Regular Paper

AVD: A Router-Assisted On-Demand Video Delivery Service

Chih-Chang Hsu,† Terumasa Aoki† and Hiroshi Yasuda†

At this time, high quality on-demand video streaming services are expected to be one of the most promising applications of Content Distribution Networks because of the wide-area deployment of broadband network and the advanced improvement of the video compression technologies. This paper proposes a simple and efficient mechanism called Active Video Delivery (AVD) that provides an efficient group communication service for on-demand video systems. This mechanism utilizes the efficiency of the forwarding paradigm provided in IP Multicast and the flexibilities in control and deployment introduced in Application-layer Multicast. Unlike the traditional IP Multicast approach, the AVD mechanism is incrementally deployable. Only the branching Active Routers (AR) are required to maintain the forwarding state. In addition, since the AVD mechanism performs the underlying Unicast forwarding to achieve Multicast-like service, no specific Multicast group needs to be defined. In this paper, we describe how to construct an AVD tree where a Multicast-like delivery service can be achieved. Additionally, we present a *true* on-demand algorithm called Video Merging that can be built on top of the AVD tree. Both in our analytic model and simulation results, we have demonstrated that with a properly chosen AVD stream regeneration threshold, this AVD mechanism allows for significant reductions in the video server bandwidth requirement, the network cost and the average link load. In addition, we have evaluated the AVD performance according to varied percentages of routers that are ARs deployed in the simulation network.

1. Introduction

While the current IP Unicast-based delivery mechanism works fine for www, email and FTP, a number of new multimedia applications, such as video-on-demand, on-line gaming, multipoint video conferencing, Internet broadcasting, and large-scale digital video archive, could not benefit from the services built on top of this delivery mechanism. As a result, an infrastructure that enables the delivery of diverse video content to meet performance objectives under limited network resource conditions is in great demand. In addition, an efficient method that assists in retrieving, accessing, and downloading video content on-demand or in real-time from any locale is also strongly demanded.

Large-scale commercial Video on Demand (VoD) systems that would provide users the ability to view high quality streaming videos at any time via the Internet are becoming more and more desirable. Currently, the majority of VoD research concerns the video server bandwidth requirement, the network delivery cost, and the startup delay during video playback. Video Batching [1], Video Broadcasting [2], and Video Caching algorithms are examples of recent schemes that address these issues. They have shown that the performance of VoD systems can be greatly improved in terms of lower server bandwidth requirement and guaranteed startup delay.

However, most of these schemes assume the presence of Multicast-enabled routers deployed in all the transmission paths. This obviously is not feasible in the current state of IP Multicast networks.

An alternative technology called Application-layer Multicast [3] was proposed to solve those issues existing in the IP Multicast. Instead of using a shared Multicast delivery tree, end hosts participating in an application-layer Multicast group create and maintain data forwarding paths for other nearby end hosts, which are located in the network hierarchy. This provides a very simple and flexible way to achieve a group communication service. However, several issues still remain.

In this paper, we propose a simple and efficient mechanism called Active Video Delivery (AVD), which is intended to obtain the advantages of both IP-layer and Application-layer Multicast approaches. Instead of solving all the existing problems of IP Multicast at once, we focus this work on how to construct a simple mechanism by which a group communication service can be achieved. Based on the Active Networks (AN) technology [4] (i.e., programmable network), our proposed mechanism is capable of utilizing the efficiency of the for-

† RCAST, The University of Tokyo

warding paradigm provided in hardware-based (IP-layer) Multicast. This also allows for flexibilities in control and deployment introduced in Application-layer Multicast. In addition, a new video merging algorithm which utilizes the constructed AVD tree is also presented. This algorithm is suitable for the reduction of the network link load and for the optimization of the video server bandwidth utilization, while delivering the heavy traffic and lengthy nature of video streaming.

Section 2 presents the AVD tree construction algorithm that includes the introduction of the User JOIN and the AVD Tree Optimization processes. Section 3 describes the video merging algorithm and provides an analysis of required video server bandwidth and network cost related to this algorithm. Section 4 outlines our simulation design and evaluation results. Section 5 describes the related work. Finally, we conclude the paper.

2. Active Video Delivery

Unlike the conventional IP Multicast approach, the AVD mechanism does not require all the routers that set along the path to be Multicast-capable. Although this idea is similar to that of the *Tunneling* technology, it differs from the *Tunneling* technology in that the AVD mechanism does not incur the complicated encapsulated/decapsulated processes. These processes result in a significant increase in overheads at the routers. In addition, our proposed AVD mechanism requires only parts of routers (i.e., those acting as BPs) to maintain the per-group forwarding state. As a result, the overhead of the maintenance of forwarding state can be greatly reduced.

To find a BP, the AVD mechanism exploits a simplified *Traceroute*-like program to detect the available Active Routers (AR) deployed along the path. Unlike the underlying *Traceroute* program which detects the path between two systems by sending consecutive UDP packets with increasing TTLs, the simplified *Traceroute-like* program acts much like that of *Router Record* defined in IP option 7 [12]. That is, upon receiving user's JOIN request, the server first generates this detection program towards the user. When an intermediate AR receives this program, it checks to see if the "AR_detection" option is present. If so, it inserts its own ID as known in the environment and forwards this program towards the user. The same procedure will be repeated when the downstream ARs receive this program.

One possible drawback for using this simplified *traceroute-like* program is that the server should generate this program once for every receipt of the user's request. Although this may cause a significant overhead if the number of simultaneous user's arrivals is large, we think that this issue can be solved by utilizing some of the existing technologies, such as clustering and load-balancing. We see there are a great number of similar models which are successfully deployed in the current Internet, such as Akamai [13], Yahoo and Google. We believe that our proposed mechanism remains efficient and effective for a large number of IP Multicast applications. Services models particularly in the field of secure digital content delivery require a simple and effective digital rights management system. This can be easily achieved by our AVD mechanism compared to those proposed in the conventional IP Multicast as well as other distributed-based approaches.

