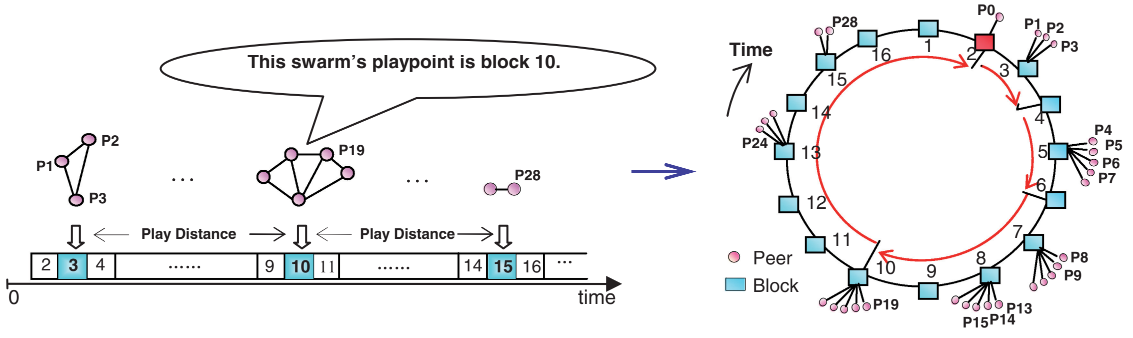
2. RELATED WORK

In this section, we briefly review and discuss the previous efforts on distributed lookup for P2P-VoD systems.

P2P-VoD systems can be categorized into two groups: 1) tree-based and 2) mesh-based according to their organization topologies. The techniques of searching providers (parents) are largely different.

Two typical tree-based systems are P2Cast and P2VoD. P2Cast [Guo et al. 2003] uses patching technique to stream video, while relying on unicast connections among peers. However, the single parent delivery model is not efficient enough in a heterogeneous network. Moreover, it is difficult to maintain a tree structure in a dynamic environment. P2VoD [Do 2004] organizes each video session tree into layers. Peer departure is handled by finding another parent in the upper layer and new client can join the lowest layer of the tree or creating a new layer. However, P2VoD does not provide any mechanism for random jump.

Due to its fine resilience to network churn and peer heterogeneity, the mesh-based paradigm has currently become the mainstream P2P-VoD design model. The few structured lookup methods for P2P-VoD are mostly built on such mesh-based systems. The typical approaches are based on the DHT networks (e.g., Chord [Stoica et al. 2001], Tapestry [Zhao et al. 2004]), where peers publish the hash value of their sharing file name to the closest peer. PROP [Guo et al. 2004] uses a DHT to provide VoD service. In PROP, when a peer gets a video block, it publishes the information to the DHT. If another peer wants to fetch a block, it searches in the DHT for a supplier. A problem of this approach is that continuously receiving data brings high publishing overhead. Another problem is that, when a peer moves on and discards some blocks from its buffer, it needs to send deleting messages to update the DHT. If the deleting messages can not be received in time, it will cause the node failure problem in the figure table. PROMISE [Hefeeda et al. 2003] exploits network bandwidth to provide peer selection. It



(a) Play distance is unchanged unless some peers perform VCR operations or network churn happens.

(b) Playpoint overlay where every swarm corresponds to a playpoint.

Fig. 2. An example of playpoint overlay. Peers with the same playpoint are grouped into one swarm. All swarms are indexed on a ring.

uses the actual media data to probe available bandwidth during the peer's playing. This method cannot handle random jump in VoD because its probe process is performed after connection establishment. This is why the above method is called supplier "select" rather than "lookup". OBN [Liao et al. 2006] and RINDY [Cheng et al. 2007] introduce the idea of attribute lookup. OBN considers the DHT update problem. It uses the buffer relationship between peers to construct a non-DHT overlay. Nevertheless, how to compute the distance between peers accurately is not discussed in the paper. Furthermore, OBN does not exploit the network quality of peers. RINDY uses a similar approach to construct a multiring lookup network. A requester first finds the ring containing the target peers. Then these peers can be located through the gossip protocol. However, the loose relationship between peers can not locate the target accurately. Without a clear definition of distances between peers, we will not know how to perform the lookup operation or locate the accurate routing peers. Additionally, using gossip protocol in RINDY is costly when the ring contains a large number of peers and it does not consider the network properties either. Recently Yiu et al. [2007] propose VMesh, which integrates three mechanisms, that is, scheduling, storage, and lookup, into one solution. For the lookup method, media content is divided into large segments (up to 5-minute video), and a segment is published after completely downloaded. Nevertheless, which segment should be stored depends on a complicated data cache and replacement mechanism. This mechanism is based on popularity design and must utilize hard disk to store large segments. On the contrary, our method is a general solution, independent of cache mechanism and cache volume.

3. SYSTEM MODEL

Now we describe the system model of Mediacoop. Essentially, a distributed lookup is built on the P2P network overlay, which indexes the peers using routing information. Then, the lookup operation can be performed along the network overlay. Therefore, how to construct an efficient overlay is the fundamental framework to provide lookup service. In Mediacoop, two problems need to be solved. One is how to provide content match lookup efficiently; the other is how to find close peers with low delay. To address these two issues, we divide the lookup process into two stages along hierarchical overlays. The basic idea of these two overlays is depicted in this section.

3.1 Playpoint Overlay

The content of a video is segmented into M blocks and each block corresponds to a playpoint. We group peers with the same playpoint into one swarm (see Figure 2(a)). In view of the fact that the distance of playpoints is unchanged due to the constant playback rate for a given movie, our idea in the first stage is to utilize the distance of playpoints to index all swarms on a ring (Figure 2(b)). Therefore, every swarm is acquainted with the status of each other even though they are moving on. A requester can easily find peers according to a Chord-like structured overlay [Stoica et al. 2001]. The difference is that we use the distance of playpoint rather than hash value, and the unchanged distance leads to no requirement for state update messages, which reduces communication overhead.

3.2 Sub-Overlay: Indexed Swarm

The results of the first stage are a series of *seed* peers, from which our goal is to find some peers who have low delay with the requester (*quality match*). Inspired by the delay detection approach which can get the AS level or IP prefix level delay table [Ren et al. 2006] (a global delay table), in the second stage, we index all the peers in the target swarm using their IP prefixes as a sub-overlay. The target IP prefixes with low delay can be picked up from the global delay table (such as Table I). Then we can find the peers within the target IP prefixes along the sub-overlay using IP-prefix information stored on each peer. The details about how to get the delay table without ISPs' help will be explained in the following section.

4. DESIGN OF MEDIACOOP

In this section, we first address the importance of the end-to-end delay for P2P-VoD and give the method of how to obtain the delay in Section 4.1. Then we present the finger tables utilizing the distance of playpoint and IP prefix in Section 4.2. How to conduct lookup operation using the finger table is shown in Section 4.3. Finally, in Section 4.4 we describe the maintenance of Mediacoop.

4.1 Exploit Network Properties

In real-time media systems, lower end-to-end delay leads to less waiting time and hence improves the interactivity [Liu 2007; Noh et al. 2008; Ren et al. 2006]. Especially in VoD services, the user interactivity frequently happens [Huang et al. 2008], and the large delay between peers results in long response time. Therefore, the end-to-end delay is adopted to measure the network property of supplying peers. One might argue how to get the exact value of the end-to-end delay, because it varies with time. In practice, we only need to record the *average* delays for different candidates, and choose the smaller ones as suppliers. So we do not require the exact value of the delay. Then the question is how to get the end-to-end delay. The Internet is composed of many Autonomous Systems (ASes). Peers within one AS are usually close to each other and inter-AS routing is specified by Border Gateway Protocol (BGP). Thus, we only need to know the AS-AS delays or cluster level delays and choose suppliers from those clusters whose delay is the least with the initiator. We use the method proposed in [Ren et al. 2006] to construct the AS topology and obtain AS-AS delays. A sketch of the method is briefly described as follows (cf. Ren et al. [2006] for more details).

