Chapter 1

POLICY-BASED RESOURCE SHARING IN STREAMING OVERLAY NETWORKS

K. Selçuk Candan*, Yusuf Akca, and Wen-Syan Li†
NEC Laboratories America,
10080 North Wolfe Road, Suite SW3-350, Cupertino, CA 95014, USA.
{candan,yakca,wen}@sv.nec-labs.com

Abstract In this chapter, we discuss peer-to-peer media streaming overlay network architectures and introduce a policy-based architecture for streaming live media from media sources to end-users over independently owned and operated networks. This architecture (mSON) efficiently supports multiple simultaneous media streams, with different sources and user populations, through *shared* overlay resources. The overlay network infrastructure takes into account the existence of multiple content providers (media sources) and minimizes its footprint to use available resources most effectively. In the meanwhile, it prevents resources from being overutilized to prevent congestions, service rejections, and jitters in the streams users receive. We report experimental results that show that the policy-based mSON achieves these tasks efficiently and effectively.

Keywords: Overlay networks, multicasting, live media streaming, policy-based QoS, peer-to-peer adaptation, multi-source, multi-sink

Introduction

Wide-area data dissemination of streaming media poses significant challenges in the context of current Internet structure. Without appropriate multicasting mechanisms, network routes are quickly congested.

*Current address: CSE Dept., Arizona State University, Tempe, AZ 85287, USA. This chapter describes work performed during author's visit at NEC.
†Current address: IBM Almaden Research Labs, San Jose, CA 95120, USA. This chapter describes work performed while the author was employed at NEC.

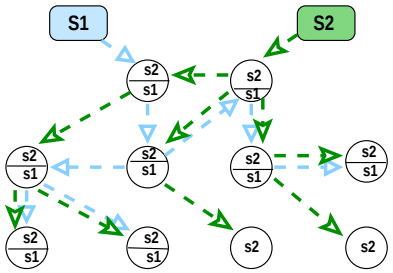


Figure 1.1. A multicasting overlay network supporting multiple content providers (media sources) s1 and s2 (proxies and network resources are shared)

IP multicasting based solutions do not go far due to the incompatibility of the various independent network elements. Overlay based solutions address the incompatibility and interoperability problems at the physical network layer by building virtual networks on top of the underlying physical network. By distributing the user 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 network. Also, by regulating the number of proxies on the delivery path, they limit the jitter in the packets being delivered to end-users. Since they maintain service agreements with the ISPs and since they oversee the creation and maintenance of multicast hierarchies (with predictable lifetimes), they deploy policies and resource reservation protocols to prevent fluctuations in the service.

In this chapter, we present a policy-driven peer-to-peer media streaming overlay network (mSON) architecture for streaming live media over independently owned/operated networks (Figure 1.1). The physical network on which the overlay operates consists of multiple autonomous systems (ASs), each of which can host proxy servers that can act as Multicast Service Nodes or (MSNs) [Shi and Turner, 2002, Banerjee et al., 2003]. Each AS is an independent entity; therefore, the overlay has to compensate ASs for the resources consumed. The last mile (i.e., the final leg of the streams from the source to the end-users' systems), on the other hand, is charged to end-users by their ISPs (usually in terms of monthly cable modem or DSL subscription charges). Therefore, the goal of the overlay is to deploy its network-wide proxies to bring streamed media closer to the last-mile, while minimizing its own operational costs. The dynamic adjustment of proxy network is based on parameters including (1) concurrent media sources, (2) distribution of user requests, (3) network conditions, and (4) capacity of proxy servers.

mSON approach to peer-to-peer media streaming differs from most existing approaches in its optimization criteria. Most systems aim at creating multicast trees in a way that latency observed by the end-users of the corresponding media source is minimized. For instance, [Banerjee et al., 2003] formulates the overlay networking problem as a *degree constrained minimum average latency problem* [Blum et al., 1994] and develops multicasting algorithms that addressed this problem. Most existing work, however, misses two critical constraints faced by media overlay networks:

- An overlay network has to support multiple simultaneous media streams, each with a different source and user population. The network infrastructure should take into account the existence of multiple media sources *sharing* the same resources (Figure 1.1).
- The overlay network should minimize its footprint on the network to minimize its operational costs and use its resources most effectively. However, resources should not be overutilized; otherwise, end-users will observe congestions, service rejections, and jitters.

The mSON architecture considers these two aspects of an effective media streaming architecture in addition to the average latency provided to the end-users. In other words, unlike most existing work, the overlay is explicitly optimized for resource (link and node) stress [Castro et al., 2003b] (or tension) along with the depth of the multicast trees. Unlike related work in resource allocation for overlay multicasts, such as [Cui et al., 2003] which presents optimal resource allocation schemes for single source network, mSON is specifically designed for shared, multi source, networks.

Another aspect that differentiates mSON style overlay networking from most recent peer-to-peer multicasting [Xu et al., 2002, Banerjee et al., 2003, Banerjee et al., 2002, Blum et al., 1994] work is that, instead of focusing on peers that independently arrive and leave the system, mSON is specifically designed to consider peer-to-peer negotiations between overlay's own proxies. Consequently, except during failures or at the edges where user arrival is not in overlay's control, the way proxies join and leave a multicast tree is controllable and is based on mutual agreements that will benefit the overlay in the long run. mSON leverages this to ensure that overlay's resources are used most efficiently.

End-Users and Network Resources

The Internet consists of autonomous systems (ASs). User requests are captured at (or redirected to) the proxies that are in the same AS

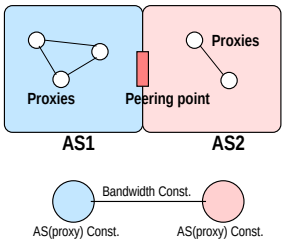


Figure 1.2. Two ASs connected with a peering point and the corresponding resource constraints

as (or close to) the users. Overlay network resources (such as proxies) are placed in an AS after resource and QoS agreements signed. Consequently, the overlay service provider has to compensate each AS for the bandwidth streamed through it, while providing certain bandwidth and QoS guarantees as specified with the agreement. ASs are connected to each other through peering points (or borders) (Figure 1.2). Each peering point has a maximum bandwidth capacity as specified in the service agreements. Therefore, one of mSON's primary goals is to minimize the amount of traffic that flows across AS boundaries. On the other hand, purchasing, deploying, and maintaining a proxy server in an AS is costly. Therefore, mSON also aims to minimize the total number of proxy servers that needs to be deployed to carry a given workload.