2.1 USER_JOIN

In our proposed model, a logical topology tree is maintained in the video server's database. Since the logical topology tree does not need to contain all the router nodes of the whole Internet, but only ARs set along the delivery paths, it does not suffer from the scalability issue. Note that it is normally assumed that ARs are likely to be deployed at the edge of the Internet. In addition, according to the research result regarding the real Multicast tree shape presented in [9),14)], a large percentage of the nodes in a Multicast tree (graph) are routers with an out-degree of only one, and the branching points are located in the upper hierarchies of the Multicast tree. This allows us to further reduce the size of the logical topology tree, which needs to be maintained in the server, by simply selecting the AR, which is closest to the client, and part of the AR nodes, which are closer to the video server, from the result of the AR detection program.

As depicted in **Fig. 1**, upon receiving a user's request, the video server generates a program to detect all the available ARs set along the path from the video server to the user. The first AR, which receives the detection program, must construct an AR_ON_PATH list where IDs (for example, IP address) of all the downstream ARs (i.e., toward the user) are appended (PUSH in) sequentially. As a result

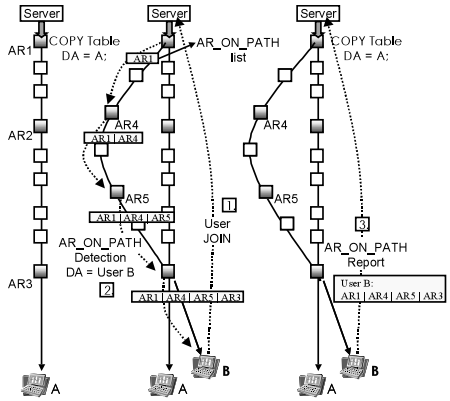

Fig. 1 Processes related to the AR_ON_PATH list construction.

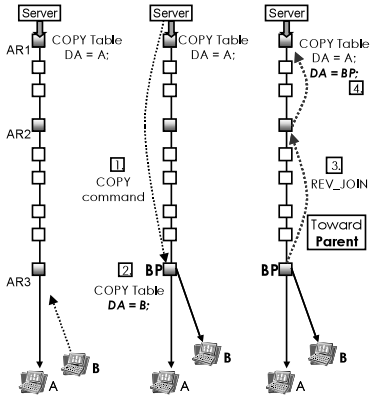

Fig. 2 Sequences related to the USER_JOIN process.

ARs set along the path from the video server to the user can be determined eventually. In addition, each AR must also learn about its parent AR by POP out of the most recently pushed entry, which is performed by its upstream AR (i.e., toward the server), from the received AN_ON_PATH list. This upstream parent information will be used in the following AVD tree optimization processes.

After constructing the AR_ON_PATH list and appending its ID into the list, the first AR forwards the detection program and an updated AR_ON_PATH list to its downstream node based on the routing information maintained in its Unicast forwarding table. The same procedure will be repeated until the user who generated the original request receives the detection program and a final updated version of the AR_ON_PATH list. In the end, the user sends the final version of the AR_ON_PATH list to the server.

Upon receiving the AR_ON_PATH report from the user, the server checks/updates the existing logical delivery tree maintained in its database. This logical delivery tree will then be compared with the newly received AR_ON_PATH information for determining a branch point (BP) where the packet duplication function is to be installed. If multiple BPs are found in this comparison, the AR which is the closest to the user will be chosen as the BP. After that, the server encapsulates an active code program containing the packet duplication initialization command into a normal IP packet (for example, Option of IPV4 header) and sends it to the discovered BP. When the BP (i.e., AR) receives the active program generated from the video server, it checks the capability of performing the COPY function. If incapable, the AR will issue a request for the necessary module to existing well-known active code servers [15] and then perform a downloading of the necessary modules. On the other hand, if the COPY module is already implemented, the AR simply initializes the duplication function operation and creates a new COPY table where a new DA entry regarding the new user is added.

2.2 REV_JOIN

As depicted in Steps 1 and 2 of **Fig. 2**, after the completion of the USER_JOIN process of User B, the destination addresses (DA) maintained in the COPY tables of the server and the BP are User A and User B, respectively. That is, packets generated from the video server are encapsulated with DA=User A while packets generated from the BP are encapsulated with DA=User B. This, however, would cause the problem that packets generated from the video server will not be received by the BP but will be directly sent to User A. As a result, the BP is incapable of performing duplication for packets passing through.

To solve this problem, we deploy a control message called REV_JOIN by which packets generated from the video server can be directed to the BP, if necessary. As shown in Step 3 of Fig. 2, when a new entry is added to the COPY table of an AR for the first time, the AR initializes and propagates a REV_JOIN message toward its upstream "parent" node with a final destination as Server (in this example, the parent is AR2). If the parent AR, which is currently performing the duplication operation, finds that the same requested video ID (VID) is being processed, it will create a new entry into its COPY table upon receiving this REV_JOIN message; On the other hand, if the same VID is

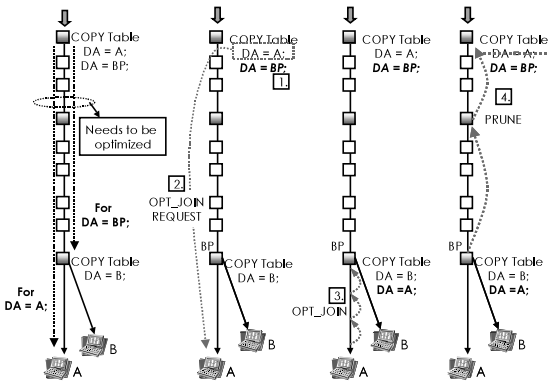

Fig. 3 OPT_JOIN and PRUNE processes.

