Virtual Active Network for Live Streaming Media Delivery

Wen-Syan Li* Divyakant Agrawal K. Selçuk Candan Yusuf Akca Murat Kantarcioglu

NEC Laboratories America, Inc. - Silicon Valley
10080 North Wolfe Road, Suite SW3-350, Cupertino, California 95014, USA
Email:{wen,agrawal,candan,yakca,kanmurat}@sv.nec-labs.com

Abstract

The large amount of bandwidth and other resources required to deliver streaming media limits the number of concurrent users. We propose a virtual active network (VAN) architecture for streaming media data delivery over wide area networks. In the proposed architecture, cooperating proxies support multiplexing and delivery of live streaming media. The hierarchical delivery structure is dynamically adjusted based on user population distribution, usage patterns, and network conditions. The proposed system architecture provides (1) reliable and high quality live streaming media delivery; (2) lower server resource requirements at the content provider sites; (3) reduced inter-ISP traffic; (4) application level routing for rapid deployment; and (5) cost-effective media data delivery. To deal with one characteristics of live broadcasting events, burst traffic at the beginning of the events, our system features a unique function that dynamically clusters multiple proxy servers to form a server farm to handle a large number of user login events when needed. Experimental results show that the proposed VAN architecture consistently delivers reliable live data streams using resources within 10 percent of the theoretically possible lower bound. The experimental results also show the effectiveness of our load balance algorithms to handle various user patterns, server capacity, and network congestion events.

Keywords: Streaming media, live broadcasting, content delivery networks, P2P, overlay networks, proxy server, active networks, server farm

1 Introduction

Streaming media delivery and wide-area data dissemination pose significant challenges in the context of current Internet structure. Without appropriate multicasting mechanisms, network routes are quickly congested as a result of the same stream being delivered from the source to its many recipients. Currently, several proposals are underway

*This work was performed when the author was with NEC Laboratories America, Inc. He is currently affiliated with IBM Almaden Research Center and can be reached at wen@almaden.ibm.com.

to mitigate this problem. One approach is to use IP multicasting to build multicast routes from sources to sinks. Although several efforts have been in progress, they are bogged down due to the incompatibility of various network elements of the Internet service providers. As a result, researchers have started to explore alternative approaches, mostly based on building virtual (or overlay) networks on top of the underlying physical network. The idea is to use application level routing and multicasting through logical links between network elements, such as proxies. This approach addresses the incompatibility and interoperability problems at the physical network layer. In this paper, we describe a virtual network architecture for aggregating routes between sources and sinks to dynamically configure a delivery network for live media streams.

In the absence of an appropriate distribution mechanism, each user request results in a connection flow set-up between the origin server and the user. Furthermore, since the current network infrastructure does not pro-actively monitor (1) congestion in the network, (2) servers, and (3) distributions of end user population, media or data streams can suffer from network congestion on delivery paths.

Recently there has been considerable interest in using proxy caching for streaming data delivery [1, 2, 3, 4, 5, 6]. Overlay networks of proxy servers address these concerns in several ways. First, by distributing the load intelligently across alternative sources and by creating multicast hierarchies within the networks, the overlay networks reduce the overall load on each server and congestion on the underlying IP network components. Secondly, by constantly monitoring the network, overlay networks are capable of identifying (and predicting) congested network components and dynamically redirecting streams through less loaded parts of the network. More importantly, since they maintain service agreements with the ISPs and since they oversee the creation and maintenance of multicast hierarchies (with predictable lifetimes), they can use resource reservation protocols that prevent unpredictable fluctuations (congestions and burst) in the underlying network. By introducing intermediary proxies that each packet has to visit, the overlay network can limit the length of individual TCP-routed path segments, thus reducing the unpredictability and variations in packet inter-arrival times significantly.

1.1 Proposed Approach

In this paper, we are proposing an application level distribution network, Virtual Active Network (VAN), for distribution of live streaming data and wide area data dissemination. We describe how application level protocols among network proxies can be used to support efficient distribution of live data streams. VAN is built on top of an overlay of a proxy network deployed strategically in the network infrastructure. A Proxy Network Coordinator (PNC - a logical entity that can be implemented centrally as a single component or in a distributed fashion across multiple components) is used to determine appropriate routes across the proxy network for delivering data streams. Our virtual active network for data distribution is architected at the application layer, which is different from many existing active network or virtual active network systems, such as [7, 8], that are architected in the network/service layers.

Stream data traffic consists of two parts: the data part and the control part. The control part could include splitting and merging of data streams, buffer management, communication between the recipient and the sender for negotiating the transmission speed. The control part must be implemented at the application level, while the data part can be implemented at either the hardware level or the software level. We deploy a network of cooperating proxies that can request for media content and if required, multiplex a connection to serve multiple

clients. Similarly, network proxies use a peering arrangement with other proxies to merge live connections when the workload shrinks. The dynamic adjustment of proxy network is based on parameters including (1) end-user population, (2) geographical distribution of user requests, (3) network conditions, and (4) location and capacity of proxy servers. To the best of our knowledge none of the existing proposed schemes considered maintaining a proxy hierarchy to minimize network bandwidth requirements. Our approach dynamically adjusts the proxy hierarchy to account for varying workloads.

1.2 Contribution of Our Work

Many CDN solutions are commercially available. These solutions (1) treat streaming media as an object to cache at edge caches to deliver to the near by end users[9]; (2) require network operators to change the CDN configuration manually based on system load and user distributions[10]; or (3) support pre-configured and fixed proxy networks[11]. In our solution, the collaborative proxy network hierarchical structure is dynamically adjusted according to the geographical distribution of load and network traffic conditions without human operators. Our solution also has an advantage of ease of deployment since the routing scheme of our solution is based on application level functions rather than network level functions.

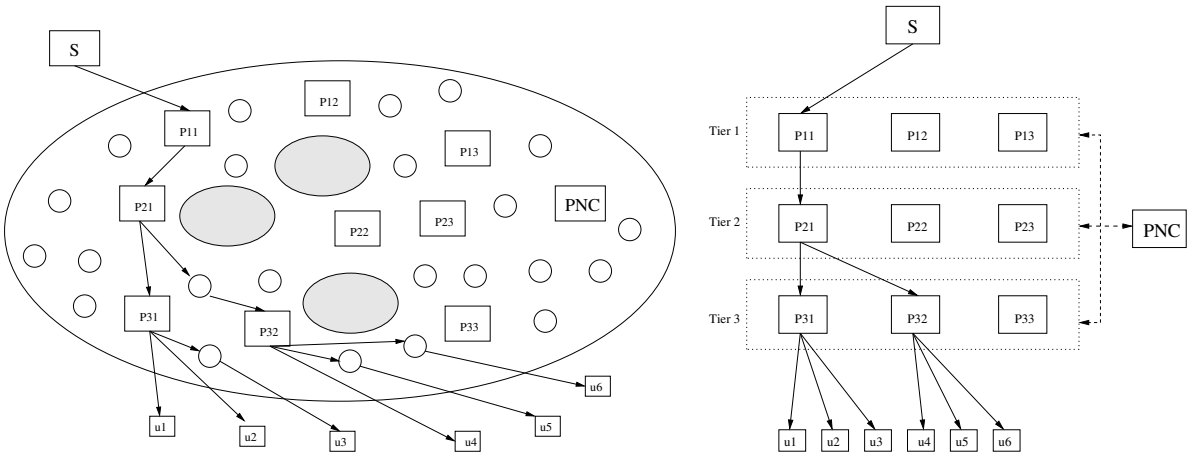
One of the key characteristics of our solution is that the proposed streaming media delivery mechanism is designed for live broadcasts. In that it is especially designed to deal with the burst characteristics of user logins (and user logoffs). To deal with burst traffic at the beginning of live broadcasting events, our system features a unique function that dynamically clusters multiple proxy servers to form a server farm to handle a large number of user login events when needed. Furthermore, unlike most other proposals which assume that proxy activation is instantaneous, our proposed approach explicitly accounts for the delays involved in proxy activation and connection migration. Another contribution is that our solution has been implemented based on Real Network Proxy and tested under various conditions, such as proxy server capability, domain name server setup time, and network condition downgrade events. It is also extensively tested for various user login patterns and stream broadcast characteristics, including more than one hour long broadcast events with more than half million user logins.

1.3 Paper Organization

The rest of the paper is organized as follows. In Section 2, we describe the design of VAN and its components and operations that coordinate splitting and merging of live streams. In Section 3 we present the detailed algorithm used by the proxy servers. In Section 4 we present the detailed algorithm used by the proxy network coordinator. In Section 5, we present the experimental results of the evaluation of the proposed architecture. In Section 6 we discuss related work and contrast it with our approach. We conclude the paper in Section 7.

2 Virtual Active Network

In this section, we describe the Virtual Active Network (VAN) architecture and give an overview of its two main components; proxy servers and the proxy network coordinator.



(a) (b)

Figure 1: (a) Streaming Media Delivery with Virtual Active Network; and (b) Architecture of VAN

2.1 Architecture of a Virtual Active Network

Figure 1(a) illustrates the proposed virtual active network architecture, composed of a number of proxy servers deployed across the Internet. During the initialization phase when a media server is introduced to the Virtual Active Network, the proxy network is partitioned into multiple tiers based on the distribution of the end user population and the relative distance among the media server, proxy servers, and end users.

Figure 1(b) shows a three-tiered overlay network, which consists of a single origin server and nine proxies. P_{ij} indicates the j^{th} proxy server in the i^{th} tier of the overlay network. A proxy network coordinator (i.e. PNC) is deployed to coordinate traffic between proxy servers. Note that the links in the overlay network are all logical, the actual communication between two proxy servers still requires routing at the level of the network routers. End users, labeled as u_i, connect to the overlay network via domain name resolution based redirection. Figure 2 illustrates how multiple sources result in multiple overlays over the same set of cooperating proxies. We adhere to the following terminology:

- *overlay* refers to a network of proxies (proxy network) deployed strategically on top of an existing network;
- an *overlay network* refers to the static partitions of the proxies organized as a multi-tier hierarchy with respect to a given data stream; and
- a *virtual active network* refers to the live or active component of an overlay network.

Although Figure 1 illustrates an overlay network with a single media, like typical routers, each proxy server can also serve multiple media streams originating from one or more media sources. Figure 2 shows an overlay network architecture consisting of nine proxy servers shared by two media servers, $S1$ and $S2$, to deliver streaming data. The solid lines denote the streams from server S_1 and the dashed lines denote the streams from server S_2. Note that many *virtual proxies* (for example P_{11} of the first VAN and Q_{31} of the second VAN) share the same physical server. The virtual active network for each data stream may have different number of tiers. In this example, the numbers of tiers of the virtual active network for $S1$ and $S2$ are 3 and 4 respectively. Also note that the number of proxy servers in each tier may be different.

