oStream: Asynchronous Streaming Multicast in Application-Layer Overlay Networks

Yi Cui, Baochun Li and Klara Nahrstedt

Abstract

Existing multicast-based streaming solutions are based on the existence of IP multicast, as well as the assumption of sequential access patterns from the clients. In order to accommodate non-sequential access patterns and alleviate server load when the requests are bursty targeting "hotspots", we take advantage of the strong buffering capabilities of end hosts in application-layer overlay networks, and propose a novel overlay multicast strategy, *oStream*, to perform streaming multicast on application overlay networks. We have performed extensive analysis and performance evaluation with respect to the scalability and the efficiency of oStream. With respect to the required server bandwidth, we show that oStream defeats the theoretical lower bound of traditional multicast-based approaches, under both sequential and non-sequential access models. oStream is also robust against bursty requests. With respect to bandwidth consumption on the backbone network, we show that the benefit introduced by oStream overshadows the topological inefficiency (e.g., link stress and stretch) introduced by using application overlay multicast.

I. Introduction

The undisputed popularity of on-demand multimedia distribution in current and next-generation Internet applications has stimulated extensive research work to ensure its scalability and efficiency, especially when a large number of clients are involved and dispersed over wide-area broadband networks. Scalable solutions have been proposed for the purpose of multimedia streaming. As an example, with *Hierarchical Stream Merging*, the most efficient streaming strategy based on IP multicast, it has been shown that [1] server bandwidth only increases logarithmically with respect to rate of new client requests. Existing work has assumed that: (1) existing support for IP multicast is present; (2) the access patterns of clients are highly sequential (*i.e.*, most media streams are played back from the beginning to the end); and (3) the variations of client request rate is stable and predictable over time.

However, recent studies on the streaming access workloads [6] have revealed that (1) client access is, in fact, non-sequential; and (2) the request rate is bursty for 'hotspots', which may be accessed by a 'flash crowd' drawn by an event of widespread interest. It is further revealed that with non-sequential client access patterns, and for any multicast-based streaming strategies, scalability is not as satisfactory as the case of sequential access. Other studies also show that if the client load overwhelms the designed server bandwidth capacity by a small fraction (above 10%), the system performance will degrade dramatically [7]. These findings suggest that IP-based multicast streaming may not be the panacea to achieve scalability under realistic problem settings.

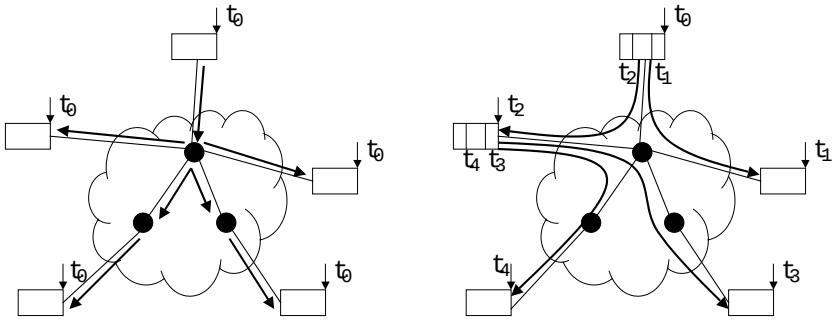
The problem is further exacerbated if we consider the fact that support for IP multicast is still severely limited today and in the foreseeable future. Application-layer multicast in overlay networks is proposed [8][9][11][13] due to the absence of IP multicast. The main idea is to build multicast trees in application-layer overlays, and to minimize the topological overhead introduced by the sub-optimality of the trees, compared with the corresponding IP multicast trees. Though the results are encouraging, very few previous work has explored the feasibility of using application-layer multicast trees to perform on-demand media streaming. One unique problem that distinguishes such work from existing application-layer multicast approaches is the asynchronous arrivals and arbitrary durations of client requests. Further, the requests are for different segments of the media stream, if non-sequentiality is assumed. This is in conflict with the synchronous nature of multicast, where the source pushes data to all receivers simultaneously. Traditional IP-multicast based solutions address this conflict by repeating the same content on multiple multicast channels over time. Asynchronous clients are either enforced to be synchronized at the price of service delays, or required to participate in several multicast sessions at the same time.

We argue that this is not necessarily the case in the context of overlay networks. In fact, we should leverage the temporal correlation of asynchronous requests to resolve the conflicts between the asynchronous nature of arrivals and the synchronous nature of multicasts. The key to our solution is *cooperative buffering* on end hosts. By enabling data buffering on the relaying nodes in an application-layer multicast tree, requests at different times can be satisfied by the same stream, thus achieving efficient media delivery. Based on this foundation, we propose *oStream*, an application-layer asynchronous streaming multicast mechanism to achieve the goals of scalability and efficiency under realistic assumptions. While impossible in IP multicast since the buffering capabilities of routers are limited, asynchronous streaming multicast can be realized in application-layer overlays, where relaying nodes are end

This work was supported by the National Science Foundation under contract number 9870736, the Air Force Grant under contract number F30602-97-2-0121, National Science Foundation Career Grant under contract number NSF CCR 96-23867, NSF CISE Infrastructure grant under contract number NSF EIA 99-72884, and NASA grant under contract number NASA NAG 2-1250.

Yi Cui and Klara Nahrstedt are affiliated with the Department of Computer Science, University of Illinois at Urbana-Champaign. Their email addresses are {*yicui, klara*}*@cs.uiuc.edu.*

Baochun Li is affiliated with the Department of Electrical and Computer Engineering, University of Toronto. His email address is *bli@eecg.toronto.edu*.



(a) IP multicast (b) Asynchronous multicast

Fig. 1. Conceptual comparison between IP multicast and asynchronous multicast

hosts with strong buffering capabilities. The main contributions of introducing oStream include the following favorable properties, supported and verified by extensive analytical and experimental results:

(1) *Scalability:* We derive the required server bandwidth in oStream, which defeats the theoretical lower bound of traditional multicast-based approaches, under both sequential and non-sequential access models. This may be achieved when we allow each relaying node in the multicast tree to buffer at most 10% of the media streams. Furthermore, over a certain threshold, the required server bandwidth no longer increases as the request rate grows, which suggests the robustness of oStream against 'flash crowds'.

(2) *Efficiency:* In previous works, server bandwidth has been used as the sole metric to evaluate system scalability and performance. However, bandwidth consumption on the backbone network for any streaming scheme has not been evaluated. This issue is of particular interest due to the following conjecture: although overlay networks inevitably introduce topological inefficiency (link stress and stretch) compared to IP multicast, the benefit introduced by asynchronous streaming multicast in oStream may overshadow such inefficiency. Towards analyzing the efficiency of oStream, we investigate this conjecture analytically, the results of which have been confirmed by our experiments.

The remainder of the paper is organized as follows. Sec. II compares oStream with related work. Sec. III presents the temporal dependency model. Sec. IV presents the asynchronous multicast algorithm. We extensively analyze the scalability and efficiency of oStream in Sec. V and VI, respectively. Sec. VII evaluates the performance of oStream and verify our analytical results. Finally, Sec. VIII concludes the paper.

II. Related Work

A. Application Overlay Multicast

Application-layer overlay multicast originates from the study on end-system multicast by Chu *et al.* [8], which is proposed as a deployable alternative to IP multicast. In their work, end hosts self-organize into an overlay multicast tree using a fully distributed protocol. End hosts within the overlay relay data to each other via unicast. This approach has been proved successful for supporting small to medium sized conferencing applications in dynamic and heterogeneous network environments. Similar ideas have been applied into other areas such as content distribution [9][10] and network service [11][12]. Recently, hierarchical end system multicast [13] is also proposed to make the application-layer overlay scalable at supporting multicast applications of with a larger number of participating hosts.

With respect to link cost and bandwidth efficiency, application-layer overlay multicast may not be as efficient as IP multicast, since data are replicated at end hosts, rather than routers in the backbone. Therefore, it becomes an important criterion to evaluate whether an overlay approach can minimize this factor of inefficiency compared to IP multicast. In this paper, we continue to use the link cost to evaluate the performance of oStream, especially answering the question on if the benefits of oStream may overshadow such topological inefficiency, inherent in overlay multicast.

B. Multicast-based On-Demand Media Distribution

The problem of delivering high-quality multimedia stream to asynchronous clients have been well studied. To achieve system scalability especially on the server side, IP multicast is widely adopted to serve multiple clients with one single server stream. However, the asynchrony of client requests is in conflict with the nature of multicast, which was originally designed to support applications of synchronous data transmission, such as conferencing. Various solutions have been proposed to address this conflict. In *batching* [2], client requests from different times are aggregated into one multicast session. However, users have to suffer long playback delay since their requests are enforced to be synchronized. The approaching of *patching* [3] attempts to address this problem by allowing clients to "catch up" with an ongoing multicast session and patch the missing starting portion via server unicast. With *merging* [1], a client repeatedly merge into larger and larger multicast sessions. In periodic broadcasting [4], the server separates a media stream into segments and periodically broadcasts them through different multicast channels, from which a client chooses to join.

These solutions largely fall into two categories: true on-demand or immediate services, and near on-demand services. Solutions in the first category (patching and merging) serves the client immediately once the request is issued. For solutions in the second category (batching and periodic broadcasting), a client has to wait for a bounded delay time. In this paper, we only consider the true on-demand media distribution solutions. However, it is worth pointing out that the near on-demand solutions are in fact not superior than on-demand solutions at saving system cost. Eager *et al.* [1] reveal that using the approach of merging, the server bandwidth consumption grows at least logarithmically as the request rate increases. They also reveal that in periodic broadcasting, the server bandwidth requirement grows at least logarithmically as one tries to shorten the service delay bound. Therefore, the scale factors of both approaches are the same. Jin *et al.* [5] further confirms that this conclusion holds when the clients' media access patterns are non-sequential.

C. Media Caching and Buffering

Besides multicast, an orthogonal technique for reducing server loads is media caching. A large body of work in this area includes proxy caching. These works are similar to the proxy-based web caching in that they both use proxies to serve clients on behalf of the server, if the requested data is cached locally. However, since the media objects tend to be of large sizes, a proxy usually caches only a portion of the object. There are different ways of partial caching, such as prefix-based caching [14], and segment-based caching [15][16]. Besides proxy caching, client-side caching is also proposed, such as chaining [17] and interval caching [18].

It is well observed that multicast and caching can help reduce the server load in media distribution. In addition, we believe that they also have the potential capability of reducing bandwidth consumption, thus increasing efficiency. In the case of multicasting, aggregating n clients into one multicast group can conserve more link bandwidth than issuing n unicast sessions from the server to clients. In the case of caching, if a client can stream data from a proxy or client in its proximity rather than from a remote server, bandwidth can also be conserved. While previous works have extensively evaluated the server bandwidth reduction of different solutions, their performance at reducing link cost and increasing bandwidth efficiency largely remains uninvestigated. The only work we are aware of is by Jin *et al.* [21], which analyzes the link cost of a client-based caching approach and shows its scalability. In this paper, we seek to evaluate the performance of multicast-based and cache-based solutions with respect to reducing link cost via in-depth analysis and extensive experiments. Our study is conducted under different network settings, including IP multicast and application overlay. We also investigate the impact of user access patterns (sequential or non-sequential) on the performance of both approaches.

III. oSTREAM: PRELIMINARIES

A. Temporal Dependency Model

Consider two asynchronous requests R_i and R_j for the same media object. The size of the object is T bytes. The object is played back at a constant bit rate of one byte per unit time. Therefore, the playback time of the object is also T time units. R_i starts at time s_i from the offset o_i. R_j starts at time s_j from the offset o_j. With respect to these *offsets*, the unit is *bytes*. Since the playback rate is one byte per unit time, the unit is equivalently *time units* as well. This is convenient to compare these offsets with time instants (e.g., s_i and s_j). The requests may last for a specific period of time, and the time lengths of R_i and R_j are l_i and l_j. We define R_i as the *predecessor* of R_j, R_j as the *successor* of R_i (denoted as $R_i \prec R_j$), if the following requirements are met:

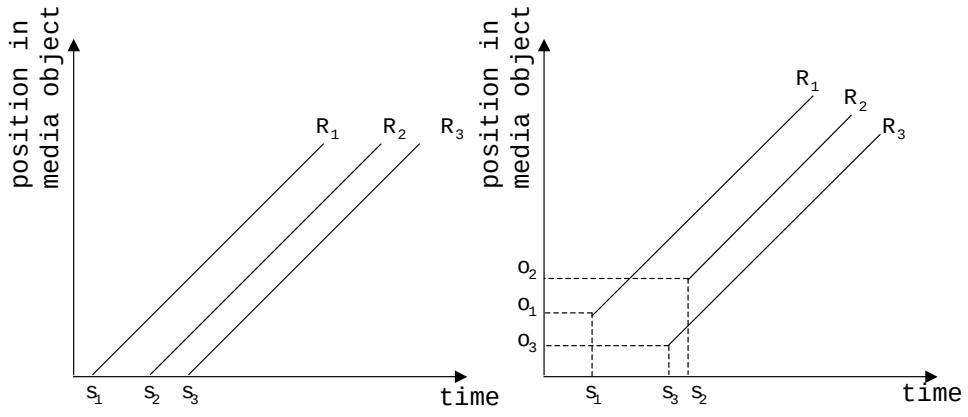
$$\begin{cases} o_i + l_i > o_j & \text{if} \quad o_i \leq o_j \\ o_j + l_j > o_i & \text{if} \quad o_i > o_j \end{cases} \tag{1}$$

$$s_j - o_j > s_i - o_i \tag{2}$$

Inequality (1) demands that the media data requested by R_i and R_j must (partially) overlap. Inequality (2) means that R_i must retrieve the data before R_j does.

Fig. 2 (a) and (b) show examples of temporal dependencies under sequential and non-sequential access patterns. In both figures, $R_1 \prec R_2 \prec R_3$. Notice that in case of sequential access, $o_1 = o_2 = o_3 = 0$. From the definition, we may observe that R_1 has the potential to benefit R_2 in that, R_2 could (partially) reuse the stream from R_1 rather than obtaining it directly from the server. Note that in Fig. 2(b), R_2 is the predecessor of R_3, even though R_3 starts earlier than R_2. This case will not occur when all requests observe sequential access patterns (Fig. 2(a)).

We claim that all on-demand media streaming algorithms could be described based on the temporal dependency model. In all algorithms, R_2 utilizes its predecessor/successor relation with R_1 to conserve server bandwidth, only with different ways of reusing the stream from R_1.