Firstly, we collect a large number of public BGP routing tables and BGP updates, such as those from RouteViews [RouteViews 2010] and RIPERIS [RIPERIS 2010]. From these routing tables, we build an AS graph and group IPs with the same longest prefix into one cluster. Then, we randomly choose one IP out of each cluster as the cluster delegate. A delay measurement between each pair of cluster delegates can be estimated by the tool King [Gummadi et al. 2002]. Finally, we obtain two tables: (1) IP to cluster mapping table (ICMT); (2) Cluster to cluster delay table (CCDT). From ICMT, a peer can learn about its own cluster, and get the target clusters from CCDT, which has the least delay with itself. Figure 3 illustrates the work procedure of the delay detector, and Table I shows a CCDT example in CoolFish [CoolFish 2011]. In Mediacoop, the delay detector program runs and updates ICMT and CCDT all the time. When a peer joins the system, it gets ICMT and CCDT from the probing server. In order to

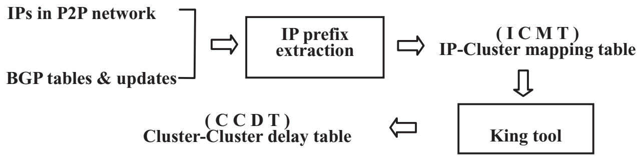


Fig. 3. Cluster (IP prefix) level delay detector. Using this detector we can get the global tables: ICMT and CCDT.

Table I. CCDT of CoolFish. Time: 15:00, 2/9/2009

Delay from X to Y	159.226.40.*	202.127.200.*	210.72.15.*	...
159.226.40.*	0	31(ms)	8(ms)	...
202.127.200.*	31(ms)	0	58(ms)	...
210.72.15.*	8(ms)	58(ms)	0	...
...	...	...	...	...

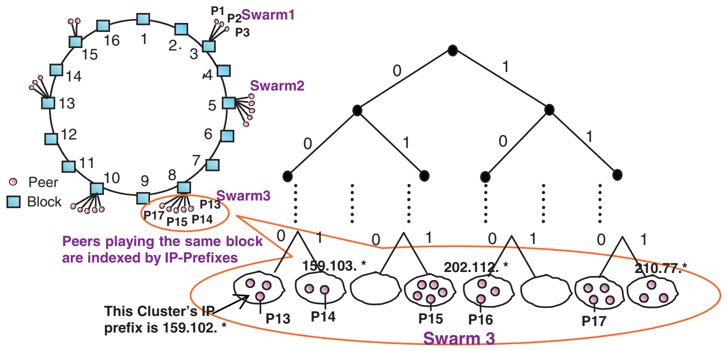


Fig. 4. Illustration of binary tree overlay. Each swarm on the ring corresponds to a binary tree overlay. A leaf node is annotated with the IP prefix which is represented by the prefix code. Peers in the swarm is assigned with corresponding leaf nodes according to their IP prefixes.

prevent frequent requests in normal lookup operations, the peers will ask the server to update ICMT and CCDT at intervals. Now, the key problem is how to organize these clusters to provide efficient lookup.

In our approach, clusters are organized into a binary tree, among which each leaf node represents a cluster. Therefore, the number of leaf nodes K is equal to that of the clusters. Every leaf node is annotated with the network address (IP prefix), which is represented by its prefix code (see Figure 4). The distance between the leaves $node_1$ and $node_2$ is calculated by the XOR operation of their prefix. Specifically, the distance is defined as follows:

$$D_{IPprefix}(n_1, n_2) = Prefix_{n_1} \oplus Prefix_{n_2}. \tag{1}$$

Some might argue why we use the leaf nodes of a binary tree to organize the IP prefixes. The reason is that the distance between the leaf nodes with prefix code is consistent with the measurement of XOR operation.

4.2 Peer State

How to perform distributed lookup depends on the routing information stored by each peer. In this section we discuss the detailed structure of peers' figure table. Mediacoop performs the lookup operations

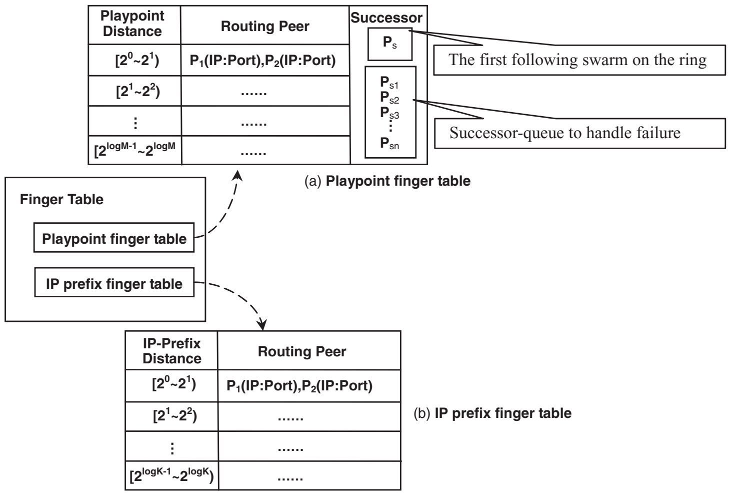

Playpoint Distance
Routing Peer
Successor
$[2^0\sim2^1)$
P_1(IP:Port),P_2(IP:Port)
$[2^1\sim2^2)$
......
$[2^{logM-1}\sim2^{logM}$
P_s
The first following swarm on the ring
P_{s1} P_{s2} P_{s3} P_{sn}
Successor-queue to handle failure
(a) Playpoint finger table
Finger Table
Playpoint finger table
IP prefix finger table
IP-Prefix Distance
Routing Peer
$[2^0\sim2^1)$
P_1(IP:Port),P_2(IP:Port)
$[2^1\sim2^2)$
$[2^{logK-1}\sim2^{logK})$
(b) IP prefix finger table

Fig. 5. Finger table. Peer's Finger table includes two sub-tables: playpoint finger table (a) and IP prefix finger table (b). There is a successor in playpoint finger table. Successor-queue is designed for failure handling.

along hierarchical overlays. Accordingly, every peer has two finger tables: 1) Playpoint finger table, and 2) IP prefix finger table.

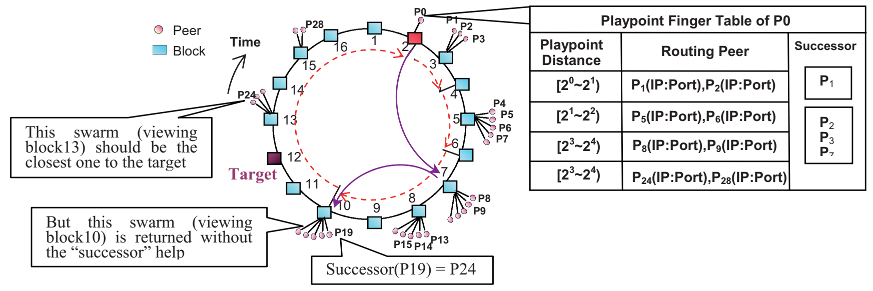
Playpoint finger table (PPFT). PPFT stores playpoint information about other swarms. It has $\log M$ (M is the number of blocks) entries. For each entry i ($0 \le i < \log M$), it keeps information for k peers of playpoint (play block) distance between 2^i and 2^{i+1} from itself. The distance from p_1 to p_2 is defined as the number of hops from p_1 to p_2 in clockwise direction along the ring:

$$D_{playpoint}(p_1, p_2) = ((PP_{p_2} - PP_{p_1}) + M) \quad mod \quad M \tag{2}$$

where M is the number of blocks, PP_{p_i} is p_i's playpoint. Figure 5(a) presents the structure of PPFT. Every entry records k peers (in our system we set k to 2) whose playpoint is within the distance interval $[2^i, 2^{i+1})$. If the number of peers in the interval is larger than k, we only record k peers' information according to LRU (Least Recently Used) replacement strategy. Figure 6 shows an example of PPFT. p_0's playpoint is $block_2$. It has 4 entries in its PPFT and each entry is responsible for an interval as shown as the red dash line. Each interval records 2 routing peers.

In addition, for every peer (with playpoint of PP_{p_i}), it records a **successor** whose playpoint *just follows* PP_{p_i}. For example, as shown in Figure 6, p_0's playpoint is $block_2$, and the first following swarm is the one playing $block_3$. Thus, p_1 or p_2 or p_3 in this swarm can be the successor of p_0. For another example, p_{19}'s playpoint is $block_{10}$, but there is no peer playing $block_{11}$. So the first following swarm is the one playing $block_{13}$. Thus, p_{24} in this swarm can be the successor of p_{19}. Such successor design plays an important role to find the "real closest" peers, which is discussed in Section 4.3.