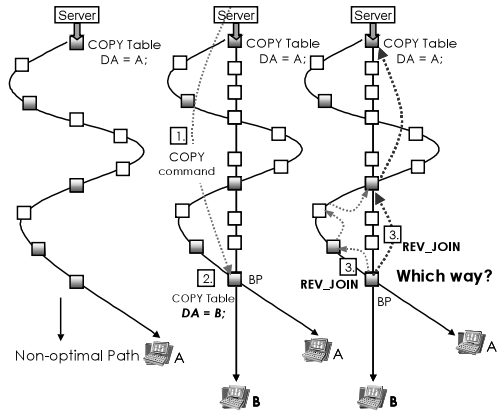

Fig. 4 The RTT estimate process -1.

not currently being processed, the AR simply forwards the REV_JOIN message to its parent node.

2.3 OPT_JOIN and PRUNE

After the REV_JOIN process is completed, the BP is capable of receiving (snooping) packets passing through, performing duplication, and generating those received packets to the DA entries recorded in its COPY table. However, the construction of an optimal AVD tree is not completed at this point. As shown in **Fig. 3**, packets with the same content are delivered via a single link twice. This will greatly increase the total network delivery cost when the number of the downstream nodes is large.

The OPT_JOIN process, implemented in the AVD mechanism, is intended to solve this problem. As depicted in Step 2 of Fig. 3, when a new entry has been added to the COPY table of an AR after the REV_JOIN process, the AR issues an *OPT_JOIN_Request* to all the entries (i.e., User's DA) recorded in its COPY table, except the most recently added entry (i.e, BP). The user (i.e., User A) who receives the *OPT_JOIN_Request* subsequently performs an OPT_JOIN process to its "parent" node. As a result, a new entry of User A is added to the COPY table of the BP (Step 3, Fig. 3). Another process called PRUNE is performed after the OPT_JOIN process. That is, after the completion of the OPT_JOIN process, when a new entry has been added to an AR node's COPY table, the AR initializes and generates a PRUNE message to its upstream AR. This is intended to remove entries that might be out-of-date, thus no redundant packets will be delivered more than twice in a single link.

2.4 RTT Estimate

As described above, the AVD tree is constructed step by step according to the sequence of the user arrivals. We call the path constructed between the server and the first arrived user the *fundamental path* (see **Fig. 4**: The path from the server to User A). *Since* the *fundamental path* is built based on the Unicast forwarding, it is highly possible that this path may not be optimal (i.e., *the shortest*). If the *fundamental path* is not optimal and will not be updated or revised, the final established AVD tree (i.e., after the completion of all user's JOIN processes) will result in an inefficient Multicast delivery tree.

To construct and maintain the *fundamental path* as optimally as possible, we deploy a Round Trip Time (RTT) Estimate process into our AVD mechanism. This RTT Estimate process is straight forward, that is, we use this process to calculate either RTT or Hop counts among different paths, then choose the path with the least RTT or Hop counts (see Fig. 4) as the new *fundamental path* where the subsequent REV_JOIN message will flow through and the packet duplication function will be allocated.

As depicted in **Fig. 5**, if an AR acquires a new "parent" from the AR_ON_PATH list during the AR detection process, we call this AR "BP_Standby_AR". When this BP_Standby_AR node receives a COPY command from the server, it will first perform the RTT Estimate process, then determine which "parent" will be used for performing its REV_JOIN process by choosing the path with the lowest measured RTT or hop counts. As a result, the non-optimal fundamental path (if it exists) can be updated and revised.

2.5 AVD Tree Maintenance

To keep the logical tree maintained in the

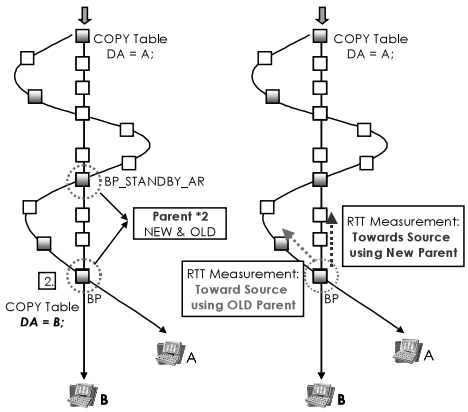

Fig. 5 The RTT estimate process -2.

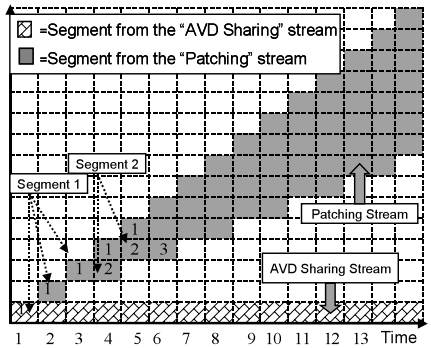

Fig. 6 The Single_AVD mechanism.

server and the soft-state maintained in the ARs up-to-date, each parent AR (i.e., BP) periodically sends a "*heartbeat*" message to all its children, and each child (i.e., User or BP) responds to the *heartbeat* message with an "*alive*" message after a randomly selected period of time.

If a child does not hear from its parent for a period of time, and if the child is a BP: (i) the BP originates a "PARENT_LOSS" message to the server; then (ii) The server initiates an AR detection program with the destination equal to the BP; after that (iii) the BP performs "REV_JOIN" to its nearest upstream new parent according to the AR information recorded in the newly received AR_ON_PATH list. But if the child is a user, the user simply performs a new JOIN process.

If a parent does not hear from its child for a period of time, and if the child is a BP: (i) the parent simply removes the entry related to the child (BP) from its COPY table; then (ii) the parent sends the server a TREE_UPDATE message which explains that "the BP & its downstream nodes" are unreachable; after that (iii) The server updates the logical tree by removing the tree related to the BP, after a selected period of time. But if the child is a user, the parent simply removes the entry related to the child from its COPY table.

3. VIDEO MERGING

In this section we discuss the basic video merging algorithm of the AVD mechanism. This algorithm can be implemented as a component of any on-demand video services. In addition, it can be constructed either on top of the AVD tree or other group communication service models. The video merging involves combining a number of pure Unicast (Patching) streams and a Multicast-like (AVD_Sharing) stream for achieving an efficient sharing service (see **Fig. 6**). In addition, we present an off-line analysis (video stream initiations are known ahead of time) for both the video server bandwidth requirement (i.e., the maximum number of concurrent out-going connections) and the total network cost.