Policy Description

Most existing work on utility-based multicast routing, such as [Kelly et al., 1998] and [Kar et al., 2001], are focussed mainly on the use of pricing mechanisms to efficiently manage network resources. [Cui et al., 2003] provides a rate based overlay multicasting model based on the maximization of a utility function over all receivers in the overlay network. In overlay systems, however, resources of the proxies are as valuable as that of the underlying network [Amir et al., 2002]. In a streaming overlay network, the role of each overlay proxy server is to multiplex the stream it receives from another proxy and serve the resulting streams to other proxies or to end-users.

Without a loss of generality, let us model the proxies in a given AS as a single virtual proxy, analogous Multicast Service Nodes, as in [Banerjee et al., 2003]. Each AS has a multiplexing capacity, determined by the processing power of the available proxy servers in the AS and the local bandwidth availabilities. Given the amount of resources in a given AS, the overlay needs to maintain a level of resource utilization. Underutilization of the resources, such as bandwidth, is not desirable as in

many cases the overlay service provider has to pay for resources allocated even if they are not utilized. Like underutilization, overutilization of the available resources is also not desirable as overutilization can lead to congestions, rejection of user requests, jitters, and vulnerability to sudden changes in the load. mSON models the effects of these two factors, underutilization and overutilization, in terms of an intuitive *policy* concept, which represents the underlying *costs* associated with deviating from the intended proxy resource utilization rate:

- resource tension is minimal (t_{pref}) when a resource is used at its intended load (l_{pref}).
- tension rises rapidly (to ∞) when the load on the resources reaches unacceptable high rates, l_{limit}.
- tension also increases as the utilization drops below the preferred rate as the overlay has to pay for these underutilized resources, and
- the resource tension is 0 when the resource is not utilized at all. If the workload can be maintained while keeping a particular resource at 0 utilization, then the resource can be removed from the system or can be re-allocated for other services the overlay provides.

Thus the concept of tension captures not-only real-time optimization needs (prevention of overutilization of resources), but it also favors releasing of resources altogether if possible, promoting mid- to long-term efficient allocation of resources. In general, an interaction between two resources, r_i and r_j is *desirable* if and only if

$$tension_i(l_{before,i}) + tension_j(l_{before,j}) > tension_i(l_{after,i}) + tension_j(l_{after,j}).$$

Although resource interactions are limited to those that reduce the overall tensions, user-resource interactions can increase the overall tension in the system. Although for an underutilized resource, addition of a new user will reduce its tension, for a resource being utilized at or above its preferred load, a new user will cause an increase in the tension. After a user interaction which increases the tension (such as a user leaving an already underutilized resource or a new user requesting an overutilized one) proxies can interact to reduce the overall tension in the system.

Given two alternative loads, l_1 and l_2, the policy graph describes which load is more desirable (i.e., $tension(l_1) < tension(l_2)$ or vice versa). Figure 1.3 presents two sample policy (tension) graphs, corresponding to different overlay service provider policies. These two policy graphs

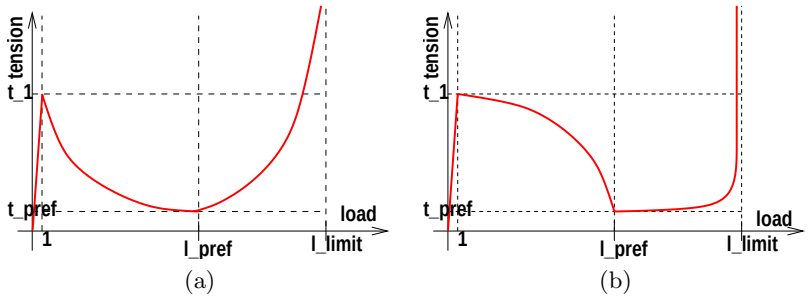


Figure 1.3. (a) An example resource policy graph favoring balanced utilization of available resources; (b) an alternative policy graph favoring the creation of a small *core* of resources carrying the current workload

differ in that they represent different cost structures for underutilized and overutilized proxies. The graph in Figure 1.3(a) gets flatter as the utilization increases towards the preferred rate. If an overlay is using this policy graph, all *used* proxies are likely to fill up at the same rate. The slope in Figure 1.3(b) on the other hand gets steeper as the load increases towards the perfect utilization rate. Consequently, the second graph favors those proxies whose utilization is already close to perfect when choosing which proxy to use for new requests. The proxies that are almost empty are not loaded with additional loads. Thus, a few proxies would reach the perfect utilization quickly, and only after that other proxies would start being utilized. Both of these graphs provide high penalties to resources that are very lightly used; in other words, both graphs prefers a non-utilized proxy to a very lightly utilized one. In both cases, there is a cost associated with activating a new proxy in the proxy network and maintaining that proxy at a very light utilization rate. The two graphs also differ in the way they treat overutilized proxies. While the penalty of overutilizing a proxy increases sharply for the resource presented in Figure 1.3(a), the policy modelled in Figure 1.3(b) allows proxies to be loaded very close to their limits.

Policy-based Resource Allocation on Shared Networks

Given the resource utilization policies described above, the overlay's task is to delivering multiple streams to end users, using as little resources as possible, while preventing congestions and reductions in QoS. Formally, at a given point in time, the state of the overlay network is described as follows:

- overlay owns and operates a set, $\mathcal{P}$, of proxies.