(a) Sequential access (b) Non-sequential access

Fig. 2. Temporal dependency model

Symbol	Defnition
$R_i \prec R_j$	request R_i is the predecessor of request R_j, R_j is the successor of R_i
s_i	starting time of R_i
o_i	starting offset of R_i
l_i	time length (duration) of R_i

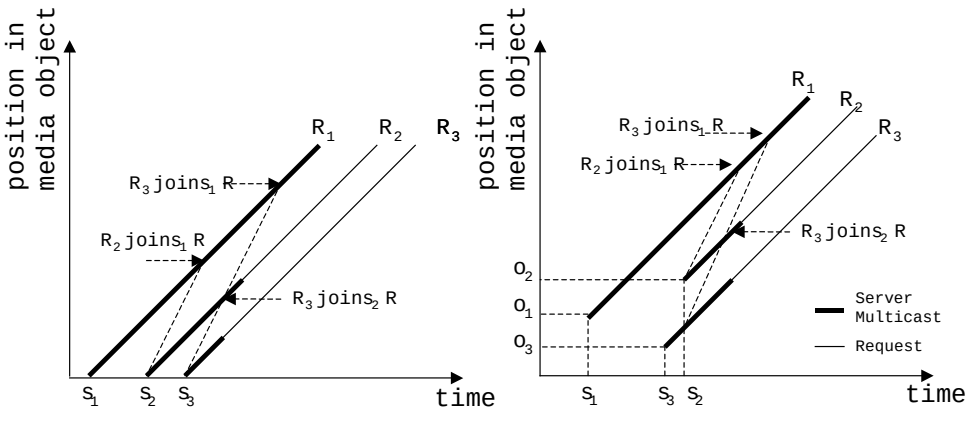
TABLE I
NOTATIONS USED IN SEC. III

B. Hierarchical Stream Merging: a Review

We first review the hierarchical stream merging (HSM) algorithm. In [1], Eager *et al.* point out that HSM can achieve the theoretical lower bound of server bandwidth consumption for all multicast-based on-demand streaming algorithms, if the client has unlimited receiving bandwidth and buffering space. In this paper, we evaluate HSM based on these optimal yet unrealistic assumptions. Our purpose here is to establish a theoretical baseline, against which our proposed algorithm can be evaluated and compared.

We illustrate the algorithm using the requests shown in Fig. 2(a) and (b) as examples. In Fig. 3(a), since R_1 has no predecessor, it opens a multicast session from the server to retrieve the requested data. R_2 initiates another multicast session upon its arrival. Meanwhile, it also listens to the session of R_1 to prefetch data from R_1. R_2 stays in both sessions until the point when the prefetched data starts to get played. From this point on, R_2 continues to prefetch data from R_1's session and withdraws from its own session. We claim that R_2 "joins" R_1 at this point. Similarly, R_3 initially opens its own multicast session and retrieves data from sessions of both R_1 and R_2 at the same time. It then withdraws from its own session and joins R_2 at the point when the data prefetched from R_2 may be used. Finally, R_3 joins R_1. Within the algorithm, each request repeatedly joins the ongoing multicast session of its closest predecessor until it catches up with the earliest one. The initial missing portion is offered by opening a "make-up" session from the server. Similar results are obtained in the non-sequential case, illustrated in Fig. 3(b).

As shown in the figures, the multicast sessions initially opened by R_2 and R_3 are terminated once all their members are merged



(a) Sequential access (b) Non-sequential access

Fig. 3. Examples of hierarchical stream merging

into R_1's session. The requirement on server bandwidth is thus significantly reduced, compared to opening three separate unicast sessions to accommodate each request. Obviously, to reuse the stream from R_i, R_j must satisfy $s_i + l_i > s_j$, in addition to Inequality (1) and (2).

C. Asynchronous Multicast

We proceed to propose and illustrate the concept of *Asynchronous Multicast* (AM), again using the example in Fig. 2. We assume that each request is able to buffer the streamed data for a certain amount of time after playback. This can be achieved by using a circular buffer to cache the stream. For example, as shown in Fig. 4(a), R_1 has a buffer capable of storing data for time length W_1. In other words, any data cached in the buffer is kept for a time length of W_1, after which it is replaced by new data. Obviously, all requests that fall within this window may potentially be served by R_1. In this case, R_2 directly retrieves its stream from the buffer of R_1. R_3 is unable to stream from R_1 since it falls outside the buffer window of R_1. Instead, it streams from the buffer of R_2. The only difference in the case of non-sequential access (Fig. 4(b)) is that, R_3 needs to stream from the server for the initial portion, which is not available at R_2.

This mechanism is referred to as *asynchronous multicast*, mainly because it needs only one server stream to serve a group of requests, similar to traditional multicasts. However, the difference is that members within the group receive data asynchronously. An important feature of asynchronous multicast is that it is purely end-host based. For example, in Fig. 4(a), R_1 streams from the server via unicast. R_2 also uses unicast connection to retrieve data from R_1. Another major difference is that in our approach, although a request may have multiple sources, the streaming from different sources is sequentialized. For example, in Fig. 4(b), R_3 first streams from the server, then switches to R_2. It keeps only one live connection throughout its entire session. In HSM, a request needs to stream from multiple sources in parallel.

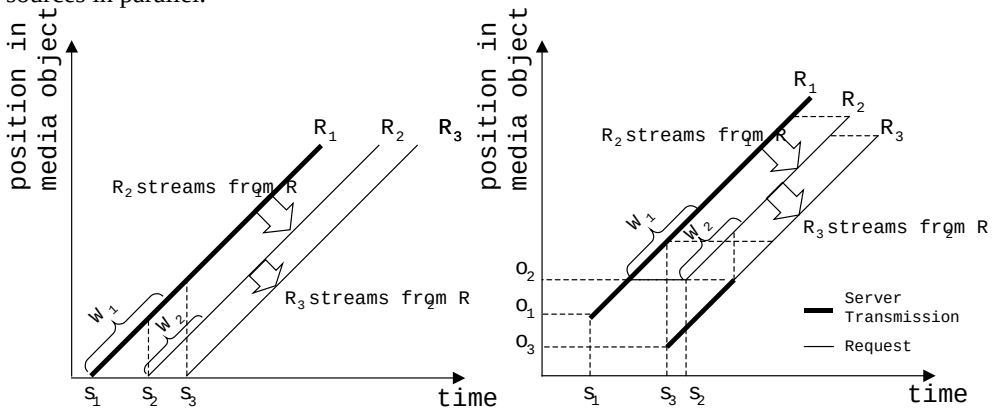


Fig. 4. Asynchronous multicast

To summarize, in our algorithm, in order to reuse stream from R_i, R_j must meet the following requirement besides those listed in Inequality (1) and (2):

$$(s_j - o_j) - (s_i - o_i) < W_i \tag{3}$$

$(s_j - o_j) - (s_i - o_i)$ is also referred to as the *buffer distance* between R_i and R_j. If Inequality (3) is satisfied, we further define R_i as the *buffer-constrained predecessor* of R_j, R_j as the *buffer-constrained successor* of R_i, denoted as $R_i \overset{W_i}{\prec} R_j$. Henceforth in this paper, we simply refer to R_i as the predecessor of R_j, and R_j as the successor of R_i.

IV. oSTREAM: ALGORITHMS

In this section, we present algorithms of asynchronous multicast. The algorithms are fully distributed and operate in application-layer overlay networks. In Sec. IV-A, we first formulate the problem of on-demand media distribution as a graph theoretic problem. We then present the basic algorithm in Sec. IV-B. Finally, we revise the basic algorithm to accommodate practical issues discussed in Sec. IV-D. We summarize notations used in this section in Table II.

A. Problem Formulation

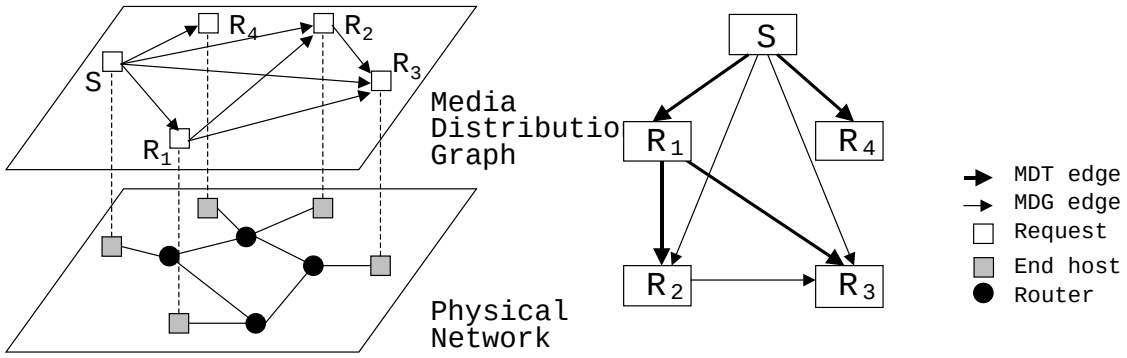
Definition 1 (Media Distribution Graph). Let $\mathbb{R}$ be a set of asynchronous requests. We define the *Media Distribution Graph* (MDG) for $\mathbb{R}$ as a directed weighted graph $MDG_{\mathbb{R}} = (\mathbb{R}, E)$, such that $E = \{(R_i, R_j) \mid R_i \overset{W_i}{\prec} R_j, R_i, R_j \in \mathbb{R}\}$. Each edge (R_i, R_j) has the weight $c(R_i, R_j)$, which is the transmission cost between two end hosts carrying R_i and R_j, respectively.

Symbol	Definition
$\mathbb{R}$	a set of asynchronous requests
$R_i \overset{W_i}{\prec} R_j$	R_i is the buffer-constrained predecessor of R_j
$MDG_{\mathbb{R}} = (\mathbb{R}, E)$	media distribution graph for $\mathbb{R}$
$MDT_{\mathbb{R}} = (\mathbb{R}, E^T)$	media distribution tree for $MDG_{\mathbb{R}}$ ($E^T \in E$)
$c(R_i, R_j)$	weight of edge $(R_i, R_j) \in E$, the transmission cost between R_i and R_j
$MDG_{ins} = (\mathbb{R} \cup R_{ins}, E_{ins})$	resulting graph after R_{ins} is added to $MDG_{\mathbb{R}}$
$MDT_{ins} = (\mathbb{R} \cup R_{ins}, E^T_{ins})$	media distribution tree for MDG_{ins} ($E^T_{ins} \in E_{ins}$)
$MDG_{del} = (\mathbb{R} - R_{del}, E_{del})$	resulting graph after R_{del} is deleted from $MDG_{\mathbb{R}}$
$MDT_{del} = (\mathbb{R} - R_{del}, E^T_{del})$	media distribution tree for MDG_{del} ($E^T_{del} \in E_{del}$)

TABLE II
NOTATIONS USED IN SEC. IV

We use an example to illustrate the concept of MDG. Consider a set of requests $\mathbb{R} = \{R_1, R_2, R_3, R_4\}$. $R_1 \overset{W_1}{\prec} R_2$, *i.e.*, R_2 falls within the buffer window of R_1. Similarly, we have $R_1 \overset{W_1}{\prec} R_3$ and $R_2 \overset{W_2}{\prec} R_3$. The MDG for $\mathbb{R}$ is shown in Fig. 5(a). Each MDG node represents a request[1]. An edge directed from node R_1 to R_2 means that R_1 is the predecessor of R_2, *i.e.*, R_2 could reuse the media stream from R_1. $c(R_1, R_2)$ is the transmission cost between two end hosts holding R_1 and R_2, respectively. A special node S represents the server. Since the server can serve any request, it can be regarded as the predecessor of all members in $\mathbb{R}$. Thus, S has directed edges to all other nodes in the graph.

The media distribution graph changes dynamically: (1) when a new request arrives, a new node is inserted into the graph; (2) when a request is terminated, its corresponding node is removed from the graph. As such, MDG represents the dependencies among all asynchronous requests, which forms a virtual overlay on top of the physical network. The weight of edges in MDG reflects the communication cost between end hosts holding the requests.



(a) Media distribution graph (b) Media distribution tree

Fig. 5. Illustration of the media distribution graph

Given the definition of MDG, now we can formulate the problem of on-demand media distribution as the issue of constructing and maintaining a spanning tree on MDG, defined as the *Media Distribution Tree*.

Definition 2 (Media Distribution Tree). For a request set $\mathbb{R}$ and its MDG $MDG_{\mathbb{R}} = (\mathbb{R}, E)$, the corresponding MDT is denoted as $MDT_{\mathbb{R}} = (\mathbb{R}, E^T)(E^T \in E)$, and it is a *spanning tree* on MDG. For two nodes $R_i, R_j \in \mathbb{R}$, if $(R_i, R_j) \in E^T$, then R_i is the *parent* of R_j, R_j is the *child* of R_i.

Given a MDG, the optimal solution for MDT, *i.e.*, to minimize the overall transmission cost of media distribution, is to find the *Minimal Spanning Tree* (MST) on MDG. An example is shown in Fig. 5(b).

B. Basic Algorithm: MDT Operations

We proceed to present our basic algorithm on MDT construction and maintenance. Our algorithm is fully distributed and incremental. First, we do not assume the existence of a centralized manager to control the tree construction. Second, each new request joins the tree based on its local decision, which only needs partial knowledge of the existing tree. Third, upon request departure, the tree is quickly recovered since no global reorganization is required.

[1]We use terms request and node interchangeably in the remainder of this section, depending on the context.

The algorithm is executed each time when a new request joins the graph, or when an existing request departs. To tackle problems in both cases, our algorithm is split into two operations: **MDT-Insert** and **MDT-Delete**.

We first present **MDT-Insert**. Let $MDG_{ins} = (\mathbb{R} \cup R_{ins}, E_{ins})$ be the resulting graph after the request R_{ins} is added to $MDG_{\mathbb{R}}$, **MDT-Insert** is able to return $MDT_{ins} = (\mathbb{R} \cup R_{ins}, E^T_{ins})$ as the new MDT for MDG_{ins}.