IP prefix finger table (IPFT). IPFT records the IP prefixes of other peers within the same swarm. IPFT has $\log K$ (K is the number of the leaf nodes) entries. For each entry j ($0 \le j < \log K$), it keeps information for the peers of distance between 2^j and 2^{j+1} from itself. The distance is calculated according to Equation (1). Figure 5(b) shows the structure of IPFT. Because the distance calculation of IP prefix is different from that of playpoint, it is not necessary to design successor for IP prefix finger table. The detailed reasons are discussed in the following section.


Peer
Block
Time
P28
P0
P1
P2
P3
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
P24
P4
P5
P6
P7
P8
P9
Target
P19
P15
P14
P13
This swarm (viewing block13) should be the closest one to the target
But this swarm (viewing block10) is returned without the "successor" help
Successor(P19) = P24

Playpoint Finger Table of P0		
Playpoint Distance	Routing Peer	Successor
$[2^0 \sim 2^1)$	P_1(IP:Port),P_2(IP:Port)	P_1
$[2^1 \sim 2^2)$	P_5(IP:Port),P_6(IP:Port)	P_2 P_3 P_7
$[2^3 \sim 2^4)$	P_8(IP:Port),P_9(IP:Port)	
$[2^3 \sim 2^4)$	P_{24}(IP:Port),P_{28}(IP:Port)	

Fig. 6. Content match lookup with the successor's help. Successor(p) is a peer in the first swarm following p clockwise on the ring. When p_{19} is found at the end of content match lookup, in order to get the real closest peers, its successor p_{24} is also returned.

4.3 Lookup along Hierarchical Overlays

Content Match Lookup. This is the first stage lookup along the playpoint overlay. A recursive algorithm is employed starting from the initiator p. On receiving a lookup message, the recipient firstly calculates the distance $D_{content}$ from itself to the required block according to Equation (2). Then, the recipient picks the closest peer from its PPFT and relays the message to it. This algorithm terminates when the closest peers are found. That is to say, the termination is when $D_{content}$ is equal to 0 or there is no routing peers' distance smaller than $D_{content}$. Then the recipient itself is the closest peer according to Equation (2).

However, the "closest" peer we found above is probably not the real closest one. For example, in Figure 6 when the lookup target is $block_{12}$, but there are no peers playing $block_{12}$. According to the lookup procedure, we finally find p_{19} as the "closest" peer. However, apparently, the more suitable peer is p_{24}, because p_{24} is only one block away from the target. Actually, according to Equation (2), the closest peers are in the first swarm that *most closely precedes or equals to the target*, which we refer as to *predecessor*, such as predecessor($block_{12}$)=p_{19}. If the predecessor has a long distance from the target, it probably can not provide the required data. Such problem mainly stems from the buffer length and the download speed of a peer. Again, take an example of Figure 6. Although p_{19} has a relative long distance with the target of $block_{12}$, if its download speed is high, it could have more "prefetched" data and probably have $block_{12}$. On the other hand, assume that if a peer could cache all the data it has watched, every swarm after $block_{12}$ could meet the "content match" to provide the required data of $block_{12}$. Unfortunately, different P2P-VoD systems employ a variety of cache mechanisms.

As a general solution, we design a "successor" scheme to find the real closest peers. In our scheme, peer p_k with playpoint PP_{p_k} records a peer in the first swarm whose playpoint just follows PP_{p_k} in the playpoint overlay. This peer is called the successor of p_k, denoted by successor(p_k). Because the playpoint overlay is represented as a circle of numbers, successor(p_k) should be a peer in the first swarm clockwise from p_k. In Figure 6, p_{19}'s successor can be any peer playing $block_{13}$. In this case, we set successor(p_{19})=p_{24}. When the content lookup completes based on Equation (2), the peers we find and their successors are all returned. In this example, when p_{19} is found, its successor p_{24} is also returned. Therefore, we get the closest peers *around* the target. The closest peers returned are used as the seed peers for the second stage lookup.

Quality Match Lookup within the Candidate Swarm. After the initiator p obtains the seed peers from the first stage, p gets α target IP prefixes which have the least delay from itself in CCDT, where α is a system parameter (in simulation we set $\alpha = 3$). Then, p sends an IP prefix lookup message to the seeds. Each seed relays the message to the closest peers in its IPFT. The distance $D_{quality}$ is calculated according to Equation (1). The relay peers perform the same operation recursively until there are no

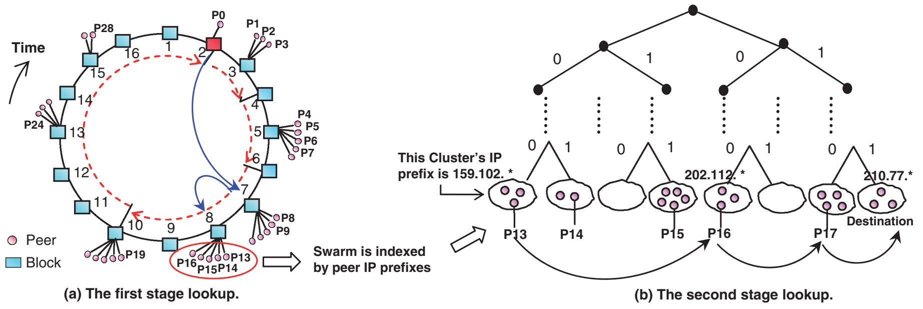

Time
Peer
Block
Swarm is indexed by peer IP prefixes
(a) The first stage lookup.
This Cluster's IP prefix is 159.102.*
202.112.*
210.77.*
Destination
(b) The second stage lookup.

Fig. 7. An example of hierarchical lookup. P_0 firstly finds p_{13} playing $block_8$. Then, from p_{13} refined lookup is conducted along the swarm ($block_8$).

routing peers with closer distance than itself. Therefore, if there are no peers matching the target IP prefixes, the lookup will return the peers who have the closest IP prefixes with the target.

Note that the terminate condition is different from that of content match lookup. In the content match lookup, when $D_{content}$ is equal to 0, the procedure will terminate. However, in the quality match lookup, any peer that meets quality match should be returned, even if $D_{quality}$ is 0. That is to say, the lookup will continue until there are no routing peers with $D_{quality}$ equal to 0. Furthermore, quality match does not have the "real closest" problem involved in content match. So it is not necessary to design "successor" in IP prefix finger table.

Someone might argue that whether we can use a global peer list to contain all peers within a swarm rather than using IPFT to lookup. According to our practical experience, some swarms are very large (100,000∼400,000 peers), such as in BitTorrent. Therefore, we must design structured lookup mechanism (IPFT) for such large amounts of peers.

An example. Figure 7 shows an example of the whole lookup process from p_0. In Figure 7(a), p_0 initiates a lookup with the goal to find suppliers who can provide $block_8$. It calculates $D_{playpoint}(p_0, block_8) = 6$ and sends the "content match" lookup message to the closest neighbor p_8. In the same way, p_8 calculates $D = 1$ and finds that p_{13} belongs to the candidate swarm. Therefore, p_{13} and its successor p_{19} are returned to the initiator as seeds. Then, p_0 looks up its CCDT, and picks out the target IP prefix 210.77.* which has the least delay from p_0. Next, p_0 sends the IP prefix lookup message to p_{13} and p_{19} to start the second stage (see Figure 7(b), we do not show p_{19} in the second stage because it is similar with p_{13}). On receiving the message, p_{13} looks up IPFT and relay it to p_{16} which is the closest one. In the same way p_{16} relays the massage to p_{17} and finally finds the target peers within cluster 210.77.*.

4.4 Maintenance

The maintenance of Mediacoop includes three components: peer joining, peer state updating, and failure handling.

4.4.1 *Peer Joining*. To join the network, a peer p must contact an already existing peer j. p inserts j's information into its own finger table. Then p performs a peer lookup for its own attributes (playpoint and IP prefix). After that, p gets the closest neighbors: predecessor p_a and successor p_s. obviously, p's finger table is the same as that of its predecessor. So p refreshes its fingers further away than p_a with p_a's finger table. During the refreshes, p also inserts itself into others' finger tables. Also the successor relationship should be changed. If p and p_a are not in the same swarm, p notifies p_a the successor should be changed to p. At the same time, p assigns p_s as its successor.

4.4.2 *Peer State Updating.* When a peer p performs VCR operations, such as pause, stop and jump, p will notify its neighbors (in both finger tables) of its new playpoint and its neighbors will update the information.