Ying Cai's work on *video patching* [16] contributes greatly to the design of our video merging mechanism. He exploited a regular stream (used for delivering an entire video) and several patching streams (used for delivering parts of a video) as a Multicast group to perform delivering requested videos recursively. By using the *Grace Patching* algorithm and taking the arrival rate of the user's requests into consideration, his proposed technique could find its optimal patching performance. The key of this system is that the generation of a regular stream or a patching stream, which is determined by the video server, depends on the client's buffering capability. However, the client's buffering capability is no longer an issue today given the low cost of personal computers and the current maker-specific set-up boxes.

In contrast, our proposed mechanism takes the total number of *active* out-going server connections as the main consideration for determining the regeneration of a new AVD stream. Unlike Ying Cai's approach, we believe the main challenge of today's large-scale VoD developments (applications) should concentrate on how to effectively utilize the available network bandwidth, especially in its relationship to the video server.

3.1 Analytic Model

Throughout the paper, we made a number of assumptions. These are: (i) the video length

can be suitably partitioned into L units; (ii) the user arrival access pattern is "fully loaded" (i.e., every time slot contains at least one arrival request); (iii) the time required for transmitting one segment is one unit of time (i.e., the total time T for delivering the whole segments is equal to L); (iv) the network cost for delivering a segment via one channel is 1; and (v) a new stream, either in the form of AVD_Sharing or Patching, always begins at segment 1 of the video.

For example, as depicted in Fig. 6, User A arriving at $time = 1$ receives only the AVD_Sharing stream originated by the video server directly (*Note that these segments will playback immediately*). User B arriving at $time = 2$ will be required to receive the video segments from two streams. These are: (i) the AVD_Sharing stream: for video segments "from 2^{nd} to the end" that are duplicated and forwarded by the intermediate ARs. (*Note that these segments are originally destined to User A only, In addition, these segments will be stored on local disk for latter playback temporally*); and (ii) the patching stream: for video segment 1. This segment *will playback immediately.* (Note that this segment is originated from the video server and directed to User B directly. Similarly, User C, arriving at $time = 3$, will also be required to receive both the AVD_Sharing stream which contains video segment "from 3^{rd} to the end" and the patching stream which contains video segments 1 and 2.

Given the above rules by which users receive the streams either from the video server directly or from the AR, we can have the total number of the active logical channels (connections), $Active\_Srv\_Channel(t)$, originated from the video server at any time t:

$$Active\_Srv\_Channel(t) = AVD\_Sharing(t) + \sum_i Patch_i(t)$$

$$where \quad AVD\_Sharing(t) = 1 \quad t = [0, T] \tag{1}$$

$$Patch_i(t) = \begin{matrix} 1 \\ 0 \end{matrix} \quad \begin{matrix} t = [i, 2*i+1] \\ else \end{matrix} \tag{2}$$

From Eq.(2), we can obtain the following time series

$$\Big\{ \sum_i Patch_i(t)|_{t=0 \sim T} \Big\} = \{0, 1, 1, 2, 2, 3, 3, ..., A\}$$

$$where \quad A = \begin{cases} \frac{T-1}{2} & if \quad T \in odd \\ \frac{T}{2} & if \quad T \in even \end{cases}$$

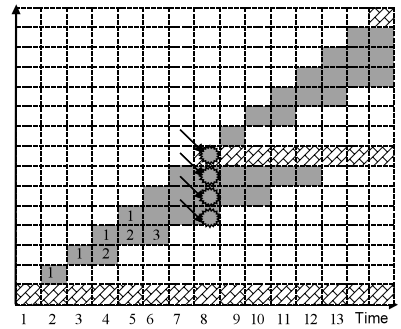

Fig. 7 The Multiple_AVD mechanism, i.e., Multiple_AVD_Patching (P=4).

For simplicity, we choose $T \in odd$ for the following analysis. Therefore, we can obtain the number of the active server channels at any time t and the total network cost for delivering all the video segments:

$$Active\_Srv\_Channel(t) = \frac{t-1}{2} + 1 = \frac{t+1}{2} \quad 0 < t \leq T \tag{3}$$

where 1 = the AVD_Sharing stream channel.

$$\begin{aligned} &Total\_Network\_Cost \\ &= \sum_{t=0}^{T} AVD\_Sharing(t) \\ &\quad + \sum_i Length\_Patch_i(t) \\ &= T + [1 + 2 + 3 + ... + (T-1)] \\ &= \frac{T(T+1)}{2} \end{aligned}$$

The simplest algorithm that can be used in the AVD mechanism is shown in Fig. 6. In this case, the video server generates only one main stream (i.e., AVD_Sharing stream) to perform the Multicast-like sharing and several independent Unicast streams for performing patching. We call this algorithm Single_AVD. Similar to the underlying Unicast mechanism, this Single_AVD sharing algorithm, however, clearly would not perform effectively because of the high demand on the video server bandwidth and the heavy network cost given the presence of a lengthy video (i.e., the time duration is large).

To solve this problem, we can consider another mechanism called the *Multiple_AVD_Patching* mechanism (i.e., **Fig. 7**). That is, If the number of the fan-out (out-going) *Patching* streams originated from the video server reaches the threshold ($P_{threshold}$), which is set in advance, for the first time (t_{1st}), i.e.,

$$\sum_i Patch_i(t) \geq P_{threshold}$$

the video server then regenerates a new, full-length main stream (i.e., from segment 1 to the end segment) to perform a new AVD sharing source for those users who arrive later. *Note that we allow this form of a new AVD_Sharing stream regeneration to repeat continuously after a fixed time interval.* As a result, the usual traffic produced by the Patching streams can be greatly reduced.

