Efficient Search in P2P-based Video-on-Demand Streaming Service

Huicheng Chi and Qian Zhang
Department of Computer Science
Hong Kong University of Science and Technology, Hong Kong, China
{addison, qianzh}@cs.ust.hk

Abstract — *Providing video-on-demand streaming service to a large population of clients using peer-to-peer approach is drawing great interest recently. Since clients' demands are asynchronous and the buffered contents are continuously changing, how to find partners with expected data and collaborate with each other for future content delivery are very important and challenging. In this paper, we propose a generic buffer-assisted search (BAS) scheme to improve partner search efficiency. Extensive simulation results demonstrate that BAS can provide faster response time with less control cost than the existing search methods.*

I. Introduction

With the widespread deployment of broadband access, Video-on-Demand (VoD) streaming on the Internet has received increasing attention recently. In VoD streaming service, video can be delivered to asynchronous users with low delay and VCR-like operation support (e.g., pause, fast-forward, and rewind). However, streaming to a large population of clients is very challenge due to the limited server capacity and little deployment of IP multicast [5]. Peer-to-peer (P2P) technology is considered to be one of the promising solutions for the streaming service [1]. However, applying these techniques into VoD streaming is not a trivial task due to the following fundamental differences between the two types of streaming [2, 3, 6].

In a typical P2P-based VoD system, cooperative peers [1] are organized into an overlay network via unicast tunnels. The streaming content is split into a sequence of segments, each of which is a small playable unit, and the server distributes these segments among clients of asynchronous demands. Each client caches a few segments around its *play offset*. The client exchanges the available segments with the *partners*, who have close play offset and thus the client can fetch the expected data from its partners with high probability. Therefore, *partner search* is one of the main components in the P2P-based VoD system. For the users in the VoD system, when joining or taking VCR operations, it needs to search for the partners. Since different users' demands may be asynchronous, and the contents buffered in one peer are continuously changing, an additional index structure is needed for the partner search upon peer joins or VCR operations. In general, we can assume the playing speed is identical for all the peers, so their partner relationship will not change unless they leave, fail, or take VCR-jump. It is noticed that there are two overlay networks in the VoD system: the index overlay for partner search and the data overlay for media transmission.

It is very challenging to develop an efficient partner search structure for a large P2P-based VoD system. Due to scalability concerns, the search structures should be distributed ones with sub-linear search time efficiency. Some distributed structures have been proposed with logarithmic search efficiency, e.g., AVL tree [8] and skip-list [7]. In those structures, all peers are sorted by the play offset and maintained in the index overlay. However, indexing all the peers incurs non-trivial maintenance overhead, especially considering the insertion, deletion, and rebalancing cost, since VCR operations are frequent and the nodes join or leave at will in the dynamic P2P-based VoD systems.

In this paper, we propose a novel Buffer Assisted Search (BAS) structure to address the above challenges. It can be observed that there is buffer overlapping among different nodes, and removing the nodes whose buffer range is fully covered by other nodes does not reduce the total buffer coverage of the search structure. Therefore, we introduce BAS structure to exploit this buffer coverage redundancy to reduce the size of search structure. By indexing a small subset of peers, BAS provides constant search time efficiency and low control overhead. In addition, the BAS structure is generic and can be implemented based on existing data structures, e.g., AVL-tree and skip-list.

The rest of the paper is organized as follows. The background and motivation is reviewed in Section II. Section III presents the BAS structure. Simulation results are given in Section IV, followed by the conclusions in Section V.

II. Background and Motivation

In this section, we briefly review the related works on partner search in VoD service and present one concrete example that motivates our study.

Developing the efficient search schemes for P2P-based VoD service has been a very active research. CollectCast [4] and oStream [1] record the play progress of all the nodes in the centralized server. Though it is simple and easy to manage, the server is easily overloaded in the presence of frequent peer join, leave, and VCR operations. P2Cast [3], P2VoD [2], TAG [8], and DSL [7] logically organize all peers into linear, tree, or skip-list structures. Given the play offset as a key, these methods support random search and VCR operation in a distributed manner. Each peer maintains constant or logarithmic number of index neighbors and the search is performed by tracing the peers hop by hop according to the neighbor information. Although some of them provide sub-linear search efficiency, the maintenance cost is not trivial since all peers need to be indexed in the structure. The frequent VCR operations and dynamic P2P environment further aggravate this maintenance overhead.