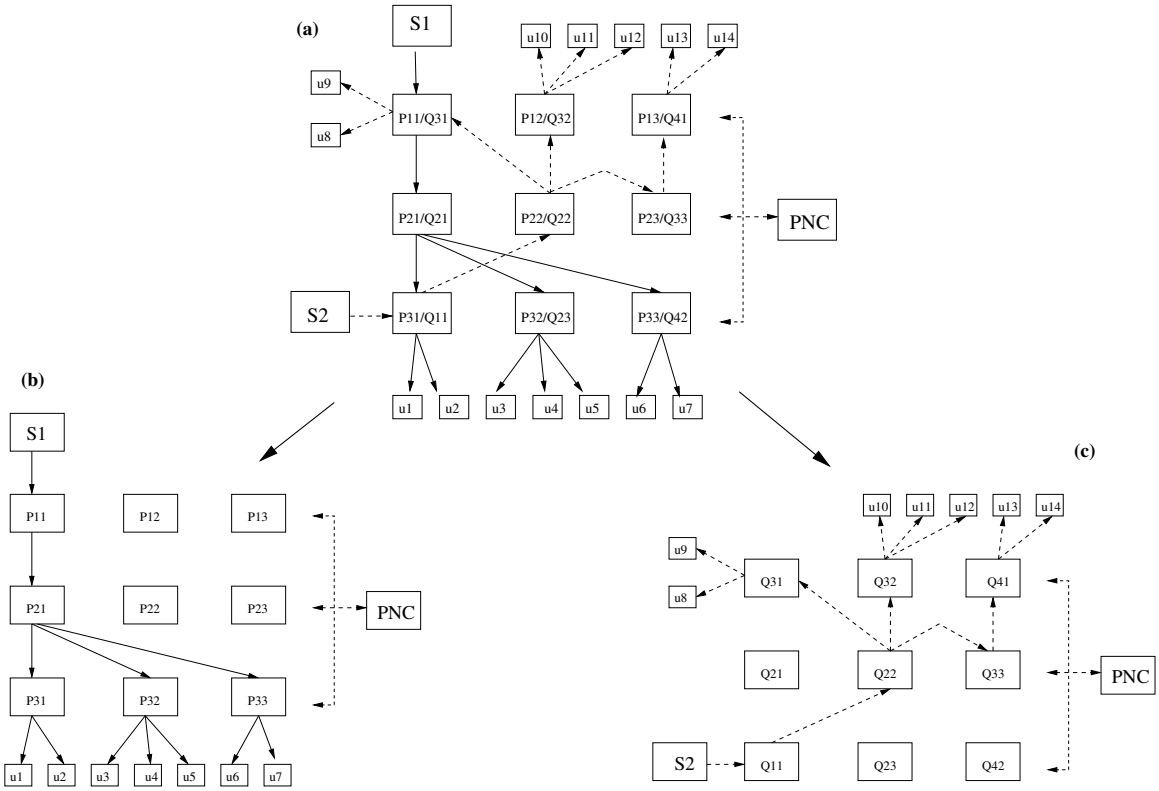


Figure 2: Multi-Source Virtual Active Network

In the VAN architecture, redundant retrieval capability is utilized during the restructuring of the multicast network. When a proxy needs to change its parent due to varying network conditions, the proxy establishes a connection with the new parent proxy before disconnecting from the old parent proxy. A single physical proxy server can be shared by multiple VANs each for a different media source or for multiple streams from the same source (Figure 2). Since the traffic between two proxy servers is more crucial than the traffic between a proxy server and end users (loss of a single inter-proxy connection may affect multiple users adversely), proxy servers retrieve multiple streams from the proxy servers in its parent tier for the same stream. This ensures a higher quality of streaming data delivery.

A proxy server that is serving a stream is an *active proxy server* (with respect to that particular stream). A proxy server that is active with respect to a given stream could be in the following phases:

- Expansion Phase: When a proxy server is activated by the PNC, it is in the expansion phase until PNC initiates the shrinking phase. During the expansion phase the proxy server continues to accept new connection requests.
- Shrinking Phase: After an active proxy server notifies the PNC that its load is too high, PNC will activate additional proxy servers in the same tier or a higher tier to serve new or migrated traffic. After such a load *distribution* operation, a proxy server transits from the expansion phase to the shrinking phase and will stop receiving new connection requests. After the load of a proxy server drops below a certain threshold, the proxy server requests consolidation from PNC.
- Idle Phase: When the load falls to zero with respect to the given stream, the server becomes *idle*.

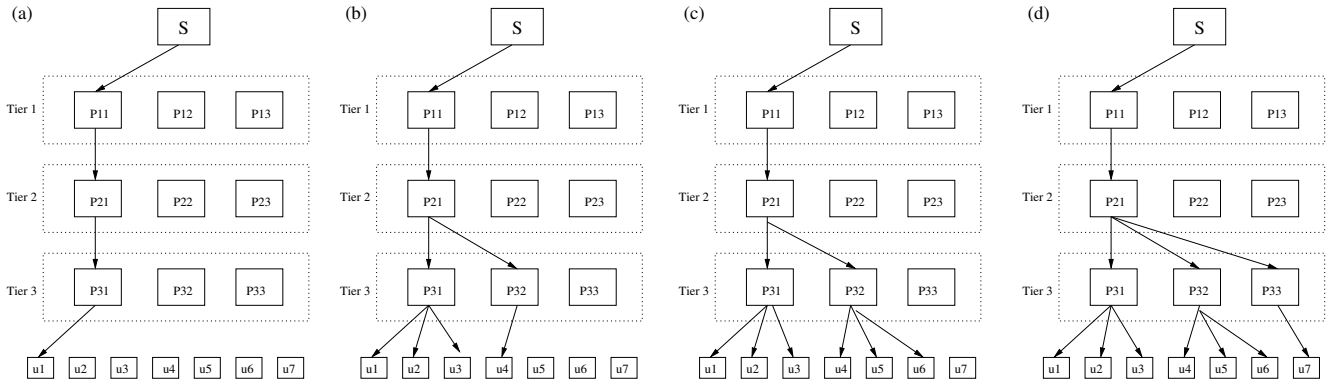


Figure 3: Load Distribution

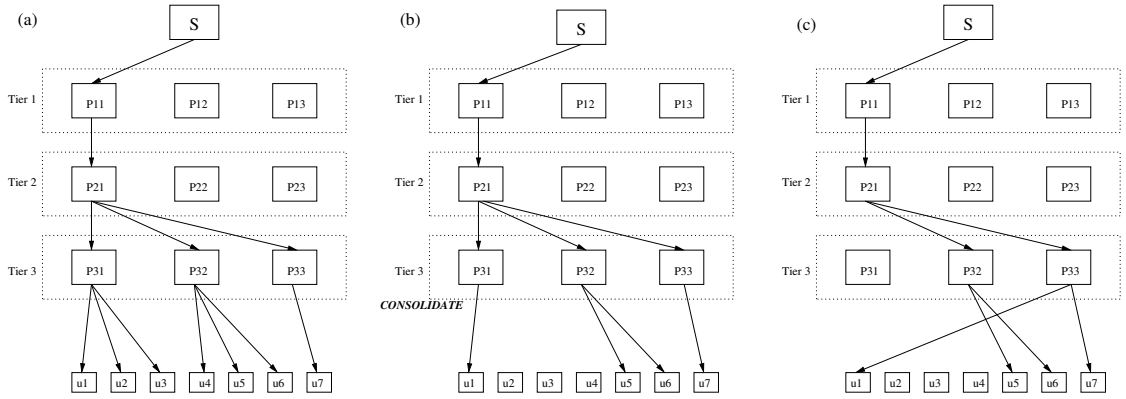


Figure 4: Load Consolidation

2.2 Coordination of Proxy Network

The proxy network coordinator (PNC) is a logical entity that can be implemented centrally as a single component or in a distributed fashion across multiple components. PNC coordinates the connections between proxy servers using *load distribution* and *load consolidation* mechanisms. For each streaming data server, the information PNC maintains in order to establish and manage the VAN include (1) the number of tiers and the proxy servers in each tier; (2) network status between proxy server pairs in adjacent tiers; (3) the list of active proxy servers in each tier; and (4) the hierarchical structure of virtual active network (identified by proxy server pairs in adjacent tiers).

The main task of PNC is to maintain the hierarchy corresponding to VAN component for each media server. At first, PNC initiates the minimal number of proxies required at each tier to ensure a coverage of all potential end-users for a given media. The IP addresses of proxies where the end-users should be redirected to when they request for the media stream is registered using DNS. As the number of end-user requests to a proxy server increases, the proxy server sends a message to PNC to expand the VAN hierarchy. The number of proxies activated then is based on the rate at which end-users arrive into the system. Similarly, when end-users depart, PNC's role is to shrink the VAN hierarchy to minimize bandwidth usage. Due to the hierarchical nature of VAN, most of the changes occur at the bottom tier of the overlay network. The number of changes reduce significantly as one moves through the upper tiers in the overlay. This approach results in the media server being cushioned from the adverse effects of the abrupt changes in the network and the workload.

Distribution: When a proxy gets overloaded, it sends a `DISTRIBUTE` request to PNC, which in turn allocates new proxies to handle the increased load.

Figures 3(a) through 3(c) show the load distribution process. In this example, there is a single origin server, below which we have tier 1 proxy servers labeled P_{11}, P_{12}, and P_{13}. Below tier 1 proxy servers, we have tier 2 proxy servers labeled P_{21}, P_{22}, and P_{23}. Finally, at the lowest tier we have tier 3 proxy servers, P_{31}, P_{32}, and P_{33}, below which we have the end users. In this example, the capability of each proxy servers is assumed to be limited to three simultaneous connections. Without any loss of generality, the users are assumed to arrive in the order of u_1, u_2, u_3, u_4, and so on.

When user u_1 arrives, this causes one proxy at each level (P_{11}, P_{21}, and P_{31}) to be activated to form a streaming path. When user u_3 arrives, however, P_{31} reaches the limit of its capacity and sends a `DISTRIBUTE` request to the PNC. As a result, PNC selects P_{32} from the overlay network and activates it by sending P_{32} a message to inform P_{32} that it is activated as part of the VAN and its parent is P_{21}. PNC then updates the DNS resolution mechanism so that future user logins are directed to P_{32} instead of P_{31}. Later, the arrival of u_6 would trigger a `DISTRIBUTE` request by P_{32} and the arrival of u_7 would trigger a `DISTRIBUTE` request by P_{21}, which now becomes full. The above sequence of events illustrate how the virtual active network expands gracefully as the load in the system increases.

Consolidation: As the number of users in the system drops, the VAN structure has to contract to make sure that the proxies in the system and the network resources are not under-utilized.

Figure 4(a) shows a configuration the VAN with active proxies and users. When users u_4, u_2, and u_3 log off one after another (Figure 4(b)), the reduction in the load at P_{31} triggers a `CONSOLIDATE` request from P_{31} to PNC. As a result, the PNC processes this event by sending a message to the children of P_{31} (u_1 in this case) to switch to the most recently activated proxy server, i.e., P_{33}. Consequently, u_1 logs off from P_{31} and logs on to P_{33}, which results in P_{31} logging off from P_{21} (Figure 4(c)).

2.3 Initialization of Virtual Active Network

When a media server is introduced to the virtual active network, PNC needs to perform the following tasks for initialization:

- Determine the regions where users in various ISPs will be directed to be served only by the proxy servers in the same region. For example, as show in Figure 5, the users in region 1 will only be directed or redirected to the proxy servers in region 1. This is necessary since users in one region may be too distant from the proxy servers in the other regions. Directing users to a proxy server in a faraway region may cause a higher packet drop rate.
- Determine the proxy servers in each region.
- Determine the layers in each region and assign proxy servers to each layer.

The region and layering information is stored at PNC. Note that the algorithms described later in this paper are designed for traffic direction for a single region. There exist several approaches to discover the network topology over the Internet, such as [12, 13].

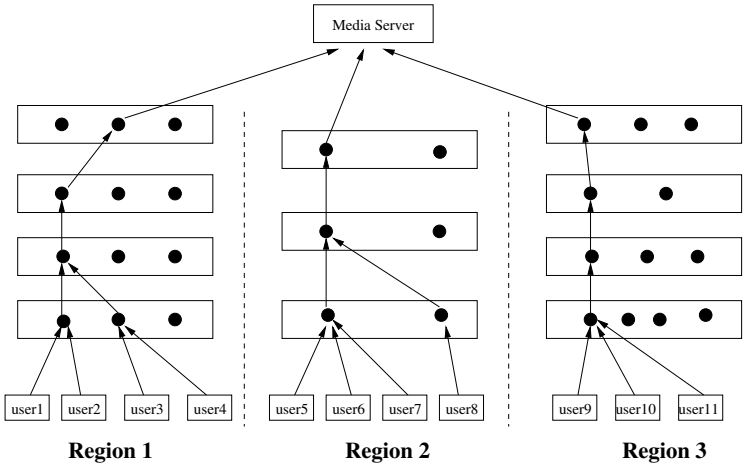


Figure 5: Initialization of Virtual Active Network

In order to build the overlay structure for a given media source, we need to identify the relationship of the media source with respect to the ISPs (Internet Service Provider) / ASs (Autonomous Systems) in the network. Using the media source as the center and the fanout of the overlay structure as a parameter, we can use a BFS (i.e. Breath First Search) approach to embed the overlay structure using the AS connectivity graph. The depth of the overlay structure will depend upon the diameter of the connectivity graph.