From Eq.(3), we can compute the time, t_{1st}, at which the server regenerates a new AVD_Sharing stream and obtain the total round of the AVD stream regeneration during the time period T:

$$\frac{t_{1st}-1}{2} = P_{threshold}$$
$$\rightarrow \quad t_{1st} = 2 * P_{threshold} + 1$$
$$Round_{avd} = L/t_{1st} = L/(2 * P_{threshold}+1) \quad (4)$$

In the $Multiple\_AVD\_Patching$ case, the $Active\_Srv\_Channel(t)$ at any time t (where $t < T$) and the $Total\_Network\_Cost$ during a period of time from generating the 1^{st} video segment of the AVD_Sharing stream to receiving the last video segment of the Patching stream (i.e., the time duration will be $2T$ in this example) can be expressed as:

$$Active\_Srv\_Channel(t) = No.\,of\,the\,AVD\,Sharing\,Streams + \sum_i Patch_i(t) \quad (5)$$

$$Total\_Network\_Cost = \sum_{j=1}^{round} Length\_AVD_j + \sum_{j=1}^{round} \Big[\sum_i Length\_Patch_i(t)\Big] \quad (6)$$

From Eq.(5), we can further obtain both the maximum number of on-going channels generated from the video server and the total network cost for delivering the whole video as the following

$$Max\_Active\_Srv\_Channel \approx Round_{avd} + P_{threshold} \quad (7)$$

$$Tota\_Network\_Cost \approx L(Round_{avd} + P_{threshold}) \quad (8)$$

For simplicity, we use R and P to denote $Round_{avd}$ and $P_{threshold}$ respectively. Therefore, from Eq.(7), we have

$$Max\_Active\_Srv\_Channel = R + P$$

since $L = R * (2P + 1)$,
the $Max\_Active\_Srv\_Channel$ can be expressed as:

$$= (2R^2 - R + L)/2R \quad \rightarrow \quad F(R)$$

Let $F'(R) = 0$, we can observe that the *minimum* number of maximum active server channels during the time period $2T$ can be obtained while we set

$$R = \sqrt{\frac{L}{2}} \quad (9)$$

3.2 Comparison Results

In **Figs. 8** and **9** we compare the performance metrics in terms of the number of the maximum active server channels (i.e., we call it the server bandwidth requirement) and the total network cost for three different delivery mechanisms. (Note that we set $L = 1001$). Clearly, the $Multiple\_AVD\_Patching$ mechanism requires considerably less network delivery cost and at an overall lower video server bandwidth requirement than that of the Unicast and the original AVD mechanism (Note that we set $P_{threshold} = 22$).

Figure 10 shows the relationship of the

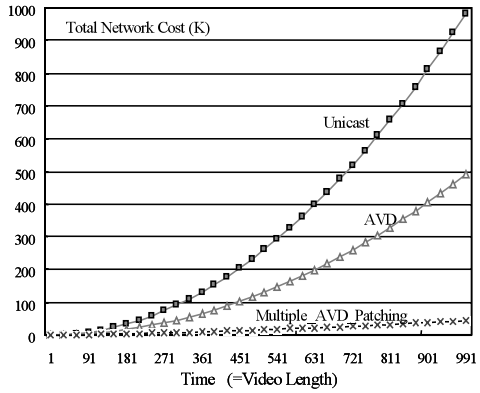

Fig. 8 Number of active server channels over time.

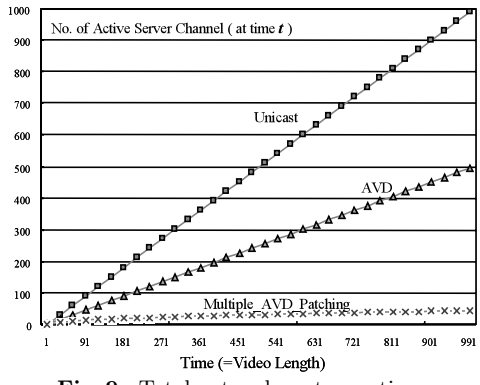

Fig. 9 Total network cost over time.

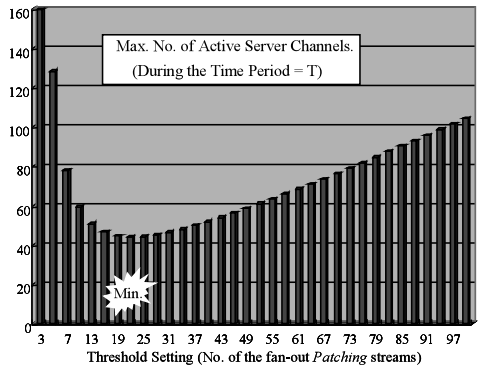

Fig. 10 Max. number of active server channels v.s. threshold setting.

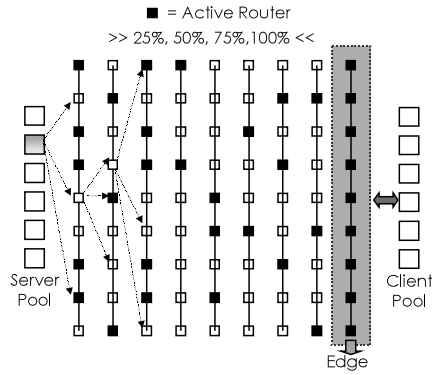

Fig. 11 The simulation topology.

Table 1 The main parameters deployed in the simulations.