Recent studies show that it is not necessary to index all peers because finding a small number of partners is sufficient for the media streaming [2, 3, 7, 8]. Therefore, it is possible to prune the indexing structure without scarifying the search performance. We propose BAS scheme to exploit the client buffer coverage in order to keep the indexing structure within a small scale. In the VoD system, each peer has a buffer to maintain contents around its play offset and acts as a potential partner for the peers whose play offset lies within its buffer range. If a peer, whose buffer range is fully covered by other peers, has been removed from the indexing structure, the other peers who use this removed peer as a partner can still find other possible partners instead. Thus, in the BAS structure, those nodes with redundant buffer coverage can be safely removed from the indexing structure. Fig 1 shows an illustration example for the effectiveness of BAS scheme, where AVL tree is used as the basic structure. In this example, seven peers join the system in the sequence from *P1* to *P7*. Fig 1(a) and Fig 1(b) show

[1] User, client, peer and node are used interchangeably in this paper.

the AVL tree and BAS structure respectively after seven peers joined. As shown in Fig 1(a), *P1* is covered by *P3* and *P4*, *P2* is within *P5*, *P3* is covered by *P5* and *P6*, and *P4* is covered by *P6* and *P7*. Assume that each new peer examines the buffer overlap of the existing peers in the BAS structure, and then *P1*, *P2*, *P3,* and *P4* can be removed upon the arrival of *P4*, *P5*, *P6,* and *P7* accordingly. Thus in Fig 1(b), only *P5*, *P6* and *P7* are indexed while the total buffer coverage is not affected. We will present how to minimize the number of index peers while ensuring the search effectiveness in detail in Section III.

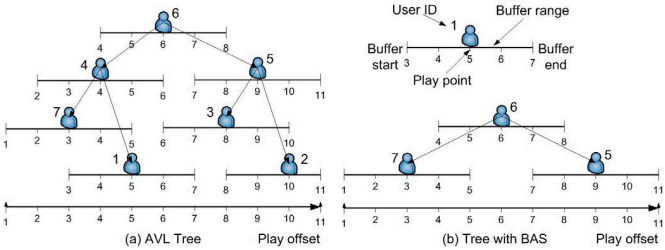

Fig 1. The effectiveness of BAS on an AVL tree

III. Buffer-Assisted Search Overlay

The objective of BAS is to maintain as few index peers as possible for better search efficiency. In other words, we want to minimize the search overlay size without affecting the search effectiveness. We first formulate this as the Minimum Buffer Cover (MBC) problem and present a globally optimal solution. Concerning the system scalability, we then design a distributed algorithm and show how to apply it in the overlay construction, maintenance, and VCR operations accordingly.

A. Minimum Buffer Cover Problem and Optimal Solution

Each peer has a buffer range around its play offset and may overlap with other peers in buffer coverage. By observing there is buffer coverage redundancy, BAS aims to select as few index peers as possible without reducing search effectiveness, i.e., total buffer coverage. It is formulated as the minimum buffer cover (MBC) problem as follows. We consider a collection C of buffers totally covering an integer range R [1, 2, …, M][2], and every buffer is a consecutive subrange of R. A buffer cover for R is a subset $C' \subseteq C$ such that every element in R belongs to at least one member of C'. The goal is to find the buffer cover C' with minimal cardinality $|C'|$.

We design a dynamic programming algorithm to compute the optimal solution. Let $A(0) = \Phi$ and $A(k)$ be the MBC to cover R_k [1, 2, ..., k] for $k \geq 1$. Obviously, $|A(i)| \leq |A(j)|$ for $i \leq j$. We denote the set of buffers covering number k as Y_k, the smallest number in the buffers of Y_k as m_k and the corresponding buffer containing m_k as $\chi_k \in Y_k$. If there are more than one buffers containing m_k, choose the one with longest length.

Theorem 1: If $k \geq 1$, $A(m_k\text{-}1) \cup \{\chi_k\}$ is also the MBC for R_k.