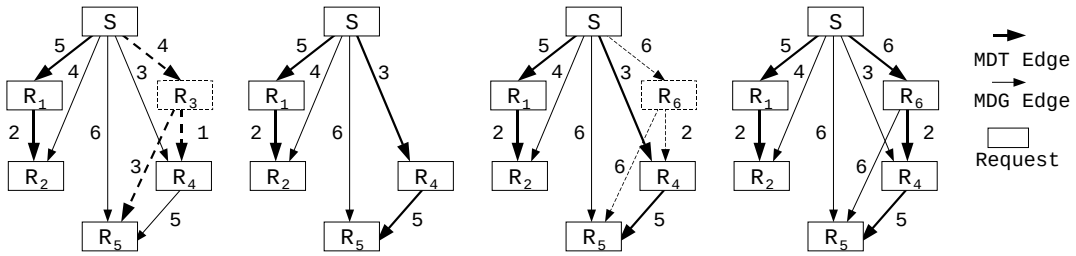
MDT-Insert$(R_{ins}, E_{ins}, E^T, E^T_{ins})$
/* From all its predecessors, R_{ins} finds parent R_{min}, whose transmission cost to R_{ins} is minimal */
1 $R_{min} \leftarrow \arg\min\{c(R_{pre}, R_{ins}) \mid (R_{pre}, R_{ins}) \in E_{ins}\}$
2 $E^T_{ins} \leftarrow E^T \cup (R_{min}, R_{ins})$

/* For all its successors R_{suc}, R_{ins} compares if the transmission cost from itself to R_{suc} is less than from R_{suc}'s current parent R_{par}. If so, R_{suc} is asked to switch parent to R_{ins}. */
3 **for each** $R_{suc} \in \{R_{suc} \mid (R_{ins}, R_{suc}) \in E_{ins}\}$
4 **if** $c(R_{ins}, R_{suc}) < c(R_{par}, R_{suc}) \mid (R_{par}, R_{suc}) \in E^T$
5 **do** $E^T_{ins} \leftarrow (E^T_{ins} - (R_{par}, R_{suc})) \cup \{(R_{ins}, R_{suc})\}$

Second, we present **MDT-Delete**. Let $MDG_{del} = (\mathbb{R} - R_{del}, E_{del})$ be the resulting graph after R_{del} is removed from $MDG_{\mathbb{R}}$, **MDT-Delete** is able to return $MDT_{del} = (\mathbb{R} - R_{del}, E^T_{del})$ as the new MDT for MDG_{del}.

MDT-Delete$(R_{del}, E_{del}, E^T, E^T_{del})$
/* R_{del} deletes the tree edge from its parent R_{par} */
1 $E^T_{del} \leftarrow E^T - \{(R_{par}, R_{del}) \mid (R_{par}, R_{del}) \in E^T\}$
2 **for each** $R_{chi} \in \{R_{chi} \mid (R_{del}, R_{chi}) \in E^T\}$
do
/* R_{del} deletes the tree edge to each of its children R_{chi} */
3 $E^T_{del} \leftarrow E^T_{del} - (R_{del}, R_{chi})$
/* From all its predecessors, R_{chi} finds the new parent R_{min}, whose transmission cost to R_{chi} is minimal */
4 $R_{min} \leftarrow \arg\min\{c(R_{pre}, R_{chi}) \mid (R_{pre}, R_{chi}) \in E_{del}\}$
5 $E^T_{del} \leftarrow E^T_{del} \cup (R_{min}, R_{chi})$

We use an example to illustrate these two algorithms. In Fig. 6 (a), when R_3 leaves, it first deletes itself from the MDT (Lines 1 to 2 in **MDT-Delete**), then notifies its children R_4 and R_5 to find their new parents as S and R_4, respectively (Lines 3 to 5 in **MDT-Delete**). In Fig. 6 (b), when a new request R_6 joins, it first finds S as its parent (Lines 1 to 2 in **MDT-Insert**), then notifies its successor R_4 to switch parent from S to R_6, since $c(R_6, R_4) < c(S, R_4)$ (Lines 3 to 5 in **MDT-Insert**).



(a) Before R_3 is Deleted (b) After R_3 is Deleted (c) Before R_5 is Inserted (d) After R_5 is Inserted

Fig. 6. Illustration of the Media Distribution Graph

C. Algorithm Properties

We prove the correctness and optimality of the above algorithms as follows.

Correctness

Lemma 1: MDG is directed acylic graph (DAG).

Proof: By Inequality (2), $R_i \prec R_j$ if and only if $s_i - o_i < s_j - o_j$. Therefore, a loop $R_i \rightarrow \ldots \rightarrow R_k \rightarrow R_i$ in $MDG_{\mathbb{R}}$ would mean that $s_i - o_i < \ldots s_k - o_k < s_i - o_i$, which forms contradiction. Thus $MDG_{\mathbb{R}}$ is a DAG. □

Theorem 1 (Correctness). Both **MDT-Insert** and **MDT-Delete** generate loop-free spanning trees.

Proof: At line 2, **MDT-Insert** adds one edge (R_{min}, R_{ins}) to the original tree. On the contrary, **MDT-Delete** deletes one edge at line 1. For the rest operations, both algorithms only replace old edges with new edges sharing the same destination. Therefore, the inbound degree of these nodes remains unchanged as in the old spanning tree. To this end, we conclude that both algorithms ensure the newly formed tree to have $|\mathbb{R}| - 1$ edges. Moreover, every node except S in the tree is ensured to have one and only one inbound edge. Thus the tree must cover every node of $MDG_{\mathbb{R}}$. □

Optimality

Lemma 2: Given $MDG_{\mathbb{R}}$ and its MDT $MDT_{\mathbb{R}}$, which the MST of $MDG_{\mathbb{R}}$. If a node and the edges incident to it are deleted from $MDG_{\mathbb{R}}$, or an edge is deleted from $MDT_{\mathbb{R}}$, then correspondingly $MDT_{\mathbb{R}}$ breaks into several subtrees. Each subtree is a MST.

Proof: By **Lemma 1**, a MDG is a DAG. Therefore, a tree on a MDG is the MST if it satisfies that: (1) except for the source node, each node R_i has one and only edge directed to itself, and (2) this edge is the smallest weighted one of all edges directed to R_i. These properties still hold for all subtrees MDT_0 through MDT_n. Therefore, they remain to be MSTs. □

Let G_{del} be the resulting graph after the deletion of a node from $MDG_{\mathbb{R}}$ or the deletion of a tree edge from $MDT_{\mathbb{R}}$. Let $T_0 = (\mathbb{R}_0, E_0^T)$, ..., $T_n = (\mathbb{R}_n, E_n^T)$ be the resulting subtrees of $MDT_{\mathbb{R}}$ (T_0 through T_n are sorted according to the predecessor/successor order of their root nodes, i.e., $root(T_0) \prec \ldots \prec root(T_n)$). These subtrees can be connected into a spanning tree according to the following method.

For each T_k ($k = 1, \ldots, n$), among all edges in G_{del}, which are directed to $root(T_k)$, find out the smallest weighted one and attaches it to T_k. This will result in a spanning tree T_{del} of G_{del}.

Lemma 3: T_{del} is the MST of G_{del}.

Proof: We first show that only edges destined to $root(T_k)$ ($k \neq 0$) are qualified to be the candidate MST tree edges besides those ones in T_k. We exclude other cases one by one. First, for each T_k ($k = 0, \ldots, n$), any edge $e \in \{(R_i, R_j) \mid R_i, R_j \in \mathbb{R}_k, (R_i, R_j) \notin E_k^T\}$ is not a tree edge of T_{del}. Otherwise, it will contradict with the fact that T_k is MST, which is proved in **Lemma 2**. Second, consider an edge e destined to a non-root node $v \in T_k$ from another subtree. If e is a tree edge of T_{del}, then e must have a smaller weight than the current edge pointing to $v \in T_k$. If so, e should have appeared in $MDT_{\mathbb{R}}$ and remained in T_k. This again contradicts the fact that T_k is MST. Finally, edges pointing to $root(T_0)$ do not exist since $root(T_0) = S$.

To this end, we conclude that each $root(T_k)$ still remains in the MST of G_{del}. To finish the entire MST, we only need to connect the roots of these subtrees together. The method to construct T_{del} (presented before **Lemma 3**) is guaranteed to return a tree satisfying the properties of a MST over MDG, which are listed in the proof of **Lemma 2**. Thus, T_{del} is the MST of G_{del}. □

Theorem 2 (Optimality). The trees returned by **MDT-Insert** and **MDT-Delete** are MSTs.

Proof: Proof for **MDT-Delete** can be directly achieved from **Lemma 3**, since MDG_{del} is the resulting graph of $MDG_{\mathbb{R}}$ after the removal of R_{del} and **MDT-Delete** implements the method presented before **Lemma 3**.

To prove **MDT-Insert**, we first remove from $MDT_{\mathbb{R}}$ those tree edges in the set $O_{ins} = \{(R_i, R_j) \mid (R_{ins}, R_j) \in E_{ins}\}$. This will result in several subtrees of $MDT_{\mathbb{R}}$, namely T_0, ..., T_n ($root(T_0) \prec \ldots \prec root(T_n)$). Except T_0, all other subtrees are rooted at different R_j. Note that R_{ins} is also a one-node tree rooted at itself. Now we add edges in O_{ins} as well edges in $N_{ins} = \{(R_{ins}, R_j) \mid (R_{ins}, R_j) \in E_{ins}\}$ back to the old graph. All edges in $O_{ins} \cup N_{ins}$ are from one subtree to the root node of another. According to **Lemma 3**, they are qualified as the candidate MST tree edges to be added on. Now we can use the same method presented before **Lemma 3** to construct MST. **MDT-Insert** implements this method in two steps. According to their predecessor/successor order, The subtrees are sorted as T_0, R_{ins}, T_1, ..., T_n. Therefore, the algorithm first finds the smallest weighted edge connected to R_{ins}, as done in lines 1-2. Line 3-5 find the smallest edges for the rest subtrees T_k ($k = 1, \ldots, n$). Since the existing edge pointing to $root(T_k)$ is already the smallest one in $MDG_{\mathbb{R}}$, we only need to compare it with the newly added edge $(R_{ins}, root(T_k))$ to determine which is smaller. □

D. Practical Issues

Content Discovery Service

The MDT algorithms require that each request must have knowledge of all its predecessors and successors. There a publish/subscribe service needs to be in place for the purpose of information exchange and update. Intuitively, we could use the server as a centralized manager to keep track of information of all active requests. In this way, each new request R_{ins} must register itself to the server before joining. The server then feedbacks R_{ins} with information of all its predecessors and successors. Then R_{ins} is able to run the **MDT-Insert** algorithm. When a request R_{del} leaves, it also needs to deregister itself from the server, before running **MDT-Delete**.

This solution obviously suffers from drawbacks of all centralized approaches. Traditional static content-based discovery solutions do not fit here, mainly for the following reasons. First, different to other caching schemes, where a content of a buffer is fixed, we allow the content of a buffer to be time varying. Second, each buffer is associated with its corresponding request, which has

a lifetime. Therefore, events of buffer birth/death and buffer content changing will constantly invalidate the content availability information, which results into higher volume of update messages.

To address these problems, we should leverage the temporal dependencies among different requests. Our solution is summarized as follows. (1) We choose a number of end hosts as discovery servers. Each server has a unique ID. (2) For each request R_i trying to register to the discovery service, its call is directed to a subset of discovery server, which will keep the record of R_i until R_i removes itself. This subset is determined by mapping the timing information of R_i into a set of set of server IDs. (3) Likewise, if R_i tries to query its successors or predecessors, its call is directed to a subset of discovery servers, which keep the record of all successors and predecessors of R_i. This subset is also determined through timing information mapping for R_i. (4) The subsets in (2) and (3) contain constant number of servers. Thus the message overhead for the above update or query operations is constantly bounded.

We use $N = \{h_0, h_1, \ldots, h_{n-1}\}$ to denote all discovery servers. If R_i needs to register to the discovery service, it first calls the hashing function $r(R_i)$ to return a subset of discovery servers. Then R_i sends update messages to these servers, which will keep its record. Similarly, function $p(R_i)$ returns a subset of servers, to which R_i sends the query messages about its predecessors. Function $s(R_i)$ is designed the same way for successor queries.

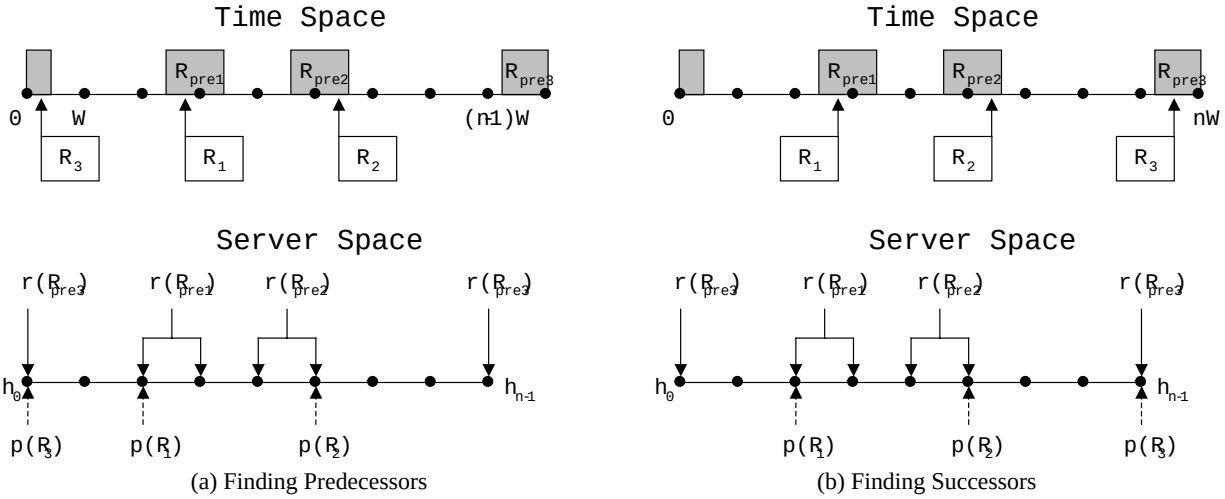
Let W be the buffer size, n the number of discovery server, s_i the starting time of R_i, o_i the starting offset of R_i, the hashing functions are specified as follows.

$$r(R_i) = \{h_m, h_{(m+1) \bmod n}\} \quad \text{if } m \cdot W \leq (s_i - o_i) \bmod (n \cdot W) < (m+1) \cdot W \tag{4}$$

$$p(R_i) = \{h_m\} \quad \text{if } m \cdot W \leq (s_i - o_i) \bmod (n \cdot W) < (m+1) \cdot W \tag{5}$$

$$s(R_i) = \{h_m\} \quad \text{if } m \cdot W \leq (s_i - o_i + W) \bmod (n \cdot W) < (m+1) \cdot W \tag{6}$$

In Fig. 7 (a), we illustrate the relationship between function r and p in three cases. In particular, R_{pre1} is the predecessor of the request R_1, R_{pre2} is the predecessor of the request R_2, R_{pre3} is the predecessor of the request R_3. In Fig. 7 (b), we illustrate the relationship between function r and s in three cases. In particular, R_{suc1} is the predecessor of the request R_1, R_{suc2} is the predecessor of the request R_2, R_{suc3} is the predecessor of the request R_3.



