Overlay Multicast for Video On Demand on the Internet *

Kien A. Hua
School of Electrical Engineering and Computer Science
University of Central Florida
Orlando, FL 32816-2362, U.S.A
kienhua@cs.ucf.edu

Duc A. Tran
School of Electrical Engineering and Computer Science
University of Central Florida
Orlando, FL 32816-2362, U.S.A
dtran@cs.ucf.edu

Roy Villafane
School of Electrical Engineering and Computer Science
University of Central Florida
Orlando, FL 32816-2362, U.S.A
villafan@cs.ucf.edu

Keywords

Overlay multicast, Video on demand, Patching.

ABSTRACT

Patching is an attractive technique for building efficient video-on-demand systems. However, since Patching assumes the existence of IP Multicast, implementing Patching on the current Internet is a challenging task because the Internet is based on IP Unicast only. In this paper, we propose an overlay technique called *Vcast* for enabling multicast services on the application layer, as a way to support the implementation of Patching. Unlike earlier overlay multicast schemes, Vcast does not introduce any global topology for the overlay and therefore avoids control overhead to maintain it. Vcast can configure itself adaptively to the changing network traffic, and is tolerant to failures prone to happen frequently in dynamic environments such as the Internet. In addition, Vcast provides load balancing among network nodes by employing the Round-robin approach in selecting delivery paths for clients.

1. INTRODUCTION

We are interested in the problem of streaming bandwidth-intensive media on demand from a source to a large quantity of receivers on the Internet. The simplest solution dedicates an individual connection to stream the content to each receiver. This method consumes a tremendous amount of costly bandwidth and leads to an inferior quality stream for the receiver, making it nearly impossible for a service provider to serve quality streaming to large audiences while generating profits. IP Multicast [6] could be the best way to overcome this drawback since it was designed for group-oriented applications. For clients requesting the same video, the server creates a multicast address and sends a stream of the video to all of them on a multicast tree. An advantage is the optimal use of network bandwidth since a single copy of video data is delivered on each physical link to all the multicast clients. Another advantage is that they all share a single stream emanated from the server, hence a significant saving on the server bandwidth. On the other hand, since clients participating in a multicast group receive the same data at any point of time, the server has to create new multicast addresses for late-coming clients in order to provide them with the full content of the requested video. In reality, client requests usually reach the server at different times. The server may have to create a large number of separate multicasts, and therefore, the bandwidth bottleneck remains severe.

Patching [8, 3] is a remedy for the aforementioned limitation of IP Multicast. In this technique, when a new client requests a video, it joins the multicast group that was created most recently for the same video, and receives data destined for this group. Since a portion of beginning video data cannot be delivered from that multicast, the client gets it from the server over a unicast, or a "patching" stream. A proper buffering algorithm is pursued at the client to synchronize the playback of these two data streams (i.e., multicast stream and patching stream). An obvious advantage of Patching is that instead of creating a long-lasting multicast stream for the new client, the server just creates a short-term unicast stream. Therefore, Patching is highly efficient in reducing the demand on server bandwidth while providing true VOD services to the clients. Due to its simplicity in concept and efficiency in performance, Patching has been used in a large number of works (e.g., [14, 7, 1, 2, 13]) that deal with various fundamental issues relevant to VOD systems.

We are interested in implementing Patching on the Internet. Since Patching assumes the existence of IP Multicast, the main barrier in our problem is due to the infeasibility of deploying IP Multicast on the current Internet. Therefore, we examine building multicast services on the application layer, which assumes only the IP Unicast protocol. The basis for our work is to install a number of machines at the end-system nodes, which play as application-layer routers. These nodes and the server are organized into a logical overlay whose topology is built on top of the Internet. Video data are transmitted on unicast paths involving the overlay nodes only. A new client joins an existing video stream

* This research is partially supported by US National Science Foundation under grant ANI-0088026

SAC 2003 Melbourne, Florida, USA

by connecting to an overlay node appearing on the delivery path to an earlier client. This approach is a kind of *overlay multicast* which has spurred many recent works on providing group-oriented services on the Internet [4, 5, 9, 12, 10, 11, 15, 17].

One has to cope with several issues in designing a good overlay multicast scheme. Firstly, transmissions of data from one end-system to another on the overlay may consume network bandwidth inefficiently and prolong end-to-end delay. Secondly, since an overlay path is created without knowing about the underlying network routers, some routers may become congested if having to send out many video streams. It is desirable to distribute the network traffic evenly among the network routers. Most existing overlay multicast schemes do not mention this issue. Lastly, since the overlay topology should be dynamically adjusted according to changes in underlying traffic, overlay state updates/probes may need to be propagated across the network, which results in significant communication overhead and in high processing complexity at each overlay node. In this paper, we propose *VCast* (*V*ideo Overlay Multi*cast*), a simple yet efficient overlay multicast scheme to support Patching and to address the above issues.

The remainder of this paper is structured as follows. In Section 2, we present the network model used in Vcast. In Section 3, we present the protocol details of VCast. In Section 4, we report performance results from our simulation study. Finally, we give concluding remarks in Section 5.