The tasks may be carried out during run time if it is necessary to adjust the region and layer partitions. In an extreme case in which each proxy server is assigned to only one region and one layer and the region and layer partition is done during run time, this design of virtual active network is considered fully distributed. The major difference between the fully distributed version of algorithms and the algorithms described in this paper is the selection of parent proxy server for proxy servers. The load management and redirection remain the same.

3 Design and Implementation of Proxy Servers

In this section we describe the details of the proxy server interface employed to deliver streaming media efficiently. We start by describing the internal architecture of the proxy servers and data-structures employed to maintain the connection states of multiple streams. We then describe the message-based API that is used by the proxy server to communicate with other proxy servers as well as with PNC.

3.1 Internal Architectural Design

Figure 6 illustrates the internal architecture of the proxy servers. The main resource of the proxy server is a buffer used for streaming media. The buffer is shared and accessed by two modules, the incoming stream handling module (ISHM) and the outgoing stream handling module (OSHM). As their names suggest, ISHM handles the incoming streams from the media server or proxy servers in the tier immediately above in the overlay network (referred to as the parent tier and parent proxies) and OSHM handles streams delivered to the end users or the proxy servers in the tier immediately below (referred to as the child proxies). ISHM is responsible to manage connections, disconnections, and reconnections to the parent proxy servers as specified by PNC. It has the capability

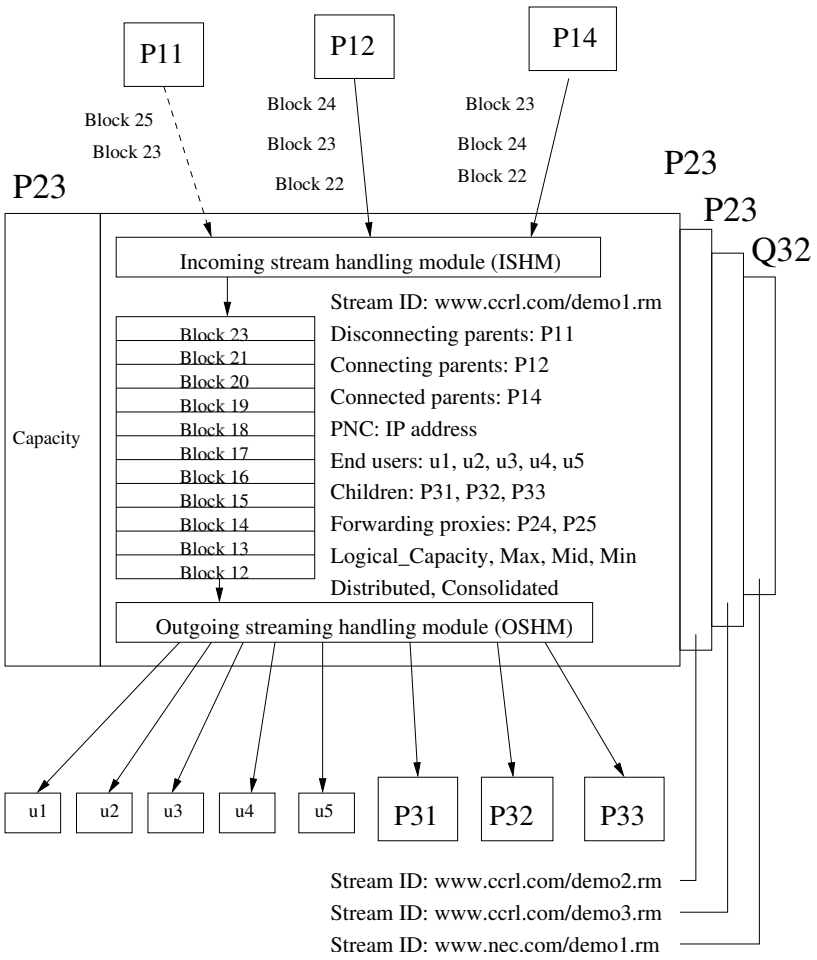


Figure 6: Proxy Server

of connecting to multiple parents for a single stream to enable redundant, and therefore robust, media delivery. For example, in Figure 6, the proxy has three parents, $P11$, $P12$, and $P14$. ISHM fetches streams from $P11$, $P12$, and $P14$ block by block and eliminates the redundant blocks by checking their sequence numbers or time stamps. The proxy server also keeps track of the end users and proxy servers who are retrieving the streaming media. Note that in the VAN architecture, redundant retrieval capability is utilized during the restructuring of the multicast network. When a proxy needs to change its parent due to varying network conditions, the proxy establishes a connection with the new parent proxy before disconnecting from the old parent proxy.

A single physical proxy server can be shared by multiple VANs each for a different media source (`www.ccrl.com` and `www.nec.com` in this example) or for multiple streams from the same source (`www.ccrl.com` in this example). For delivering the three different streams, $demo1.rm$, $demo2.rm$, and $demo3.rm$, from `www.ccrl.com`, we only need to have a single virtual active network tiering assignment in which all proxy servers have a prefix of P. On the other hand, we have a separate virtual active network tiering assignment for another media server `www.nec.com`. Note that although $demo1.rm$, $demo2.rm$, and $demo3.rm$ are delivered through the same virtual active network, the delivering proxy server assignment in each tier is different. If the geographical distributions of end users for $demo1.rm$, $demo2.rm$, and $demo3.rm$ are different, it is beneficial to have different virtual active network tiering assignments for them.

Although the proxy server is logically partitioned into multiple virtual hosts as shown in the figure, it needs to enforce machine-wide resource constraints, such as the number of simultaneous connections and the bandwidth usage.

The proxy server keeps track of the IP addresses of end users and proxy servers who request down streams from the proxy server. These IP addresses can be extracted from the media protocol (such as RTSP) headers. When redirection of end users and proxy servers in the lower tier is needed, the proxy server sends out redirection messages to all end users to switch to the newly assigned proxy server while PNC sends out the messages to redirect the proxy servers in the lower tiers to switch to the new parent proxy server. The IP address of each end user is maintained in proxy servers rather than PNC while the proxy network hierarchical information is maintained only at PNC.

For simplicity, the physical capacity and the capability to handle the number of logins at the proxy server is assumed to be the same for all media streams. To summarize, a proxy server P maintains the following information:

- Stream ID: the unique identification for the stream.
- Disconnecting parents: the proxy servers at the upper tier that P is disconnecting from.
- Connecting parents: the proxy servers at the upper tier that P is connecting to.
- Connected parents: the proxy servers at the upper tier that P is currently connected to.
- PNC: the IP address of PNC to which P sends `DISTRIBUTE`, `CONSOLIDATE`, and `MIGRATE` messages.
- End users: the IP addresses of the end users who login at P. This information can be extracted from the RTSP protocol header.
- Children: the IP addresses of the proxy servers at the lower tier who login at P. This information can be extracted from the RTSP protocol header that is customized to identify that the party that logs in is a proxy server.
- Forwarding proxy servers: the IP addresses of the proxy servers at the same tier where the requests should be forwarded to.
- Physical capacity: the maximum number of down streams that a machine can support.
- Logical capacity: the maximum number of down streams that is assigned to a proxy server for delivering a unique stream.

3.2 Data Structures

The first data structure `ConnectionState` shown in Figure 7 maintains the current state of all live connections that are passing through a given proxy. The connection state at proxy PS maintains the following information: the URL of the stream source for which this connection is maintained; the IP address and other connection related information of the parent to which this proxy is connected; the IP addresses of all the child hosts that are being served by PS. Note that a child of PS can be either another proxy or an end-user. PNC offloads the task of end-user maintenance to the proxy PS itself. The other data-structure is maintained to keep track of current parent proxy that a proxy must connect to for each stream source.

```
MODULE DynamicMultiSourceProxyServer;
/* Data Structure to maintain connection states */
struct ConnectionState{
    int                 State; /* IDLE, CONNECTING */
    URL                 StreamSource; /* Source Identifier */
    NetHost             Parent; /* Parent Node of this Connection */
    int                 ChildCount; /* Number of children */
    LIST OF NetHost     Children; /* Children of this Connection */
    LIST OF NetHost     Waiting; /* Awaiting parent connection */
    int                 Load, Dist, Cons, Max, LinRate, LoffRate;
                            /* Connection Management; */
} struct ProxyParent{
    URL                 StreamSource; /* Source Identifier */
    NetHost             ParentCache; /* my parent for this source? */
}
SET OF ConnectionState  Conns; /* global variables */
            /* INITIALIZATION HERE */
/* Event Handling */
MONITOR(P) from sender S for sourceURL;
LOGIN(params) from sender S for sourceURL: See Figure 8
LOGOFF(params) from sender S for sourceURL: See Figure 9
CONNECTED() from S for sourceURL;
ConnRefused() from S for sourceURL;
SwitchToParent(P) from PNC for sourceURL;
```

Figure 7: The Proxy Server Module for Handling Media Streams

3.3 Events

The main events that a proxy server supports are `login` and `logoff` requests from the users. Note that depending upon the active network structure, the user may be another proxy server or an end-user. In the same vein as `login` and `logoff` are events triggered by a child proxy to the parent proxy, `connected` event is triggered by a parent proxy to the child proxies. The remaining events are primarily used to maintain the virtual active network for each stream source. The event `SwitchToParent` is triggered by PNC if PNC detects that there are either network problems or if the proxy load needs to be balanced. Similarly, the event `MONITOR` is initiated by PNC so that each proxy can monitor the health of network links between consecutive tiers in a distributed manner. PNC requires that for each stream source at a proxy in tier $k+1$ monitor all the proxies in tier k.

The `login` event (Figure 8) is triggered either by a proxy or by an end-user. If the event is triggered by a proxy this means that the proxy has been assigned the current proxy PS as a parent proxy for the specified media stream. On receiving the login event from sender S for $sourceURL$, PS first checks if the connection is already active for

```
LOGIN(params) from sender S for sourceURL:
{
 Let CC be in Conns such that CC.StreamSource = sourceURL;
 if there is no ConnectionState then
      · setup the structure to maintain the ConnectionState;
      · update load information appropriately;
 endif
 update CC.Load, CC.LoginRate using double smoothing;
 if (New ConnectionState)
      send(LOGIN(params+MyParams)) to CC.Parent for sourceURL;
      mark this request as WAITING for
      a connection setup with the parent Proxy;
 else if (ConnectionState indicates login requests to parent is pending)
      mark this request as WAITING for
      a connection setup with the parent Proxy;
 else /* parent connection already exists */
      SetUpLocalConnection(params, S); /* allocate buffers etc. */
      send(CONNECTED) to S for sourceURL; }
 update CC.Load, CC.LoginRate using double smoothing;
 if (OVERFLOW possible) /* check if the current load+login rate-logoff rate will cause overflow */
      send(DISTRIBUTE(MyParams)) to ProxyNetworkCoordinator;
}
```

Figure 8: The LOGIN Event

the specified sourceURL. If this is not the case, PS initiates the connection by setting up a local data-structure as well as forwarding S's request for a connection to the parent of PS. Also, when a new media begins to share PS, the server dynamically adjusts the load control parameters appropriately to account for the fact that there is more contention for physical resources at PS. If the connection is already active then the login request of S is served locally by including S in the local connection state. As a part of the maintenance activity, if PS's load reaches beyond the maximum threshold then PS sends a `DISTRIBUTE` event to PNC requesting that additional proxies be activated in PS's tier. The condition for the `DISTRIBUTE` event is determined by estimating two parameters at the proxy: the `login` rate and the `logoff` rate. If the two rates are such that in the next ΔT seconds the proxy may reach its maximum capacity then the proxy triggers a `DISTRIBUTE` event to the proxy network coordinator. Note that the parameter ΔT is the estimate for the time it takes for PNC to activate a new set of proxy servers.