- overlay operates in an AS network, $G(\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is the set of ASs the overlay has agreements with and $\mathcal{E}$ is the set of peering points (or borders) across these ASs. Each proxy $p_i \in \mathcal{P}$ is associated with a an as denoted as $loc(p_i)$. As stated earlier, without a loss of generalization, we assume that each network node contains at most one proxy).
- overlay's resource utilization policy that applies to a proxy $p_i \in \mathcal{P}$ is captured by a policy graph, $ptension_i$, as described earlier. Specifically, $pcap_i$ denotes the total multiplexing capacity (due to buffer, CPU, or local bandwidth limitations), of the proxy and $putil_i$ denotes its intended rate of utilization.
- each peering point $e_j \in \mathcal{E}$, has a maximum bandwidth capacity denoted as $bcap_j$ determined based on the service agreements. For each peering point, e_j, the overlay associates a policy $btension_j$ and an intended utilization $butil_j$.
- overlay simultaneously serves media from its media providers, $\mathcal{S}$, where each provider has an associated source $s_i \in \mathcal{S}$ hosted at an origin proxy $host(s_i) \in \mathcal{P}$.
- end-users access streaming media by forwarding requests to the proxies of the overlay network. Therefore, at each proxy p_i, there is a $load(p_i, s_j)$ associated with the media served by source s_j.

The state of the network is subject to changes. The goal of the overlay is to adapt to these changes and deliver media streams using as little resource as possible, while providing low end-to-end delays. To achieve this task, mSON deploys proxy-based overlay multicasting structures. Given a set of proxies, an AS-level network structure, a set of media sources, and user access loads, mSON creates multicast trees such that

- resources are not overutilized,
- resources are not underutilized, and
- end-to-end delays are small.

Overlay proxies are shared resources, serving multiple streams simultaneously. Therefore, unlike other multicasting approaches, mSON does not focus on individual multicasts, but takes a holistic approach, where the shared resources are allocated in a way to support all streams the overlay serves at a given point in time. Therefore, mSON creates and maintains a set, $\mathcal{T}$, of multicast trees for the overlay network over the shared resources. For each active media source s_i mSON maintains a (directed) multicast tree, $T_i(\mathcal{V}_i, \mathcal{C}_i) \in \mathcal{T}$, where $\mathcal{V}_i \subseteq \mathcal{V}$ and each $\mathcal{C}_i$ is

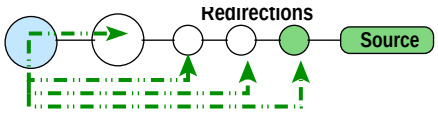


Figure 1.4. Tunneling redirections instead of activating proxies close to the edge

a path on $\mathcal{E}$. The root of the each tree T_i is a media source and the leaves of the tree are the proxies which serve end-users. A single proxy-to-proxy connection on the overlay network may pass through multiple ASs depending on the actual network structure. Therefore, each overlay connection $c_j \in \mathcal{C}_i$ between two proxies has a corresponding set of edges (peering points), $path(c_j) \in 2^{\mathcal{E}}$, denoting the path it follows on the actual AS structure $\mathcal{G}$. The trees maintained by mSON respect the overlay's proxy and network resource utilization policies:

$$\forall p_i \in \mathcal{P} \;\; pload_i = \sum_{T_j \in \mathcal{T}} out_{j,loc(p_i)} \simeq putil_i,$$

and

$$\forall e_k \in \mathcal{E} \;\; bload_k = \sum_{T_j \in \mathcal{T}} [c_l \in \mathcal{C}_j][e_k \in path(c_l)] \simeq butil_k.$$

In other words, the overall resource tension in the system should be minimal:

$$(\sum_{e_j \in \mathcal{E}} btension_j) + (\sum_{v_j \in \mathcal{V}} ptension_j).$$

Note that this formulation implicitly minimizes the latency of the resulting multicast trees; since each edge (even those edges that are used at their perfect utilization levels - see Figure 1.3) adds to the overall tension, the use of unnecessary edges is strongly discouraged. Thus, the latency of the resulting trees are implicitly minimized (subject to resource constraints and preferred utilization rates) as well.

Policy Cross-Talk

As described above, mSON identifies on two different types of resources: proxies and bandwidth. Although these are different, their policies need to be related. First of all, irrespective of its non-network resources (such as CPU and memory), a proxy can not multiplex more streams then the network bandwidth available at its outgoing peering points allow. Also, although an AS could tunnel streams to other ASs without having to use local multiplexing resources, the amount of available peering point resources bound the multiplexing power of the proxy;

thus, since it consumes network bandwidth at the peering points, tunneling would impact (reduce) the multiplexing capacity of the proxy.

On the other hand, if network tunneling is cheap (i.e., network resources are not constrained, then overlay may not have an incentive to activate proxies closer to the edge and may choose to use such tunnels instead (Figure 1.4). As a result, the overlay may create shallow multicast trees with long proxy-to-proxy network connections, where most edge proxies are simply connected to the proxies closer to the root. Therefore, the cost of creating a new proxy, the network cost, and the multicast trees overlays create are interdependent.

In the next section, we provide an overview of the decentralized protocols mSON uses to achieve policy-driven resource allocation.

Protocols needed by Peer-to-Peer Media Streaming Overlay Network

Operation of a peer-to-peer media streaming overlay network requires three complementary protocols:

- a media streaming protocol,
- an information exchange protocol between peers to keep them aware of the status of the overlay network, and
- a multicast management protocol to create and manage the multicast trees for efficient utilization of resources.

In this section, we provide an overview of these three protocols. Since the underlying peer-to-peer decision making processes are embedded in the multicast management protocol, in the rest of the chapter, we focus on the third of these three protocols and discuss it in greater detail.

Media Streaming Protocol (MSP). The media streaming protocol is specific to the media types delivered through the overlay and specifies how media is transmitted from the source to the end-users through a chain of proxies established by the overlay. These include control protocols, such as *Realtime Streaming Protocol* (RTSP) and *Realtime Control Protocol* (RTCP), and transport-level protocols, such as *Realtime Transport Protocol* (RTP). In this chapter, we do not assume any specific media streaming protocol.