2. VCAST OVERLAY MODEL

The Vcast system is an overlay network consisting of three components: server node, client nodes, and internal nodes. The *server node* is the source of video streams and assumed to remain up all the time. *Client nodes* are the nodes where clients request for service. They can be anywhere on the Internet. A client always sends its request to the server node. *Internal nodes* are machines installed across the Internet to act as the backbone nodes of the overlay. They are responsible for relaying video streams to the clients. The number of internal nodes may change over time due to addition of new nodes or removal of existing nodes. Internal nodes and the server node are called *overlay nodes*.

If no internal node is installed, using a dedicated stream for each requesting client seems to be the only simple way to implement Patching on the Internet. This is obviously not a desirable solution. By employing internal nodes in the network fabric, Vcast offers a better solution. Indeed, when a client requests a video stream, the data may pass through a number of internal nodes before finally reaching the client. An internal node involving in a delivery of a video stream is called a *relay node* of that stream. Subsequent clients instead of getting a video stream from the server are able to get it from these relay nodes. Clearly, no more server bandwidth is consumed in this case. One might argue that the complexity and cost arise from deploying and maintaining the internal nodes. Sharing the same insight in [10], we believe that one-time hardware costs do not drive the total cost of the system. Maintenance costs are cut by simplifying node deployment. As shown later in the rest of this paper, Vcast provides a highly scalable solution while being kept simple and lightweight.

We do not establish any topology for connecting overlay nodes. The delivery path for a client is built on demand and adjustable over time based upon the underlying traffic or node failure. This path can be a direct path from the server or a path consisting of several internal nodes.

Table 1: Service Table

streamID	parent	childList	rate
Sport News	B	E, F, G	100
Titanic	H	B, E, A, K	40
Tonight Show	A	C	2048

An overlay node X is assigned a unique identifier by the server when first added to the overlay. Maintained at X are a *node list* and a *service table*. The node list contains the ID of every other overlay node. This is updated only when an internal node is removed or a new internal node is added. The service table has four attributes $streamID$, $parent$, $childList$, and $rate$. Each row corresponds to an active stream $streamID = V$ currently coming to node X. X receives this stream from node $parent(X, V)$ and forwards it to all nodes in $childList(X, V)$ on the corresponding unicast paths. The value of $rate(X, V)$ is the estimated arriving data rate (in KB/s) at node X and computed for a particular stream V. This value, initially set to zero, is observed locally and updated periodically without any communication across the network. Table 1 gives a sample look of this table.

Given the above structure, an overlay node only knows its parents, each for an individual stream, and the corresponding children. Even though computed locally, the service table reflects the underlying network traffic well. By looking at the entry corresponding to a video stream V, a high value of $rate(X, V)$ indicates that X should keep its connection with the parent $parent(X, V)$. On the contrary, a low value indicates congestion or disconnection in the upstream. A criterion for Vcast to ensure continuous client playback is to keep the value of $rate(.,.)$ above a threshold T_{min}. The value of T_{min} should equal $\lambda + \Delta$, where λ is the minimum arrival data rate that a client must sustain for a conceivable playback and $\Delta > 0$ is conservatively chosen to ensure that the additional delay due to overlay self-adjustment does not cause client playback to be jittery.

By joining an existing multicast tree, a client can only watch the video from the point when the request is generated. For the missing portion, the client gets it from the server on a direct unicast path as in the original Patching design [8]. For the rest of the paper, we focus on how to maintain overlay multicast trees, which is the most challenging task toward our purpose of implementing Patching on the Internet.

3. VCAST PROTOCOL DESCRIPTION

In this section, we describe how Vcast functions on the following events: a client requests a video, a client quits (purposely or accidentally), a new internal node is added to the overlay, an internal node is purposely removed from the overlay, and an internal node fails. We also present how Vcast is self-organized to adapt with changing underlying network traffic.

3.1 A New Client Requests

Servicing a client request has two phases: (1) finding a

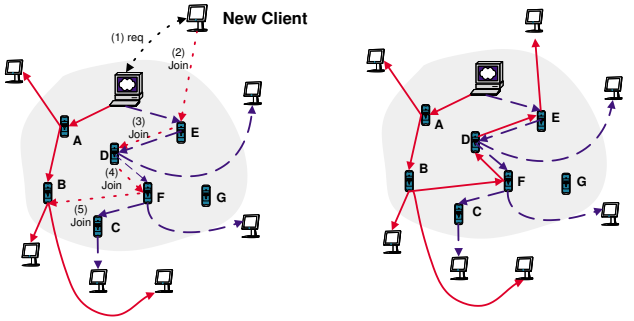


Figure 1: A Client Requests A Video

representative for the client, and (2) having the representative find a path for the client to get data. These are explained as follows.