4.4.3 *Failure Handling.* The ability to detect, handle and recover from failures is important for any distributed system. Usually, failures are an expected part of normal operations rather than a set of special case failure handlers. The cache information are updated by periodic refreshment messages as appearing in Tapestry [Zhao et al. 2004] and Berkeley Service Discovery Service [Hodes et al. 2002]. To detect peer failures during normal operations, each Mediacoop peer sends periodic heartbeats on UDP packets to peers including the successor in its finger table. By checking the status of each peer when a message arrivs, we can quickly detect the failures and replace the failure peers via ordinary gossip messages or by searching some new peers.

In addition, peer failures must not be allowed to disrupt queries especially when they are in progress. We use surrogate or replication to handle this. Each entry in the finger table stores m backup neighbors (m is 2 in our system implementation) in addition to the primary routing peers, which is similar to Plaxton [Plaxton et al. 1997]. When the primary routing peers fail, we turn to the alternate peers in a sequential order until a correct route is found.

To handle the successor failure, each Mediacoop peer maintains a "successor-queue" of its n nearest successors (n is 3 in our system implementation) on the ring, as shown in Figure 6. If a peer notices that its successor has failed, this peer replaces the fail successor with the first live entry in its successor-queue. The maintenance of successor-queue can be done through the normal failure detection and replacement discussed above.

5. PERFORMANCE EVALUATION

In this section, we first analyze the lookup efficiency of our method, followed by the simulation results based on the NS2 simulator.

5.1 Theoretical Analysis

Our method includes two-stage lookups. Accordingly, we model the performance as follows:

$$P(M, K) = P_{FirstStage}(M) + P_{SecondStage}(K),$$

where $P(M, K)$ is the total lookup hop counts of Mediacoop; $P_{FirstStage}(M)$ and $P_{SecondStage}(K)$ is the hop counts of the first and second stage respectively; M is the number of blocks and K is the number of clusters.

Firstly, we analyze $P_{FirstStage}(M)$. Mediacoop is a structured method and the lookup procedure is similar to the DHT protocols like Chord. What is different is that we use playpoint (play block) instead of peer identifier in DHTs. Therefore, the performance is related to not only the traditional $O(logN)$ in DHTs, but also the number of playpoints (i.e., the number of blocks):

$$P_{FirstStage}(M) = \min\{O(logM), O(logN)\},$$

where N is the total number of peers. In general, the number of peers for a popular P2P system is large. In contrast, the number of blocks for a video is severely limited. For example, according to our experience, 720 blocks are enough for a common 2-hour movie. That is, $M \ll N$, and thus $P_{FirstStage}(M) = \min\{O(logM), O(logN)\} = O(logM)$.

For the second stage, the lookup procedure is essentially a binary search along the search tree. Therefore, the lookup performance of the second stage is equal to the time-complexity of the binary search:

$$P_{SecondStage}(K) = O(logK),$$

where K is the number of valid IP clusters. Invalid IP clusters mean that they are empty with no peers within them and will not be taken into consideration. In fact, the second stage is conducted in the target swarm with the number of peers $n = \frac{N}{M}$ (we can assume that all peers are uniformly distributed among play blocks). Thus, K is actually equal to or less than n, where n is the maximum value of K when each cluster only has one peer. Therefore, the total complexity of our hierarchical lookup method is:

$$P(M, K) = P_{FirstStage}(M) + P_{SecondStage}(K)$$

$$\leq O(logM) + O\left(log\frac{N}{M}\right) = O(logN),$$

that is

$$P(M, K) \leq logN.$$

That is to say, in no more than $O(logN)$ hops, we can find out the peers meeting both content and delay requirements through only one lookup operation.

5.2 Simulation Model

5.2.1 *Churn Model.* A number of properties of Mediacoop under different network churns are evaluated. The network churns include network congestion, varied streaming rates and peer churns. Usually, the primary source of inconsistencies is peers' joining and online time. Guo et al. [2005] prove that the user arrival follows an exponentially decreasing rule rather than Poisson distribution assumed in the previous studies. Therefore, we set the exponentially decreasing of the peer arrival rate as:

$$\lambda(t) = \lambda_0 e^{-\frac{t}{\tau}}, \tag{3}$$

where λ_0 is the initial arrival rate when the video is published, and τ is the attenuation parameter. Accordingly, we set average inter-arrival time as 5s.

As for peers' online time, several previous studies use Pareto distribution to model it [Leonard et al. 2005; Sen and Wang 2004] and some others use Weibull [Atalla et al. 2008; Stutzbach and Rejaie 2006] and exponential [Li et al. 2005; Rhea et al. 2004] distributions. Exponential distribution is adopted in this paper to model peers' online time due to its convenient calculation of mean value, which is used as the parameter in the simulation experiments. The simulations are divided into several groups with different online time (detail results are shown in Section 5.3).

5.2.2 *Simulation Configurations.* In the simulation, we generate a transit-stub topology including 860 routers using GT-ITM [Zegura et al. 1996]. And we randomly select 100 stub nodes as the IP prefix cluster routers[1]. The delay between any two nodes is 10ms$\sim$60ms. We generate 8,000 peers attached to IP cluster routers following uniform distribution.

The NS2 simulation program was performed on our super computing cluster called Dawning 4000A [Xu 2007], which has 640 nodes and 2,560 AMD Opteron processors connected by Myrinet 2000 with 5TB main memory.

[1]GT-ITM does not have the function to separate transit nodes from stub nodes, so we develop a tool to support this.

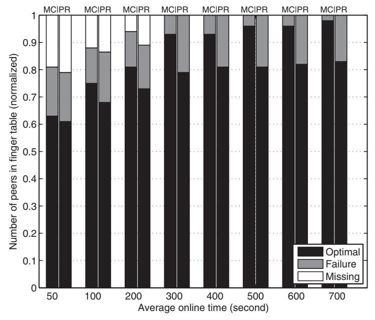


Fig. 8. Quality of the finger tables under different online time. The higher peer failures in PROP (PR) due to the frequency of publishing messages.

5.3 Basic Performance Evaluation

In this section, we first discuss the finger tables under peer churns and varied streaming rates. Then, the lookup failures are considered. Finally, we evaluate the packet loss in Mediacoop's routing under network congestion. In order to compare Mediacoop with the traditional DHT methods, we implement PROP [Guo et al. 2004], a representative DHT-based VoD system. We do not implement the replica servers in PROP, because it is out of the scope of this paper.

5.3.1 *Finger Table Status.* In this experiment, we evaluate the finger tables of Mediacoop (MC) and PROP (PR).

Figure 8 shows the quality of the finger table with different online time. We set the video time = 1200s with playback rate = 600Kbps, and a block is 256K Bytes. The download bandwidth is 1Mbps and any peer has the ability to serve one stream. Each entry in the finger table stores two peers with status update every 60 seconds. The "quality" of finger table includes three levels: Optimal, Failure and Missing. *Optimal* means that finger table is full and every peer in the finger table is available; *Failure* means the peer is in the position but failed; *Missing* means that the position in finger table is empty. Because PROP has only one finger table, we accumulate the results of playpoint finger table and IP prefix finger table in order to compare with PROP.

The results show that Mediacoop always outperforms PROP. The quality of finger table tends to improve with the online time increasing. When the average online time is larger than 300 seconds, the finger table status of both Mediacoop and PROP becomes stable, and "Optimal" takes up more than 92% in Mediacoop while less than 82% in PROP. That is because in PROP when a peer moves on and discards some blocks from its buffer, it needs to send deleting messages to update the DHT. Furthermore, the packet loss is higher as background traffic increases (see Figure 11), and if the deleting messages can not be received in time under network congestion, it will cause peer failures in the figure table.

Figure 9 shows the quality of the finger tables with different streaming rates. The simulations are divided into 5 groups with streaming rate from 200Kbps to 1,000Kbps. The average online time is 60 seconds and the block size is still 256K Bytes. The results are shown in Figure 9. We can see that the number of peers in finger table is similar between Mediacoop and PROP. However, with the increase of streaming rate, the quality is stable for Mediacoop while decreasing for PROP. The reason is that high streaming rate leads to high frequency of old block deleting messages. High frequency has high probability to fail. In addition, high steaming rate will cause increased network congestion which blocks the deleting messages in turn. All the reasons result in high peer failures in PROP.