Information Exchange Protocol (IEP). In a decentralized overlay operating through negotiations between proxies in the network, there is a need for an information exchange protocol that will enable proxies to collect up-to-date information about the environment they operate

Inform.	*Meaning*
$\mathcal{N}_i$	the list of proxies that are its *neighbors*. The concept of neighborhood is logical. Initially, the list contains its physical neighbors; however, as the load increases, the neighbor list may extend to include proxies that are not immediate neighbors.
$bload_k$ $btension_k$	the current load and the tension(respectively) of each path, e_k, to its neighbors,
$pload_j$ $ptension_j$	the current load and the tension (respectively) of each proxy $p_j \in \mathcal{N}_i$ in its neighborhood.
$minbcost_{i,j}$	the minimum cost of sending a stream from source s_j to p_i.

Table 1.1. Information collected and maintained by proxy p_i

in. Based on the formalism introduced earlier, in Table 1.1, we highlight the information that each proxy, $p_i \in \mathcal{P}$, collects from its neighbors and from the sources in a distributed fashion. Each proxy collects this information through regular communications with the other proxies it knows. This regular information exchange also helps proxies to identify when the connection between two proxies becomes broken. Note that, a link may be declared failed, when the QoS over the link becomes unacceptably low even though control-messages travel without any problem. This is taken care of network monitoring processes that run on each proxy. In addition to periodic information exchange, each proxy can also piggyback its current list of values to other messages it exchanges with its neighbors or can explicitly demand new information when it senses that the status may have been changed (due to a failure or insertion of new sources).

Although most of the information in Table 1.1 requires knowledge about only the immediate neighborhood. In other words, mSON needs to estimate current source to proxy distances that will be used by the multicast management protocol to guide multicast tree creation process. On the other hand, estimating the minimum cost, $minbcost_{i,j}$, of sending a stream from a source, s_j to a proxy, p_i, (the minimum cost of sending a stream from source s_j to p_i based on the current network and proxy utilizations) requires a distributed routing (or distance-vector) style algorithm running on the overlay proxies. To eliminate errors during the estimation of the source to proxy distances that will be used by the multicast management protocol to guide multicast tree creation process, mSON uses distance estimation protocols (such as DUAL [Garcia-Lunes-Aceves, 1993], EIGRP [Albrightson et al., 1994], RIP-MTI [Schmid and Steigner, 2002], and LPA [Murthy and J.J.Garcia-Luna-Aceves, 1995]) which eliminate loops or minimize the undesirable effects of routing loops. Note that, even though the physical network may not be changing, the distance values (which measure the cost of streams) can vary

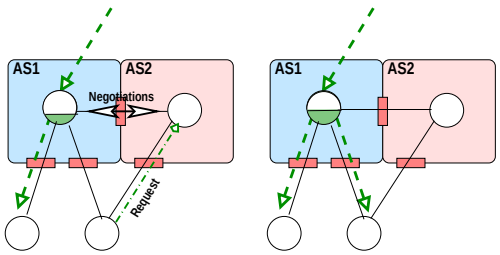


Figure 1.5. Request redirection: (a) before and (b) after

during the operation of the overlay based on the current load on the peering points. One way to eliminate the need for constant information exchange among proxies and to limit such exchange to only significant changes in the network structure, such as addition or removal of a proxy or network edge, is to calculates the shortest distance values based on the assumption that the network and the proxies are used at their preferred levels.

Multicast Management Protocol (MMP)

The multicast management protocol has two major tasks; creating the multicast trees that will be used for delivering streams from the sources to the end-users, and updating these multicast trees as the load and request characteristics in the overlay network changes. In the rest of this chapter, we provide an overview of how mSON achieves these tasks in a decentralized manner through communications and negotiations among neighboring proxies. mSON operates in a request-driven fashion. When a new source is inserted into the system, the only required *registration* or *initialization* process is to make the proxies aware of the new source. This can be achieved through a central lookup registry, by pushing the list of the sources to the proxies, or by letting proxies to discover sources in the system through the information exchange protocol discussed earlier. Otherwise, the operation of each proxy is driven by

- join requests from other proxies,
- stop requests from other proxies, and
- messages that initiate changes in the multicast tree structures.

In the sections, we discuss the various aspects of the Multicast Management Protocol (MMP) in detail.

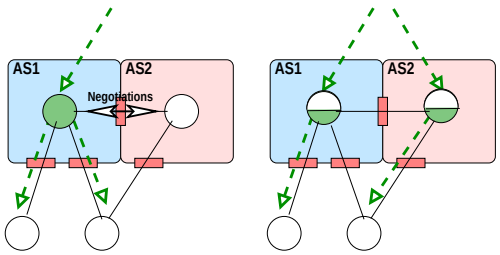


Figure 1.6. Load redistribution to prevent overutilization: (a) before and (b) after

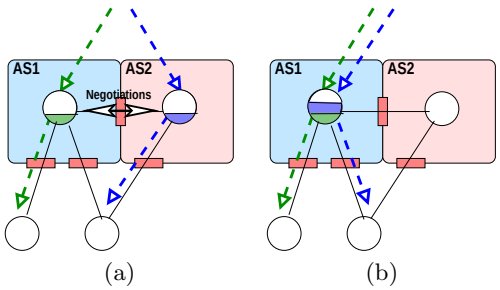


Figure 1.7. Load consolidation to prevent underutilization: (a) before and (b) after

1. Policy-Driven Peer Actions in a Shared Overlay Network

The mSON architecture uses decentralized proxy to proxy collaborations, including request redirections, load *redistributions*, and load *consolidations*, for optimizing resource utilization in the overlay network. Based on the available policy descriptions, a proxy which receives a request may redirect the request to a more suitable proxy in its neighborhood (Figure 1.5). An overloaded resource (Figure 1.6) may initiate a redistribution process which ships part of the load to a different proxy. Similarly, an underutilized resource may initiate consolidations (Figure 1.7) of overlay resources.