(a) Finding Predecessors (b) Finding Successors

Fig. 7. Illustration of Hashing Functions

From the figure we can see, the update message is directed to two consecutive discovery servers, thus the request update overhead is 2. For both query operations (predecessors and successors), the message overhead is 1.

Simplifying Session Switching

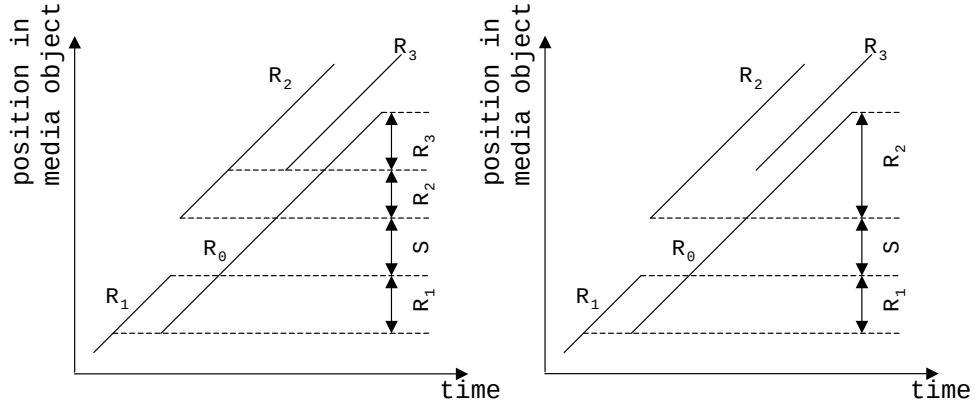
Under sequential access model, a request retrieves data from one predecessor (either server or another request) throughout the entire session (recall Fig. 4(a)), if we preclude the condition that the predecessor may fail. However, in case of non-sequential access, a request may have to switch its session among different predecessors, e.g., R_3 in Fig. 4(b).

Fig. 8(a) illustrates conditions which can cause session switching. As shown in the figure, R_0 initially receives stream from R_1. When R_1 finishes before R_0 does, R_0 has to find a new predecessor to retrieve data from[2]. We claim that this case is unavoidable since R_0 cannot know when R_1 will finish beforehand. If no predecessor is found, R_0 directly streams data from the server S, until a new predecessor R_2 appears[3]. At this point, R_0 switches from the server to R_2. This is consistent with the primary goal of our algorithm:

[2]Notice that there is a time delay from the termination of R_1 until R_0 starts streaming from its new predecessor, which is the buffer distance between R_0 and R_1. During this period, R_1 flushes the data remained in its buffer to R_0.

[3]Although R_2 arrives later than R_0, it can still be R_0's predecessor. A similar example is illustrated in Fig. 2(b)

to maximally save the server bandwidth, *i.e.*, R_0 streams from the server only when it has no predecessor. Finally, if a new predecessor R_3 appears and the transmission cost between R_0 and R_3 is smaller than the cost between R_0 and R_2 ($c(R_0, R_3) < c(R_0, R_2)$), R_0 switches from R_2 to R_3. This is consistent with the secondary goal of our algorithm: to save link cost as much as possible, *i.e.*, R_0 always chooses to stream from its closest predecessor. However, to achieve this goal, R_0 has to keep updated of newly arrived requests and compares if they are closer than its current predecessor (recall lines 3 to 5 in **MDT-Insert**), which will incur intensive message overhead. It also increases the number of session switching times. To save session switching overhead, we simplify the basic algorithm as shown in Fig. 8(b): R_0 continues to stream from R_2, without considering to switch to a closer predecessor, namely R_3. However, the price is that the basic algorithm's optimality at saving link cost is sacrificed.



(a) Basic algorithm (b) Simplified algorithm

Fig. 8. Simplifying session switching

Degree Constrained MDT

In the basic algorithm on MDT operations, we did not constrain the outbound degree of a tree node. Our assumption here is that each end host has unlimited outbound bandwidth to have as many children as possible. However, when we limit the outbound degree of a tree node, then regarding **Definition 2**, the MDT problem has to be modified as follows.

For a request set $\mathbb{R}$ and its MDG $MDG_{\mathbb{R}} = (\mathbb{R}, E)$, find the MDT $MDT_{\mathbb{R}} = (\mathbb{R}, E^T)(E^T \in E)$, which is the *minimal degree-constrained spanning tree* on $MDG_{\mathbb{R}}$.

This problem is NP-complete [22]. Our heuristic is as follows: for each newly arrived request, it finds one among its candidate predecessors with smallest transmission cost, whose outbound degree has not reached the constraint yet. Obviously, the heuristic algorithm is not optimal at reducing link cost. However, its optimality at conserving server bandwidth remains intact. We postpone the proof to Sec. V-E.

V. Scalability: Server Bandwidth Savings

In this section, we analyze the scalability of HSM and asynchronous multicast with respect to conserving server bandwidth. We first introduce the analytical methodology, then derive the required server bandwidth of each approach under different stream access patterns.

A. Analytical Methodology

We consider the distribution of a single media object. The size of the object is T bytes. The object is played out at a constant bit rate of one byte per unit time. Therefore the playback time of the object is T time units. Client requests follow a Poisson process with arrival rate λ. We consider an arbitrarily small portion of the object, say, a byte. This byte is located at the offset x of the object. This byte is denoted as $\boldsymbol{x}$.

Assume that $\boldsymbol{x}$ is multicast by the server at time 0, we need to know till when $\boldsymbol{x}$ needs to be multicast again. let X be the event of $\boldsymbol{x}$ being requested, Λ_X be the average arrival rate of X. We use a random variable w to denote the interarrival time of events in $\{X\}$. Let Z be the event of server multicast of $\boldsymbol{x}$. Clearly events in $\{Z\}$ are a subset of events in $\{X\}$, since not every request for $\boldsymbol{x}$ will trigger the sever multicast. We further use a random variable τ to denote the interarrival time of events in $\{Z\}$. If we know the expected value of τ, denoted as $E(\tau|x)$ (the expected value is conditional, depending on $\boldsymbol{x}$'s location in the media object, i.e., x), then with respect to the above raised question, x will be multicast again after time length $E(\tau|x)$. It means that on average, $\boldsymbol{x}$ is multicast for $\frac{1}{E(\tau|x)}$ times per unit time. Therefore, the required server bandwidth for $\boldsymbol{x}$ is $\frac{1}{E(\tau|x)}$ per unit time. Summarizing the bandwidth for all bytes in the object, the total required server bandwidth B is given by

$$B = \int_0^T \frac{\mathrm{d}x}{E(\tau|x)} \tag{7}$$

Therefore, the main goal of our analysis is to acquire $E(\tau|x)$. To facilitate reading, we illustrate useful notations that are previously introduced in Table III.

Symbol	Definition
λ	average request rate
T	object length
$\boldsymbol{x}$	a byte of the object
x	$\boldsymbol{x}$'s location in the object
$\{X\}$	events of request for x
$\{Z\}$	events of server multicast of x
Λ_X	average arrival rates of events in $\{X\}$
Λ_Z	average arrival rates of events in $\{Z\}$
w	random variable of interarrival time of events X
τ	random variable of interarrival time of events Z
$E(\tau\|x)$	expected value of τ
W	buffer size
B	required server bandwidth
S	request duration in simple access model

TABLE III
NOTATIONS USED IN SEC. V

B. Hierarchical Stream Merging

We start with the HSM algorithm. Analytical results in this subsection have appeared in related work [1][5]. We therefore omit the detailed derivations. Assume that R_1 requests the $\boldsymbol{x}$ at time 0 and triggers a server multicast. x is requested later by R_2 at time t. As outlined in Sec. III-B, if R_2 starts before time 0, it can catch up the multicast of $\boldsymbol{x}$. Otherwise, it misses the multicast at time 0, and x needs to be multicasted again at time t.

The answer to whether R_2 starts before time 0 varies depending on the stream access models we assume. In our analysis, we choose two models: *simple access* and *sequential access*. They are studied respectively as follows.

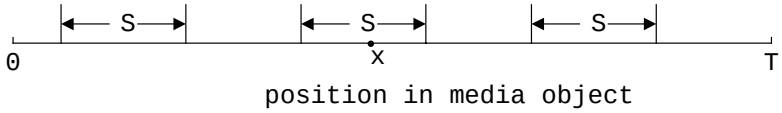


Fig. 9. Simple access model

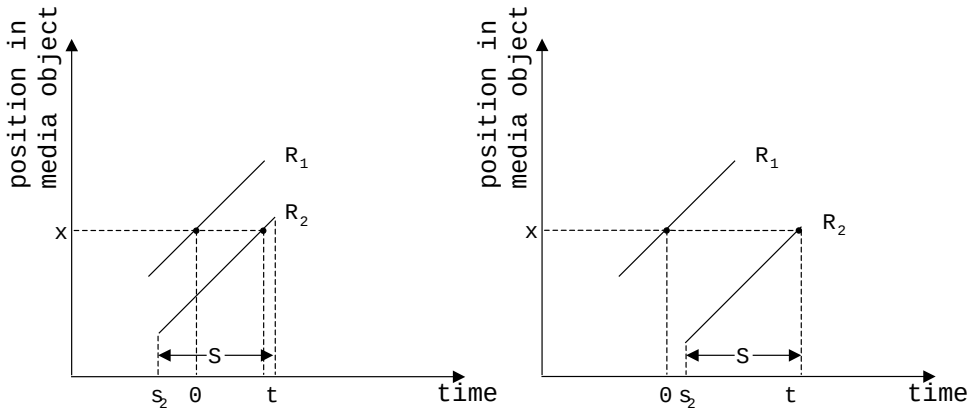
1) Simple Access Model: Under simple access model, each request lasts for time length S ($S < T$). The request starts from an arbitrary offset of the object. For simplicity, we assume that the object is cyclic, which means that the access may proceed past the end of the object by cycling to the beginning of the object. As shown in Fig. 9, a request would contain $\boldsymbol{x}$ only if its starting offset is ranged within $(x - S, x)$. Assuming the starting offset of a request is uniformly distributed within $(0, T)$, then the probability that this request contains $\boldsymbol{x}$ is S/T. Consequently, the arrival rate of event X is

$$\Lambda_X = \lambda S/T \tag{8}$$

R_2's starting time s_2 is ranged within the time interval $(t - S, t)$. R_2 will trigger a new multicast of x if $s_2 > 0$. As shown in Fig. 10(a), if $t \leq S$, then $s_2 > 0$ with probability t/S. In this case, with probability t/S, an event X will trigger an event Z. If $t > S$ (Fig. 10(b)), $s_2 > 0$ is always true. In this case, an event X will definitely trigger an event Z. Now we can derive the arrival rate of event Z as follows

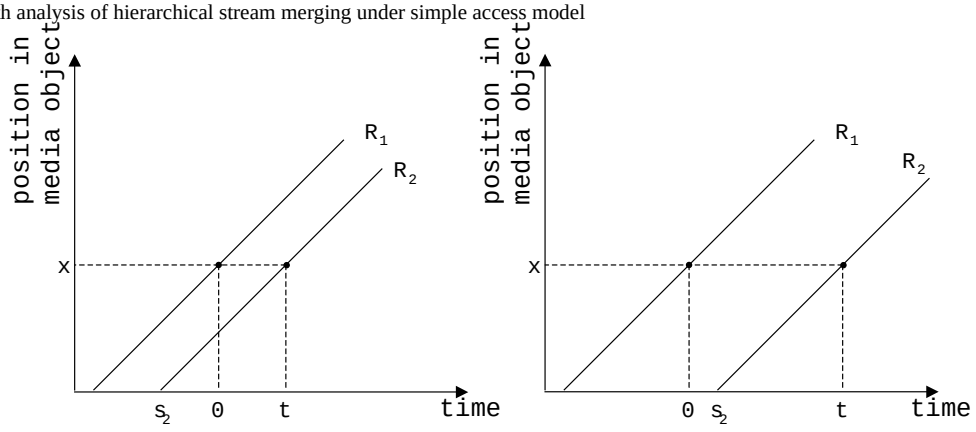
$$\Lambda_Z = \begin{cases} \Lambda_X t/S & \text{if } t \leq S \\ \Lambda_X & \text{if } t > S \end{cases}$$

With Λ_Z, we can derive the expected number of events Z during the interval $(0, t)$ as



(a) $t \leq S$ (b) $t > S$

Fig. 10. Server bandwidth analysis of hierarchical stream merging under simple access model



(a) $t \leq x$ (b) $t > x$

Fig. 11. Server bandwidth analysis of hierarchical stream merging under sequential access model

$$N_Z = \int_0^t \Lambda_Z \mathrm{d}t = \begin{cases} \lambda t^2/2T & \text{if } t \leq S \\ \frac{\lambda S}{T}(t - \frac{S}{2}) & \text{if } t > S \end{cases}$$

Then the probability of no arrival of Z during interval $(0, t)$ is $P(\tau > t) = e^{-N_Z}$, since events in $\{Z\}$ are independent. Hence, the conditional distribution function of τ is

$$F_\tau(t|x) = P(\tau \leq t) = \begin{cases} 1 - e^{-\frac{\lambda t^2}{2T}} & \text{if } t \leq S \\ 1 - e^{-\frac{\lambda S}{T}(t - \frac{S}{2})} & \text{if } t > S \end{cases}$$

Then the conditional density function can be derived as

$$f_\tau(t|x) = \frac{\mathrm{d}F_\tau(t|x)}{\mathrm{d}t} = \begin{cases} \frac{\lambda t}{T} e^{-\frac{\lambda t^2}{2T}} & \text{if } t \leq S \\ \frac{\lambda S}{T} e^{-\frac{\lambda S}{T}(t - \frac{S}{2})} & \text{if } t > S \end{cases}$$

Now we can derive the required server bandwidth based on Eq. (7):

$$B_{HSM}^{simple} = \int_0^T \frac{\mathrm{d}x}{E(\tau|x)} = \int_0^T \frac{\mathrm{d}x}{\int_0^\infty t f_\tau(t|x)\mathrm{d}t} \approx \sqrt{\frac{2\lambda T}{\pi}} \tag{9}$$

The detailed derivations of Eq. (9) may be found in [5].