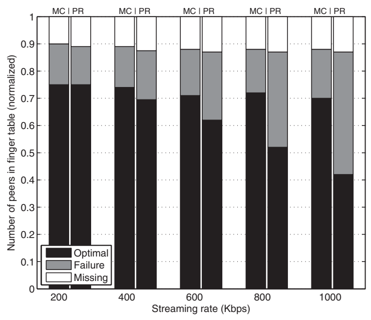


Fig. 9. Quality of the finger tables with different streaming rates. High streaming rate causes high frequency of publishing which results in high probability to fail in PROP (PR).

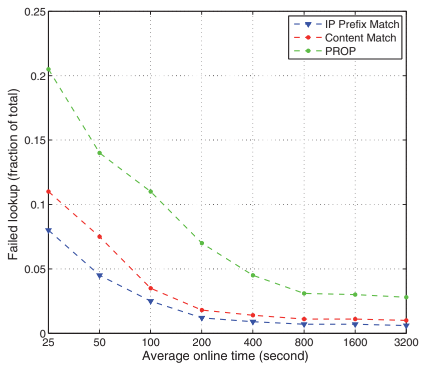


Fig. 10. Lookup failures with different online time.

5.3.2 *Lookup Failures.* In this experiment, we evaluate the lookup ability of Mediacoop with respect to peers' online time. As in [Lv et al. 2002; Castro et al. 2004; Stoica et al. 2001], lookup operations are generated according to a Poisson process at a rate of one per second. Mediacoop peers use ping messages to estimate reliability of the next-hop. A failed lookup is the one that can not be forwarded, that is, if the forwarded entry is empty or the peers in this entry are all failed, the lookup fails. In addition, the simulator does not retry queries. Thus, the results given in this section can be viewed as the worst-case for the lookup failures. We also set two peers in one entry and updates all finger table entries every 60 seconds. In the simulation, we examine content lookup and quality lookup separately with PROP.

Figure 10 plots the average failure rates with different peers online time. Intuitively, the lookup failure should be the same as the rate of peer failures if every entry has one peer, since this is just the fraction of routing peers expected to be lost due to the failure of the responsible peers. We conclude that there is no significant lookup failure in the Mediacoop network. For example, when the average online time is 200 seconds, there should be 19% peer failures in Mediacoop and 28% in PROP (Failure + Missing, as shown in Figure 8). Our results do not show this, suggesting that peer redundance is useful when deploying real system and we also find that Mediacoop is more robust than PROP due to the lower peer failures.

5.3.3 *Packet Loss.* This experiment demonstrates the packet loss in the Mediacoop network. Mediacoop peers use ping messages to estimate reliability of the neighbor links in the finger table and

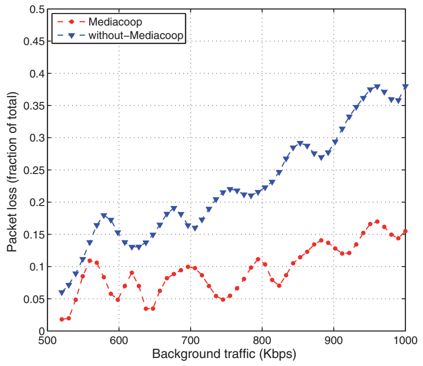


Fig. 11. Packet loss. Mediacoop has low packet loss because it estimates reliability of the next-hop using ping messages.

pick out the winner as the next-hop. In our simulation, we generate the background traffic as follows: any two peers transferring a stream at a rate between 400Kbps ∼ 1500Kbps. As the average background packets increases, router queues begin to overflow and packet loss increases. Figure 11 shows the simulation results of packet loss as average background traffic increases on the network. We found that Mediacoop significantly reduces the packet loss as the packets route around congestive regions. Furthermore, the reduced packet loss results in fewer retransmissions and overall higher throughput. This result is useful for the deployment of the real-world system, since the routing optimization is not complicated (just using ping messages).

5.4 Comparisons with Popular Systems

In order to compare Mediacoop with the popular approaches, besides PROP based-on the DHT, we also simulate a typical "cache and relay" system, P2VoD [Do 2004], to compare with our approach. We choose P2VoD because it also uses playpoint to divide peers into layers. Peers in upper layer deliver data to the lower ones, organized in a tree-like structure. We add random jump function and peer search along parents and siblings. We have evaluated two versions of Mediacoop using the NS2 simulator. The first one is Mediacoop (no-DA), which does not include delay-awareness (DA); the second is Mediacoop(DA) with "DA." Thus, Mediacoop(no-DA) actually does not have IPFT. Instead, it employs gossip protocol to exchange the information of peers. We separate DA in simulations in order to study the performance of the proposed mechanisms in two stages respectively. In our simulations, we set the video time = 3600s with playback rate = 500Kbps, and the time of a block is 10s. The average peer on-line time = 1800s. The download bandwidth is 1Mbps and any peer has ability to serve two streams. The startup and jump periods both buffer 5-second media data. 5 seconds is quite short, which is sensitive to examine our scheme. The simulations are divided into 12 groups with peers from 100 to 8,000 and the total runtime is 4 days on our supercomputer.

We evaluate Mediacoop using the following performance metrics.

(1) *Average number of hops*: the average number of routing hops for a complete lookup process.

(2) *Control overhead*: all control message overhead including the messages of joining, jump, data scheduling and content publishing.

(3) *Server stress*: the outgoing bandwidth required at the media server to support the whole system. We use the peak stress to examine the system scalability.

(4) *Playback continuity*: the ratio of pieces that arrive before or on the playback deadlines.

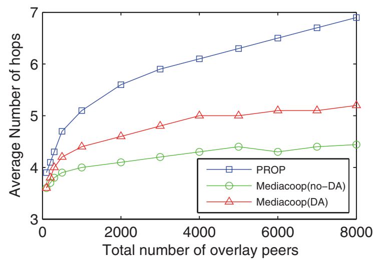


Fig. 12. Average number of hops for varying overlay sizes.

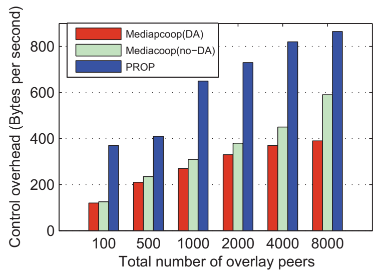


Fig. 13. Control overhead for varying overlay sizes.

(5) *Startup latency*: the latency from the moment a user sends a request for a video to the moment it starts playing the required video, after buffering 5-second data.

(6) *Jump latency*: the time from the moment a user launches a jump operation to the moment it starts playing the video from the jump position after buffering 5-second data.

In this section, we present the simulation results. In summary, compared with the typical DHT-based method, Mediacoop(DA) decreases about 50% in the jump latency and 27% in the startup delay. Furthermore, the overhead is reduced by about 55% on average. Compared with the traditional tree-based method, our mechanism has considerable advantages. The detailed results are presented as follows.

5.4.1 *Average Number of Hops.* For this metric, we do not consider the gossip stage in Mediacoop (no-DA), because the purpose of comparing hops is only for structured methods. We do not mention P2VoD either due to its unstructured overlay. Figure 12 shows the average number of routing hops as a function of the overlay sizes from 100 to 8000 with 12 groups. The results show that PROP takes "$logN$" rule as the traditional DHT-based methods. Both versions of Mediacoop perform better than PROP, because the lookup hops in our method are related to the number of blocks and clusters. We have $\frac{3600}{10} = 360$ blocks and $\frac{8000}{360} \approx 22$ peers for each block. Accordingly, the number of valid clusters for each block are less than 22, reducing the routing hops subsequently.

5.4.2 *Control Overhead.* In order to compare the update overhead of DHT-based methods, we compare our scheme with PROP. As shown in Figure 13, because Mediacoop(no-DA) adopts gossip protocol, which brings extra control messages, there are more control traffic than Mediacoop(DA) as network expands. In PROP, the overhead is considerable because peers continuously send the publishing and deleting messages. In contrast, Mediacoop(DA) can reduce the overhead by about 40–70%. The overhead of Quality Match Lookup accounts for about one-fifth of the total. Because the proportion varies little with the number of peers, we do not show it in the figure.

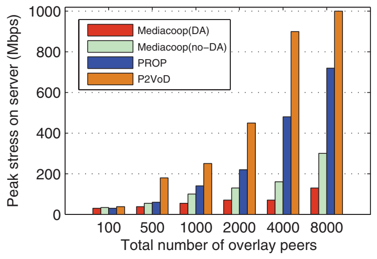


Fig. 14. Peak stress on media server for varying overlay sizes.