Proof: In any buffer cover A' for R_k, there is at least one buffer $\chi' \in (Y_k \cap A')$. Assume the smallest number in χ' is m', then the set $A'\text{-}\{\chi'\}$ at least covers $R_{m'\text{-}1}$ [1,2,…,m'-1], thus

$$|A'\text{-}\{\chi'\}| \geq |A(m'\text{-}1)|, \text{ i.e., } |A'| \geq |A(m'\text{-}1)|+1.$$

Recall that $m_k \leq m'$, we have

$$|A'| \geq |A(m'\text{-}1)|+1 \geq |A(m_k\text{-}1)|+1 = |A(m_k\text{-}1) \cup \{\chi_k\}|.$$

Thus, $|A(k)| \geq |A(m_k\text{-}1) \cup \{\chi_k\}|$. From the optimality of $A(k)$ for R_k, $|A(k)| \leq |A(m_k\text{-}1) \cup \{\chi_k\}|$. So $A(k)$ and $A(m_k\text{-}1) \cup \{\chi_k\}$ must have the same cardinality, and $A(m_k\text{-}1) \cup \{\chi_k\}$ is also a minimum buffer cover for R_k. □

Theorem 1 indicates the recurrence in the dynamic programming solution is given by:

$$A(k) = \begin{cases} \Phi & \text{if } k = 0, \\ A(m_k - 1) \cup \{\chi_k\} & \text{if } 1 \leq k \leq M. \end{cases}$$

Note that $A(M)$ is the optimal solution to MBC problem. There is a trick in computing $A(M)$. Instead of computing all $A(i)$'s for $i<M$, we only need to trace from $A(M)$ to $A(0)$. Since each step requires $O(|C|)$ comparison time for m_k and the number of steps is $O(|C'|)$, the total running time is $O(|C|^2)$. If the average buffer length of the peers is much smaller than $|C|$ (constant relative to $|C|$), we can further reduce the running time by pre-assigning the buffers to their containing numbers using $O(|C|)$ time. Thus upon computing $A(k)$, only the buffers containing k are compared. Since each buffer is compared only once, the running time is $O(|C|)$. Thus, the total running time can be reduced to $O(|C|)$. The latter solution is described in Fig 2.

```
1:  for i = 1 to M  { Y_i = Φ, A(i) = Φ; }
2:  for each buffer χ∈C
3:      for each i∈χ
4:          Y_i = Y_i ∪ {χ};
5:  j= M;
6:  while (j>0) {
7:      χ = argmin_χ{the smallest number s_χ in χ, χ∈ Y_j };
8:      A(M) = A(M) ∪{χ}; j = s_χ-1;   }
```

Fig 2. The globally optimal Dynamic Programming algorithm

Although this algorithm is efficient and provides optimal solution, it requires global information of all peers' buffer. This is not practical in a large and distributed system. We propose a distributed algorithm which makes decision only based on local information in the next subsection.

B. Distributed Algorithm

The large and distributed systems require a distributed algorithm that adjusts the index overlay based on local information. Since the peers come asynchronously and not all the peers are maintained in the index overlay, we design a distributed algorithm to calculate the optimal solution only based on the existing index peers plus the new comer, upon each new peer insertion. Note that, at any time, every index peer contains at least one *unique* number, which is not covered by other index peers, or else it will be removed from the index overlay.

The basic flow of our distributed algorithm is to divide the existing index peers into two groups according to whether they overlap with the new comer, and then apply the dynamic programming algorithm to the new comer plus the group overlapping with the new comer. We denote the collection of buffers of existing index peers by C' and the newly added buffer by β. Let $LAP = \{\alpha | \alpha \in C'$ and $\alpha \cap \beta \neq \Phi\}$ and $UNLAP = C'\text{-}LAP$. Assume $UNLAP$ covers [1,…, i] and [j,…, M]. Let $LAP' = \{\alpha \cap \{i+1,.., j-1\} | \alpha \in LAP\}$ and record the mapping between LAP and LAP'. Let $D = \{\beta\} \cup LAP'$. The dynamic programming algorithm is to compute the optimal solution D' for D. Restore the buffers in D' according to the mapping between LAP and LAP'. Let $C'' = D' \cup UNLAP$ and it is the optimal solution for $\{\beta\} \cup C'$. The complexity of the distributed algorithm is determined by the dynamic programming algorithm,

[2] If the covering area consists of several discontinuous ranges, the problem can be deduced into some sub-problems, each for a range and the corresponding buffers.

which is O(|D|) = O(|*LAP*|), i.e. the number of existing index peers overlapping with the new peer, instead of the system size.