2) Sequential Access Model: Under the sequential access model, every request contains the entire object. Then it is obvious that

$$\Lambda_X = \lambda \tag{10}$$

As shown in Fig. 11(a), if $t \leq x$, R_2 will definitely arrive no later than time 0. If $t > x$ (Fig. 11(b)), then R_2 can never arrive before time 0. Therefore, we have

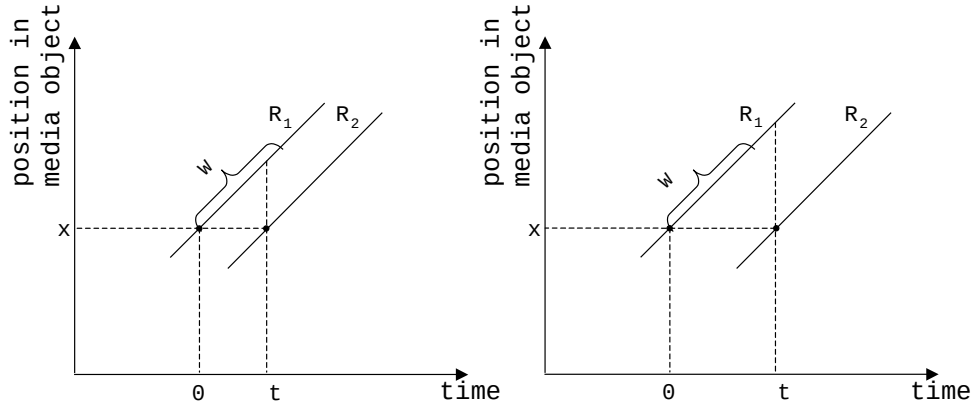
$$\Lambda_Z = \begin{cases} 0 & \text{if } t \leq x \\ \lambda & \text{if } t > x \end{cases}$$

With Λ_Z, following the same procedure in Section V-B.1, B becomes

$$B_{HSM}^{sequential} = \int_0^T \frac{\mathrm{d}x}{E(\tau|x)} = \int_0^T \frac{\mathrm{d}x}{x + 1/\lambda} = \ln(\lambda T + 1) \tag{11}$$

This result agrees with former analysis result in [1].

C. Asynchronous Multicast



(a) $t \leq W$ (b) $t > W$

Fig. 12. Server bandwidth analysis of asynchronous multicast

We use the same analytical model to derive the required server bandwidth for asynchronous multicast. However, we rephrase event Z as the server transmission (unicast) of the byte x. For simplicity, we assume that all requests have a unified buffer size of W. Suppose R_1 requested $\boldsymbol{x}$ at time 0, which triggered the server transmission of $\boldsymbol{x}$. Then $\boldsymbol{x}$ will stay in the buffer of R_1 for time W. Thus, if the next request R_2 requests $\boldsymbol{x}$ within interval $(0, W)$ (Fig. 12(a)), $\boldsymbol{x}$ can be retrieved from the R_1 and further buffered at R_2 for time W. Otherwise (Fig. 12(b)), $\boldsymbol{x}$ has to be retransmitted by the server. In other words, a "chain" is formed among consecutive requests (X event) if their interarrival times are within W. The head request at each chain triggers the server transmission of $\boldsymbol{x}$. Therefore, $E(\tau|x)$ is the average time length of the chain. To derive $E(\tau|x)$, we first calculate the expected value of w, which is used to denote the interarrival time of events X. Suppose we know Λ_X, the expected number of events X during interval $(0, t)$ is

$$N_X = \int_0^t \Lambda_X \mathrm{d}t = t\Lambda_X$$

Then the probability of no arrival of X during interval $(0, t)$ is $P(w > t) = e^{-N_X}$, since events in $\{X\}$ are independent. Hence, the conditional distribution function of w is

$$F_w(t|x) = P(w \leq t) = 1 - e^{N_X} = 1 - e^{-t\Lambda_X}$$

Then the conditional density function can be derived as

$$f_w(t|x) = \frac{\mathrm{d}F_w(t|x)}{\mathrm{d}t} = \Lambda_X e^{-t\Lambda_X}$$

We then calculate the expected value of w when $w \leq W$. In this case, the "chain" is prolonged.

$$E_{w \leq W}(w|x) = \frac{\int_0^W t f_w(t|x)\mathrm{d}t}{P\{w \leq W\}} = \frac{\Lambda_X}{1 - e^{-\Lambda_X W}} \int_0^W te^{-t\Lambda_X}\,\mathrm{d}t$$

Similarly, the expected value of w when $w > W$ is derived as follows. In this case, the "chain" is broken.

$$E_{w > W}(w|x) = \frac{\int_W^\infty t f_w(t|x)\mathrm{d}t}{P\{w > W\}} = \frac{\Lambda_X}{e^{-\Lambda_X W}} \int_W^\infty te^{-t\Lambda_X}\,\mathrm{d}t$$

Let $p = P\{w \leq W\}$, we can derive $E(\tau|x)$ as

$$E(\tau|x) = \sum_{i=0}^{\infty} p^i(1-p)(iE_{w \leq W}(w|x) + E_{w > W}(w|x))$$

Based on Eq. (7), we now have the unified form of B for asynchronous multicast.

$$B = \int_0^T \frac{\mathrm{d}x}{E(\tau|x)} = \int_0^T \frac{\Lambda_X(e^{\Lambda_X W} - 1)}{e^{2\Lambda_X W} - \Lambda_X W - 1}\mathrm{d}x \tag{12}$$

When we substitute Λ_X with Equations (8), and (10), we can obtain the required server bandwidth under simple access and sequential access models as follows:

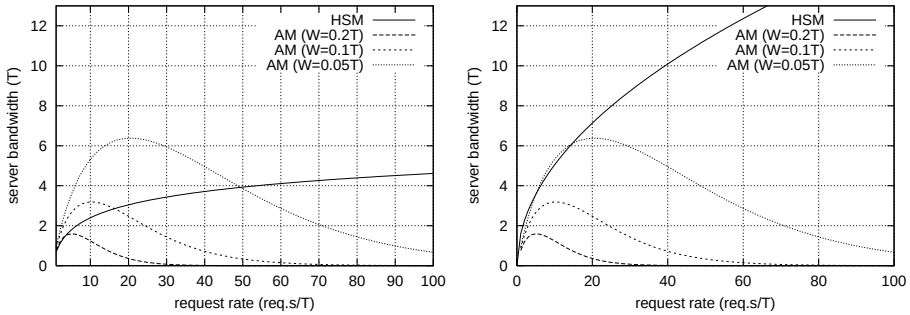
$$B_{Asyn}^{simple} = \frac{\lambda S(e^{\frac{\lambda SW}{T}} - 1)}{e^{\frac{2\lambda SW}{T}} - \frac{\lambda SW}{T} - 1} \tag{13}$$

$$B_{Asyn}^{sequential} = \frac{\lambda T(e^{\lambda W} - 1)}{e^{2\lambda W} - \lambda W - 1} \tag{14}$$

D. Comparison

We plot Eq. (11) and (14) in Fig. 13(a), Eq. (9) and (13) in Fig. 13(b), as function of the average value of Λ_X, which is calculated as

$$\Lambda_X^{avg} = \frac{1}{T}\int_0^T \Lambda_X \mathrm{d}x$$



(a) Sequential model (b) Simple model ($S = \frac{T}{4}$)

Fig. 13. Server bandwidth analysis results

For sequential access and simple access models, Λ_X^{avg} is λ and $\lambda\frac{S}{T}$, respectively. In both figures, request rate is the number of requests issued per T time units, which is the object's playback time. Server bandwidth indicates how many T bytes of data are streamed from the server during this time. As Λ_X^{avg} grows, the cost of HSM is asymptotically increasing at different speeds (logarithm growth for sequential model and square root growth for non-sequential model). The cost of asynchronous multicast is identical under both models because of its unified form in Equation (12). The cost reaches its maximum value when $\Lambda_X = 1/W$ (the point where $\frac{dB}{d\Lambda_X} = 0$). The reason is that the time length of the "request chain" ($E(\tau|x)$) increases exponentially, which overcomes the linear growth of Λ_X^{avg} after this threshold. This means that B can be finite in the face of unpredictable client request rate. The maximum of B is further controllable by tuning W.

Note: We also consider a *Random Access Model*, where the request starts and ends at arbitrary offset of the object. We found it difficult to analytically derive the closed forms with respect to server bandwidth for both HSM and asynchronous multicast under this model. However, experimental results suggest that the curves of random model are very closed to their counterparts in Fig. 13(b) (simple model). For space constraints, we briefly introduce the random model in Appendix A.

E. Optimality of Degree-Constrained MDT

We now have sufficient preparation to return to the issue of optimality of the degree-constrained MDT algorithm that we have previously proposed.

Theorem 3: Degree-constrained MDT algorithm consumes the same amount of server bandwidth as the basic MDT algorithm.

Proof: In the basic MDT algorithm, a request streams from the server, only when it has no other predecessor. If the degree-constrained MDT algorithm is not equally efficient at saving server bandwidth as the basic MDT algorithm, then the following condition must happen. We consider $\boldsymbol{x}$, a arbitrarily byte of the object. As shown in Fig. 14, $\boldsymbol{x}$ is first requested by R_1 at time 0, then by R_2 at time t. The buffer distance between R_1 and R_2 is less than the buffer size W. Yet R_2 is unable to stream from R_1 because R_1 is fully occupied by other request(s). As a result, R_2 has to stream from the server.

Without loss of generality, we assume that R_1 is the only predecessor of R_2, and R_1 can only stream to one of its successors at the same time, *i.e.*, the degree constraint is 1. We now show that this condition does not exist. Let us assume that R_1 is occupied by another request R_1', then R_1' must appears between R_1 and R_2. Therefore, the buffer distance between R_1' and R_2 is even closer than the one between R_1 and R_2. Since we assume that all requests have unified buffer size, then R_2 has another predecessor R_1', from which it can stream from. Therefore, if R_2 has to retrieve $\boldsymbol{x}$ from the server, the only reason is that R_2 does not fall into the buffer window of any of its predecessors. □

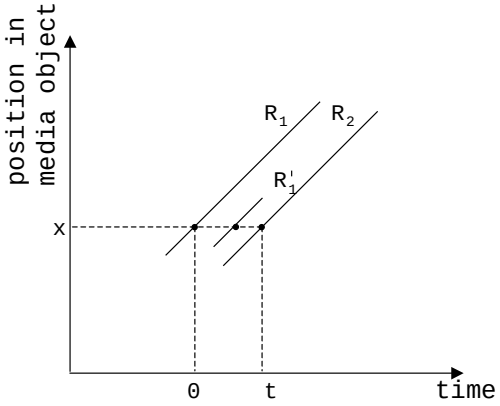


Fig. 14. Degree-constrained media delivery

VI. Efficiency: Link Bandwidth Reduction

Besides service scalability, multicast also offers the network efficiency at reducing link cost. It is well observed that the network overhead increases sublinearly as the multicast group size grows [19][20]. Meanwhile, the per-member cost continues to decrease, which implies that the optimal network efficiency can be achieved by maximally enlarging the client group. This is also the goal to achieve service scalability. Therefore, a scalable multicast solution should also be efficient at saving network cost. To this end, one may intuitively conclude that asynchronous multicast is superior than traditional approaches based on the analysis results of the last section. However, this conjecture needs to be carefully investigated for the following two reasons. First, asynchronous multicast is an application-layer multicast approach, which inevitably introduces topological inefficiencies, namely stress and stretch. Second, asynchronous multicast differs from conventional multicast in that there exist temporal dependencies among group members, which puts more constraints on attempts to optimize the multicast tree. Our solution needs to overcome these two factors in order to claim network efficiency.

A. Analytical Methodology

We again use the analytical model in Sec. V. Let us consider a byte $\boldsymbol{x}$ located at offset x of the object. Suppose $\boldsymbol{x}$ is multicast at time 0, then according to our analysis in Sec. V, $\boldsymbol{x}$ will be multicasted again after $E(\tau|x)$. Therefore, all requests for $\boldsymbol{x}$ that fall within the interval $(0, E(\tau|x))$ are served by the server multicast at time 0. In other words, they are aggregated into one multicast group. We use $G(x)$ to denote the number of receivers in this group. Clearly, $G(x) = E(\tau|x) \cdot \Lambda_X$. We further use $L(n)$ to denote the link cost of a multicast group with n receivers. Then the average per-member link cost is $\frac{L(n)}{n}$. Substituting n with $G(x)$, the average link cost per request is $\frac{L(G(x))}{G(x)}$. Summarizing the link cost for all bytes in the object, the total link cost C can be formulated as

$$C = \int_0^T \frac{L(G(x))}{G(x)} \mathrm{d}x \tag{15}$$

Since we already know $E(\tau|x)$, $G(x)$ can be easily derived. The main goal of our analysis in this section is to acquire $L(n)$. We summarize notations used in this section in Table IV.

B. Hierarchical Stream Merging

The derivation of $L(n)$ varies depending on the network topology. Even within the same topology model, $L(n)$ still takes different forms, since HSM depends on IP multicast, and asynchronous multicast is end-host based. In our analysis, we use the k-ary tree model, which was also adopted in [19][20]. Consider a k-ary tree of depth D. Each tree node is a router. The source is attached to the root node, while clients are attached to the leaf nodes. Since we mainly care about link cost on the backbone, the cost from client to the leaf node is ignored. Therefore, only data traveled within the k-ary tree is counted.