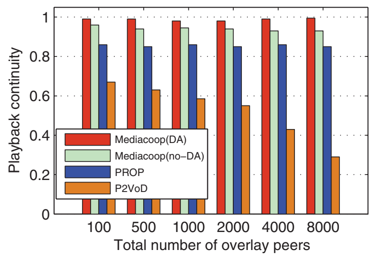


Fig. 15. Playback continuity for varying overlay sizes.

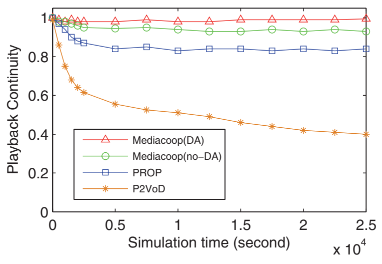


Fig. 16. Playback continuity against simulation time with the overlay size=4000.

5.4.3 *Server Stress.* There is a media content server with 1000 Mbps upload bandwidth in the system to support unsatisfied peer requests. If a peer has not received the required data when timeout occurs, or there are no suppliers to serve it, it will ask the server to send data. We use peak stress (Mbps) on the server as the metric of system scalability. Figure 14 shows the peak stress against different overlay sizes from 100 to 8000. Mediacoop(no-DA) achieves higher server stress than Mediacoop(DA). This is because some suppliers in Mediacoop(no-DA) cause higher delays, which leads to the urgent requests timeout. Then the server is asked to send the missing data again. For PROP, with buffer moving on, the published blocks are discarded. This causes the requests for these blocks unsatisfied, which results in higher server stress than Mediacoop. In P2VoD, server stress increases linearly with the size of network. This is because the system is organized in a tree-like structure, with upper-level peers providing data to lower-level ones. Therefore, the parents are so lacking that most peers directly request data from server which makes the server stress so high.

5.4.4 *Playback Continuity.* Firstly, we track the playback continuity with different overlay sizes in Figure 15. Then we examine it against simulation time with 4000 peers, as shown in Figure 16. Our schemes perform better than the other two systems (P2VoD and PROP) and improve the playback

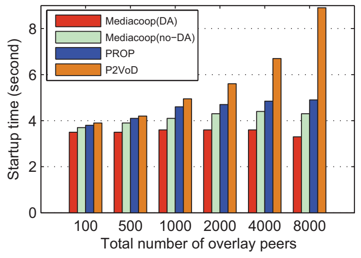


Fig. 17. Startup time for varying overlay sizes.

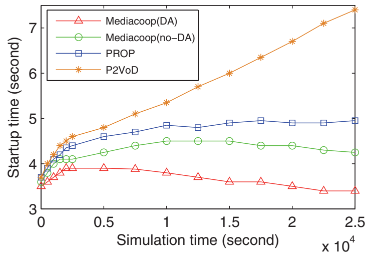


Fig. 18. Startup time against simulation time with the overlay size=4000.

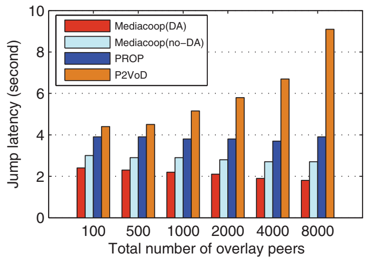


Fig. 19. Jump latency for varying overlay sizes.

continuity much closer to 1.0. Besides the factor explained in Server Stress, another reason is that the DHT methods must wait to publish the sharing messages until the data block is completely downloaded. This waiting leads to lower sharing level of the available data resources.

5.4.5 *Startup and Jump latency.* Actually, these metrics involve two parts: the lookup latency and the buffering time. The first part is explained in Average Number of Hops. The second part depends on the quality of suppliers, which is also the critical factor for Playback Continuity. Note that Mediacoop(DA) could find close suppliers and hence fill up its buffer more quickly. Figure 17, 18, 19 and 20 illustrate the results. For the 5-second buffering time, Mediacoop(DA) only needs about 3.5 seconds for startup time and 2 seconds for jump latency on average.

6. EMPIRICAL STUDY IN COOLFISH

We implement Mediacoop in the current version of our real-world P2P-VoD system called CoolFish [CoolFish 2011]. In this section, we first give the system overview of CoolFish, then we discuss the optimization of Mediacoop for BT-like system, and finally the comparison results are presented.

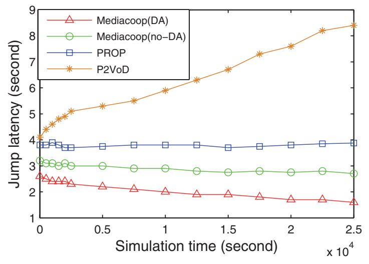


Fig. 20. Jump latency against simulation time with the overlay size=4000.

6.1 System Overview

CoolFish is the first P2P-VoD system deployed in China Science & Technology Network (CSTNet), one of the four major ISPs in China. From Oct. 2008 to July. 2010, there had been over 4.9 million user visits and the number of recent daily visits has exceeded 7000. In the hot time, there are 700 simultaneous online viewers. CoolFish is able to support an average video bit rate of 700Kbps, which is about 50% higher than that of most commercial P2P-VoD systems with a video bit rate less than 450Kbps [PPStream 2011; Liu et al. 2010; Huang et al. 2008; Cheng et al. 2008]. The details of CoolFish are presented in the Appendix.

6.2 Adaptive Mediacoop for BT-like P2P-VoD Systems

When implementing Mediacoop in CoolFish, we observed Mediacoop could be optimized in dealing with the “total-cache” system. Total-cache means all the data blocks will be cached in the users’ disk space, typically used by BT-like systems. Actually, CoolFish is a BT-like P2P-VoD system, where peers cache the played-out data in their disk space. In such systems, the playpoint is not adequate to represent a supplier’s contents. For example in Figure 6, if p_0 wants to find out some peers who hold $block_{12}$, using Mediacoop p_{19} with playpoint of $block_{10}$ and its successor p_{24} will be returned as the closest peers according to our general design. However, returning P_{24} is not enough because all the peers follows $block_{12}$ clockwise on the ring are likely to provide the data. If we return more peers, there would be more opportunities to select “good” suppliers for the upper layer software.

Driven by the above observations, we improved Mediacoop in its implementation on CoolFish: Mediacoop should return y peers whose playpoint follows the target in clockwise direction, not only the peers whose playpoint is the closest to the target. These y peers can be obtained through the successor-queue. We call this optimization of Mediacoop as adaptive Mediacoop for BT-like P2P-VoD systems. The notable thing is that our original Mediacoop design is still practical and applicable for most applications as a general solution. Therefore it is unnecessary to modify it in Section 4.

6.3 Evaluation on Finger Tables

In the real-world system, we can not control user behaviors such as online time and access frequency. Therefore, we take the inter-arrival time between two consecutive peers as the access frequency and record the quality of finger tables under different inter-arrival time. Because CoolFish is not a large-scale system and the peak online number of peers is about 700, this will cause some entries empty in the finger tables. Therefore we only consider the Failure and Optimal scenarios. Figure 21 shows the results. We can see that the finger tables have high quality even under small inter-arrival time (such as 1s and 2s), and the quality is stable (Optimal takes up more than 95%) with different inter-arrival time.

The most important reason for such high quality of finger tables is shown in Figure 22. This figure plots the PDF of inter-arrival time and the corresponding average online time. We can see that the

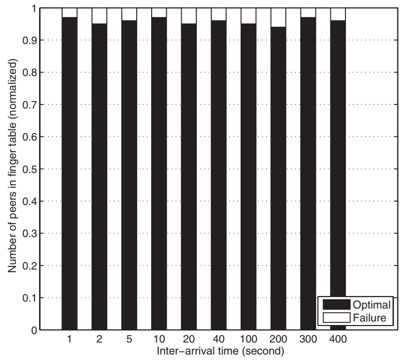


Fig. 21. Finger tables under different inter-arrival time (in CoolFish). The quality is high even under small inter-arrival time (such as 1sec and 2sec).