The `logoff` event shown in Figure 9 is analogous to the `login` event. When a sender S triggers this event at PS, PS removes S from the connection state. Furthermore, if S is the last child to request `logoff`, in that case PS sends the `logoff` event to its parent since it no longer needs to keep that connection alive. If the workload of PS falls below a certain threshold due to multiple `logoff` events, PS indicates to PNC that PS should be made dormant via the `CONSOLIDATE` event. On receiving such an event, PNC deactivates PS by moving its workload

```
LOGOFF(params) from sender S for sourceURL:
{
 Find entry CC in Conns s.t. Entry.StreamSource=sourceURL;
 TearDownLocalConnection(params, S); /* deallocate buffers etc. */
 Remove S from the list of children of CC;
 Conn.Children=Conn.Children \ {S};
 update load parameters: Load, LogoffRate using double smoothing;
 if UNDERFLOW possible /* check if the current load+login rate-logoff rate will cause overflow */
      send(CONSOLIDATE(MyParams)) to ProxyNetworkCoordinator
      for sourceURL;
 endifif (this is the last user to logoff)
      send(LOGOFF(params,MyParams)) to Conn.Parent for sourceURL;
      DEALLOCATE Conn;
 endif }
```

Figure 9: The LOGOFF Event

to another active proxy in the same tier. Note that if the proxy server that sends out the `CONSOLIDATE` message is the only remaining proxy server serving the content in the same tier, PNC will ignore the `CONSOLIDATE` message.

In the case of receiving the `connected` event from the parent proxy, PS appropriately updates its state and begins the session to deliver the media content. On the other hand, `ConnRefused` results in the login request being rejected. The event `SwitchToParent` is triggered by PNC to move PS to a new parent proxy. Each proxy periodically triggers `ProxyStats` event to send the current load statistics (e.g. login and logoff rate) to PNC.

4 Design and Implementation of Proxy Network Coordinator

In this section, we present the details of the event driven code that is deployed to maintain the application level proxy network for multiple media streams from different sources. Figure 10 depicts the module `DynamicMultiSourcePNC`, which primarily includes data-structures to maintain both static system information as well as the dynamic relationship among the proxies. In addition, it provides the message-based API that is used by the proxy servers to coordinate their activities. The data-structure `SourceProxyPair` maintains the relationship between stream sources and the corresponding proxy structures.

4.1 Overview

As discussed earlier the set of proxies deployed in our system are partitioned into different tiers with respect to each media source. Proxies in tier 1 are closest to the media source and the proxies in tier N are closest to the end-users. Proxies in tier k+1 are served by the proxies in tier k. The actual establishment of parent-child relationships among the proxies occurs during the dynamic restructuring of the virtual active network via `DISTRIBUTE` and `CONSOLIDATE` events. An active proxy server initiates a `DISTRIBUTE` event when its load reaches above a

```
MODULE DynamicMultiSourcePNC;
struct Proxy{
    NetHost                 Id;
    int                     State; /* FREE, INUSE, FAILED */
    int                     Tier; /* Tier Identifier */
    NetHost                 Parent;
    List of NetHost         Children;
}
struct SourceProxyPair{
    NetHost                 StreamSource;
    Proxy                   Overlay[NTiers][NProxies];
    INFO                    ProxyMaint[NTiers];
} /* The proxies are maintained in a layered manner */
SET OF SourceProxyPair ProxyNet; /* Global Variables */
 BEGIN
/* Initialization */
/* Initiate Link Monitoring Activity */
/* Event Handling */
DISTRIBUTE() from S for sourceURL: See Figure 11.
CONSOLIDATE() from S for sourceURL: See Figure 13.
END.
```

Figure 10: Proxy Network Coordinator

certain threshold. This event has an effect of activating an appropriate number of proxies in the same tier. Similarly, a proxy server generates a `CONSOLIDATE` event when its load falls below a certain threshold. This event indicates to PNC that the proxy should be made idle. A sequence of `DISTRIBUTE` and `CONSOLIDATE` events results in the proxy hierarchy shrinking and expanding with the changes in demand. Our goal is to maintain this hierarchy in such a way that the network resource (e.g., bandwidth) usage is minimized while the active set of end-users change in real time. At the time of initialization, VAN initiates one proxy at each tier, forming a thin pipe across the virtual active network from the source to the lowest tier in the hierarchy. At the same time, PNC also initiates monitoring activity to make each proxy at tier k+1 start monitoring the network connections to all proxies in tier k to ensure reliable delivery of media. If the proxy discovers a failed connection, it reports this to PNC via the LinkDownGrade API call. PNC, then, depending on the status of the proxy takes appropriate corrective actions.

4.2 Dynamic Resource Management

We now describe the main adaptive part of our system. Based on the patterns of user traffic, the system automatically allocates required resources for streaming media broadcast. On demand resource allocation is especially important in an environment where the user traffic shows huge variations. For instance, burst traffic tends to occur at the beginning of a live broadcasting event. We now describe dynamic resource management in detail below.

DISTRIBUTE() from S for sourceURL:

1. $sp \Leftarrow$ stability period
2. $SysLinRate \Leftarrow \Sigma_{p \in Proxies[S.tier]} LinRate_p$ /* Calculate total expected login rate */
3. $SysLoffRate \Leftarrow \Sigma_{p \in Proxies[S.tier]} LoffRate_p$/* Calculate total expected logout rate */
4. if $((SysLinRate - SysLoffRate) \cdot sp) \geq \sum_{p \in Proxies[S.tier]} (Max_p - Load_p)$
5. $\quad Load_1 = ((SysLinRate - SysLoffRate) \cdot sp) - \sum_{p \in Proxies[S.tier]} (Max_p - Load_p)$
6. $\quad$ /* Calculate the extra needed resources during the next sp */
7. $\quad Load_2 = SysLinRate \cdot \Delta T$; /* We also need to at least handle the load during the next ΔT period */
8. $\quad AnticipatedLoad = MAX(Load_1, Load_2)$;
9. $\quad$ FIND new m proxies in S.tier such that $\Sigma_{i=1}^{m} Max_i \geq AnticipatedLoad$ and m is minimum;
10. $\quad$ /*Note that these proxies are newly added to this layer */
11. $\quad$ Let this set be $\mathcal{P} = \{P_1, P_2, ..., P_m\}$;
12. else /* the current load can be handled by currently active proxies, no need to add new proxies */
13. $\quad \mathcal{P} = CreateServerFarmFromActiveProxies()$;
14. $\quad$ if $\mathcal{P} = \emptyset$ {
15. $\quad\quad$ /* Sometimes due to fragmentation, we may not be able create server farm */
16. $\quad\quad AnticipatedLoad = SysLinRate \cdot \Delta T$;
17. $\quad\quad$ FIND m proxies in S.tier such that $\Sigma_{i=1}^{m} Max_i \geq AnticipatedLoad$ and m is minimum;
18. $\quad\quad$ Let this set be $\mathcal{P} = \{P_1, P_2, ..., P_m\}$;
    $\quad$ }
19. for (each proxy $T \in \mathcal{P}$) do
    {
20. $\quad$ Activate T in p.Overlay;
21. $\quad$ /*find parent; use a round robin allocation if multiple parents*/
22. $\quad$ P = FindCurrentActiveProxy(p.Overlay, ParentTier(T.tier));
23. $\quad$ /* the following steps maintains the active part of the overlay */
24. $\quad$ p.Overlay[P.i][P.j].Children=* $\cup$ {T};
25. $\quad$ p.Overlay[T.i][T.j].Parent=*$\cup${P};
26. $\quad$ if (T is at the lowest level) then add T to the DNS;
27. $\quad$ send(SwitchToParent(P)) to T for sourceURL;
    }

Figure 11: Proxy Network Coordinator: `DISTRIBUTE`

4.2.1 `Distribute` Event Handling and Dynamic Construction of Server Farms

User traffic can be redirected to any servers. However, there is a time delay, denoted as ΔT, associated with setting appropriate DNS (Domain Name Server) lookup resolution and buffering the streaming media in new proxy servers. If the server S predicts that during the next ΔT time units, the total user logins will exceed its

CreateServerFarmFromActiveProxies:

1. for all active proxies s do{
2. find minimum Min_s such that
3. $((\frac{SysLinRate}{Min_s} - Loffrate_S) \cdot sp) \leq Max_S - Load_S$;
4. ADD $\langle Min_s, s \rangle$ to the List L ;

}

5. Sort L
6. $S = \emptyset$;
7. $count = 0$;
8. for $(i = 0; i < n; i++)$ do{
9. $count = count + 1$;
10. $S = S \cup \{L[i].s\}$;
11. if $(count \geq L[i].Min_s)$ RETURN S;

}

12. RETURN $\{\}$;

Figure 12: Proxy Network Coordinator: CreateServerFarmFromActiveProxies

capacity, it will send a distribute request to PNC. Note that PNC needs to be requested ΔT time units in advance so that it can redirect the user traffic to an available server before users start being rejected due to overload.

Although our goal is to design an adaptive system, we do not want to sacrifice the stability of the system. In particular, we do not want to deactivate a server from the system if it needs to be reactivated soon in the future/ For this reason, we introduce a parameter, denoted sp, to control the stability of the system. When we add or remove a server from the system, we take into account the constraint that during the next sp time units, no additional servers need to be activated or de-activated.

To deal with burst traffic or proxies with low capacity, PNC may establish a server farm consisting of multiple proxies if there is not enough time to activate proxies one by one to serve the burst traffic. After a server farm is established, incoming load is distributed among all proxies in the server farm group in a round robin fashion. Thus, at time t, the number of user logins to the proxy s in a server farm is $login_s(t) = login(t)/group\_size(s)$ where $login(t)$ is the number of logins to the system. In consideration of user logoffs, we subtract expected user logoffs from expected user logins to calculate net additional users to the system for the next ΔT time units. After calculating the expected number of net additions, if the proxy will be over-loaded in the next ΔT period, it sends a distribute request to PNC.

Figure 11 illustrates how PNC handles the `DISTRIBUTE` event. When PNC receives a distribute request from the proxy, it first checks whether, during the next sp time, it has to activate a new server; in other words whether the capacity available in the system (via active proxies) will not be enough to serve all the users (Figure 11, Lines 1 to 4). If this is the case, it identifies and adds the minimum number of servers so that these new servers will be able to handle the excess load during the sp time units. If it fails to activate any new proxy, it searches for the proxy that has the least load and redirects the traffic to that proxy (Figure 11, Lines 5 to 11).

If PNC sees that the remaining capacity of the system is sufficient for the next sp time units, it allocates a suitable proxy or a group of proxies (i.e. a server farm) to which the traffic can be redirected. In the case that a group of proxies are needed, PNC allocates the minimum number of proxies such that each proxy in the group will be able to handle the load during the next ΔT time units. Note that if there are m proxies in the group, then each proxy will observe $login(t)/m$ logins. A suitable server farm of the minimum number of proxies can be found in $O(n \log(n))$ time where n is the number of active proxies in the system. More details are given in Section 4.2.2.

Since round robin is used for load distribution among proxies in a server farm, it is necessary to identify proxies with close small capacity to avoid resource fragmentation. For example, assume that during the next ΔT time units, resources are needed for serving 100 new users. Although grouping two proxies which has 80 and 20 user capacity left respectively seems like a good choice, due to round robin load distribution policy in a server farm, the proxy with 20 user capacity left will be overloaded before the end of ΔT time units. If we want to create a group with only two proxies, each of the proxies must be able to handle 50 users during the next ΔT time units. By grouping the existing servers, we need to overcome the restrictions due to the fragmentation (Figure 11, Line 13). The procedure used to create the server groups is shown in Figure 12.