These policy-driven proxy actions ensure that the overlay network scales and adapts itself as media sources and end-users come and go and as multi-cast trees are created and torn apart. Consequently, unlike IP-multicasting [Eriksson, 1994], mSON understands and integrates AS-level service agreements. Furthermore, unlike most work in overlay multicasting (for example [Shi and Turner, 2002, Banerjee et al., 2003, Banerjee et al., 2002]), it is aware of the shared nature of the overlay network and aims optimizing the utilization of the network and proxy resources along with the average delay *within an integrated framework*. Unlike static overlay-network based solutions [RealNetworks, 2005], mSON

adapts to the varying data, network, and user conditions dynamically in a truly distributed, peer-to-peer manner.

Handling Join Requests

When a proxy p_i receives a request from a user or another proxy for access to the media stream multicasted at source s_j, it has to decide whether it should serve this request or forward the request to another, more suitable proxy (Figure 1.5). Since, proxies are working with incomplete knowledge of the status of the entire mSON, this step is especially important for ensuring that an appropriate search space is covered during the creation of the multicast trees. The decision to accept a new join request is based on the following criteria:

1 p_i checks if the request causes a service loop. Since loops are not desirable, if a loop is detected, then the request needs to be redirected to another proxy.

2 if p_i will get too overloaded when this request is admitted, p_i checks if redistributing its existing load to other servers can enable it to accept this new request.

   (a) if so, then it accepts the request and redistributes its load

   (b) if redistribution is not possible, then it checks if there is a suitable proxy in the neighborhood to serve the join request. If it finds one then p_i redirects the request to that proxy; otherwise, it denies the request.

3 even if p_i is not getting too overloaded, if there is a more suitable proxy in the neighborhood to serve the join request, p_i redirects the request to that proxy.

4 otherwise, if p_i is already serving the stream, then it admits the new request.

5 if p_i is not serving the stream yet,

   (a) if it has a more suitable neighbor, then it redirects the request to that neighbor,

   (b) otherwise, it admits the request and sends a join request to a *candidate* parent based on its neighbors' distances (costs) from the source and their current tensions.

Below, we discuss the underlying processes in greater detail.

Overload Detection and Elimination. Before admitting a join request, p_i has to check whether it is becoming too overloaded. Obviously, if the proxy resource utilization reaches the absolute maximum, the proxy can not accept new requests. To promote overall system performance, instead of defining overload in absolute terms, we say that the proxy is too overloaded if shedding a fraction (Θ) of its load to some other server (say p_h) will be beneficial to the overall operation of mSON; in other words, p_i is overloaded when

$$t\_reduction\_at\_i > (1 + \gamma_d) \times t\_increase\_at\_h$$

or

$$ptension_i(pload_i) - ptension_i((1 - \theta) \times pload_i) > (1 + \gamma_d) \times ptension_h(\theta \times pload_i).$$

Here, $0.0 \leq \gamma_d$ is the threshold for initiating a load redistribution. If this condition is satisfied, then mSON uses redirection and redistribution negotiations (Sections 1.0.0 and 1.0) between proxies to identify and handle detected overloads.

Identification of a More Suitable Neighbor to Serve a Request and Redirection. Before admitting a join request for a stream from source s_j, p_i checks if there is any proxy in its neighborhood better suited to admit this request than itself. p_i first calculates a self-cost of admitting the request (a negative cost denotes reduced tension; hence *benefit*). If p_i is already serving the stream, then the self-cost can be calculated as follows:

$$selfcost = ptension_i(new\_pl_i) - ptension_i(old\_pl_i)$$

If p_i is not yet serving the stream, on the other hand, the expected network cost (increase in the tension in the network resources) must be taken into account as well. Therefore, in that case, p_i estimates the cost of admitting this request as follows:

$$selfcost = ptension_i(new\_pl_i) - ptension_i(old\_pl_i) + minbcost_{i,j}$$

As discussed earlier , $minbcost_{i,j}$ is the cheapest distance based on the current tension values in the network between servers s_i and s_j. Clearly, this value is only an estimate of the actual network and costs: the request for a particular stream may be routed through a more costly route depending on actual resource loads or routing may cost much less if a multicast tree serving the stream is found nearby.

The costs of the request for the neighbors (say p_h) of p_i are also calculated in a similar manner:

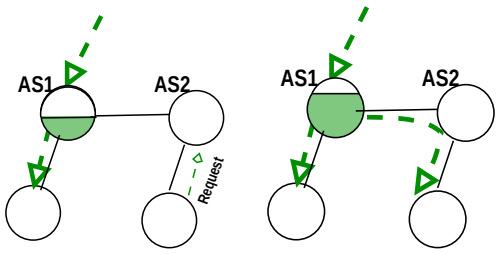


Figure 1.8. Request redirection and tunneling: proxy resources of AS2 are not utilized

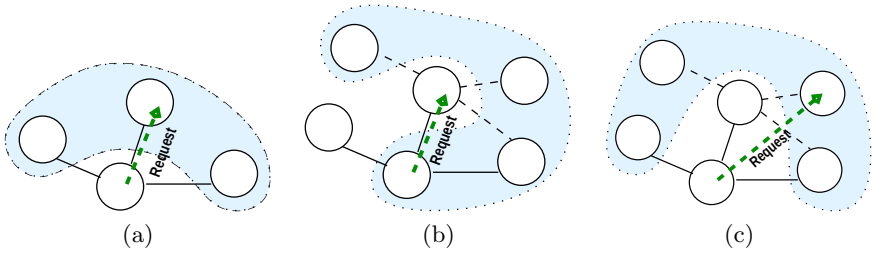


Figure 1.9. Expansion of the neighborhood through redirection: (a) A proxy sends a request to the best of its neighbors; (b) the neighbor chooses to redirect the request to one of its neighbors (it is either loaded or it knows that one of its neighbors can be better suited for this request; and (c) the original proxy now has a wider view of the neighborhood and can send a request to the best available proxy

$$cost_h = ptension_h(new\_pl_h) - ptension_h(old\_pl_h) + minbcost_{h,j} + btension_{i,h}(new\_bl_{i,h}) - btension_{i,h}(old\_bl_{i,h})$$

The main difference from the calculation of the self-cost is that, while estimating the cost for a neighboring proxy, p_h, p_i also considers the cost associated with acting as a network tunnel for the request, in case the requesting proxy does not have a direct connection to p_h (Figure 1.8). Of course, if the proxy originating the request and p_h have a direct connection, then that connection may be used after redirection; however, at this point p_i does not have this information and has to account for the worst case scenario. Also, in order to speed up the process, the cost estimation for neighbors and networks are regularly pre-computed, refreshed, and kept in a local table in each proxy.