We first derive $L(n)$ for IP multicast. For this purpose, we introduce the reachability function $U(s)$, which denotes the number of tree nodes that are exactly s hops away from the source. In k-ary tree topology, $U(s) = k^s$, which is the number of nodes at tree level

Symbol	Definition
$G(x)$	number of receivers which receive the multicast of x from server
$L(n)$	link cost of a multicast group of n receivers
C	total link cost
$U(s)$	reachability function denoting number of nodes s hops away from the server (HSM)
$F(s)$	probability distribution function of distance s between two end hosts (asynchronous multicast)
D	depth of the k-ary tree

TABLE IV
NOTATIONS USED IN SEC. VI

s. As shown in Fig. 15(a), consider a client H_0 attached to a random leaf node. The multicast path from the source to H_0 passes tree nodes of all levels. Then for an arbitrary node N_s at level s, the probability that the path goes through N_s is $\frac{1}{U(s)} = \frac{1}{k^s}$. If there are n clients, then the probability that none of their paths goes through N_s is $(1 - \frac{1}{U(s)})^n$. Therefore, the probability that N_s belongs to the multicast delivery tree is $(1 - (1 - \frac{1}{U(s)})^n)$. $L(n)$ (the size of the multicast tree) is thus given by

$$L_{IP}(n) = \sum_{s=1}^{D} U(s)(1 - (1 - \frac{1}{U(s)})^n) = \sum_{s=1}^{D} k^s(1 - (1 - k^{-s})^n) \approx n(\frac{1}{\ln k} + D - \frac{\ln n}{\ln k}) \tag{16}$$

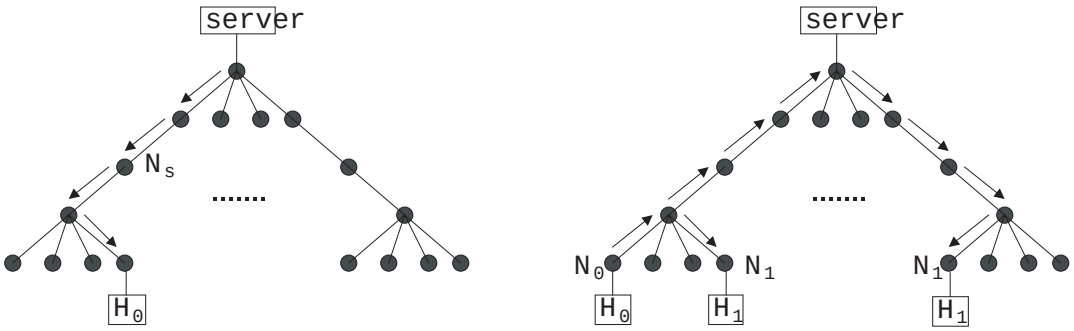
The detailed derivation of Eq. (16) can be found at [19]. Based on Eq. (15) and (16), we can derive the unified form of link cost for HSM as follows

$$C_{HSM} = \int_0^T (\frac{1}{\ln k} + D - \frac{\ln(G_{HSM}(x))}{\ln k})\mathrm{d}x \tag{17}$$

Under sequential access model, $G_{HSM}(x) = \lambda x + 1$ (derived from Eq. (11)). Under simple access model, $G_{HSM}(x) = S\sqrt{\frac{\pi\lambda}{2T}}$ (derived from Eq. (9) and (8)). Now we can derive the link cost of HSM under these two models.

$$C_{HSM}^{sequential} = T(\frac{1}{\ln k} + D) + \frac{\lambda T - (\lambda T + 1)\ln(\lambda T + 1)}{\lambda \ln k} \tag{18}$$

$$C_{HSM}^{simple} = T(\frac{1}{\ln k} + D - \frac{\ln(S\sqrt{\frac{\pi\lambda}{2T}})}{\ln k}) \tag{19}$$



(a) IP multicast (b) Asynchronous multicast

Fig. 15. K-ary tree topology model

C. Asynchronous Multicast

Now we derive $L(n)$ for asynchronous multicast. Since this approach is end-host based, we need to derive the average unicast path length between two clients. As shown in Fig. 15(b), for a given client H_0 attached to leaf router N_0, let s be the distance between H_0 and another client H_1. s is 0 if H_1 is also attached to N_0 (ignoring local link cost), which happens with probability $1/k^D$. If the router of H_1 shares the same parent with N_0 (probability k/k^D), s is either 2 or 0. In general, s is no more than $2h$ if H_0 and H_1 reside in the same subtree of height h. Therefore, we can summarize the probability distribution of s as

$$F(s) = k^{s/2-D} \tag{20}$$

If H_0 could receive data from m other clients, then the probability that none of them is within distance s to H_0 is $(1-F(s))^n$. Then the distribution function of the distance from H_0 to the nearest one of these clients is given by

$$F_m(s) = 1 - (1 - F(s))^m$$

We can further acquire the probability density function $f_m(s) = \frac{\mathrm{d}F_m(s)}{\mathrm{d}s}$. Then the expected value of s can be derived as

$$E_m(s) = \int_0^{2D} s \cdot f_m(s) ds \approx 2(D - \frac{\ln m}{\ln k}) \tag{21}$$

We illustrate the detailed derivation of Eq. (21) in Appendix B. As mentioned in Sec. V-C, in asynchronous multicast, a "chain" is formed among consecutive requests, whose interarrival times are within the buffer size W. All requests on this chain form a multicast group. The header request unicasts data from the server. The link cost of this path is D. Each following request streams from one of its predecessors, whose buffer distance to itself is no more than W. The number of such candidate predecessors is $m = W\Lambda_X$. Then the link cost for a multicast group of n requests is given by

$$L_{AM}(n) = D + (n-1)E_m(s) = D + 2(n-1)(D - \frac{\ln m}{\ln k}) \tag{22}$$

Based on Eq. (12), (15) and (22), the unified form of link cost for asynchronous multicast is

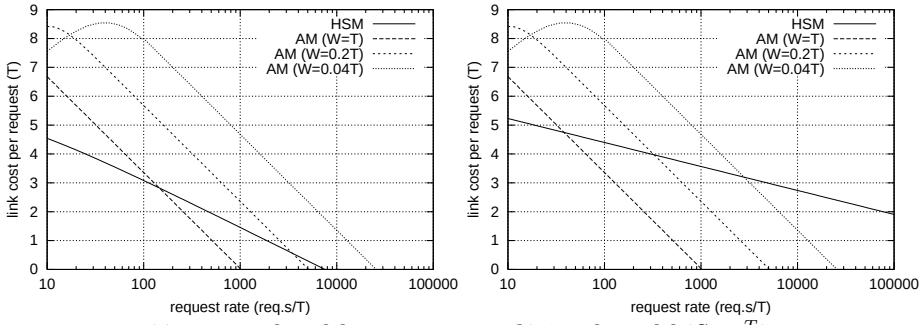
$$C_{AM} = \int_0^T \frac{D + 2(G_{AM}(x)-1)(D - \frac{\ln m}{\ln k})}{G_{AM}(x)} \mathrm{d}x = \int_0^T \Big[\frac{e^{\Lambda_X W} - 1}{e^{2\Lambda_X W} - \Lambda_X W - 1}(\frac{2\ln \Lambda_X W}{\ln k} - D) + 2(D - \frac{\ln \Lambda_X W}{\ln k})\Big]\mathrm{d}x \tag{23}$$

Substituting Λ_X with Eq. (10) and Eq. (8), we have obtained the link cost under sequential and simple access model as follows:

$$C_{AM}^{sequential} = T\Big[\frac{e^{\lambda W} - 1}{e^{2\lambda W} - \lambda W - 1}(\frac{2\ln \lambda W}{\ln k} - D) + 2(D - \frac{\ln \lambda W}{\ln k})\Big] \tag{24}$$

$$C_{Asyn}^{simple} = T\Big[\frac{e^{\frac{\lambda SW}{T}} - 1}{e^{\frac{2\lambda SW}{T}} - \frac{\lambda SW}{T} - 1}(\frac{2\ln \frac{\lambda SW}{T}}{\ln k} - D) + 2(D - \frac{\ln \frac{\lambda SW}{T}}{\ln k})\Big] \tag{25}$$

D. Comparison



(a) Sequential model (b) Simple model ($S = \frac{T}{4}$)

Fig. 16. Analysis of multicast link cost per request in k-ary tree network($k = 4$, $D = 5$)

We plot Eq. (18) and (24) in Fig. 16 (a). As shown in the figure, the first term of Eq. (24) is negligible unless the buffer size is small (e.g., $W = 0.04T$). Furthermore, this term exponentially approaches zero as the request rate λ increases. Therefore, Eq. (24) can be simplified to $2T(D - \frac{\ln \lambda W}{\ln k})$. The scale factor here is the minus term $(-\frac{2T\ln\lambda}{\ln k})$. The remaining part of the equation is constant. For Eq. (18), its scale factor is $(-\frac{T\ln(\lambda)}{\ln k})$. The remaining part is constant. Reflected in the figure, the slope of asynchronous multicast is steeper than HSM since the scale factor of Eq. (24) is two times the scale factor of Eq. (18). Although decreasing more slowly, the cost of HSM is still the smallest, unless the buffer size W of asynchronous multicast becomes very large. Also note that increasing W for asynchronous multicast can help move down the curve, but has nothing to do with the scale factor. Furthermore, the diminishing return of increasing W is logarithmic. This is determined by the minus term $(-\frac{T\ln W}{\ln k})$ in Eq. (24). Finally, we note that the above equations become inaccurate as they approach 0. This is because $L(n)$ (Equations (16) and (22)) is invalid when the group size n reaches its saturation point [19], *i.e.*, the number of clients exceeds the number of leaf routers ($n \geq k^D$).

Fig. 16(b) plots Eq. (19) and (25). The costs of asynchronous multicast are the same in Fig. 16(a) and (b) because of their unified form in (23). However, the scale factor of HSM (Eq. (19)) reduces to $(-\frac{\ln\sqrt{\lambda}}{\ln k})$. Reflected in the figure, the curve of HSM decreases more slowly. This gives asynchronous multicast a better chance to outperform HSM. As shown in the figure, when $W = 0.2T$, the cost of asynchronous multicast becomes the smallest as the request rate is greater than 300.

E. Discussions

The analysis of multicast link cost heavily depends on different network topology models. In K-ary tree topology, the neighborhood of a node expands exponentially. To further confirm our observation in Sec. VI-D, we also consider the power-law topology model [29]. In this model, the neighborhood expansion follows a power law. Furthermore, we do not restrict receivers to be on the leaf routers. Instead, receivers are spread over the entire network. The analysis results (Appendix C) show the similar outcome as in Figure 16: under sequential access model, the link cost reduction of asynchronous multicast has little chance to outperform HSM, unless W becomes unreasonably large. However, the gain becomes significant under non-sequential access model. The intuitive explanation for such phenomenon is that, when switched from sequential to non-sequential model, HSM can aggregate fewer number of client requests into one group, while the multicast group size of asynchronous multicast stays unchanged. This observation ensures the universality of cost link reduction gain of asynchronous multicast to HSM, although the gain varies under different network topologies.

VII. Performance Evaluation

In this section, we compare the performance of asynchronous multicast and HSM at saving server bandwidth and link cost. To examine the practicability of our approach, we also study its operation complexity. Note that in our experiment, asynchronous multicast is end-host based, while HSM assumes the existence of IP multicast. Furthermore, in HSM experiments, we assume that each client is able to simultaneously join unlimited number of multicast groups and calculate joining sequences offline. Therefore, the performance results of HSM is optimal but impractical. Our purpose here is to make it the theoretical baseline, along which the performance of end-host based asynchronous multicast can be evaluated.

A. Experimental Setup

We consider the case of a single CBR video distribution. The video file is one-hour long, *i.e.*, $T = 1$ hour. We do not specify the streaming rate. Instead, we use playback time to indicate the server and link bandwidth consumption. Each 12-hour run is repeated under sequential and non-sequential (simple and random) access models. Since the results obtained from simple and random models are very closed, we only show results from the random model for space constraints. A brief introduction of the random model can be found at Appendix A.

To study the impact of network topology on link cost, we run experiments on a diversified set of synthetic and real network topologies. Our selection largely falls into three categories:

1) **K-ary Tree Topology**: We choose this topology to confirm our analysis in Sec. VI-D, which was conducted on the same topology. We set $k = 4$, $D = 5$, as was specified in Fig. 16. The topology size is 1365. We first experiment the case in which receivers are only located on leaf routers. In the second experiment, we allow receivers to be located on non-leaf routers as well.
2) **Router-level Topology**: We choose an Internet map (Lucent, November, 1999)[23] to represent the router-level topology. The topology size is 112969. We also use GT-ITM topology generator [24] to create a topology of 500 nodes based on transit-stub model. In this set of experiments, receivers can be located on any nodes in the topology.
3) **AS-level Topology**: We use a real AS map (March, 1997) obtained from NLANR [26]. The topology size is 6474. We also use the Inet topology generator [25] to create a topology of 6000 nodes. In both topologies, the distribution function of network distance between two nodes follows the power-law. In this set of experiments, we also allow receivers to be located on any nodes within the topology.

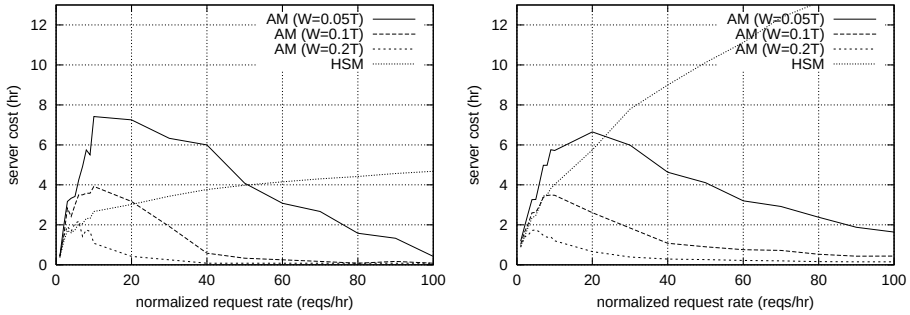
We assume that the IP unicast routing uses delay as its routing metric. IP multicast routing is based on shortest path tree algorithm (DVMRP[27]).

B. Server Bandwidth Consumption

We first evaluate the server bandwidth consumption (average amount of data streamed per hour). Since network topology has no impact on this metric, we only show results obtained on NLANR topology. Note that under non-sequential model, the request rate is normalized the same way the analysis does in Sec. V-D. The curves in Fig. 17(a) show the same growing trend as those analyzed in Fig. 13(a). The curves of both figures do not match exactly. This is mainly caused by the fact that for each specific request rate, the number of requests generated in our simulation cannot be the same as the statistical average number of requests, upon which curves in Fig. 13(a) are calculated. With regard to such statistical error, we believe that it is convincing enough that our results confirm our analysis in Sec. V. When comparing Fig. 17(b) and Fig. 13(b), we draw the same conclusion.