If there exists no proxy or group of proxies suitable to redirect the traffic, (even when the total capacity left is enough, due to fragmentation, it is possible that the resources cannot be used as a whole), PNC adds the minimum number of proxies such that these newly activated proxies will satisfy the requests of these new users (Figure 11, Lines 14 to 18). Again these newly added proxies will serve as a group.

4.2.2 Finding Eligible Proxies for Re-grouping

If PNC expects that the remaining capacity of the system (i.e. capacity of all activated proxies) is sufficient for the next sp period, PNC will allocate a suitable proxy or a group of proxies where the incoming traffic. Our goal is to find the minimum number of proxies that can handle the incoming users for at least ΔT time units.

For example, let us assume that 100 (i.e $SysLinRate = 100$) users are logging into the system per second and ΔT is 10 seconds. Assume that we want to create a server farm with 2 proxies only, then each proxy will need to handle 500 (i.e. $(100/2) * 10$) users in the next 10 seconds. Therefore a proxy can be a part of this server farm if only if it has the capacity to handle at least 500 additional logins.

The above observation can be formalized as follows: Let $SysLinRate$ denote the average predicted login rate to the system and $Loffrate_s$ denote the average logoff predictions for each proxy s. If a particular proxy is part of a group that has m proxies, then it will observe $SysLinRate/m$ number of logins. If a proxy can handle the load assigned to it, then it means that, either logoffs are higher then logins($(SysLinRate/m) < Loffrate_s$) or it has available space for ΔT period ($\Delta T * ((SysLinRate/m) - Loffrate_s) < Max_s - Load_s$). For any given proxy s, let Min_s denotes the minimum value of m such that the above condition holds. A proxy s can only be part of a group that has at least Min_s servers in it (Figure 12, Lines 1 to 3). After calculating these Min_s values, each proxy can be ranked according to its own Min_s (Figure 12, Line 5). An eligible group of size l exists if and only if the total number of the proxies that has Min_s value less than equal to l is greater or equal to l. Since any suitable minimum size group is acceptable, we can choose the first l of them. The group with the minimum size and the Min_s values can be evaluated in the worst case in $O(n \log(n))$ time where n is the number of active

CONSOLIDATE() from S for sourceURL:

1. $sp \Leftarrow$ stability period
2. $SysLinRate \Leftarrow \Sigma_{p \in Proxies[S.tier]} LinRate_p$
3. $SysLoffrate \Leftarrow \Sigma_{p \in Proxies[S.tier]} LoffRate_p$
4. if $\Big((SysLinRate - SysLoffRate) \cdot sp \geq \sum_{p \in ProxiesinS.Tier \wedge p \neq S} (Max_p - Load_p)\Big) \wedge$
5. ($load_S$ can be handled by the current group) Then S is deactivated;

Figure 13: Proxy Network Coordinator: `CONSOLIDATE`

proxies (Figure 12, Lines 7 to 11).

4.2.3 Consolidate

Figure 13 illustrates the code to render proxies idle when the workload reduces in the system. When a proxy expects to drop to a certain level, it sends a consolidate request to PNC. Each proxy checks whether during the next $cons_t$ time, it will drop to zero load level . If this is the case, it sends a consolidate message. The $cons_t$ parameter can be adjusted to achieve more aggressive or less aggressive consolidation effect. In either case, the proxy must be in a shrinking mode (log-offs must be higher than logins) to initiate a consolidate request.

Through this request, the proxy requests PNC to remove itself from the active part of the virtual active network. When PNC receives a consolidate request, it checks whether the consolidation of this server may result in a creation of a new server in the next sp time units (Figure 13, Lines 1 to 4). It also checks whether there is enough resources in the current active proxy group to handle the additional traffic that will occur due to the requested consolidation (Figure 13, Line 5). If both of these conditions are satisfied, then the proxy is consolidated; otherwise consolidation request is declined and the requesting proxy will send another consolidate request after a period of time. If a proxy is consolidated, the current users on the proxy is moved to the other machines in the current server farm.

4.2.4 Predicting Login and Logoff Rates

As mentioned above, our adaptive resource allocation methods depend on the prediction of the expected number of users in the very near future. Since we do not try to predict too far in the future, a time-series prediction method like *double exponential smoothing* can be effective for our purpose. *Double exponential smoothing* uses two equations to predict the future values.

$$S_t = \alpha y_t + (1 - \alpha)(S_{t-1} + b_{t-1}) \quad (1)$$

$$b_t = \beta(S_t - S_{t-1}) + (1 - \beta) b_{t-1} \quad (2)$$

where y_t is the last value observed, $0 < \alpha, \beta < 1$, and S_t is the average future prediction.

In all experiments in this paper, we set $\alpha = 0.9$ and $\beta = 0.1$. We update the parameters of the double exponential smoothing in every $\Delta T/10$ period, based on the number of logins or logoffs seen in the last $\Delta T/10$ period. Since we need to predict the values for at least ΔT time in the future, we would like to update our predictions more often then ΔT to capture the sudden changes in the user pattern, on the other hand we do not want to update it too often and yield a heavy overhead. The choice of $\Delta T/10$ period is the result of these considerations.

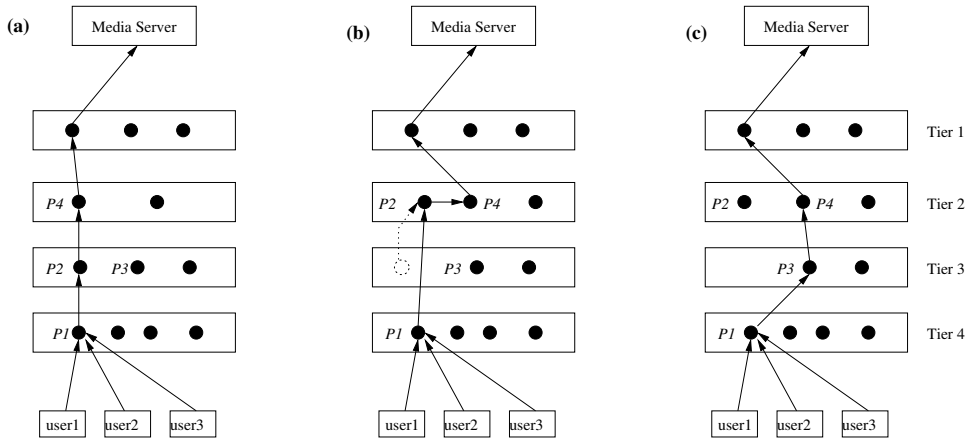


Figure 14: Adjustment to Changes of Network Topology

4.3 Network Monitoring and Migration

As discussed earlier, a proxy in a child tier monitors the network connectivity to all the proxies in the parent tier based on the static overlay structure. If the network connectivity from child proxy S to a parent proxy P is lost, S sends a `LinkDowngrade` event to PNC. In this case, PNC first tries to find an alternate parent proxy P' for S and sends a message `SwitchToParent` to S asking it to establish P' as the new parent. If an active parent other than P does not exist, in that case a sibling proxy S' is located and the children of S are migrated to S'. If both P' and S' are non-existent, then PNC activates a proxy P'' in the parent tier (similar to DISTRIBUTE) and asks S to switch to P''. In addition to monitoring the network links, PNC also monitors the network host status of each proxy in the system. If PNC detects that a proxy server has crashed (through a timeout mechanism), then the PNC migrates all the children of the failed proxy, with respect to each media stream, to alternate active proxies in the same tier. If no active proxy exists currently, then a proxy is activated in the specified tier. Through the above message based events, PNC maintains the virtual active network so that the bandwidth utilization is minimized.

4.4 Dealing with the Changes in the Network Topology

During the initialization phase, the static overlay network of proxies is initialized with respect to each registered media source. We assume that the partitioning of proxies in different tiers is pre-determined. The algorithms present in this paper work when network topology changes occur. In Figure 14, we illustrate how to deal with the changes in the network topology.

Figure 14(a) shows a virtual active network layout. In Figure 14(b), proxy $P2$ is reassigned to tier 2 instead of tier 3 due to network topology changes. As we can see, the virtual active network continues to work although it may not be optimal. Once PNC detects that $P2$ is moved to a different tier and $P2$ is active, it can initiate a $distribute$ event to $P2$ to offload its load to other active proxy, such as $P4$. The $distribute$ event then results in a stream delivered from $P4$ to $P3$, and from $P3$ to $P1$ as shown in Figure 14(c).

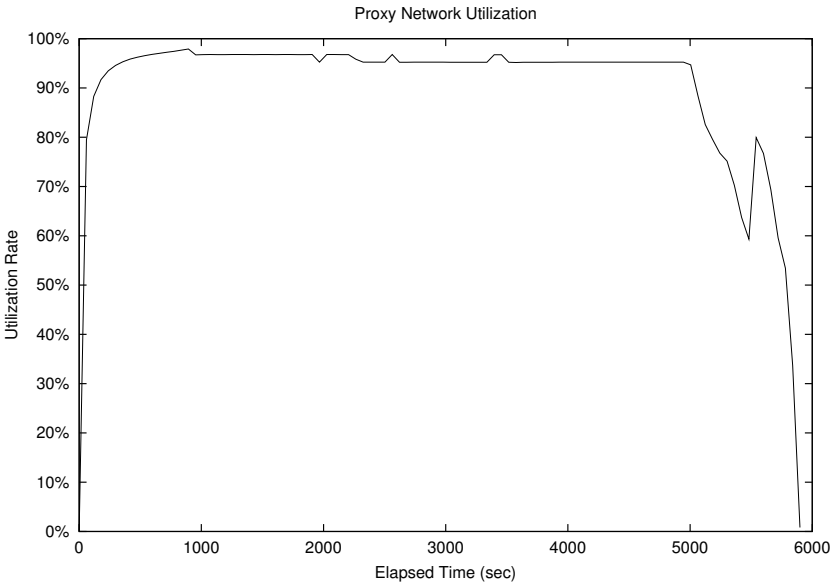


Figure 15: Proxy Network Utilization Rate for the Baseline Experiment Setting

5 Experiments

In this section, we present experimental results and evaluate the effectiveness and usefulness of the proposed virtual active network (VAN) architecture and its protocols for large streaming data delivery. The experiments were conducted using software simulation as well as real proxy deployment in a network-emulated environment

The goal of the virtual active network is to use the minimum number of proxy servers and network bandwidth (i.e., minimum cost) to serve a large streaming data broadcasting event (such as a a live media stream). For example, if the capacity of proxy servers is 500 user logins and there are one million user logins at the peak time, with an assumption of three-tiered proxy network, we require 2005 proxies (2000 proxies at the lowest tier, 4 proxies at the middle tier, and 1 proxy at the top tier). 2005 proxies is the optimal solution. To reach such an optimal solution, the utilization rates of all proxies has to be 100%.

In our experiments, we measured the proxy network utilization rate. Note that, if the proxy network utilization rate is 95%, the system requires 5.23% (i.e. 1/0.95 - 1) more proxies than the optimal solution. Note that the optimal solution could only be achieved when user logins can be perfectly predicted and can be switched from one proxy to another proxy instantaneously, which is not feasible.

The first four sets of experiments are conducted to measure the proxy network utilization and the behavior of VAN in response to the changes in server capacity, user access patterns, and restructuring overhead and constraints. We also show the positive impact of VAN in the presence of packet loss, and hence the perceived quality of the streams.

5.1 Experiment Setting

The baseline experiment settings are as follows:

- *Duration:* a 100-minute streaming media broadcast.
- *User load:* 500,000 user logins during the broadcast.