In the first phase, a client node C sends a **req** request to the server node which is the only node on the overlay C is aware of. In response, the server returns the ID of an internal node X to the client. This node is called the *representative* of the client. The selection of a representative follows a *round-robin* policy. Specifically, if node Y is the representative for the last requesting client, node X will be the next node of Y in the node list maintained at the server.

In the second phase, the client C contacts its representative X and asks it to find a relay node for C to join and get data (see Section 3 for definition of relay node). If the contact fails (e.g., due to the failure of X), the client has to roll back to phase 1 and contacts the server again to get a new representative. Without loss of generality, suppose that the contact succeeds. If X is not currently a relay node of the requested stream, it follows the steps below:

Step 1: Find the next node Y in the node list maintained at X based on the round-robin manner. For instance, if Z is a node selected for the last client request, then Y will be the next node of Z in the node list maintained at node X.

Step 2: X sends a **join** request to Y. This **join** has a "path" header containing the identifier of every node visited. The use of the path header avoids revisiting any overlay node. As a result, if a path is used later for delivery, it will be loop-free. To have an idea of how large this header is, suppose the identifiers can be represented by 10 bits (1024 overlay nodes), the maximum size for the path header would be $1024 \times 10 = 10240$ bits or 1280 bytes. Since the number of internal nodes is not very large, the path header size should have an infinitesimal impact on the network traffic.

If Y is not yet a relay node of the requested stream, Y will do the same procedure as X does above. This process repeats until the **join** reaches a relay node. In response to the **join** request, this relay node sends a **setup** message followed by the video stream, down the delivery path specified in the path header of the received **join**. As this **setup** arrives at overlay nodes along the delivery path, they update their service table accordingly.

An example illustrating the requesting event is given in Fig. 1. The left-sided diagram shows the construction of the delivery path and the right-sided diagram shows the new multicast tree. In this example, the server is currently delivering two video streams, V_{solid} and V_{dashed} on two multicast trees consisting of solid branches and dashed branches, respectively. At this time, a new client contacts the server to request V_{solid}, and is informed that E is the representative. The client then sends a **join** to E. Since E is not a relay node for video V_{solid}, E forwards **join** to D selected based on the aforementioned round-robin approach. Similarly D forwards **join** to F which in turns forwards **join** to B. Since B is a relay node for video stream V_{solid}, the delivery path $B - F - D - E$ is created destined for the new client.

In our scheme, the selection of the representative and path nodes is based on round-robin manner. Consequently, the network load is evenly distributed among the overlay nodes and inter-node paths. This helps prevent network congestion and overlay node bottleneck. Our performance evaluation results, to be presented in Section 5, show that network load is also balanced among underlying network nodes.

3.2 A Client Quits

If a client C watching video stream V wants to cancel the on-going service purposely, it explicitly reports the wish to its representative X. If a client quits the service accidentally, X must have some way to detect. Vcast enables this detection by using a "heartbeat" mechanism at the representative to check the aliveness of its clients. Without loss of generality, suppose that X knows not to forward data to client C anymore. There are two possibilities for node X to consider:

Case 1: At least another downstream (client or internal) node are receiving video stream V: This case is handled simply by removing C from $childList(X, V)$ in the service table. As a result, future data will not be sent to client C.

Case 2: No downstream node is receiving video stream V: In this case, node X is no longer needed on the multicast tree for video V and can be removed from the tree. For this purpose, X removes the entry corresponding to video V from the service table, and sends a **quit** packet to its parent Y on the tree. Y removes X from $childList(Y, V)$ and carries out the same tear-off policy as node X does.

Clearly, the above strategy saves network bandwidth. If an internal node is not needed to forward data to any other node, excluding it from the multicast tree avoids transmitting redundant information. A node when not part of any multicast tree becomes idle and is not involved in any network communication. To this extent, Vcast is distinguished from earlier overlay multicast schemes in which every overlay node always needs to communicating through the network to re-evaluate its position on the overlay.

3.3 Addition of An Internal Node

This procedure is simple in Vcast. When a new machine wants to be an overlay node, it sends an **on** request to the server. In response, the server sends back a new unique identifier for the new node together with the list containing the ID of every existing overlay node. The server also informs every other internal node of this new identifier. This ensures that any overlay node is aware of the existence of any other node.

3.4 Removal of An Internal Node

An internal node X wants to remove itself from the overlay by sending an **off** message to every other overlay node to inform the removal of X. In response, each recipient removes X from its node list. In particular, each parent Y of X, according to some on-going video stream, removes X from