Total number of users	300 300, 600, 1200 (**Fig. 15**)
User request arrival pattern	Poisson
Average user arrival time	300 requests /sec (**Fig. 12**) 1 request /sec
Total number of selectable videos	1 16 (**Fig. 15**)
Pattern of user's request on diverse videos	Zipf-like Dist. (skew =0.7)
Percentage of routers that are ARs	25%, 50%, 75%, 100%
Threshold (AVD stream re-generation)	20

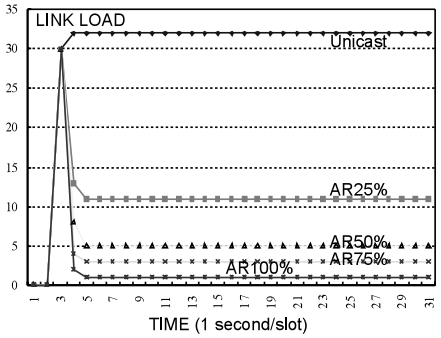

Fig. 12 Average link load over time (300 requests/sec).

threshold setting to the total number of the active server channels (i.e., concurrent on-going connections originated from the video server). We observe that the *minimum* "maximum number of active server channels during the time period $2T$", generated by the video server, could be obtained if the threshold (i.e., the number of the fan-out *Patching* streams) is set to $19 \sim 25$. The Eq.(9) derived in our analytic model provides a simple method to the same optimal threshold setting. That is, from Eqs.(9) and (4), we can also easily obtain the following:

$$R = \sqrt{\frac{1001}{2}} \cong 22.5 \quad \Rightarrow \quad P_{threshold} \cong 22$$

4. EVALUATION

We have conducted simulations to evaluate the effectiveness of our proposed mechanism using a simulation tool called OPNET [17]. For simplicity, and for the purpose of analyzing the effectiveness of the performance of the AVD tree and the video merging algorithm, we assume that all the links in the simulation graph are "lossless". As depicted in **Fig. 11**, we use a partial mesh (i.e., each router has only a limited number of neighbors in the mesh) type simulation topology in which different percentages of routers that are ARs can be varied. Links established between routers in different hierarchies are determined randomly at the beginning of the simulation. The total number of routers is set at 100.

Table 1 shows a summary of the main parameters used in these simulations. Two metrics are used for evaluating the performance. These are: (i) Link Load: the number of packets with the same content delivered by a single link; and (ii) Server Bandwidth Requirement: the maximum number of out-bound channels (connections) generated from the video server.

The first simulation is intended to evaluate if the delivery tree constructed by the AVD mechanism could achieve the same efficiency as that of conventional IP Multicast approaches. Therefore, we assume that (i) all users request the same video; and (ii) all users arrive within 1 second (i.e., the server will receive 300 user requests simultaneously). **Figure 12** shows how the average Link Load changes over time according to the varied percentages of routers that are ARs. As expected, the result indicates that

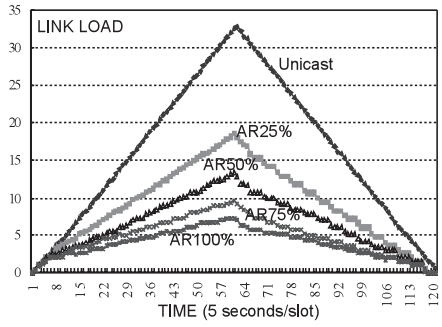

Fig. 13 Average link load over time (1 request/sec).

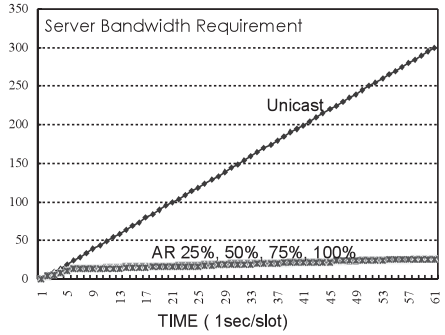

Fig. 14 Average requirement on the server bandwidth over time.

the average Link Load can be reduced to 1 if the percentage of routers that are ARs is set at 100%. That is, the proposed AVD mechanism can achieve the same efficiency as that of conventional IP Multicast approaches if all the routers are ARs. In addition, we observe that the higher the percentage of routers that are ARs, the lower the average Link Load. Another finding in this evaluation result is that the average Link Load obtained in all the AR-presence cases is reduced after the 1^{st} time slot, while an increase is found in the Unicast case. This is due to the fact that the duplication function performed by the ARs successfully reduces the average link load after the completion of all user's JOIN processes.

The second simulation is intended to evaluate the effectiveness of the *Multiple_AVD_Patching* mechanism (i.e., as shown in Fig. 7, multiple AVD_Sharing streams are generated for performing sharing service). **Figure 13** shows how the average load of the network link changes over time according to the varied percentages of routers that are ARs. As expected, we observe that the greater the percentage of routers that are ARs, the lower the average Link Load. In addition, the average Link load can be reduced to approximately half of that measured in Unicast even if the percentage of routers that are ARs is only set at 25%.

Figure 14 shows how the average requirement on the server bandwidth changes over time while performing our proposed *Multiple_AVD_Patching* mechanism. We observe that this mechanism significantly reduces the server bandwidth requirement (i.e., over 90%) in all the AR-presence cases (i.e., AR = 25%, 50%, 75%, and 100%) compared to that measured in the Unicast. This behavior is likely to occur if the router, which is the closest to

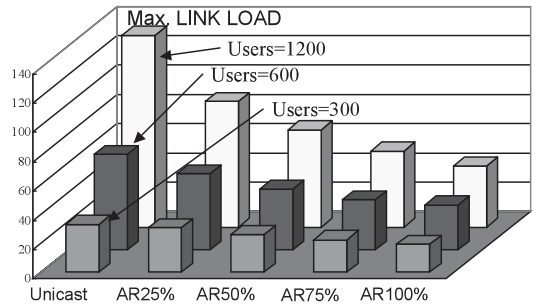

Fig. 15 Max. average link load v.s. different total number of user requests.

the video server, is AR. That is, the server only needs to generate an AVD_Sharing stream to its nearest downstream AR that would subsequently duplicate and deliver the subsequent video segments to users who request the same content.

With the results obtained from Figs. 12, 13, and 14, we can conclude that our proposed solution offers the following performance savings: (i) the capability of achieving the same efficiency as that of conventional IP Multicast approaches; (ii) the higher the percentage of routers that are ARs set in the network, the more benefit we gain; (iii) Even if users with different arrival times request the same videos that are currently being delivered, we benefit from using the *Multiple_AVD_Patching* mechanism; and (iii) the video server loading can be shared generously by a reasonably small number of ARs that are set closer to the video server.