- *Access pattern:* Each user is created with random duration of stay in the system. The time each user stays in the system is simulated as a Gaussian random variable. On average, users login for 15 minutes (avg) with a standard deviation (std) of 4 minutes. In these experiments, we used fixed arrival rate. When we conducted the same experiments with Poisson arrival rates, we observed very similar results. The user login rate is much higher than the user logoff rate at the beginning of the broadcast event. In contrast, the user logoff rate is much higher at the end of the broadcast event. For the remaining part of the event, the login and logoff rates are comparable.
- *Restructuring overhead:* The time required to switch streams across proxies (ΔT) is 7 seconds. This is a physical constraint limited by the network infrastructure and the proxy implementation.
- *Stability:* The stability parameter (sp) is a logical constraint to control the frequency of delivery network restructuring. It can be controlled by system administrators to reduce the number of times a new server is added to the system. A larger value of sp implies that each *DISTRIBUTE* request leads to a larger number of servers to be allocated for future requests. In these experiments, sp is set to 15 seconds.
- *Server capacity:* The capacity (cap) for each proxy server is 1,500 simultaneous users.

In addition to this baseline setup, in order to observe the effects of different parameters on the system performance, we varied the values of cap, avg, std, ΔT, and sp parameters one at a time to create 20 additional setups. The alternative values for the parameters are as follows:

- cap: 500, 1000, 2000, and 2500 users.
- ΔT: 4, 10, 13, and 17 seconds.
- sp: 5, 10, 20, and 25 seconds.
- avg: 5, 10, 20, and 25 minutes. For these four avg values, the numbers of user logins are 30,000, 60,000, 120,000, and 150,000 users respectively.
- std: 1, 2, 3, and 5 minutes.

5.2 Proxy Network Utilization

Figure 15 shows the proxy network utilization for the duration (100 minutes) of the experiment. We can see that during most part of the broadcasting event, the resource utilization is kept very close to 100%. The very slight downwards trend in utilization shows that there is a small overhead with the dynamic routing approach that VAN uses for dealing with the dynamic nature of the streaming environment. However, the large benefits that such restructuring can bring is apparent during the last 1000 seconds of the broadcast, where there is a large amount of logoffs. As the number of users in the system (and hence the system utilization) drops, VAN takes a corrective action by shrinking the streaming network. Consequently, as seen in the figure, resource utilization jumps upward once more despite of the continuing logoff trends. Since the logoff trend does not stop in our experiments, on the other hand, utilization eventually drops to 0%, when there are no more users.

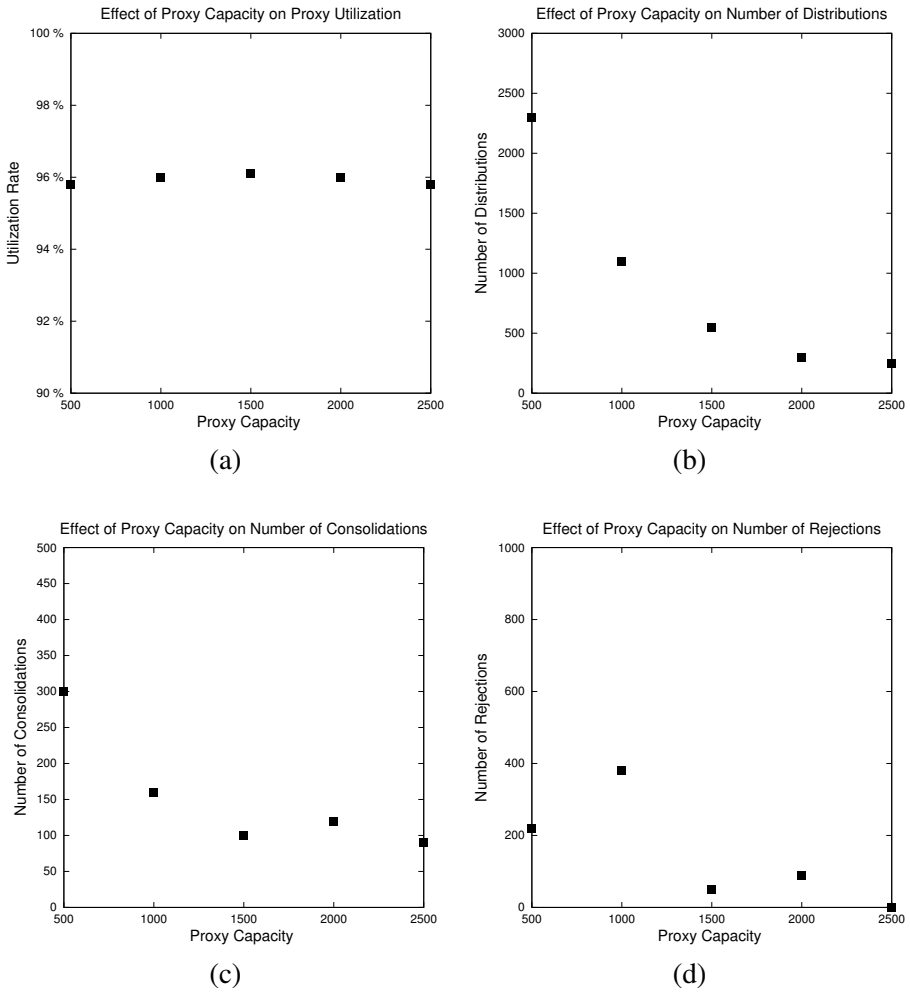
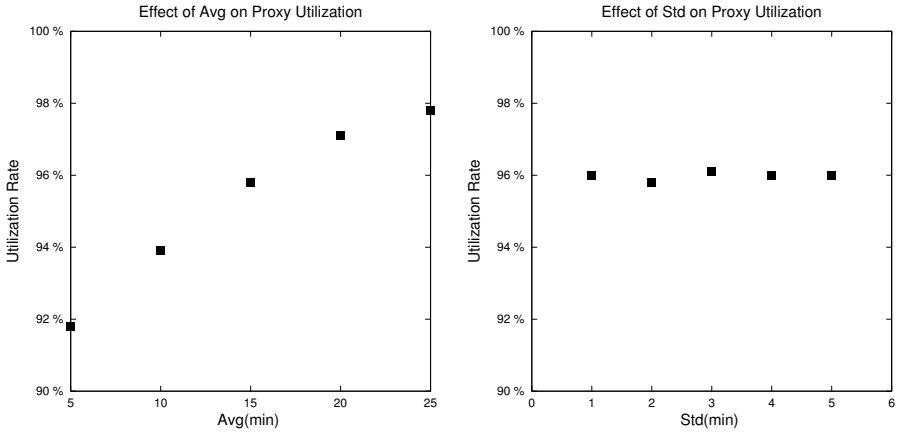


Figure 16: Effect of Proxy Server Capacity on (a) *Proxy Utilization*, (b) *DISTRIBUTE*, (c) *CONSOLIDATE*, and (d) *REJECT* Events

5.3 Effects of Proxy Server Capability

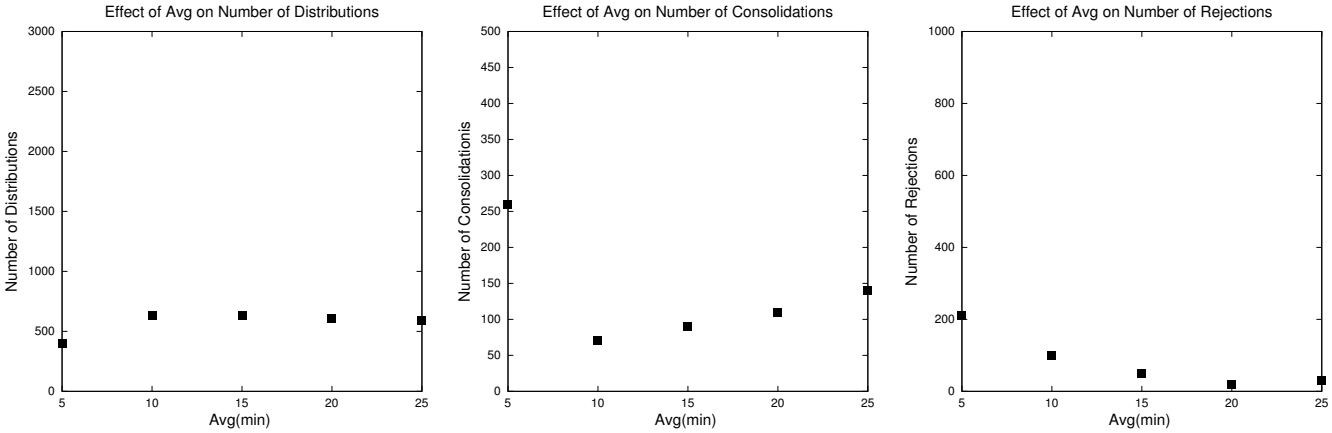
Figure 16(a) shows the impact of the individual proxy server capacity on the resource utilization. It is clear from this figure that available server capacity does not have a large impact on resource utilization and the system is capable of providing around 96% utilization, irrespective of the server capacity. This demonstrates that VAN can easily scale its structure to handle available proxy capacities. Figure 16 shows how this scaling is achieved: as the proxy capacity increases, the need for dynamic distributions (and hence consolidations) decreases (Figures 16(b) and (c)).

Note that the number of consolidate events in the system is around only 10% of the distribute requests generated by the proxies. This is due to the fact that a round-robin scheme is used for allocating additional servers as they are needed; after the initial stage of the broadcast (where a large number of users log into the system and not many users log off) is over, new users can be accommodated by the resources left by the departing users. Furthermore, due to the round-robin nature of the server allocation, all users a given server serves have approximately the same arrival time. Hence, the servers can be ranked with respect to the arrival times of the users, where the server that sends out a *DISTRIBUTE* message contains most recent arrivals and the users of the *oldest* server in the system



(a) (b)

Figure 17: Effect of User Access Pattern on Proxy Network Utilization



(a) (b) (c)

Figure 18: Effect of User Access Patterns on (a) *DISTRIBUTE*, (b) *CONSOLIDATE*, and (c) *REJECT* Events

are logging off. Therefore, instead of allocating a completely new proxy server, PNC can reuse this oldest server (avoiding proxy setup and tree-restructuring costs) before this old server becomes too underutilized to send out a *CONSOLIDATE* request. This property provides a very high stability to the VAN architecture.

Figure 16(d) shows that, as expected, when larger proxy capacities are available, since the number of *DISTRIBUTE* and *CONSOLIDATE* events are lower, the number of users that are rejected from the system due to overloads (while system is restructuring) also drops. On the other hand, the fluctuation in the graph shows that not all increases in the proxy capacity have a positive impact on rejections.