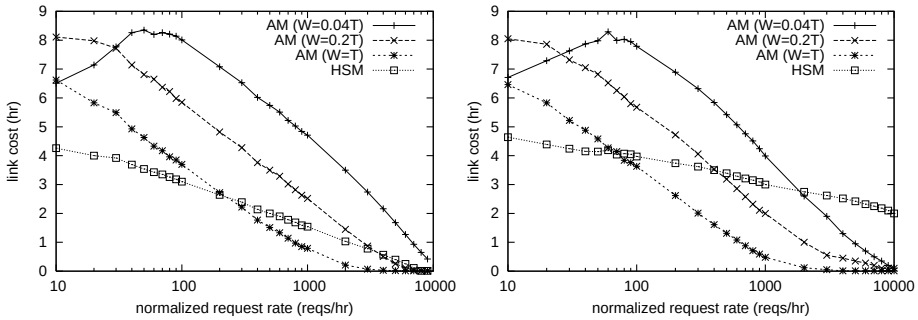
C. Link Cost

In order to validate the analysis of link bandwidth consumption in Sec. VI-D, we first show the experimental results obtained from the same K-ary tree topology . The curves in Fig. 18 are nearly identical with those in Fig. 16. This experiment confirms our observation that under non-sequential accesss model, asynchronous multicast's ability of reducing link cost is stronger than HSM.



(a) Sequential model (b) Random non-sequential model

Fig. 17. Server bandwidth consumption



(a) Sequential model (b) Random non-sequential model

Fig. 18. Multicast link cost per request in k-ary tree topology($k = 4$, $D = 5$)

To simplify our illustration, we define a new metric *Link Cost Ratio*, which is the ratio of link cost of asynchronous multicast to HSM. With this metric, what we are mainly concerned about is the growing trend of the curves: if the link cost ratio drops as we increase the request rate. If the answer to this question is affirmative, our next question is when the "crossing point" (the point at which link cost ratio equals to 1) is reached.

Fig. 19 shows the experimental results on link cost ratio under the random non-sequential access model. From the results, we have the following observations. First, the link cost ratio heavily depends on the network topology. However, all curves have negative slopes, which suggests the universality of link cost reduction gain of asynchronous multicast to HSM. The location of "crossing point" also greatly varies for different topologies. For example, in case $W = T$, the location of the point ranges from 20 reqs/hr (Fig. 19 (d)) to 300 reqs/hr (Fig. 19 (f)). Several factors may play important roles here, such as the network size, its topological properties and location of the server. Second, when we exponentially increase the buffer size (in the experiment, we set $W = 0.04T$, $0.2T$, T, respectively), the link cost reduction gain is almost linear, which confirms the same observation in our analysis (Sec. VI-D). This finding suggests that small to medium sized buffers can be greatly helpful at saving link cost. Further increases with respect to the buffer size may be less beneficial. Third, the simplified algorithm (presented in Sec. IV-D) increases the link cost by a fixed fraction. This impact can be offset by increasing the buffer size.

Fig. 20 shows the experimental results on link cost ratio under the sequential access model. A common observation is that all curves become less steep than their counterparts in Fig. 19. Plus, the "crossing point" can be hardly reached, unless the buffer size becomes large ($W = T$), or the request rate gets extremely high. Also, constraining the outbound degree of each end host does not greatly degrade the performance. When we set the constraint to 4, the curve is very similar to the one with no constraint. To summarize, this set of experiments reveal that under the sequential access model, the link cost reduction gain of asynchronous multicast to HSM is minor. The main reason is — as revealed in Sec. VI-D — under the sequential access model, HSM is able to aggregate more requests into one multicast group than under the non-sequential access model.

D. Operation Complexity

Now we evaluate the operation complexity of asynchronous mulicast under non-sequential access model. Fig. 21 shows the average number of predecessors a request needs to retrieve data from during its entire session. For the basic algorithm, this number is larger than 3. The simplified algorithm reduces this number to 2, which means that on average a request only needs to switch its predecessor once. Fig. 22 shows the cumulative distribution of requests with different number of predecessors.

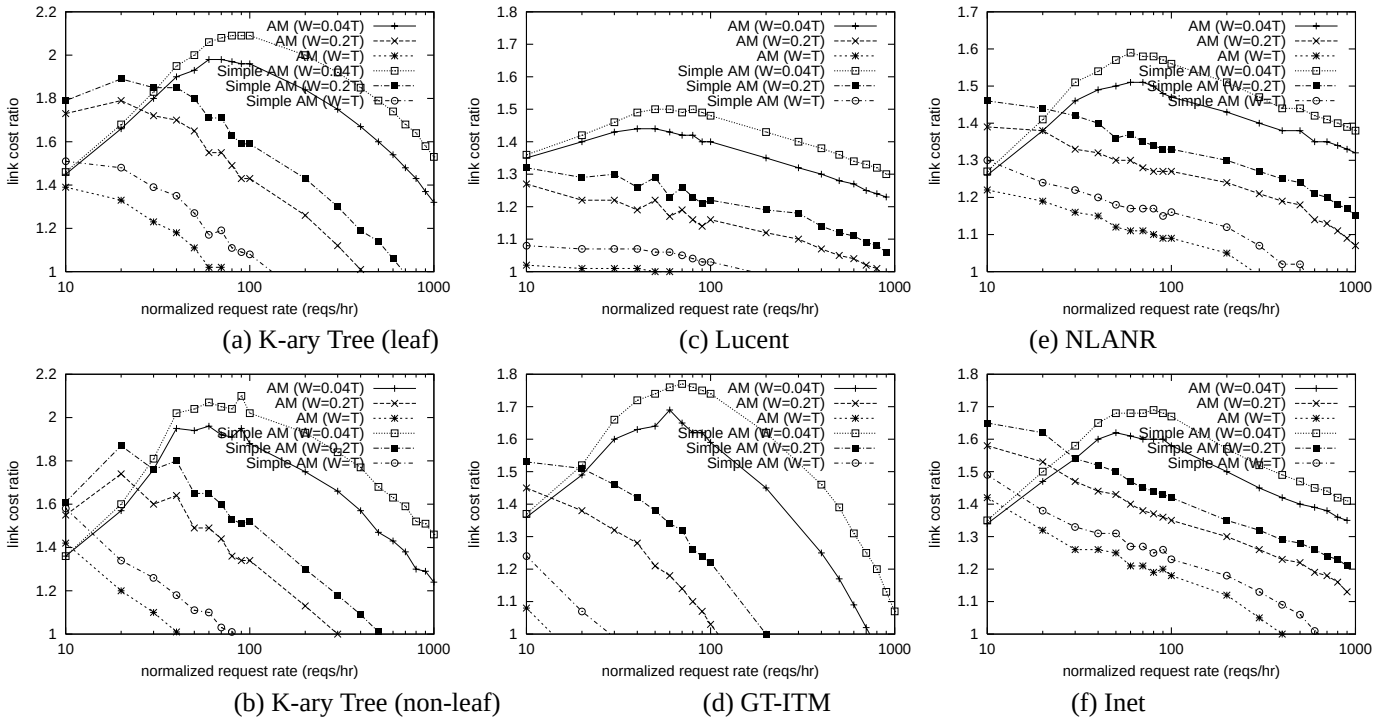


Fig. 19. Link cost ratio under random non-sequential model

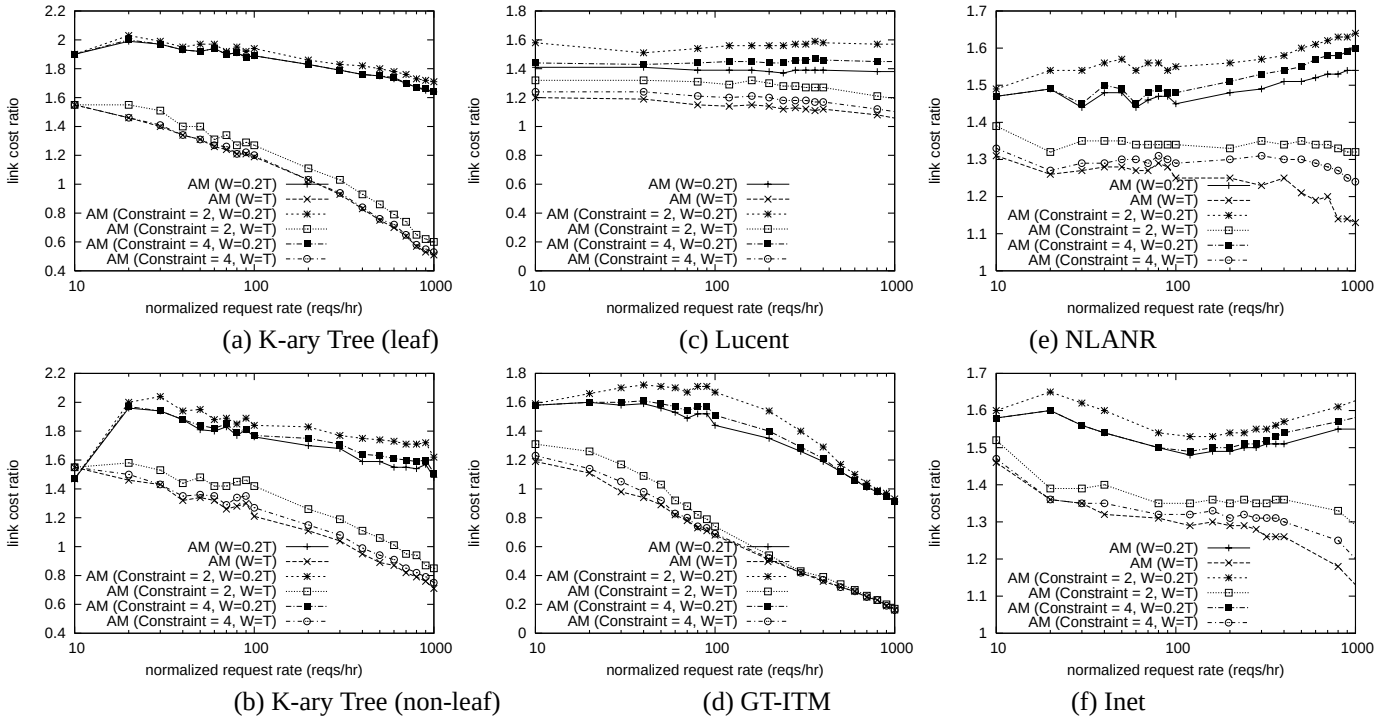


Fig. 20. Link cost ratio under sequential model

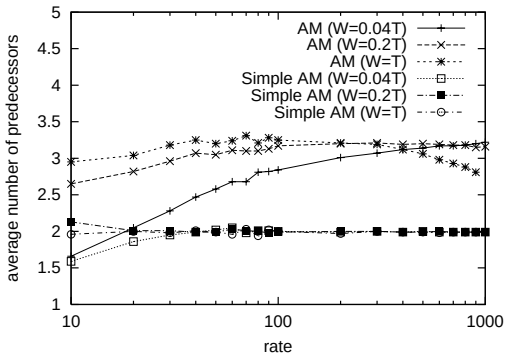


Fig. 21. Average number of predecessors per request

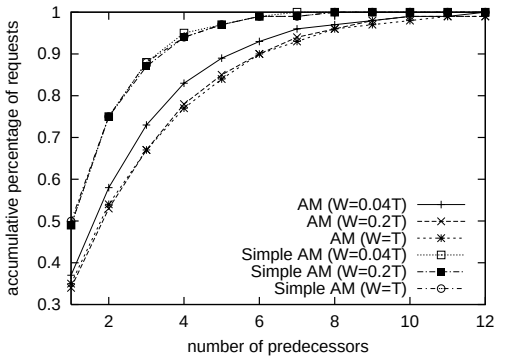


Fig. 22. Cumulative distribution of predecessor number

VIII. Concluding Remarks

Scalable and efficient on-demand media distribution is a challenging task. Existing solutions suffer from various limitations such as dependence on IP multicast, being non-scalable in the face of non-sequential client access patterns, and vulnerability to bursty requests towards "hot spots". In this paper, we propose the concept of *asynchronous multicast*. This approach takes advantage of the strong buffering capabilities of end hosts in application-layer overlay networks. Based on the concept, we propose a novel overlay multicast strategy, *oStream*, to address the on-demand media distribution problem. Through in-depth analysis and extensive performance evaluation, we are able to draw the following conclusions. First, the required server bandwidth of oStream defeats the theoretical lower bound of traditional multicast-based solutions. Second, with respect to bandwidth consumption on the backbone network, the benefit introduced by oStream overshadows the topological inefficiency of application overlay.

References

[1] D. Eager, M. Vernon and J. Zahorjan, Minimizing Bandwidth Requirements for On-Demand Data Delivery, *IEEE Transaction on Knowledge and Data Engineering*, Vol. 13, No. 5, 2001.
[2] C.C. Aggarwal, J.L. Wolf and P.S. Yu, On Optimal Batching Policies for Video-on-Demand Storage Servers, *IEEE ICMCS '96*.
[3] K.A. Hua, Y. Cai and S. Sheu, Patching: A Multicast Technique for True On-Demand Services, *ACM Multimedia '98*.
[4] K.A. Hua, S. Sheu, Skyscraper Broadcasting: A New Broadcasting Scheme for Metropolitan VoD Systems, *ACM SIGCOMM '97*.
[5] Shudong Jin and Azer Bestavros, Scalability of Multicast Delivery for Non-sequential Streaming Access, *Proceedings of ACM SIGMETRICS '01*.
[6] J. Almeida, J. Krueger, D. Eager and M. Vernon, Analysis of Educational Media Server Workloads. *Proceedings of NOSSDAV '01*.
[7] H. Tan, D. Eager, M. Vernon and H. Guo, Quality of Service Evaluations of Multicast Streaming Protocols, *ACM SIGMETRICS '02*.
[8] Y. Chu, S. Rao and H. Zhang, A Case for End System Multicast, *ACM SIGMETRICS '00.*
[9] J. Jannotti, D. Gifford, K. Johnson, M. Kaashoek and J.O. Jr., Overcast: Reliable Multicasting with an Overlay Network, *Proceedings of OSDI '00*.
[10] V. Padmanabhan, H. Wang, P. Chou and K. Sripanidkulchai, Distributing Streaming Media Content Using Cooperative Networking, *Proceedings of NOSSDAV '02*.
[11] Y. Chawathe, Scattercast: An Architecture for Internet Broadcast Distribution as an Infrastructure Service, *PhD Dissertation, University of California at Berkeley, 2000*.
[12] J. Jin, K. Nahrstedt, mc-SPF: An Application-Level Multicast Service Path Finding Protocol for Multimedia Application, *Proceedings of ICME '02*.
[13] S. Banerjee, B. Bhattacharjee and C. Kommareddy, Scalable Application Layer Multicast, *ACM SIGCOMM '02*.
[14] S. Sen, J. Rexford and D. Towsley, Proxy Prefix Caching for Multimedia Streams, *Proceedings of IEEE INFOCOM '99*.
[15] Y. Chae, K. Guo, M. Buddhikot, S. Suri and E. Zegura, Silo, Tokens, and Rainbow: Schemes for Fault Tolerant Stream Caching, *IEEE JSAC on Internet Proxy Services, 2002*.
[16] W. Jeon, K. Nahrstedt, QoS-aware Middleware Support for Collaborative Multimedia Streaming and Caching Service, *Microprocessors and Microsystems, Special Issue on QoS-enabed Multimedia Provisioning over the Internet*, 2002.
[17] S. Sheu, K. Hua and W. Tavanapong, Chaining: a Generalized Batching Technique for Video-on-Demand Systems, *IEEE ICMCS '97*.
[18] A. Dan and D. Sitaram, A Generalized Interval Caching Policy for Mixed Interval and Long Video Environments, *SPIE MMCN '96*.
[19] Graham Philips and Scott Shenker, Scaling of Multicast Trees: Comments on the Chuang-Sirbu scaling law, *Proceedings of ACM SIGCOMM '99*.