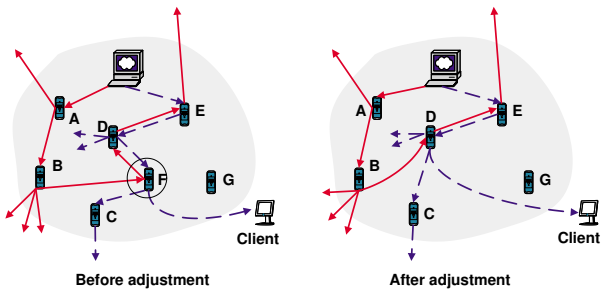


Figure 2: Removal of An Internal Node

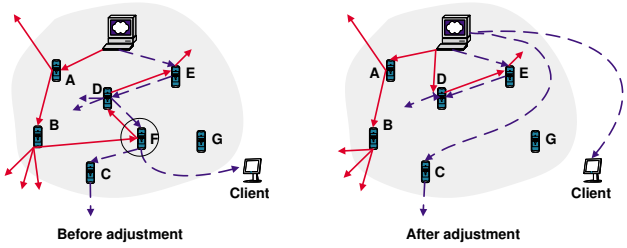


Figure 3: Failure of An Internal Node

the child list. This will stop future data from forwarded to X. Furthermore, for each stream X has been receiving, X tells its corresponding children to reconnect to the parent Y of X to get the remaining data. An example is given in Fig. 2 to illustrate the removal procedure. There are currently two multicast trees (dashed and solid) for two different streams. Since node F decides to be off, node C and a client node have to reconnect to node D which was previously their grandparent. Node D is also a child of node F, D also needs to reconnect to its corresponding grandparent B.

Following the removal strategy, the grandparent (e.g., node D) might become a bottleneck if accepting too many reconnecting nodes. This can be resolved later by reconnecting some of its child nodes to other relay nodes. We will discuss this in Section 3.6.

3.5 Failure of An Internal Node

Children of a node X can detect its failure by observing the estimated arrival data rates of the video streams from X. When a child node detects these data rates drop to and stay at the zero level for a period of time, this node requests to connect itself to the server. In response, the server picks up the service left off by X, and informs all the internal nodes to remove X from their node list, and corresponding child lists if the internal nodes are parents of X so that subsequent data will not be forwarded to this failed node. We illustrate this procedure in Fig. 3. In this example, node F goes down accidentally. All three of its children C, D and a client node have to reconnect to the server for the remainder of the service. While node D reconnects through a solid line, C and the client reconnect through a dashed line. This is due to the fact that D was connected to the solid-line tree before F failed, while C and the client were connected to the dashed-line tree. We recall that we use a solid line and a dashed line to denote video streams for two different videos.

We note that when a node fails, its children could also reconnect to the grandparent node (instead of the server). Although this approach saves server bandwidth, each node would have to keep more information and the protocol would become more complex. Since a node does not fail frequently, we opt for the simpler strategy described previously. We shall discuss shortly how to reduce the server load should the server happen to serve too many reconnecting nodes directly.

3.6 Dynamic Change of Delivery Paths

Since all delivery paths are overlay paths, their quality may change over time. For instances, congestion of a physical link negatively affects every path containing it; a node in a multicast tree may become a bottleneck as a result of removing an overlay node (as discussed in Sections 3.4 and 3.5). VCast is able to adjust the topology of delivery paths to address such problems. In particular, this is achieved without the overhead of exchanging status information among the overlay nodes as in most existing overlay multicast schemes.

We consider a node X. X monitors the value of $rate(X, V)$ for every stream V currently coming through X. If $rate(X, V) < T_{min}$, node X considers three possibilities: (1) the parent Y (i.e., $Y = parent(X, V)$) fails, (2) some physical link on the path from Y to X is congested, and (3) some physical link on the upstream paths (before reaching Y) is congested. We discuss how Vcast handles these situations below.

First, node X sends a message to Y reporting quality degradation. The message includes the value of $rate(X, V)$. If Y is dead, X will not receive any reply. In this case, X has to find a new parent node to reconnect itself to the multicast tree. This procedure is the same as in the case of a node going down accidentally as discussed in Section 3.5.

Suppose that Y is alive and has been receiving V from node Z (i.e., $Z = parent(Y, V)$). Y checks the value of $rate(Y, V)$, which subsequently leads to two cases:

Case 1: $rate(Y, V) \geq T_{min}$: This most likely implies that only the path from Y to X is poor while the upstream paths are acceptable. In this case, X sends a **quit** message to inform Y to remove X from its child list, $childList(Y, V)$. Furthermore, X attempts to reconnect to the multicast tree through its grandparent Z. This policy has two implications. First, if the low incoming data rate at X was caused by severe contention for bandwidth among the outgoing streams at Y, disconnecting X from Y would alleviate this problem. Second, since the arrival rate estimated at Y is above the threshold T_{min}, the bandwidth contention at node Z is not severe. By reconnecting X to Z, the delivery path to X is shortened and the incoming data rate at X is also improved.

Case 2: $rate(Y, V) < T_{min}$: This most likely implies that the slow incoming data rate at X is partially due to the poor data path from Z to Y. In this case, Y needs to report the value of $rate(Y, V)$ to Z and attempts to reconnect to Y's grandparent as X did in the previous case (when $rate(Y, V) \geq T_{min}$). This process is repeated recursively as necessary to reduce the load in the upstream.