5.4 Effects of User Login Behavior

Figure 17 shows the impact of user access patterns on the VAN resource utilization. As depicted in Figure 17(a), if users are staying longer in the system, they need more resource from VAN; hence, the network utilization increases. On the other hand, as the utilization becomes closer to 100%, VAN stabilizes the utilization around 98%. The standard deviation in user demand periods (Figure 17(b)), on the other hand, does not impact the utilization in any

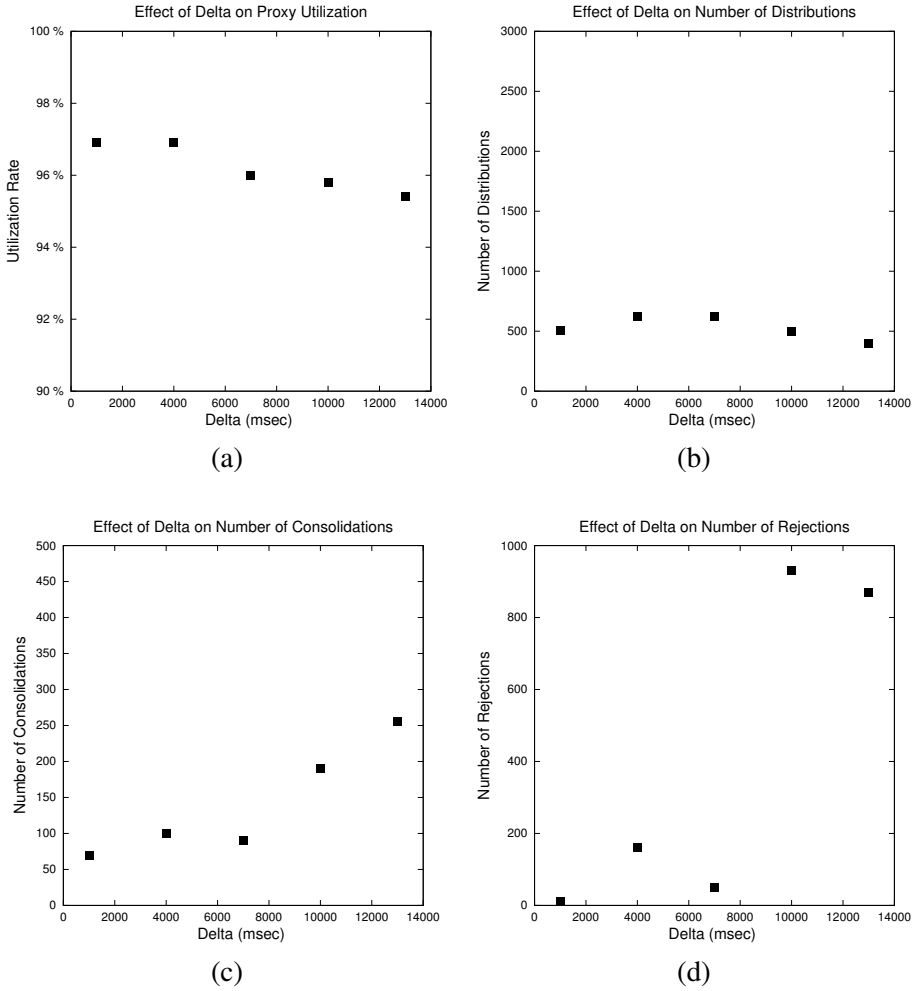


Figure 19: Effect of the Restructuring Overhead on (a) the Overall Network Utilization, (b) the Number of *DISTRIBUTE*, (c) the Number of *CONSOLIDATE*, and (d) the Number of *REJECT* Events

visible manner. This shows that the system is capable of using the resources made available by early-leavers to accommodate the long-stayers.

Figure 18(a) shows that the total number of distribute events generated is not significantly affected when users stay longer in the systems. Of course, the peak number of users in the system increases slightly when users stay longer; however, the system is capable of accommodating these users by increasing the resource utilization as was shown earlier, in Figure 17(a).

Figure 18(b) shows that the number of consolidations can be affected by user access patterns in several ways. First if all, if users are staying in the system a little longer (5mins to 10mins in the figure), this can reduce the number of consolidations required as a higher utilization of the existing resources is needed to support the additional load. On the other hand, a converse effect can be observed for large increases (10 mins to 25 mins in the figure), when additional servers have to be brought in to accommodate the additional user load. These servers will send *CONSOLIDATE* requests at the end of the broadcast, increasing the total *CONSOLIDATE* messages.

Note that, an interesting consequence of users' staying longer in the system is a reduction in the total number of rejections, as depicted in Figure 18(c). This is due to the significant reduction in the number of consolidations.

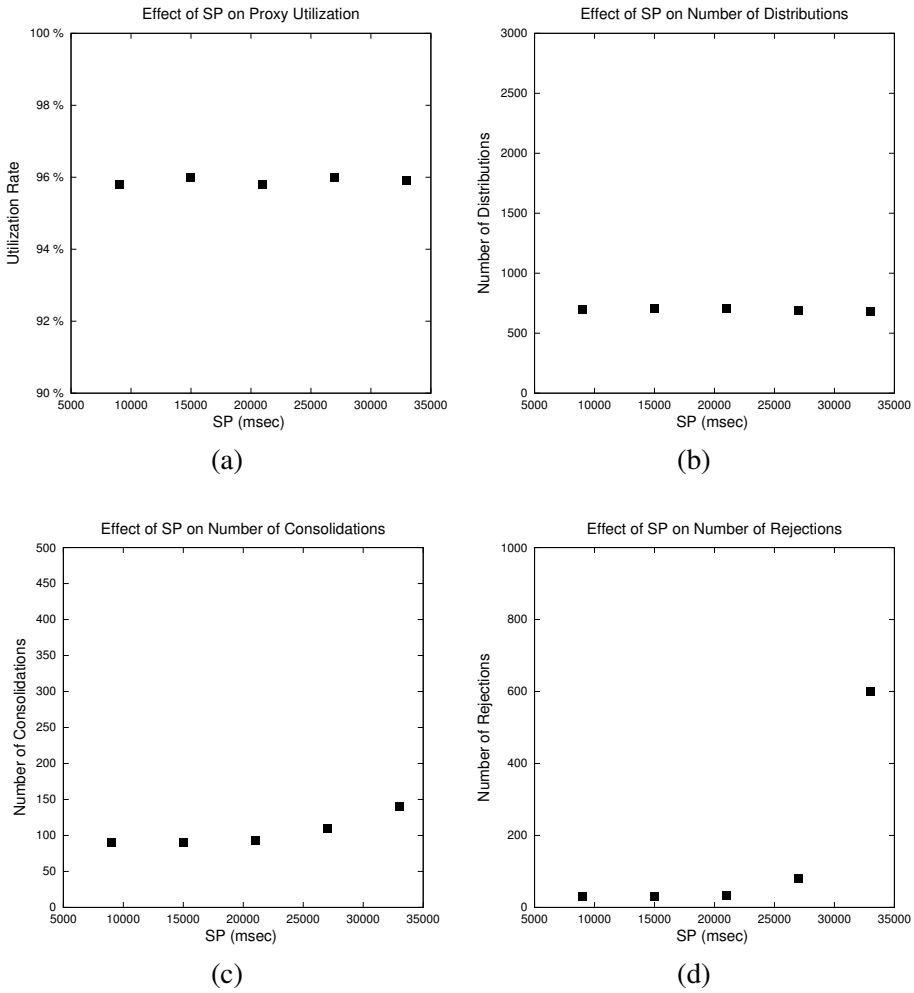
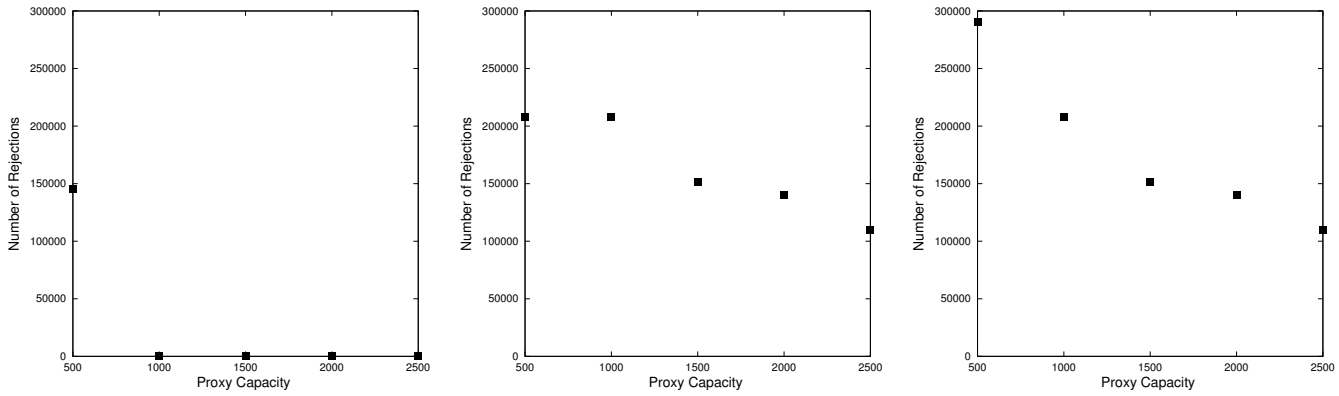


Figure 20: Effect of the Stability Parameter on (a) the Overall Network Utilization, (b) the Number of *DISTRIBUTE*, (c) the Number of *CONSOLIDATE*, and (d) the Number of *REJECT* Events

The slight increase in the number of *CONSOLIDATE* requests at the end of the broadcast (due to users' staying longer in the system) does not lead to additional rejections since the rate of arrivals is already close to zero at the end of the broadcast.

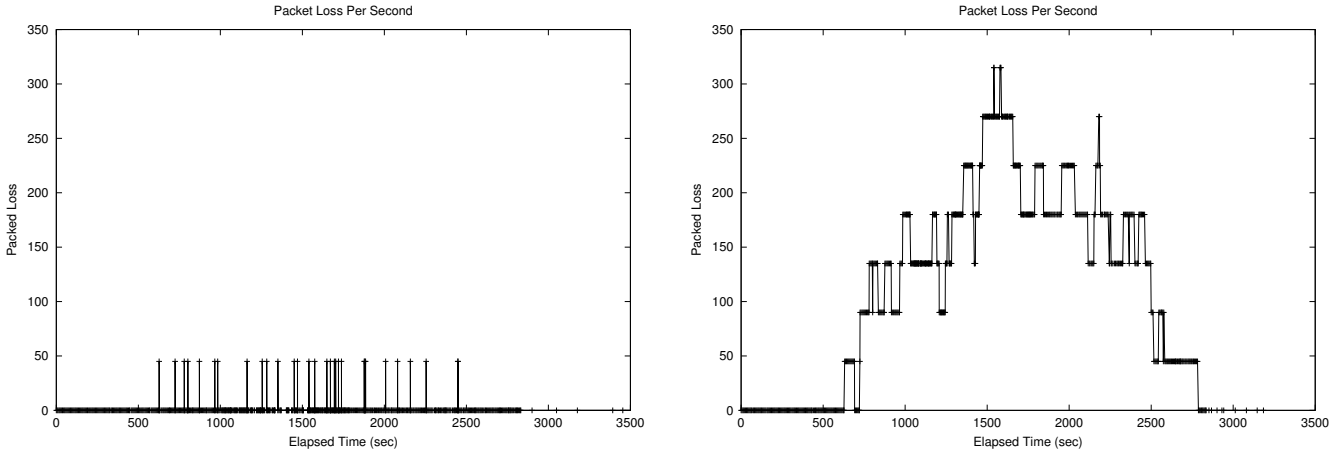
5.5 Effects of Restructuring Overhead and Stability Parameter (ΔT and sp)

As we have discussed in Section 3, dynamic rerouting does come with an overhead. Depending on the network conditions and the PNC load, this overhead may vary. Figure 19 shows the effect of this overhead on the performance of the VAN. As ΔT increases there is a slight drop in the utilization. This is the result of the time-lag between the time a server is created and the actual time it starts accepting new users. Basically, the longer the ΔT, the longer a server is idle. This causes a slight decrease in the utilization as shown in Figure 19(a). Another observation is that when switching takes longer, this leads to an increase in the total number of rejections (Figure 19(d)). This is due to the increase in the prediction difficulty. As ΔT increases, it takes longer for $CONSOLIDATE$ to occur. As a result, many proxies are included in the server farm for reuse before a CONSOLIDATE event can occur. As a result, there are fewer CONSOLIDATE (Figure 19) as ΔT values increase while the number of DISTRIBUTE



(a) (b) (c)

Figure 21: Effect of Not Consideration (a) *Server Farm*, (b) ΔT, and (c) *Server Farm* and ΔT



(a) (b)

Figure 22: Packet Loss Rate of Streaming Media Delivery (a) with dynamic routing and (b) without dynamic routing

events remain about the same.

ΔT is a physical constraint for dynamic rerouting in VAN. On the other hand, sp is a logical constraint, specified by VAN operators. The VAN operators can use the parameter sp to control the overall stability of VAN as discussed in Sections 3 and 4. Figure 20(a) shows that the effect of sp on average utilization is negligible. This shows that the system tries to adapt to the changes, by readjusting the amount of distributions and consolidations (Figures 20(b) and 20(c)). In order to have high utilization, the system does more consolidations as the sp parameter increases. The only problem associated with long sp is the increase in the number of rejections. More consolidations leads to a larger number of rejections as shown in Figure 20(d). In order to maintain a low rejection rate, the system needs to plan on and execute DISTRIBUTE earlier than it is needed to ensure that there is enough capacity for login users within the period of sp, the logical constraint that no two DISTRIBUTE or CONSOLIDATE actions are carried out within sp. Another negative effect of a large sp is that, in general, user login patterns are less predictable in a longer period. Thus, a larger sp value tends to lead to a more visible rejection rate.