Once all these estimates are available, if p_i decides that it is not the best proxy to serve the request, then p_i sends the candidates and their costs (as it knows) to the requesting proxy along with a redirection message. With this additional information, the requesting proxy chooses the best proxy in the list and forwards its join request that proxy (Fig-

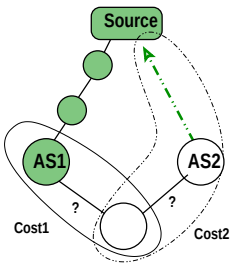


Figure 1.10. Cost estimation when selecting one of the alternative parents: one candidate parent is already delivering the stream whereas the other one has to account for the estimated distance to the source

ure 1.9). Note that the redirection protocol does not allow backtracking; i.e., if proxy p_i chooses to redirect an incoming join request (say from proxy p_l) to proxy p_h based on its estimates, p_i is dropped from further considerations by p_l even though a join request accepted by p_h may eventually lead to a more costly path. Therefore, proper estimations of the costs of the future steps is the path creation is important.

Route Establishment. If p_i decides to serve the request and if the request is for a stream that is not already served by this proxy, then p_i has to find a way to join a multicast tree that serves the stream. For this purpose, it has to evaluate its neighbors and select the best one to send a join request. As shown in Figure 1.10, p_i considers all its neighbors when choosing the next proxy. For those neighbors (say p_l) that are already on the required multicast tree, p_i estimates the cost of connection simply as

$$\begin{aligned} cost_l = & ptension_l(new\_pl_l) - ptension_l(old\_pl_l) + \\ & btension_{i,l}(new\_bl_{i,l}) - btension_{i,l}(old\_bl_{i,l}) \end{aligned}$$

That is, it accounts for the increased tension at the new parent p_l as well as the increased tension on the network connection between itself and p_l. If p_l is not serving the stream, on the other hand, it also accounts for the fact that p_l will need to establish a route to the source:

$$\begin{aligned} cost_l = & ptension_l(new\_pl_l) - ptension_l(old\_pl_l) + \\ & btension_{i,l}(new\_bl_{i,l}) - btension_{i,l}(old\_bl_{i,l}) + \\ & minbcost_{l,j}. \end{aligned}$$

After all the estimates are computed for the neighbors, p_i chooses the best neighboring proxy and forwards the request to that proxy. In order to prevent redirection loops, p_i keeps track of the set of proxies it has already redirected to (Section 1.0.0). Once p_l receives the join request, it can either deny, redirect, or admit the request as discussed above.

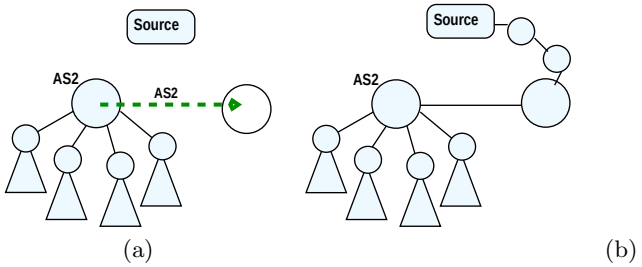


Figure 1.11. Loop avoidance in join requests

Like the redirection protocol, the join protocol does not allow backtracking; i.e., if proxy p_i chooses to send a join request to proxy p_l based on its estimates, unless p_l denies this request or explicitly redirects it to another proxy, all other neighbors of p_i are dropped from further considerations. Therefore, again, proper estimations of the costs of the future steps is the path creation is important.

Denies and Redirections. A proxy p_i that has sent a join request can receive an *admit*, *deny*, or *redirection* message. Unless the request is admitted, the proxy has to find an alternative proxy to join to. As described above, redirection messages carry information about potential candidate proxies and their costs. p_i merges this information with the information it already has about its neighborhood and sends the request to the best candidate it now has. Note that unless a limit is established, a request may be redirected too many times. Therefore, mSON uses a limit, $redir_{limit}$, of the number of times each request can be redirected. Once the redirection limit is reached, p_i fails to join to the stream; if there is a downstream proxy waiting for the establishment of the route, p_i sends a deny message to it. The downstream proxy then initiates its own deny-message handling routines.

Loop Detection and Elimination. A join request may originate from an edge proxy or from a proxy that is serving other proxies during the multicast tree establishment phase (Figure 1.11). In the latter case, it is essential to employ appropriate mechanisms to eliminate routing loops. As discussed earlier, there are various algorithms and protocols (such as DUAL [Garcia-Lunes-Aceves, 1993], EIGRP [Albrightson et al., 1994], RIP-MTI [Schmid and Steigner, 2002], and LPA [Murthy and J.J.Garcia-Luna-Aceves, 1995]) that aim at eliminating loops or minimizing the undesirable effects of routing loops.

In mSON, we opt to annotate each join request with the name of the AS where the request originates (Figure 1.11): If the receiving proxy

has already a path to the source of the stream, it only has to verify that the list of its parents does not contain the AS where the join request originated. If there is not an already established path to the source, then the receiving proxy has to make sure that the path that it will create to the source does not contain the AS that it received request from. To achieve this, each join request is *also* annotated with the list of downstream proxies *waiting* for the establishment of the path to the route. Consequently, a proxy receiving a request can easily check whether admitting the request will cause a loop.

Stop Requests

When a proxy p_i receives a stop request from a user or another proxy for access to the media stream multicasted at source s_j, it simply drops the requesting proxy from the multicast tree and makes the corresponding resources available for future requests. In case there are no more child proxies consuming a stream served by p_i, then p_i also sends a stop request to the corresponding parent for that stream so that it can release all corresponding resources for other streams' use.