To illustrate how the overlay changes upon performance degradation due to a change in underlying traffic, we give a demonstrative example in Fig. 4. There are currently two multicast trees, dashed-line and solid-line, for two different video streams V_{dashed} and V_{solid}, respectively. Suppose that

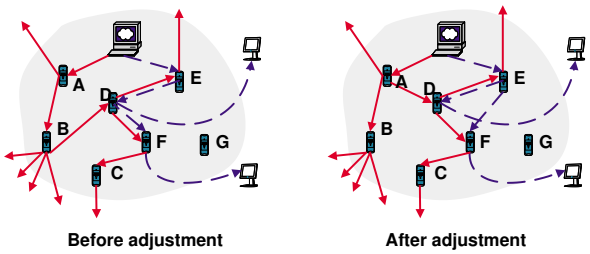


Figure 4: Dynamic Change of Overlay Organization (upward reconnection)

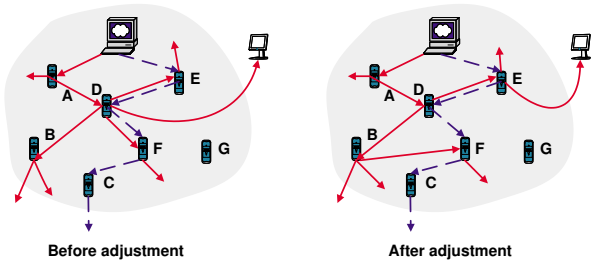


Figure 5: Dynamic Change of Overlay Organization (downward reconnection)

node F detects a degradation in the estimated arrival data rate corresponding to both streams V_{solid} and V_{dashed}, F reports this condition to node D. Node D currently receives V_{solid} from node B and V_{dashed} from node E. Node D finds that its $rate(D,\ V_{solid}) < T_{min}$ (maybe because node B is currently very busy) but $rate(D,\ V_{dashed}) \geq T_{min}$. To reduce the load on B, D reconnects to node A for stream V_{solid}; but node F remains connected to D. For V_{dashed}, however, node F disconnects from D, and reconnects to E since it is lightly loaded.

A possible case happens when a node X, directly connected to the server, is experiencing a slowdown in the incoming data from the server. Since X cannot go upward any further, it has to find a new parent in the downstream. This is achieved by sending a reconnection request with a "down" indicator to the server. In response, the server selects one of its children, say Z in a round-robin manner, and informs X that Z is X's new parent. If node X later experiences degradation in the arrival data rate again, it will reconnect to a child of Z selected in a round-robin manner as before. This process of downward reconnection will eventually reach a leave node of the multicast tree. At this time, X switches the reconnection indicator to "up"; and its reconnection process starts to occur in the upward direction.

An example of downward reconnection is given in Fig. 5. Here, both node F and the client node are experiencing a slowdown in the incoming data from D. To correct the problem, F reconnects downward to B, the next child node of D according to the round-robin order. Similarly, the client node reconnects to node E, the next child node of D based on the same round-robin mechanism.

4. PERFORMANCE EVALUATION

We used the GT-ITM generator [16] to create a 600-node transit-stub graph as our underlying network topology. The overlay nodes are selected in two approaches: random-based and backbone-based. In the random-based approach, overlay nodes are chosen randomly among underlying nodes. In the backbone-based approach, overlay nodes are randomly chosen among the underlying transit nodes and the remaining overlay nodes, if any, are randomly chosen among the underlying stub nodes. Since overlay selection is based on randomness, a single choice of overlay nodes is not sufficient to illustrate the performance accurately. Therefore, for each type of overlay selection, simulation was run multiple times and most performance outputs were collected as the averaged values over these runs. Although we studied both random-based and backbone-based location algorithms, this section only reports the results for the random-based located overlay because we observed that the results for the other location algorithm behaved similar.

We investigate the following metrics for VCast: Link Stress, Server Stress, Network Waste, Access Distance, and Node Stress Distribution. *Link stress* is the average number of duplicated packets carried by a physical link. This measures the effectiveness of overlay in distributing network load across physical links. The link stress of IP Multicast is always 1 since only a single copy of any packet is sent over a physical link. Link stress also affects the potential bandwidth availability from the source node to the internal nodes. A high stress on a physical link reduces the bandwidth availability on that link. *Server stress* is the average stress on a physical link involving the server node. This measures the severity of the server bottleneck. The server stress of IP Multicast is always 1. *Network waste* is computed as the ratio of network load imposed by overlay to a lower bound estimate of IP Multicast network load. The network load is defined as the total number of times physical links are used. *Access Distance* is the average length in hops of the delivery path between the representative of a client and the overlay node for the representative to join to get data for the client. This metrics influences how fast the client can start receiving data. *Node Stress Distribution* reflects how balanced the service load is distributed among network nodes.

Overcast [10] seems most relevant to our work. It would be better if we could compare Vcast to Overcast. However, we realized that simulating Overcast in our environment was not straightforward due to technical details, and might end up with bias results. Therefore, we opted to use Pure Unicast, in which every internal node is connected to the server node on the direct shortest unicast path, as the base reference in our study. Pure Unicast approximates the current approach to multicasting on the Internet.