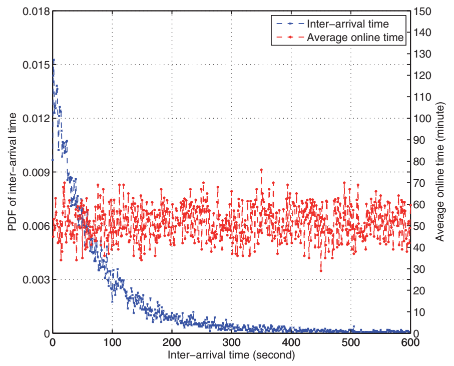


Fig. 22. PDFs of inter-arrival time are different while the average online time uniformly distributes between 30 and 75 minutes.

average online time is distributed between 30 and 75 minutes uniformly, although the PDFs of inter-arrival time are different. Furthermore, the simulation results in Figure 8 demonstrate that when the online time is larger than 500 seconds, the quality of finger tables will exceed 95%. The simulations also proves the results in the real-world system (CoolFish).

6.4 Comparison of User Experience

In this experiment, we show the comparison results between Mediacoop and without-Mediacoop in CoolFish. After the implementation of Mediacoop on CoolFish, we find that the sever stress are nearly the same with that in without-Mediacoop. Therefore, sever stress is not referred in comparison. Instead, we mainly focus on the evaluation of user experience. User experience is the most important metric to examine the real-world P2P-VoD system. As defined in Section 5, user experience includes three aspects: A. playback continuity, B. startup time and C. jump latency. *Although the goal of Mediacoop is mostly to decrease jump latency, it is necessary to examine playback continuity and startup time*. In this Section, we present the user experience comparisons between the current version of CoolFish (Mediacoop) and the old version of CoolFish (without-Mediacoop). We collect one-month (30 days) log data for both old CoolFish and current CoolFish. From Figure 27 (in Appendix), we can see that the system scales are similar for these two versions in these two months (we only represent the system

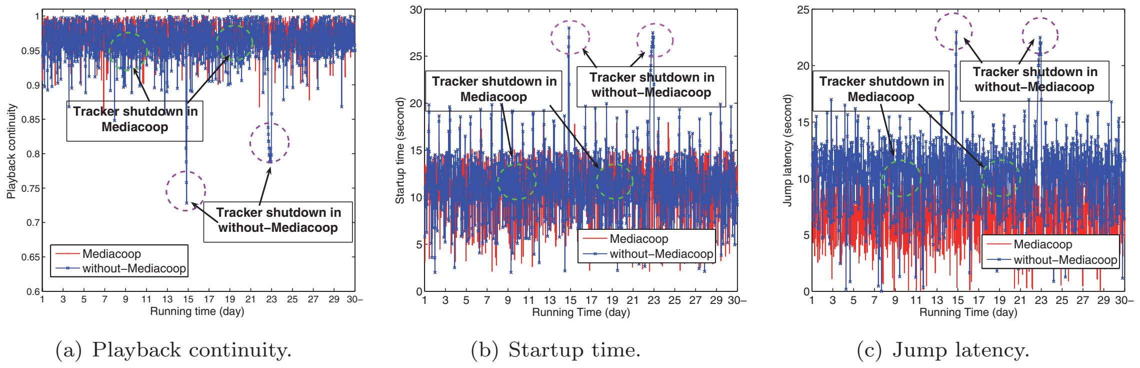


Fig. 23. User experience over running time. Mediacoop avoids the negative effects of Tracker failure (dash circle) and decreases jump latency by 40% than without-Mediacoop.

scale in one week for clarity), so the comparison is fair. In order to further guarantee the fairness of comparison, we did not add any other functions or improvements into the system.

Note that CoolFish is a well-deployed system rather than an experimental platform, and the old version of CoolFish was able to select close neighbors with less delay (quality match). Hence in our comparison, the results only reflect the performance of "content match" of Mediacoop.

6.4.1 *User Experience over Running Time.* We track the user experience during the system running time. In our system, the mean value of user experience (continuity, startup time or jump latency) is recorded every 30 minutes.

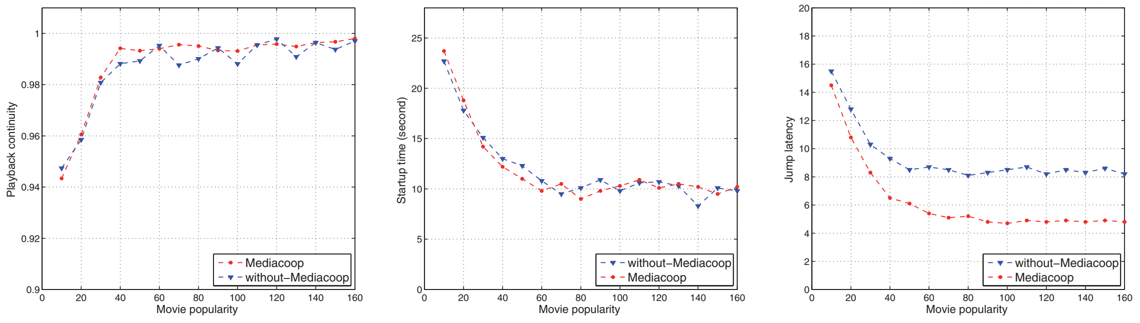
Figure 23(a) plots the playback continuity over the running time. We observe that, in general, both curves have similar playback continuity, mostly between 0.9-1.0. However, for without-Mediacoop, the playback continuity decreased sharply when the tracker shutdown happened (purple dash lines). The first shutdown happened in the 14th day (18:41 p.m.–21:03 p.m.), and the second one is in the 22th day (17:24 p.m.–23:55 p.m.). In contrast, tracker shutdown did not incur abnormal fluctuation in Mediacoop (green dash lines: the first one lasted for 10 hours and the second is 8 hours). The reason is: without Mediacoop, although peers are still able to connect to Media Server when the Tracker runs into downtime, the system degenerates to C/S architecture and the Media Server can not support so many peers. However, in Mediacoop peers are able to get "good" neighbors during the Tracker downtime.

Figure 23(b) represents the startup time over running time. Again we find the tracker shutdown problem in old CoolFish (without-Mediacoop). The reason is similar as explained above. We can see the difference of startup time between Mediacoop and without-Mediacoop is negligible. In CoolFish, the media data for startup is cached on every peer, so whether we use Mediacoop does not matter.

Figure 23(c) shows jump latency over running time. Different from startup time, we find that during Tracker downtime Mediacoop has far less jump latency (6.1s on average) than without-Mediacoop (10.2s on average). The reason is that Mediacoop could find "good" suppliers when jumping to a new position. However, without Mediacoop the peers returned by Tracker might not have the required data.

In general, Mediacoop not only significantly decreases jump latency (40%) but also avoids the negative effects of Tracker failure.

6.4.2 *User Experience over Movie Popularity.* To understand the performance of Mediacoop more deeply, we examine the user experience over different movie popularities (the number of viewers for a movie).



(a) Playback continuity. (b) Startup time. (c) Jump latency.

Fig. 24. User experience over movie popularity. Mediacoop can reduce the jump latency by about 40% (Figure 24(c)), and it also guarantees the playback continuity and startup delay (Figure 24(a)(b)).

Figure 24(a) shows the playback continuity over different popularities. Mediacoop does not show big advantage because both curves have high playback continuity from 0.94 to 1.0. However, the curve of without-Mediacoop fluctuates more greatly. This is because without Mediacoop, peers can only rely on the Tracker. But the peers randomly returned by the Tracker cannot guarantee they have the required data.

Figure 24(b) plots startup time over movie popularity. The difference of startup time between Mediacoop and without-Mediacoop is slight. The reason is similar to that of Figure 23(b), that is in CoolFish, the data for startup is cached on every peer viewing the same movie, so whether we use Mediacoop or not does not matter.

Obvious differences of jump latencies are shown in Figure 24(c). Mediacoop performs better than without-Mediacoop, especially when the popularity is large. That is because if there are more viewers watching the same video, Mediacoop has more choice to find out "good" suppliers. On the contrary, more viewers means more random peer return for without-Mediacoop. When the popularity is larger than 90, jump latency is stable for both Mediacoop (4.8s) and without-Mediacoop (8s-8.5s).

7. CONCLUSIONS

While lookup service for random jump is an essential function in P2P-VoD systems, its content match and quality match requirements need to be satisfied effectively. The proposed Mediacoop scheme, represented by its hierarchical lookup, provides both content and quality match satisfaction in a single lookup. It leverages the efficiency of structured method but avoids the state update overhead. In addition, it exploits the end-to-end delay and constructs a treelike sub-overlay to find high-quality suppliers. With these novel features, Mediacoop is able to achieve low startup and jump latency with high playback continuity. Both theoretical and simulation results confirm the effectiveness of Mediacoop. Compared with the typical DHT-based method, Mediacoop achieves 50% reduction in the jump latency and 27% in the startup delay. Furthermore, the overhead is reduced by about 55% in average. The implementation and evaluation in the real-world system also show the practicability of our design. The running results demonstrate that Mediacoop is able to achieve 40% reduction of the jump latency than the old version of CoolFish.