Redistribution

One of the scalability mechanisms used by the proxies in mSON is *redistribution*. When p_i detects any change in its neighborhood (as a result of the update messages it exchanges with its neighbors), such as nearby proxy becoming available or a configuration change, it checks if it is *overloaded* or *underloaded* relative to its neighbors. *Overload* is defined as having at least one server (p_h) in the neighborhood such that

$$t\_reduction\_at\_i > (1 + \gamma_d) t\_increase\_at\_h$$

as a result of the redistribution of one of the streams p_i is currently serving (Section 1.0.0). Here, γ_d is a threshold value to prevent very small gains to cause potentially expensive redistributions. Note that if proxy p_h is underutilized, redistribution can also decrease the tension at p_h benefiting both of the involved proxies. If p_i notices that this trigger condition is true for at least one of the streams, then it initiates a redistribution process aimed at re-balancing the overall tension.

During redistribution, p_i tries to find proxies in its neighborhood that can take a fraction (we used 50% in our implementation) of its load:

1 It first chooses the stream, s_j, whose redistribution will bring the highest benefit to the system.

2 Then, for this stream, it chooses the best proxy, p_l, to take the fraction of the load it aims to redistribute. The process p_i uses

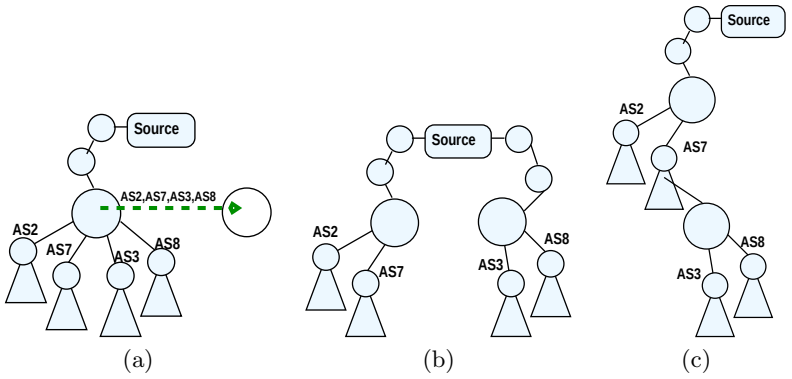


Figure 1.12. Loop avoidance in redistribution: (a) the requesting AS sends the list of the immediate children proxies to the chosen candidate; (b) if the candidate can establish a path that doesn't go through any children than the proxy can redirect any set of children; (c) if the path created by the proxy passes through a child, then a set that does not include this child is shipped to the candidate

to choose p_l is similar to the process it uses to choose the best proxy to redirect a request, except that during the redistribution more than one connection is redirected simultaneously. Hence, the loads that are used for calculating tension changes are based on the fraction of the load being shipped.

3 After it chooses the stream s_j and the proxy p_l to redistribute the load to, p_i contacts p_l and checks if it is indeed willing to take the required load. At this point p_l can either admit this load shipment request, deny it, or redirect it. Before admitting the request p_l makes sure that there is a loop free path to the source (Figure 1.12). As in the case of individual join request, if p_l chooses to redirect the shipment request, redirection messages are accompanied with the neighborhood list to help p_i choose the best one in its own and p_l's neighborhood based on its redirection past.

4 Once a proxy p_l accepts the shipment, p_i starts shipping the load. During this process it locks all resources (hence can not admit new requests). p_i chooses a fraction of its children consuming the streams s_j and redirects each one of them, individually, to p_l. p_l handles these join requests as a group and admits all of them.

5 Once the processing for s_j is over, p_i chooses another stream and continues this process until it ships the required load.

If a shipment to a proxy fails (for instance due to link failures), a timeout mechanism is used to prevent more time being wasted trying to connect to it in the future.

Consolidation

When p_i detects any change in its neighborhood (as a result of the update messages it exchanges with its neighbors), it checks if it becomes underloaded as a result of the change. If so, it tries *consolidating* its service with its neighbors. Consolidation is triggered when there is at least one server (p_h) in the neighborhood such that, if p_i ships one of its streams to p_h completely, the following would be true:

$$t\_reduction\_at\_i > (1 + \gamma_c) t\_increase\_at\_h$$

Consequently, the consolidation process is very similar to the redistribution process, except that the amount of load negotiated between the proxies and shipped from one proxy to the other is 100% of the load for each stream instead of being 50%. If the consolidation process is completed successfully for all streams, proxy p_i returns to zero utilization and can be taken out of the system.

Note that, despite their apparent similarities, there is a fundamental difference between consolidations and redistributions. In general, it is easier to sense when the system needs consolidations than it needs redistributions. This is because, consolidations aims solving a current challenge (resources are being underutilized), whereas redistributions also aim making enough resources available for future arrivals. Therefore, especially at times of rapid growth, rapid sensing of the need for redistributions is essential. To achieve this, join and redistribution requests can be augmented with expected load arrival rates based on short term statistics.

Parent Failure

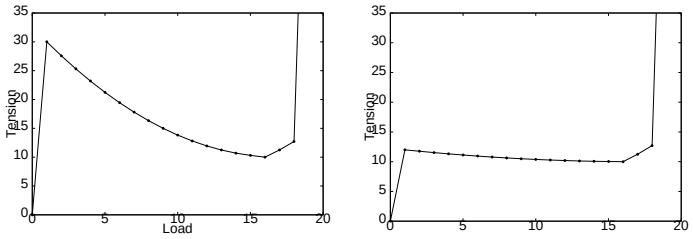
When p_i detects one of its parents (p_i can have a different parent for each stream it is serving), it has to fix the broken connection. The process of fixing the broken connection is similar to redirection, in the sense that, p_i redirects itself to another proxy in its neighborhood based on the network and proxy tensions. A timeout mechanism is used to prevent the failed parent from being used in the considerations in the future. Also, during this process p_i locks all resources (hence can not admit new requests) for the failed streams.