We assume that the server node has only one video. Hence, there is always at most a multicast tree on the overlay. Initially, all the overlay nodes are idle and no tree is built. Requests were generated and delivery paths were created accordingly until a tree was built covering all overlay nodes. At this point, we computed the metrics above. Obviously, the overlay built based on Pure Unicast has a very severe server bottleneck problem since every internal node is directly connected to the server node. As exhibited in Fig. 6(a), this problem becomes worse as the overlay size increases. In contrast, the server node in Vcast scales well with the overlay size. This study shows that even though Pure Unicast is easy to deploy, its severe bottleneck server problem would be the vital hurdle to the performance efficiency. Vcast does not have this hurdle.

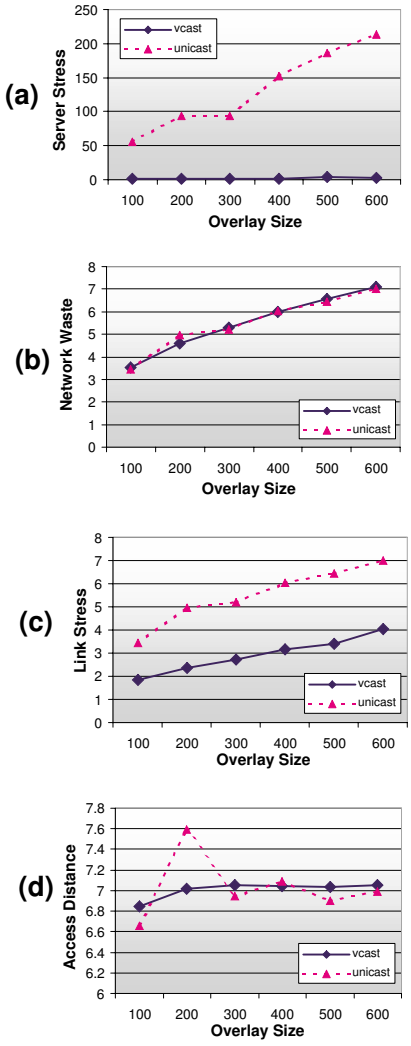


Figure 6: Effect on (a) server stress, (b) network waste, (c) link stress, and (d) access distance

Since the delivery paths in overlay multicast schemes are end-system to end-system on top of the underlying network, they must introduce additional load on the network traffic. The results in Fig. 6(b) show that Vcast increases the network waste with a factor of 3-7 times as compared to IP Multicast. This is not that severe since other good schemes such as [10] provided close numbers. Even though Vcast and Pure Unicast have similar network waste, Vcast distributes this network waste fairly among more physical links. Consequently, it has a lower link stress than Pure Unicast does (Fig. 6(c)). Therefore, Vcast provides better potential bandwidth availability. This should be seen as an advantage since having large potential bandwidth is a desirable feature of any overlay multicast scheme.

Fig. 6(d) exhibits the average distance in hops between the representative of a client and the overlay node that the representative has to join to get the requested video for the client. In a Pure Unicast overlay, this distance is the shortest path between the server node and the representative. On average, the access distance for Vcast is 7 network nodes long, approximately the same as that for Pure Unicast. This demonstrates that the joining of a new client is as fast as sending and receiving data from the server. Given the previous results showing that Vcast has a lower link stress, a lower server stress, it is convincing that the path for sending data from a tree to the client in Vcast is a lot better than the path provided to the client using Pure Unicast overlay.

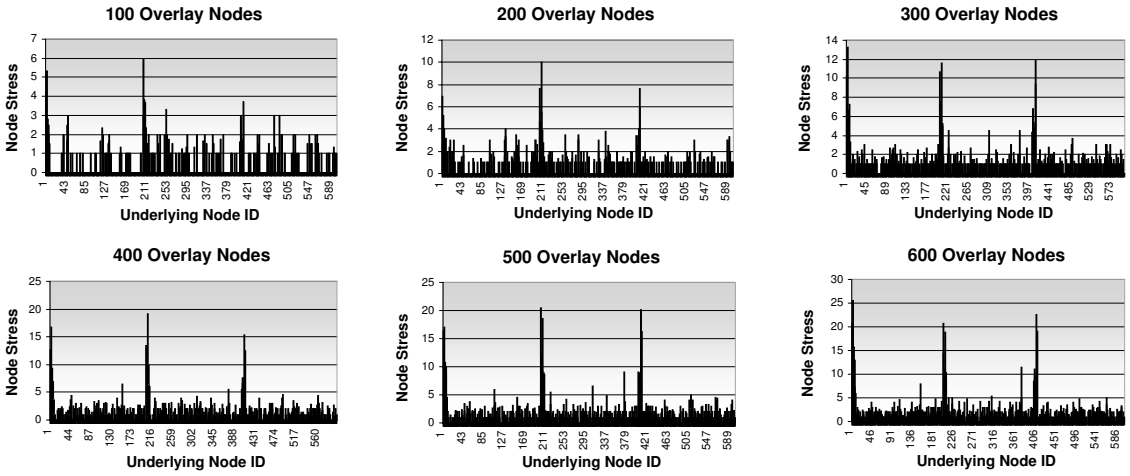
We studied how network load incurred by Vcast is distributed among the underlying network by estimating the node stress at each network node. A good scheme should have uniform node stress distribution to reduce bottleneck in the network. The stresses for all 600 network nodes for different overlay sizes are drawn in Fig. 7. It is noteworthy that a few nodes have high node stresses. These nodes are actually the inter-domain nodes that most of the delivery paths have to get through. No overlay multicast topology could avoid these hot spots. Putting these nodes aside, the node stress is highly balanced among the other network nodes (e.g., about 5 times better than Pure Unicast in terms of standard deviation). Therefore, load balancing is achieved by Vcast. This is explainable since Vcast follows the Round-robin manner to distribute the service load over all overlay nodes which are randomly located in the network.