Theorem 2: *C''* is the optimal solution for {β}∪*C'*.

Proof: Before β's arrival, each buffer α in *C'* contains at least one *unique* number, which is not covered by other buffers in *C'*. Since the new buffer β does not affect the unique numbers of the non-overlap buffers, they must remain in the optimal set. Note that *UNLAP* covers [1,…, i] and [j,…, M], and the uncovered area left becomes [i+1,…, j-1]. *LAP* is adjusted as *LAP'* to compute the minimum buffer cover *D'* for [i+1,…, j-1]. Thus, the union of *D'* and *UNLAP* is the minimum buffer cover for {β}∪*C'*. □

As more peers join the system, the same operation is applied to *C''* to get C^3, and then C^4, C^5…. Similarly, C^{k+1} is the optimal solution for {β}∪C^k. Since the complexity of the distributed algorithm is determined by |*LAP*| and C^k is the corresponding index overlay, we will study the magnitude of |*LAP*| and compare C^k with the globally optimal solution as follows.

Let $N = |C^k|$ and the buffers in C^k, denoted as $X_1, X_2, ..., X_N$, be ordered by their minimum numbers. l_i and r_i are used to represent the minimum and maximum number of X_i. B_i denotes the length of X_i, and the average length of all buffers is *B*.

Lemma 3: $r_i < r_j$ for i<j.

Proof: From the definition, we have $l_i < l_j$ for i<j. If $r_i > r_j$, then X_j can be fully covered by X_i, which contradicts the assumption. □

Lemma 4: $l_{i+2} - l_i > B_i$.

Proof: Assume $l_{i+2} - l_i \le B_i$, i.e., $l_{i+2} \le B_i + l_i = r_i$. Thus, the range from l_i to r_{i+2} is fully covered. According to the definition and lemma 1, we get $l_i < l_{i+1}$ and $r_{i+1} < r_{i+2}$, i.e., X_{i+1} is fully covered by X_i and X_{i+2}, contradicting the assumption. Thus, $l_{i+2} - l_i > B_i$. □

Theorem 5: $N \le 2M/B$.

Proof: Assume *N* is an odd integer.

$$M \ge B_N + l_N - l_1 = B_N + \sum_{i=1}^{(N-1)/2}(l_{2i+1} - l_{2i-1}) > B_N + \sum_{i=1}^{(N-1)/2} B_{2i-1} = \sum_{i=0}^{(N-1)/2} B_{2i+1} \quad (1)$$

$$M > B_{N-1} + l_{N-1} - l_2 = B_{N-1} + \sum_{i=2}^{(N-1)/2}(l_{2i} - l_{2i-2}) > B_{N-1} + \sum_{i=2}^{(N-1)/2} B_{2i-2} = \sum_{i=1}^{(N-1)/2} B_{2i} \quad (2)$$

From (1) and (2), we get

$$2M > \sum_{i=0}^{(N-1)/2} B_{2i+1} + \sum_{i=1}^{(N-1)/2} B_{2i} = \sum_{i=1}^{N} B_i = NB,$$

i.e., $N < 2M/B$. The proof is similar when *N* is an even integer. □

Theorem 6: The expected size of *LAP* is a constant.

Proof: The overlapped buffers can be categorized into two types: the ones left the index overlay and the ones remained after inserting the new buffer β. From Theorem 5, the index size is bounded by 2M/B, so β's arrival removes at most one buffer in the old index list on average. Lemma 4 ensures that any buffer in the index list X_i can only overlap with X_{i-1} and X_{i+1}. Thus on average, there are at most three overlapped buffers found by the new buffer. □

Thus, the distributed algorithm consumes constant amortized time and bounds the index overlay size within *O(M/B)*, where *M* is the media length and *B* is the average buffer length of the peers. For a typical 700MB movie, 15MB buffer ensures the index size is within 100, no matter how large the user population is. If every buffer is equal in size-B, then $\frac{|C^k|}{|OPT|} \le \frac{2M/B}{M/B} = 2$, where *OPT* is the globally optimal solution. Our simulation results show that $|C^k|$ is very close to the optimal size.

C. Overlay Construction, Maintenance, and VCR Operations

This subsection presents how the distributed algorithm is applicable to the index overlay to improve search efficiency. In this paper, we simply reuse the structure implemented using AVL tree or skip-list, which can provide sub-linear search efficiency.