In Figures 11 and 12, we show how our architecture consider the restructuring overhead (ΔT) and dynamic construction of server farm, which is essential in live broadcast events since burst traffic tends to occur at the beginning of the live broadcast events. We also implement the algorithms that do not consider the restructuring overhead ΔT and server farm to deal with burst traffic and test the algorithms with the baseline experiment settings. Figures 21(a), 21(b), and 21(c) show that there will be a large number of login rejections if the restructuring overhead (ΔT) and dynamic construction of server farm are not deployed. And the numbers of user login rejections are even higher when the proxy servers have low capacity.

5.6 Network Congestion

Figure 22 shows the impact of dynamic routing on the quality of the stream reaching the users. In this experiment, we randomly introduced network congestion (i.e. packet loss) on a certain percentage of links. Since VAN can monitor network links, once it notices network congestion or packet loss in one of the links, it dynamically adjusts the delivery path and the hierarchical structure to avoid such links. In Figure 22(a), we see that more links experience congestion at the peak time of user logins (as indicated by the higher frequency of packet loss observed). Figure 22(a) shows that the number of packets dropped in the system is limited to 50 packets per second in the entire system even with very high resource utilization. On the contrary, the number of packets dropped increases significantly, as the total number of users in the system increases during the broadcast, when dynamic restructuring is not used (Figure 22(b)).

6 Related Work

We have discussed some related work in the introduction section. Here we give an overview of additional related work and compared them with our approach.

Su and Yemini [7] have proposed a virtual active network architecture over wide-area networks. The goal of this effort is to develop an API that will eventually be incorporated in vendor-supplied black-boxes which currently perform network layer functions such as routing. Other efforts in this direction include virtual local area networks (VLANs) [14], virtual private networks (VPNs) [15], and ATM virtual path connections (VPCs) [16].

The X-Bone [8] is a system for dynamic deployment and management of Internet overlay networks. Current overlay networks include commercial VPNs [15] and IP tunneled networks M-Bone [17]. The X-Bone automatically discovers available components, configure, and monitors them. Due to its low-level implementation the X-Bone is limited in the level of functionality that is needed by advanced networking applications. Recently, a group at CMU [18] has proposed and implemented Virtual Network Service (VNS), a value added network service for deploying VPNs in a managed wide-are IP network. Their approach to provision customizable service leverages on a programmable router architecture that provides an open programmable interface [19]. The RON project [20] at MIT has built a resilient overlay network that allows distributed Internet applications to detect and recover from path changes and period of degraded performance within several seconds. Chawathe, McCanne and Brewer [21] have developed an architecture that provides application-level multicasting on top of existing IP networks with unicast capability. The architecture is referred to as *Scattercast* and is argued to be a more viable alternative to the efforts underway to integrate multicast at the IP level. The Scattercast architecture partitions a heterogeneous

set of session participants into disjoint data groups. Each data group is serviced by a strategically located network agent. Agents organize themselves into an *overlay network*. Chu, Rao, and Zhang [22] have proposed end system multicast to support small-sized applications such as audio or video conferencing. This approach is not scalable since it requires that every member maintain a complete list of every other member in the group.

Shi and Turner [23] have proposed a centralized greedy heuristic, called the Compact Tree algorithm to minimize the maximum latency from the source to a Multicase Service Node (equivalently, a proxy). Similarly, Banerjee et al. [24] have proposed a multicast overlay tree construction algorithm to minimize average-latency from the source to leaf level multimedia service nodes or proxies. In that these efforts are focused on the objective of optimization from the perspective of individual end-users. In contrast, our goal is to enable distribution of streaming media by minimizing network resources. Thus, the scheme proposed in this paper exploits the network components in such a way that a large number of end-users can be served with minimum bandwidth as well as with minimal number of proxies. This design strategy frees up network resources for simultaneous usage of multiple multicast streams effectively.

SpreadIt [25] is an application level multicast architecture for delivering streaming live media over a network of clients, using the resources of the clients themselves. ZIGZAG [26] also takes a similar approach and create media stream hierarchies over clients themselves. Other peer-to-peer structures based on clients include [27, 28]. In particular, the Bayeux approach builds on the P2P routing scheme based on Tapestry to create a highly redundant multicast tree over the peers in the network. The system also uses the Tapestry lookup mechanism to efficiently locate multimedia sessions. Although the approach is highly fault-tolerant it uses probabilistic approaches to minimize bandwidth usage. Recently, Yang [29] considers a more elaborate model for streaming media application in which the individual links do not have the capacity to accommodate a stream and hence multiple delivery path(s) need to be established between source and destination(s). Kim, Lam, and Lee [30] have recently presented a distributed algorithm that builds a multicast tree in which the average receiving rate, computed over all receivers in the tree, is maximized.

In addition to the above-mentioned research activities, many CDN vendors provide various solutions for streaming media delivery on the Internet. In the solutions provided by Akamai[9], streaming media is distributed close to the users. It also reduces bandwidth requirement at the origin media server site because media is served from the edge cache servers. RealProxy by RealNetworks[11] is software installed on a network or ISP gateway that aggregates and handles client requests for media streamed. In contrast to VAN's dynamic adjustment of delivery network hierarchical structure to user population and distribution as well as network conditions, RealProxy requires manual specification and adjustment of multiplexing structure.

7 Concluding Remarks

We have presented a peer-to-peer virtual active network (VAN) system architecture for streaming data delivery. A number of proxy servers are coordinated to deliver streams. The hierarchical structure activated for delivery is dynamically reconfigured to minimize bandwidth utilization and improve network performance. We have showed the advantages of VAN architecture over existing work and experimental results establish the effectiveness and usefulness of VAN. In this paper, we focused our discussion on centralized coordination of VAN. A major difference between a fully distributed and centralized versions of VAN is the process through which parent proxy servers

are selected. The load management and redirection schemes are essentially the same. A hybrid version could be easily implemented by assigning one proxy in each tier as the coordinator, maintaining local information and communicating with the coordinators in its neighboring tiers. Thus, the current architecture can be easily extended for decentralized coordination of the proxy network.

References

[1] S. Acharya and B. Smith. Middleman: A Video Caching Proxy Server. In *Proceedings of the 10th International Workshop on Network and Operating Systems Support for Digital Audio and Video*, 2000.

[2] R. Rajaie, H. Yu, M. Handley, and D. Estrin. Multimedia Proxy Caching Mechanism for Quality Adaptive Streaming Applications on the Internet. In *Proceedings of the 2000 IEEE INFOCOM Confeence*, March 2000.

[3] S. Jin, A. Bestavros, and A. Iyengar. Accelerating Internet Streaming Media Delivery using Network-aware Partial Caching. In *Proceedings of the International Conference on Distributed Computing Systems*, 2002.

[4] Y. Wang, Z. L. Zhang, D. H. Du, and D. Su. A Network Conscious Approach to End-to-end Delivery over Wide-area Networks Using Proxy Servers. In *Proceedings of the 1998 IEEE INFOCOM Conference*, 1998.

[5] S. Sen, J. Rexford, and D. Towsley. Proxy Prefix Caching for Multimedia Servers. In *Proceedings of the 1999 IEEE INFOCOM Conference*, April 1999.

[6] Z. Miao and A. Ortega. Proxy Caching for Efficient Video Services over the Internet. In *Proceedinsg of the PVW Conference*, 1999.

[7] G. Su and Y. Yemini. Virtual Active Networks: Towards Multi-edged Network Computing. *Computer Networks*, 36:153–168, 2001.

[8] J. Touch. Dynamic Internet Overlay Deployment and Management Using the X-Bone. *Computer Networks*, 36:117–135, 2001.

[9] Akamai Technology. *Information available at* `http://www.akamai.com/`.

[10] Inktomi. *Information available at* `http://www.inktomi.com/`.

[11] RealNetworks. *Information available at* `http://www.real.com/`.

[12] Ramesh Govindan and Hongsuda Tangmunarunkit. euristics for Internet Map Discovery. In *Proceedings of the 2000 IEEE INFOCOM Conference*, pages 1371–1380, Tel Aviv, Israel, March 2000. IEEE.

[13] Y. Breitbart and M. Garofalakis and C. Cli Martin, R. Rastogi and S. Seshadri and A. Silberschatz. Topology Discovery in Heterogeneous IP Networks. In *Proceedings of the 2000 IEEE INFOCOM Conference*, Tel-Aviv. Israel, March 2000. IEEE.

[14] D. Passmore and J. Freeman. The Virtual LAN technology. Technical Report 200374-001, 3COM, 1997.

[15] C. Scott, P. Wolfe, and M. Erwin. *Virtual Private Networks*. OReilly, Sebastopol, 1998.

[16] V. J. Friesen, J. J. Harms, and J. W. Wong. Resource Management with virtual paths in ATM networks. *IEEE Network*, 10(5):54–60, 1996.

[17] H. Eriksson. MBone: the Multicast Backbone. *Communications of the ACM*, pages 54–60, aug 1994.

[18] L. K. Lim, J. Gao, T. S. E. Ng, P. R. Chandra, P. Steenkiste, and H. Zhang. Customizable Virtual Private Network Service with QoS. *Computer Networks*, 36:137–151, 2001.

[19] E. Takahashi, P. Steenkiste, J. Gao, and A. Fischer. A Programming Interface for Network Resource Management. In *Proceedings of the 1999 IEEE Open Architectures and Network Programming*, pages 34–44, mar 1999.

[20] David G. Andersen, Hari Balakrishnan, M. Frans Kaashoek, and Robert Morris. Resilient Overlay Networks. In *Proc. SOSP 2001, Banff, Canada*, oct 2001.

[21] Y. Chawathe andS. McCane and E. Brewer. An Architecture for Internet Content Distribution as an Infrastructure Service. http://yatin.chawathe.com/ yatin/papers/scattercast.ps, 2002.

[22] Y. H. Chu, S. G. Rao, and H. Zhang. A Case for end system multicast. In *Proceedings of the ACM SIGMETRICS Conference*, June 2000.

[23] S. Shi and J. Turner. Routing in Overlay Multicast Networks. In *Proceedings of the 2002 IEEE INFOCOM Conference*, June 2002.

[24] S. Banerjee, C. Kommareddy, K. Kar, B. Bhattacharjee, and S. Khuller. Construction of an Efficient Overlay Multicast Infrastructure for Real-time Applications. In *Proceedings of 2003 IEEE INFOCOM Conference*, 2003.

[25] H. Deshpande, M. Bawa, and H. Garcia-Molina. Streaming Live Media over a Peer-to-peer Network. Submitted for publication, 2002.

[26] D. Tran, K. Hua, and T. Do. Zigzag: An Efficient Peer-to-peer Scheme for Media Streaming. In *Proceedings of the IEEE 2003 NFOCOM Conference*, 2003.

[27] Venkata N. Padmanabhan, Helen J. Wang, Philip A. Chou, and Kunwadee Sripanidkulchai. Distributing Streaming Media Content Using Cooperative Networking. Technical Report MSR-TR-2002-37, Microsoft Research, Redmond, WA, 2002.

[28] S. Q. Zhuang, B. Y. Zhao, A. D. Joseph, R. Katz, and J. Kubiatowicz. Bayeux: An Architecture for Scalable and Fault-tolerant Wide-area Data Dissemination. In *Proceedings of the 11th International Workshop on Network and Operating Systems Support for Digital Audio and Video*, 2001.

[29] J. Yang. Deliver Multimedia Streams with Flexible QoS via a Multicast DAG. In *Proceedings of the International Conference on Distributed Computing Systems*, 2003.

[30] M. S. Kim, S. S. Lam, and D. Lee. Optimal Distribution Tree for Internet Streaming Media. In *Proceedings of the International Conference on Distributed Computing Systems*, 2003.