In **Fig. 15**, we study the effect of different total numbers of users who request diverse videos. We set the total number of selectable videos is set at 16 and the total number of users at 300, 600, and 1200 respectively. According to recent research results into *requests for web pages*

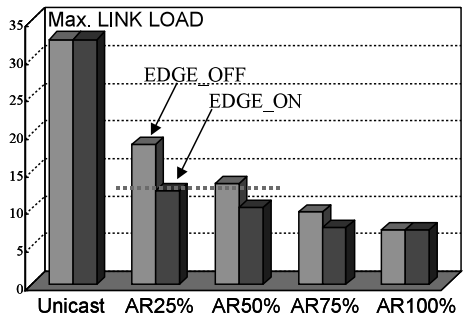

Fig. 16 Max. average link load v.s. EDGE_OFF and EDGE_ON.

and web-based video streaming [18),19)], we can assume that videos selected by different users are based on a Zipf-like distribution with a skew factor set at 0.7. Figure 15 shows that the maximum average Link Load changes over the total number of users. As expected, we observe that the more users who join the AVD tree, the greater the performance gain (i.e., the reduction in the maximum Link Load) the AVD mechanism achieves. In the case of "$Users = 300$", we observe that the performance gain achieved by the deployment of the ARs is not obvious. This is because the more the variety of videos requested by the users, the less benefit gain the AVD mechanism could achieve. On the other hand, the result in the case of "$Users = 1200$" demonstrates that even if the percentage of routers that are ARs is only 25%, the average Link Load can be reduced to around 50%. This is due to the fact that the more users who join an AVD tree, the higher the possibility the users will request the same video. As a result, we can conclude that the AVD mechanism can effectively share video segments among users with different arrival times who request diverse video content.

The final simulation is intended to evaluate the performance of setting all the edge nodes (See Fig. 11), which are deployed closer to the user site, as ARs. We call this EDGE_ON. On the other hand, if the AR is randomly allocated, we name it EDGE_OFF. We observe the result in **Fig. 16** that the greatest reduction in the maximum Link Load achieved by EDGE_ON is when AR% is set at 25%, by comparing it to that of EDGE_OFF. That is, we can conclude that if only 25% of routers that are ARs can be deployed into the Network, it would be more effective to first deploy the AR into the edge of the network. In addition, the result in Fig. 16 also demonstrates that if the percentage of routers that are ARs is 25% where all the edge routers near the user site are set as ARs (i.e., EDGE_ON), the achieved performance gain could be slightly better than the case of "AR50% without EDGE_ON", (i.e., AR50% in the EDGE_OFF case).

5. Related Work

IP Multicast was proposed to provide a group communication service that efficiently utilizes the limited network resource. Although it has been a decade since the first proposal, and a large number of research efforts continuously focus on its improvements, the current state of IP Multicast deployment is far from reaching its full potential. According to the research result of [5)], a reasonable estimate is that the actual penetration of IP Multicast into the current Internet is only around 4%.

Application layer Multicast [3),6),7)] as an alternative technology of IP Multicast provides a very simple and flexible way to achieve the group communication service. The majority of application-layer Multicast research concerns how to dynamically self-organize an overlay distribution tree where a group communication service can be deployed. However, since the overlay-based delivery tree may differ from that of the underlying physical-layer network, a non-trivial delay in packet delivery (i.e., packet forwarding) might exist. In addition, issues regarding the significant burden placed on the end host, the capability of supporting a large-scale group, and the effect of unexpected halt of end hosts still remain to be discussed more precisely.

Scattercast [19)] is another type of Application layer Multicast. The basic idea of this approach is similar to that of the AN approach, that is, adding some intelligences into the network could benefit from the support of the network infrastructure. However, instead of integrating such support into routers, Scattercast builds multiple-point delivery as an infrastructure service that leverages support from strategically placed application-layer SCXs (agents). This approach is geared towards large-scale broadcasting services where many advantages, such as the feasibility of provisioning diverse group communication services and the elimination of forwarding state maintenance problem associated with traditional IP Multicast routers, are introduced. However, the deployment of a flat single-level overlay network structure and a mesh-first [20)] routing protocol (i.e.,

Gossamer [19]) could be difficult to achieve the support of truly large audiences. The necessity of a large amount of exchange of SCX-to-SCX routing and group management information could probably limit its scalabilities.

The AVD mechanism that we have developed is based on both Su Wen's approach [8] and REUNITE [9], which performs the underlying Unicast forwarding to achieve Multicast-like service. Su Wen proposed the novel idea of using the AN technology to achieve a sophisticated group communication service. In Su Wen's centralized model, in addition to a logical Multicast delivery tree, the server needs to maintain the status of the branch points (i.e., the router where the packet duplication function is to be implemented) and the children of those branch points (BP). The Multicast delivery tree is constructed using two building blocks: Ephemeral State Processing (ESP) and Lightweight Packet Processing modules. To find an BP, the server performs the ESP process, which is performed "hop by hop" across the delivery tree to the new JOIN user, for a minimum of three times. Though highly flexible, the recursive processes of this approach could cause significant overheads in both server and router processing power.

In addition to using the AN technology, REUNITE is the first approach that utilizes only the underlying Unicast forwarding to achieve Multicast service. REUNITE proposes a very simple protocol to eliminate Multicast forwarding state at non-branching routers. It constructs a Multicast delivery tree using a Multicast Forward Table (MFT) and a Multicast Control Table (MCT). There are many advantages to using REUNITE, however, the delivery path constructed in this approach may not be the optimal one (i.e., the shortest), given the presence of asymmetric routing networks [10]. Recent research result on the Internet Topology [11] shows that the percentage of the AS path asymmetry is around 50〜70%. That is, the effect of the AS path asymmetry characteristic on REUNITE needs to be further discussed. In addition, we believe that approaches based on the AN technology are more flexible than REUNITE. That is, new services can be more easily implemented to the AN-based approaches without needing time-consuming, hardware-based modification, like that required in REUNITE.