1) Join Operation

When a new client joins the overlay, it first looks for the closest index neighbor in O(log*N*) hops and selects at most *I* partners from the found peer's data neighbors, where *I* is the number of initial partners. Then the new comer finds out the index peers with buffer overlapping by tracing backward and forward along the closest index peer's predecessor and successor. From theorem 6, the expected number of hops to be traced is a constant. At last the dynamic programming algorithm is applied to the found peers plus the new peer to figure out which peers should be pruned from the index overlay. Thus, the expected number of nodes a new client should contact is a constant.

2) Leave Operation and Failure Recovery

When a peer leaves or fails, its neighborship in the index overlay and the data overlay should be adjusted for system resilience. By maintaining a smaller index overlay, BAS reduces the control overhead due to the adjustment of the index overlay.

A peer scheduled to leave the system should first notify its neighbors, such that they can select new partners and re-connect with each other to form new neighboring relations. If the leaving peer is in the index overlay, it chooses one or more non-index neighbors that cover its unique numbers to join the index overlay.

The node failure can be easily detected after several rounds of failed scheduling or buffer information exchange. If the failed node is index peer, the neighbors that detect the failure will find one or more non-index neighbors that cover the failed node's unique numbers to join the index overlay.

Note that the leave or failure operation does not change the index overlay's property: every index peer contains at least one *unique* number. Thus the theorems in last subsection still hold.

3) VCR Operations

In general, the typical VCR operations, e.g., fast-forward or rewind movement, can be implemented with the combination of a leave and re-join operation. However, a common VCR operation may consist of a series of such jump movement, which is not far away from each other, so frequent re-join operations are not efficient. If the index structure supports horizontal shortcuts to other index peers with logarithmic-increasing distances, the VCR operation can jump to the new offset by skipping most unnecessary nodes. The skip-list is one of such examples [7]. Compared with searching from the root or top layer, following the horizontal shortcuts can reduce the search cost significantly. BAS further reduces the number of horizontal hops with a smaller index overlay.

In summary, the BAS scheme applies a distributed pruning algorithm to existing mature indexing structures such that the index overlay size is bounded within a small and stable scale O(*M/B*). A small structure ensures fast response time and low maintenance overhead.

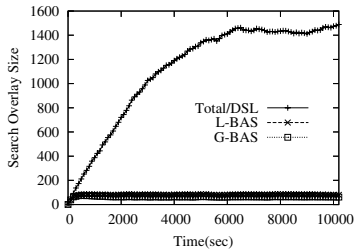

Fig 3. The Index Size for DSL and BAS

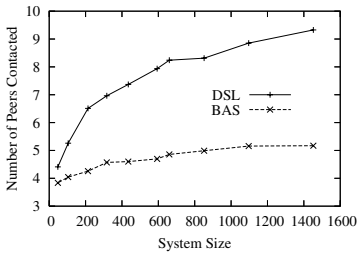

Fig 4. Join/Search cost for DSL and BAS

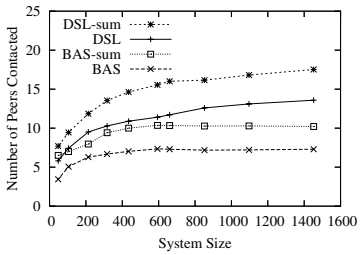

Fig 5. 30%-VCR-jump Cost Comparison

IV. Simulation Results

We evaluate the performance of the proposed VoD system by comparing it with some existing systems.

A. Simulation Configuration

We use the Sprint ISP topology collected by Rocketfuel engine [6] for system setup. It consists of 112 backbone nodes and 242 access nodes; 5 stub nodes are attached to each access node to simulate a LAN. Thus the network size is about 1,500. The default bandwidth settings between two backbone nodes, a backbone and an access node, two access nodes, two stub nodes within the same LAN are 10 Mbps, 5 Mbps, 3Mbps, and 10 Mbps, respectively.

In our simulation, the server and users are located at randomly selected stub nodes. The communication path between any two nodes follows the shortest path. The bit rate of the streaming media is 500Kbps and its length is 2 hours. The segment unit is 1 second, and the default size of the user buffer is 120 segments, i.e., less than 2% of the entire stream. There is no user in the system at the beginning, and users join the system following a Poisson process with mean inter-arrival time of 3 seconds. The start offset of each user is evenly distributed between 0 and 2 hours. Users leave when they play to the end. For each set of configuration, 10 simulation runs have been performed to mitigate the effect of randomness.