In the second simulation scenario, we allow a number of nodes to be turned off. p is the probability that a node is off due to a failure. We computed the metrics for the tree after a number of nodes are switched to the off state, and for the tree after being self-organized according to the algorithm presented in Section 3.6. For simplicity and due to the difficulty of modeling the underlying network traffic, we assumed that the arrival data rate at a node is only affected by the number of outgoing streams at the parent node. Therefore, children of a node having the most outgoing streams should be adjusted first in each self-organization phase. Fig. 8(a, c, d) found that the overlay before refinement does not change much on link stress, access distance, and network waste. Exceptionally, when most of the node removals are due to failure ($p = 0.1$), the server stress increases significantly (Fig. 8(b)). This is anticipatable since when a node fails, all of its children are reconnected to the server. However, after readjusting the burdened-server overlay, the server stress is reduced to approach to that of a normally constructed Vcast overlay tree. This study shows the potential adaptability of Vcast to changes in network status. Our preliminary prototype system also reports similar performance patterns on this adaptability of Vcast.

To investigate whether the refined overlay's service load is better balanced over network nodes, we computed node stress distribution for the refined overlay and compared it with that of the overlay before refinement. Due to the removal of nodes, the overlay load may become skewed among network nodes. However, after the self-adjustment, the node stress distribution is a lot more uniform. The results for an example scenario are drawn in Fig. 8(e, f). Illustrated in this figure is the overlay with 200 nodes removed with $p = 0.1$ from 300 nodes initially. This study shows that Vcast has a potential in keeping the overlay service load well balanced without being significantly affected by the negative changes in network traffic.

5. CONCLUSION

We have proposed a simple yet efficient overlay multicast scheme, called Vcast, as a means for putting Patching into



Num of Overlay Nodes	100	200	300	400	500	600
Vcast (Std Dev)	0.82	1.17	1.5	2.02	2.38	2.73
Pure Unicast (Std Dev)	5.85	3.27	7.7	11.99	12.49	10.12

Figure 7: Node Stress Distribution with Various Overlay Sizes

practice. The motivation behind our work is due to the lack of a deployment of multicast VOD services on the Internet while an efficient technique such as Patching assumes a currently infeasible IP Multicast. Even though different schemes on application-level multicast have been proposed for various applications, Vcast has distinguishing and desirable features:

- Topology-less: Vcast does not need to propagate state updates/probes across the network, thus avoiding the complexity at each overlay node for doing so.
- Load balancing: Even though important, this issue is not mentioned in earlier work on overlay multicast. One of the goals in designing Vcast is to give every network node an equal chance to service client requests, hence guaranteeing service load balancing.
- Lightweight self-configurable: Vcast allows adjustments of on-going delivery paths to reflect the dynamism of underlying network traffic. Other techniques also have this feature, however Vcast does not require heavy (e.g., periodic) inter-node communication to measure the congestion or the aliveness of a path.

Since Vcast is built on an overlay, it must introduce additional overhead as compared to IP Multicast. However, our simulation study found that Vcast incurs a low penalty. When compared to Pure Unicast, a centralized organization of the overlay, Vcast provides the same amount of network load but a significant reduction on server bottleneck, physical link and node stress. The delay for a client to access an existing multicast is also less.

6. REFERENCES

[1] E. Bommaiah, K. Guo, M. Hofmann, and S. Paul. Design and implementation of a caching system for streaming media over the internet. In *Proceedings of the Sixth IEEE Real-Time Technology and Applications Symposium*, May 2000.

[2] M. K. Bradshaw, B. Wang, S. Sen, L. Gao, J. Kurose, P. Shenoy, and D. Towsley. Periodic broadcast and patching services - implementation, measurement and analysis in an internet streaming video testbed. In *Proc. of ACM Conference on Multimedia*, Canada, September 2001.

[3] S. W. Carter and D. D. E. Long. Improving bandwidth efficiency of video-on-demand servers. *Computer Networks and ISDN Systems*, 31(1):99–111, March 1999.

[4] Y. Chawathe, S. McCanne, and E. Brewer. An architecture for internet content distribution as an infrastructure service. Unpublished work, February 2000.