6. Conclusion

In this paper, we have presented a novel group communication service, one based on a very simple tree construction technique and an efficient video merging algorithm. One of the main purposes of our work is to demonstrate the effectiveness in sharing data and in delivering video content among users arriving at different times. In our analytic model and simulations, we observed that the AVD mechanism significantly reduces the total network delivery cost, the video server bandwidth requirement, and the average link load; even the Active Routers are sparsely deployed and the user arrival times are diverse. In addition, we have demonstrated that the constructed AVD tree can achieve the same efficiency as that of IP Multicast tree.

References

1) Dan, A., Sitaram, D. and Shahabuddin, P.: Scheduling policies for an on-demand video server with batching, *ACM Multimedia*, USA, pp.15–23 (1994).
2) Hu, A.: Video-on-Demand Broadcasting Protocols: A Comprehensive Study, *IEEE INFOCOM2001*, USA, pp.508–517 (2001).
3) Chu, Y.-H., Rao, S. and Zhang, H.: A Case for End-System Multicast, *ACM SIGMETRICS '00*, Santa Clara, CA, pp.1–12 (2000).
4) Wetherall, D.J., Guttag, J.V. and Tennenhouse, D.L.: ANTS: A Toolkit for Building and Dynamically Deploying Network Protocols, *IEEE OPENARCH'98*, pp.1–12 (1998).
5) http://www.multicasttech.com/status/
6) Pendarakis, D., Shi, S., Verma, D. and Waldvogel, M.: ALMI: An application level multicast infrastructure, *3rd USNIX Symposium on Internet Technologies and Systems* (*USITS '01*), pp.49–60 (2001).
7) Francis, P.: Yoid: Your Own Internet Distribution, Technical Report, ACIRI, http://www.aciri.org/yoid (2000).
8) Wen, Su, Griffioen, J. and Calvert, K.L.: Building Multicast Services from Unicast Forwarding and Ephemeral State, *Computer Networks 38*, pp.327–345 (2002).
9) Stoica, I., Ng, T.S.E. and Zhang, H.: REUNITE: A Recursive Unicast Approach to Multicast, *IEEE INFOCOM2001*, USA, pp.1644–1653 (2001).
10) Costa, L.H.M.K., Fdida, S. and Duarte, O.: hop-by-hop multicast routing protocol, *ACM SIGCOMM' 2001*, USA, pp.249–259 (2001).
11) Amini, L., Shaikh, A. and Schulzrinne, H.:

Issues with Inferring Internet Topological Attributes, http://www.research.ibm.com/mda/publications/ITcom02.pdf
12) Postel, J.: Internet Protocol, *STD 5*, RFC 791 (1981).
13) http://www.akamai.com/
14) Chalmers, R. and Almeroth, K.: Modeling the branching characteristics and efficiency gains of global multicast trees, *IEEE INFOCOM2001*, USA, pp.77–86 (2001).
15) Decasper, D. and Plattner, B.: DAN: Distributed Code Caching for Active Networks, *IEEE INFOCOM 1998*, USA (1998).
16) Cai, Y. and Hua, K.A.: Sharing Multicast Videos Using Patching Streams, *Journal of Multimedia Tools and Applications*, Kluwer Academic Publishers, MTAP236-01.
17) http://www.opnet.com/
18) Chesire, M., Wolman, A., Voelker, G. and Levy, H.: Measurement and Analysis of a Streaming Media Workload, *3rd USITS*, USA, pp.1–12 (2001).
19) Breslau, L., Cao, P., Fan, L., Phillips, G. and Shenker, S.: Web Caching and Zipf-like Distributions: Evidence and Implications, *IEEE INFOCOM1999*, USA, pp.126–134 (1999).
20) Chawathe, Y.: Scattercast: An Architecture for Internet Broadcast Distribution as an Infrastructure Service, Ph.D. Thesis, U.C. Berkeley (Dec. 2000).
21) Banerjee, S. and Bhattacharjee, B.: A comparative study of application layer multicast protocols, http://www.cs.umd.edu/users/suman/publications.htm

(Received May 21, 2003)
(Accepted December 2, 2003)

Chih-Chang Hsu is a Ph.D. student at the Research Center for Advanced Science and Technology, the University of Tokyo. He was a research fellow with the TAO Yamagata Video Research Center for 2001–2003, where he was involved in simulating and designing an efficient delivery mechanism for large-scale video archival and retrieval systems. His current research interests are in the areas of active networks, video multicasting and scalable P2P network.

Terumasa Aoki is a lecturer with the Research Center for Advanced Science and Technology, the University of Tokyo. He received his B.S., M.E. and Ph.D. in Information and Communication from the University of Tokyo, Japan in 1993, 1995, and 1998 respectively. His current research interests are in the fields of Terabit IP router, access control of Gigabit LAN/WAN, next-generation video conferencing system, high efficient image coding and management of digital content copyrights. He has received various academic excellent awards such as the 2001 IPSJ Yamashita Award, the FEEICP Inose Award for 1994, and the other 4 awards.

Hiroshi Yasuda received the B.E., M.E. and Dr.E. from the University of Tokyo, Japan in 1967, 1969, and 1972 respectively. Since joining the Electrical Communication Laboratories of NTT, in 1972, he has been involved in works on Video Coding, Image Processing, Tele-presence, B-ISDN Network and Services, Internet and Computer Communication Applications. He is now Director of the Center for Collaborative Research (CCR) of the University of Tokyo. He had served as the Chairman of ISO/IEC JTC1/SC29 (JPEG/MPEG Standardization) from 1991 to 1999. He had also served as the President of DAVIC (Digital Audio Video Council) from September 1996 to September 1998. He received 1987 Takayanagi Award, 1995 the Achievement Award of EICEJ, 1995–1996 The EMMY from The National Academy of Television Arts and Science and also 2000 Charles Proteus Steinmetz Award from IEEE. He is Fellow of IEEE, EICEJ, and IPSJ, a member of Television Institute.