[20] C. Adjih, L. Georgiadis, P. Jacquet and W. Szpankowski, Multicast Tree Structure and the Power law, *Proceedings of ACM-SIAM Symposium on Discrete Algorithms (SODA) 2002*.
[21] Shudong Jin and Azer Bestavros, Cache-and-Relay Streaming Media Delivery for Asynchronous Clients, *Proceedings of NGC' 02*.
[22] M. Garey and D. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, 1979.
[23] USC Information Sciences Insititute, Internet maps, *http://www.isi.edu/div7/scan/mercator/maps.html*, 1999.
[24] E. Zegura, K. Calvert and S. Bhattacharjee, How to Model an Internetwork, *IEEE INFOCOM '96*.
[25] C. Jin, Q. Chen and S. Jamin, Inet: Internet Topology Generator, *Technical Report CSE-TR-443-00, University of Michigan*, 2000.
[26] National Laboratory for Applied Network Research, *http://moat.nlanr.net/Routing/rawdata*, 1997.
[27] S. Deering and D. Cheriton, Multicast Routing in Datagram Internetworks and Extended LANs, *ACM Transactions on Computer Systems*, vol. 8, no. 2, 1990.
[28] Y. Cui and K. Nahrstedt, Proxy-based Asynchronous Multicast for Efficient On-demand Media Distribution, *SPIE MMCN '02*.
[29] M. Faloutsos, P. Faloutsos and C. Faloutsos, On Power-Law Relationships of the Internet Topology, in *Proceedings of ACM SIGCOMM*, August 1999.

Appendix A

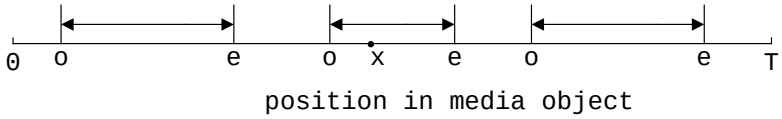


Fig. 23. Random access model

We briefly introduce the random access model, where a request starts and ends at arbitrary offsets. Corresponding notations can be found at Table III. Suppose both the starting offset o and ending offset e of the request are uniformly distributed ($o < e$), as shown in Fig. 23. Consider an arbitrary small portion x of the object, say a byte. If $o \leq x$, then in order to contain x, e must satisfy that $x < e$. The probability is $\frac{T-x}{T-o}$. If $o > x$, then by no means x can be contained in (o, e). Therefore, we have

$$\Lambda_X = \frac{\int_0^x \lambda \frac{T-x}{T-o} \mathrm{d}o + \int_x^T \lambda \cdot 0 \mathrm{d}o}{T} = \lambda \frac{T-x}{T} \ln \frac{T}{T-x} \tag{26}$$

Similar to the analysis with simple access model in Sec. V-B.1, we first derive Λ_Z. Similar to Fig. 11, if $t > x$, then R_2 can never arrive before time 0, thus will trigger the multicast of x. Otherwise, the probability that R_2 arrives after time 0 is derived as follows

$$\frac{\int_{x-t}^x \lambda \frac{T-x}{T-o} \mathrm{d}o}{T} = \lambda \frac{T-x}{T} \ln \frac{T-x+t}{T-x}$$

Therefore, the arrival rate of Z becomes

$$\Lambda_Z = \begin{cases} \lambda \frac{T-x}{T} \ln \frac{T-x+t}{T-x} & \text{if } t \leq x \\ \Lambda_X & \text{if } t > x \end{cases}$$

With Λ_X and Λ_Z, following the same procedure in Sec. V-B and V-C, we can get the server bandwidth for HSM and asynchronous multicast, which is hard to derive analytically. Instead, we showed our experimental results under this model in Sec. VII.

Appendix B

In this appendix, we derive $E_m(s)$ (Eq. (21)) in Sec. VI-C.

$$E_m(s) = \int_0^{2D} s \cdot f_m(s) \mathrm{d}s = \int_0^{2D} s \cdot m(1-k^{s/2-D})^{m-1} \frac{\ln k}{2k^D} \mathrm{d}s \tag{27}$$

Let $y = 1 - k^{s/2-D}$, then Eq. (27) becomes

$$\begin{aligned} \int_0^{1-\frac{1}{k^D}} \frac{2\ln[(1-y)k^D]}{\ln k} \cdot m y^{m-1} \mathrm{d}y &= \frac{2\ln(k^D)}{\ln k} \int_0^{1-\frac{1}{k^D}} m y^{m-1} \mathrm{d}y + \frac{2}{\ln k} \int_0^{1-\frac{1}{k^D}} \frac{\ln(1-y)}{\ln k} \cdot m y^{m-1} \mathrm{d}y \\ &= 2D \cdot y^m \Big|_0^{1-\frac{1}{k^D}} + \frac{2}{\ln k} \cdot \ln(1-y) y^m \Big|_0^{1-\frac{1}{k^D}} + \frac{2}{\ln k} \int_0^{1-\frac{1}{k^D}} \frac{y^m}{1-y} \mathrm{d}y \\ &= [2D + \frac{2}{\ln k} \ln(\frac{1}{k^D})](1-\frac{1}{k^D})^m + \frac{2}{\ln k} \int_0^{1-\frac{1}{k^D}} \frac{y^m}{1-y} \mathrm{d}y \end{aligned} \tag{28}$$

Now the only unsolved integral form of Eq. (28) is $\int_0^{1-\frac{1}{k^D}} \frac{y^m}{1-y} \mathrm{d}y$. We define it as $N(m)$. then we have

$$\begin{aligned} N(m) - N(m-1) &= -\int_0^{1-\frac{1}{k^D}} y^{m-1} \mathrm{d}y = -\frac{(1-\frac{1}{k^D})^m}{m} \\ N(0) &= \int_0^{1-\frac{1}{k^D}} \frac{1}{1-y} \mathrm{d}y = -\ln(\frac{1}{k^D}) \end{aligned}$$

Then we derive $N(m)$ as follows

$$L(m) = -\ln(\frac{1}{k^D}) - \sum_{i=1}^{m} \frac{(1-\frac{1}{k^D})^i}{i} \tag{29}$$

Since k^D (total number of leaf nodes) is usually large, we can approximate $(1-\frac{1}{k^D})$ as 1. After combining Eq. (28) and (29), we finally have

$$E_m(s) \approx 2D - \frac{2}{\ln k}\sum_{i=1}^{m}\frac{1}{i} \tag{30}$$

Since $\sum_{i=1}^{m}\frac{1}{i}$ is asymptotically close to $\ln m$, we can simplify Eq. (30) as $2(D-\frac{\ln m}{\ln k})$.

Appendix C

In this appendix, we analyze the link cost on power-law network topology. Corresponding notations can be found in Table IV. For each node in this network, it has $k \cdot s^\alpha$ neighbors of distance within s hops. k and α are constants. The maximum distance of two nodes is D, denoted as the diameter of the network. The network size is consequently $k \cdot D^\alpha$.

We first derive $L(n)$ for HSM. Suppose the multicast source is attached to a random node in the network. The reachability function $U(s) = k(s^\alpha - (s-1)^\alpha)$, which is the number of nodes reachable in exactly s hops from the source. Consider a a random client H_0 and a node N_s; N_s is s hops away from the source. If the multicast path of H_0 runs through N_s, then the following two conditions must be met: (1) H_0 must be attached to a node, whose network distance to the source is equal or larger than N_s (probability $\frac{kD^\alpha - k(s-1)^\alpha)}{kD^\alpha}$); (2) C_0 locates exactly below N_s (probability $\frac{1}{U(s)}$). Therefore, N_s is on the multicast path with probability $p = \frac{D^\alpha-(s-1)^\alpha}{U(s)\cdot D^\alpha}$. Suppose there are n clients, then the probability that at least one multicast path runs through N_s is $(1-(1-p)^n)$. $L(n)$ (the size of the multicast tree) is thus given by

$$L_{IP}(n) = \sum_{s=1}^{D} U(s)(1-(1-\frac{D^\alpha-(s-1)^\alpha)}{U(s)\cdot D^\alpha})^n) = \sum_{s=1}^{D} k(s^\alpha-(s-1)^\alpha)(1-(1-\frac{D^\alpha-(s-1)^\alpha)}{k(s^\alpha-(s-1)^\alpha)D^\alpha})^n) \tag{31}$$

Following the same procedure in Sec. VI-B, we can get $C_{HSM}^{sequential}$ and C_{HSM}^{simple}. We show their numerical results in Fig. 24, since the derivation of Eq. (31) is extremely difficult.

Now we derive $L(n)$ for asynchronous multicast. We first derive the probability distribution function of the distance between two nodes. Given the definition of the power-law network, this function is given by

$$F(s) = \frac{k\cdot s^\alpha}{k\cdot D^\alpha} = (\frac{s}{D})^\alpha$$

If a node can retrieve data from m other nodes, then following the same procedure in Sec. VI-C, we can get the probability distribution function $F_m(s) = 1-(1-F(s))^m$, and density function $f_m(s) = \frac{\mathrm{d}F_m(s)}{\mathrm{d}s}$. The expected value $E_m(s)$ is then derived as follows

$$E_m(s) = \int_0^D s\cdot f_m(s)\mathrm{d}s = \int_0^D s\cdot m(1-(\frac{s}{D})^\alpha)^{m-1}\cdot\frac{\alpha s^{\alpha-1}}{D^\alpha}\mathrm{d}s \tag{32}$$

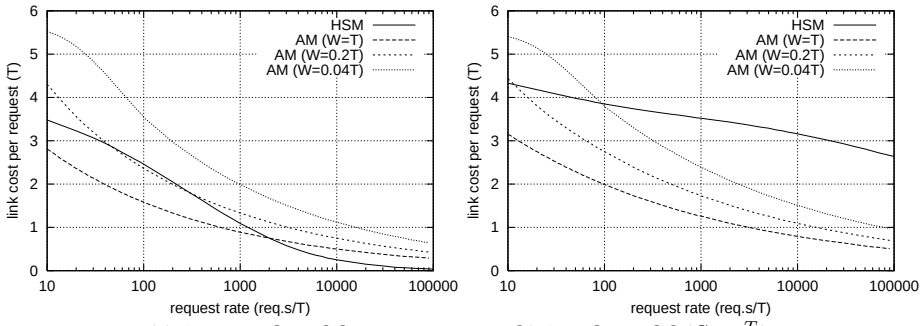
Let $y = (\frac{s}{D})^\alpha$, then Eq. (32) becomes

$$\begin{aligned}
Dm\int_0^1(1-y)^{m-1}\cdot y^{\frac{1}{\alpha}}\mathrm{d}y &= Dm\left[\frac{y^{1+\frac{1}{\alpha}}}{1+\frac{1}{\alpha}}\cdot(1-y)^{m-1}\Big|_0^1 + \frac{m-1}{1+\frac{1}{\alpha}}\int_0^1(1-y)^{m-2}\cdot y^{1+\frac{1}{\alpha}}\mathrm{d}y\right]\\
&= Dm\left[0+\frac{m-1}{1+\frac{1}{\alpha}}\int_0^1(1-y)^{m-2}\cdot y^{1+\frac{1}{\alpha}}\mathrm{d}y\right]\\
&= D\frac{m(m-1)}{1+\frac{1}{\alpha}}\left[\frac{y^{2+\frac{1}{\alpha}}}{2+\frac{1}{\alpha}}\cdot(1-y)^{m-2}\Big|_0^1 + \frac{m-2}{2+\frac{1}{\alpha}}\int_0^1(1-y)^{m-3}y^{2+\frac{1}{\alpha}}\mathrm{d}y\right]\\
&= D\frac{m(m-1)}{1+\frac{1}{\alpha}}\left[0+\frac{m-2}{2+\frac{1}{\alpha}}\int_0^1(1-y)^{m-3}y^{2+\frac{1}{\alpha}}\mathrm{d}y\right]\\
&= \cdots\\
&= D\frac{m(m-1)\cdots1}{(1+\frac{1}{\alpha})(2+\frac{1}{\alpha})\cdots(m-1+\frac{1}{\alpha})}\int_0^1 y^{m-1+\frac{1}{\alpha}}\mathrm{d}y\\
&= D\frac{m(m-1)\cdots1}{(1+\frac{1}{\alpha})(2+\frac{1}{\alpha})\cdots(m-1+\frac{1}{\alpha})}\cdot\frac{y^{m+\frac{1}{\alpha}}}{m+\frac{1}{\alpha}}\Big|_0^1\\
&= D\cdot\prod_{i=1}^{m}\frac{i}{i+\frac{1}{\alpha}}
\end{aligned} \tag{33}$$

Substituting Eq. (33) into Eq. (22), we have

$$L_{AM}(n) = D(1 + (n-1)\prod_{i=1}^{m} \frac{i}{i + \frac{1}{\alpha}})$$

where $m = W \cdot \Lambda_X$. Now $C_{AM}^{sequential}$ and C_{AM}^{simple} can be derived the same way in Sec. VI-C. We directly show the results in Fig. 24.



(a) Sequential model (b) Simple model ($S = \frac{T}{4}$)

Fig. 24. Analysis of multicast link cost per request in power-law network ($k = 2$, $\alpha = 5$, $D = 5$)