B. Search Efficiency and its Control Overhead

We first investigate the search performance for the BAS scheme. For simplicity, only the BAS using skip-list structure is shown in this section. We compare BAS with the DSL [7], which also uses skip-list structure.

Fig. 3 depicts the size of search structure for DSL and BAS during a 10,000 sec simulation. L-BAS represents the case where the distributed algorithm is used, while G-BAS is the size computed by the globally optimal algorithm. Because DSL has to maintain the play progress of all online peers, the size of DSL increases as the system grows with time until the system size keeps relatively stable around 1,400 after 7,000 sec. In contrast, BAS only maintains less than 100 peers regardless of the magnitude of system expansion. The curve of L-BAS is very close to G-BAS, indicating that the performance of our distributed algorithm is close to the optimal.

A small size index structure generates low control overhead and provides fast response time upon node joining or VCR operation. We use the number of peers contacted (peer hops) during an operation as the evaluation metric for control overhead and execution time. Fig. 4 depicts the mean peer hops during the partner search of a node join for both DSL and BAS. In DSL, though the number of peers contacted during search is logarithmic to the system size, about 9 peer hops are required when the system is about 1400. In contrast, BAS requires about 5 peer hops and the search cost is not sensitive to the system expansion, since the size of BAS index overlay is small and stable.

A VCR-jump operation consists of two steps: leave the current neighborhood, and find the partners close to the new play offset. The first step is similar to the common leave, while the second step could be implemented more efficiently than new node join operation if we exploit the horizontal shortcuts supported by the skip-list. Fig. 5 depicts the peer hops of a 30%-VCR-jump operation (jump offset is 30% of the streaming length) in both DSL and BAS. Let "DSL-sum" and "BAS-sum" denote the cases simply combining leave and rejoin operation in DSL and BAS, respectively, while "DSL" and "BAS" denote those exploiting horizontal shortcuts to realize VCR-jump interactions. It can be seen that the VCR-jump cost is lower than the sum of join cost and leave cost. This demonstrates the effect of jumping from current play point using horizontal shortcuts. BAS outperforms DSL due to its smaller size of index overlay, thus generating fewer control messages and providing faster response time.

V. Conclusion

In this paper, we have presented buffer-assisted search (BAS) scheme to improve the search efficiency for P2P-based Video-on-Demand (VoD) services. BAS exploits the redundancy of client buffer coverage to reduce the search time and maintenance cost. Simulation results have shown that our VoD system outperforms existing systems in search efficiency. Our future work is to deploy the experiments on the planet-lab test bed and build a prototype of our VoD system.

References

[1] Y.Cui, B.Li, and K. Nahrstedt, "oStream: asynchronous streaming multicast", *IEEE Journal on Selected Areas in Communications*, 22(1), January 2004.

[2] T. Do, K. A. Hua, and M. Tantaoui, "P2VoD: Providing fault tolerant video-on-demand streaming in peer-to-peer environment", *Proc. IEEE ICC'04*, Paris, June 2004.

[3] Y. Guo, K. Suh, J. Kurose, and D. Towsley, "P2Cast: Peer-to-peer patching scheme for VoD service", *in Proceedings of the 12th World Wide Web Conference (WWW-03)*, Budapest, Hungary, May 2003.

[4] M. Hefeeda, B. Bhargava, and D. Yau, "A hybrid architecture for cost effective on demand media streaming", *Journal of Computer Networks*, 44(3), pages 353-382, 2004.

[5] B. Quinn and K. Almeroth, "IP multicast applications: Challenges and solutions", *Internet Engineering Task Force (IETF) Internet Draft,* work in progress, March 2001.

[6] N. Spring, R. Mahajan, and D. Wetherall, "Measuring ISP Topologies with Rocketfuel", in *Proceedings of ACM SIGCOMM'02*, pages 133–145, August 2002.

[7] D. Wang and J. Liu, "A Dynamic Skip List based Overlay Network for On-Demand Media Streaming with VCR Interactions", *Technical Report*, May 2005.

[8] M. Zhou and J. Liu, "Tree-Assisted Gossiping for Overlay Video Distribution", to appear in *Kluwer Multimedia Tools and Applications*, 2005.