ACKNOWLEDGMENTS

We would like to thank Mrs. Yi Yi for her help in figure design, Professor Huawei Shen for his insightful advice, and Professor Jiafeng Guo for his comprehensive help.

REFERENCES

Atalla, F., Miranda, D., Almeida, J., Gonçalves, M. A., and Almeida, V. 2008. Analyzing the impact of churn and malicious behavior on the quality of peer-to-peer web search. In *Proceedings of the ACM Symposium on Applied Computing (SAC'08)*.

Castro, M., Costa, M., and Rowstron, A. 2004. Performance and dependability of structured peer-to-peer overlays. In P*roceedings of the International Conference on Dependable Systems and Networks*.

Cheng, B., Jin, H., and Liao, X. 2007. Supporting VCR functions in p2p vod services using ring-assisted overlays. In *Proceedings of the IEEE International Conference on Communications (ICC'07)*. 1698–1703.

Cheng, B., Stein, L., Jin, H., Liao, X., and Zhang, Z. 2008. Gridcast: Improving peer sharing for p2p vod. *ACM Trans. Multimedia Comput. Commun. Appl. 4*, 4, 1–31.

CoolFish. 2011. http://www.cool-fish.org.

CSTNet. 2010. http://www.cstnet.net.cn/bill.jsp.

Do, T. T. 2004. P2vod: Providing fault tolerant video-on-demand streaming in peer-to-peer environment. In *Proceedings of the IEEE International Conference on Communication*s. 1467–1472.

Google Analytics. 2010. http://www.google.com/analytics.

Gummadi, K. P., Saroiu, S., and Gribble, S. D. 2002. King: ESTIMATING latency between arbitrary internet end hosts. In *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet Measurement (IMW '02)*.

Guo, L., Chen, S., Ren, S., Chen, X., and Jiang, S. 2004. Prop: a scalable and reliable p2p assisted proxy streaming system. In *Proceedings of the IEEE International Conference on Distributed Computing Systems (ICDCS'04)*. 778–786.

Guo, L., Chen, S., Xiao, Z., Tan, E., Ding, X., and Zhang, X. 2005. Measurements, analysis, and modeling of bittorrent-like systems. In *Proceedings of the Internet Measurement Conference (IMC'05)*. USENIX Association, Berkeley, CA, 4–4.

Guo, Y., Suh, K., Kurose, J., and Towsley, D. 2003. P2cast: peer-to-peer patching scheme for vod service. In *Proceedings of the International World Wide Web Conference (WWW'03)*. ACM, New York, NY, 301–309.

Hefeeda, M., Habib, A., Botev, B., Xu, D., and Bhargava, B. 2003. Promise: Peer-to-peer media streaming using collectcast. In *Proceedings of the International Conference on Multimedia*. ACM, New York, NY, 45–54.

Hodes, T. D., Czerwinski, S. E., Zhao, B. Y., Joseph, A. D., and Katz, R. H. 2002. An architecture for secure wide-area service discovery. *Wirel. Netw. 8*, 2–3, 213–230.

Huang, Y., Fu, T., Chiu, D., Lui, J., and Huang, C. 2008. Challenges, design and analysis of a large-scale p2p-vod system. *ACM SIGCOMM Comput. Comm. Rev. 38*, 4, 375–388.

Leonard, D., Rai, V., and Loguinov, D. On lifetime-based node failure and stochastic resilience of decentralized peer-to-peer networks. In *Proceedings of the ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*.

Li, J., Stribling, J., Morris, R., Kaashoek, M., and Gil, T. 2005. A performance vs. cost framework for evaluating dht design tradeoffs under churn. In *Proceedings of the Annual Joint Conference of the IEEE Computer and Communications Societies.*

Liao, C.-S., Sun, W.-H., King, C.-T., and Hsiao, H.-C. 2006. Obn: Peering for finding suppliers in p2p on-demand streaming systems. In *Proceedings of the 12th International Conference on Parallel and Distributed Systems*. 235–242.

Liu, Y. 2007. On the minimum delay peer-to-peer video streaming: how real-time can it be? In *Proceedings of the International Conference on Multimedia.* ACM, New York, NY, 127–136.

Liu, Z., Wu, C., Li, B., and Zhao, S. 2010. Uusee: Large-scale operational on-demand streaming with random network coding. In *Proceedings of the Annual Joint Conference of the IEEE Computer and Communications Societies.*

Lv, J., Cheng, X., Jiang, Q., Ye, J., Zhang, T., Lin, S., and Wang, L. 2007. Livebt: Providing video-on-demand streaming service over bittorrent systems. In *Proceedings of the International Conference on Parallel and Distributed Computing Applications and Technologies*. 501–508.

Lv, Q., Cao, P., Cohen, E., Li, K., and Shenker, S. 2002. Search and replication in unstructured peer-to-peer networks. In *Proceedings of the 16th International Conference on Supercomputing.*

Noh, J., Mavlankar, A., Baccichet, P., and Girod, B. 2008. Reducing end-to-end transmission delay in p2p streaming systems using multiple trees with moderate outdegree. In *Proceedings of the International Conference on Multimedia and Expo.*

Plaxton, C. G., Rajaraman, R., and Richa, A. W. 1997. Accessing nearby copies of replicated objects in a distributed environment. In *Proceedings of the Annual ACM Symposium on Parallelism in Algorithms and Architectures (SPAA'97)*. ACM, New York, NY, 311–320.

PPStream. 2011. http://www.ppstream.com.

Pucha, H., Andersen, D. G., and Kaminsky, M. 2007. Exploiting similarity for multi-source downloads using file handprints. In *Proceedings of the ACM/USENIX Symposium on Networked Systems Design and Implementation*.

REN, S., GUO, L., AND ZHANG, X. 2006. Asap: an as-aware peer-relay protocol for high quality VOIP. In *Proceedings of the IEEE International Conference on Distributed Computing Systems (ICDCS '06)*. IEEE Computer Society, Los Alamitos, CA, 70.

RHEA, S., GEELS, D., ROSCOE, T., AND KUBIATOWICZ, J. 2004. Handling churn in a DHT. In *Proceedings of the USENIX Annual Technical Conference*.

RIPERIS. 2010. http://www.ripe.net/projects/ris.

ROUTEVIEWS. 2010. http://www.routeviews.org.

SEN, S. AND WANG, J. 2004. Analyzing peer-to-peer traffic across large networks. *IEEE/ACM Trans. Netw. 12*.

STOICA, I., MORRIS, R., KARGER, D., KAASHOEK, M. F., AND BALAKRISHNAN, H. 2001. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of the ACM SIGCOMM Data Communications Festival (SIGCOMM'01)*. ACM, New York, NY, 149–160.

STUTZBACH, D. AND REJAIE, R. 2006. Understanding churn in peer-to-peer networks. In *Proceedings of the 6th ACM SIGCOMM Conference on Internet Measurement.*

XU, Z. 2007. HPC research at ICT. In *Proceedings of the 2007 Asian Technology Information Program's 3rd Workshop on High Performance Computing in China. (CHINA HPC'07)*. ACM, New York, NY, 1–5.

YIU, W., JIN, X., AND CHAN, S. 2007. VMesh: Distributed segment storage for peer-to-peer interactive video streaming. *IEEE J. Select. Areas in Comm. 25*, 9, 1717–1731.

ZEGURA, E., CALVERT, K., AND BHATTACHARJEE, S. 1996. How to model an internetwork. In *Proceedings of the Annual Joint Conference of the IEEE Computer and Communications Societies*. 594-602.

ZHAO, B. Y., HUANG, L., STRIBLING, J., RHEA, S. C., JOSEPH, A. D., AND KUBIATOWICZ, J. D. 2004. Tapestry: A resilient global-scale overlay for service deployment. *IEEE J. Select. Areas in Comm. 22*, 41–53.

ZHOU, X., IPPOLITI, D., AND ZHANG, L. 2008. Fair bandwidth sharing and delay differentiation: Joint packet scheduling with buffer management. *Comput. Comm. 31*, 17, 4072–4080.

Received July 2010; revised March 2011, May 2011; accepted May 2011