[5] Y.-H. Chu, S. G. Rao, S. Seshan, and H. Zhang. Enabling conferencing applications on the internet using an overlay multicast architecture. In *ACM SIGCOMM*, San Diego, CA, August 2001.

[6] S. Deering. Host extension for ip multicasting. *RFC-1112*, August 1989.

[7] C. Griwodz, M. Zink, M. Liepert, G. On, and R. Steinmetz. Multicast for savings in cache-based video distribution. In *Proc. of ACM/SPIE Multimedia Comuting and Networking*, San Jose, USA, 2000.

[8] K. A. Hua, Y. Cai, and S. Sheu. Patching: A multicast technique for true video-on-demand services. In *Proc. of ACM MULTIMEDIA*, pages 191–200, Bristol, U.K., September 1998.

[9] S. Jain, R. Mahajan, D. Wetherall, and G. Borriello. Scalable self-organizing overlays. Technical report, Washington University, 2000.

[10] J. Jannotti, D. K. Gifford, and K. L. Johnson. Overcast: Reliable multicasting with an overlay network. In *USENIX Symposium on Operating System Design and Implementation*, San Diego, CA, October 2000.

[11] J. Liebeherr and M. Nahas. Application-layer multicasting with delaunay triangulations. In *Global*

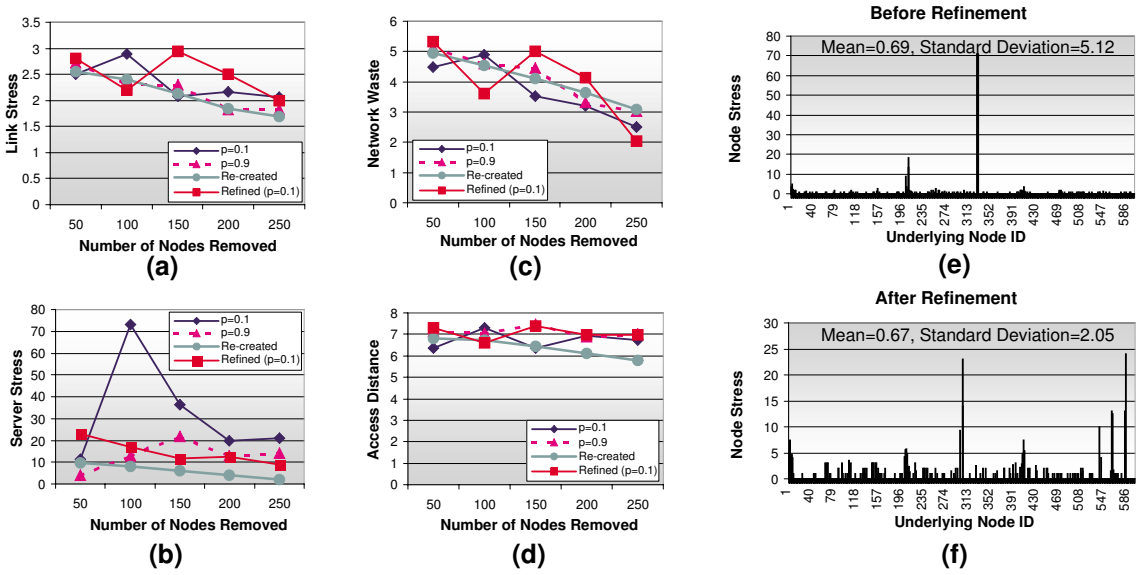


Figure 8: Overlay Performance Before and After Reconfiguration. "Re-created" represents the overlay with the same number of remaining nodes, which is constructed normally according to Vcast overlay building policy

Internet Symposium, IEEE Globecom, 2001.

[12] D. Pendakaris and S. Shi. ALMI: An application level multicast infrastructure. In *USENIX Symposium on Internet Technologies and Systems*, Sanfrancisco, CA, March 26-28 2001.

[13] S. Ramesh, I. Rhee, and K. Guo. Multicast with cache (mcache): An adaptive zero-delay video-on-demand service. In *Proc. of IEEE INFOCOM*, San Diego, USA, 2001.

[14] S. Sen, L. Gao, J. Rexford, and D. Towsley. Optimal patching schemes for efficient multimedia streaming. In *Proc. of IEEE NOSSDAV*, NJ, USA, June 1999.

[15] D. A. Tran, K. A. Hua, and T. T. Do. Zigzag: An efficient peer-to-peer scheme for media streaming. In *To appear at IEEE INFOCOM*, San Francisco, CA, March-April 2003.

[16] E. W. Zegura, K. Calvert, and S. Bhattacharjee. How to model an internetwork. In *Proceedings of IEEE Infocom*, San Francisco, CA, 1996.

[17] S. Q. Zhuang, B. Y. Zhao, and A. D. Joseph. Bayeux: An architecture for scalable and fault-tolerant wide-area data dissemination. In *11th ACM/IEEE NOSSDAV*, New York, June 2001.