2. Experimental Evaluation

In this section, we present experimental results of evaluating the effectiveness and usefulness of the mSON architecture and protocols for streaming media delivery. Each experiment has been repeated under the following five configurations and the results were compared:

Parameter	*Value*
Number of ASs(proxies)	100
Ratio of ASs(proxies) at L1 / L2 / L3	0.1/0.3/0.6
Intralayer density at (L1-L1) / (L2-L2)/ (L3-L3)	0.4/0.15/0.05
Interlayer density at (L1-L2) / (L2-L3)/ (L1-L3)	0.3/0.1/0.1
Number of proxies hit per source	80
Proxy hit ratio at L1 / L2/ L3	0.1/0.3/0.6
Number of sources	10
Source distribution at L1 / L2 / L3	0.1/0.3/0.6
Redirection limit	3
γ_d/γ_c	5%/10%

Figure 1.13. The system parameters for the first set of experiments



(a)Proxy tension graph (b)Network tension graph

Figure 1.14. (a) Proxy and (b) the network policy graphs used in the experiments

Exp.	# Redirect.	Consol.	Redist.
0nCnR	0	Off	Off
3nCnR	3	Off	Off
3nCyR	3	Off	On
3yCnR	3	On	Off
3yCyR	3	On	On

All experiments presented here were conducted using software simulation in a network-emulated environment. In order to evaluate mSON and to observe the effects of various system parameters, we needed a large number of setups. Furthermore, we needed to change the parameters in a controlled fashion. Therefore, we systematically generated a large number of scenarios with various parameters and use these in our experimental evaluation. Next, we describe these scenarios.

In the experiments, we used two network models. The first network model is a three-layered network as depicted in Figures 1.13(a) and 1.13(b). In this network, the inner layer is densest and the outer layer is sparsest. This corresponds to the fact that a smaller number of backbone networks carry a large number of smaller networks. The number of nodes in each layer and the number of connections between and across each layer are controlled through a set of parameters (Fig-

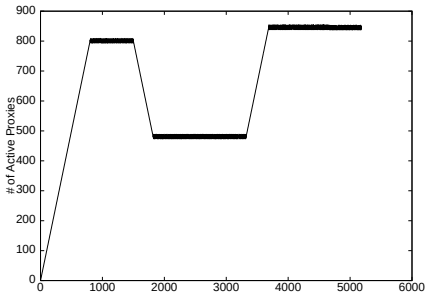


Figure 1.15. In the experiments, number of active sources in the system goes through extreme changes (50% additions/deletions)

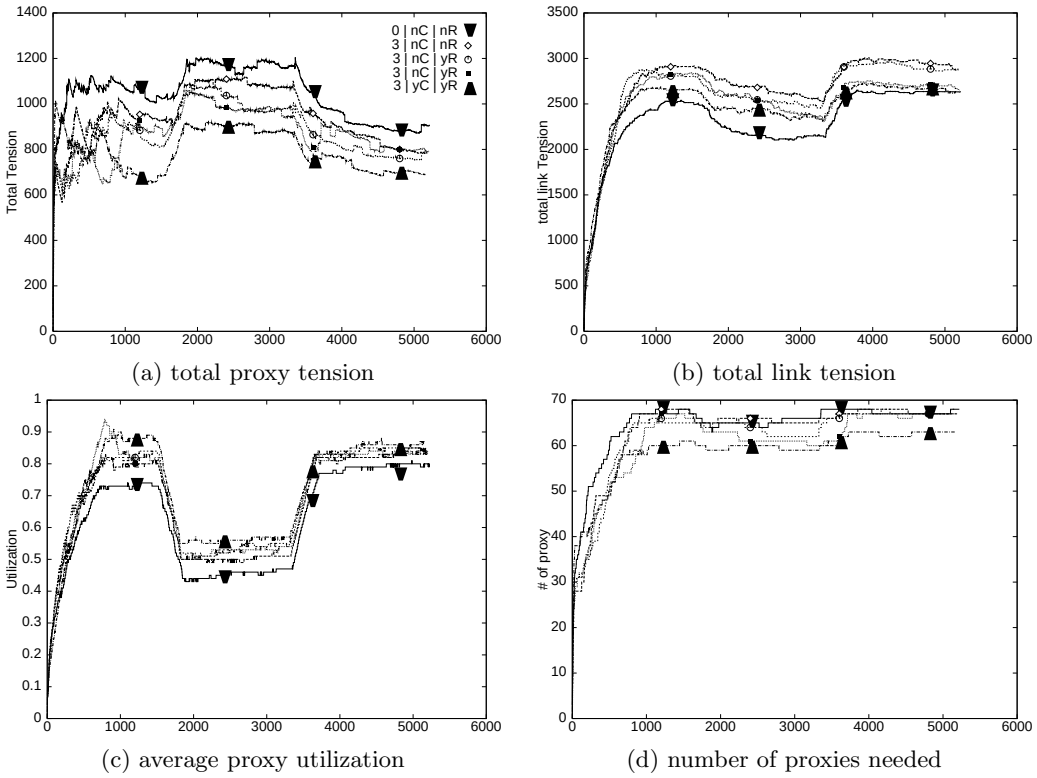
ure 1.13(c)). Results are presented in Section 2.0. In addition to this network, we have also experimented with a network having transit-stub model widely used by other researchers [Banerjee et al., 2003, Castro et al., 2003b] to model Internet connectivity. We used GT-ITM topology generator [Zegura et al., 1996] to create transit-stub networks. As described in Section 2.0, the results on this alternative network model are similar. Figure 1.14 shows the proxy and network policy graphs used in our experiments. These graphs prefer almost full utilization of the proxy resources, whereas once a network connection is active, there is not much penalty unless it is highly overloaded.

We experimented with extreme variations in the source/client arrival patterns. In the setting presented in Figure 1.15, we start with 10 active sources. 50% of the sources (and their clients) dropped off at some point and later the same amount of (different) sources joined to the system.

Results on the Multi-layered Network Topology

In the first set of experiments, we aimed at observing the impact of proposed techniques on the overall performance of mSON.

As shown in Figure 1.16 (a) and (b), a combination of redirections, consolidations, and redistributions consistently achieve good proxy and link tensions. Note that, in Figure 1.16 (b), the lowest tension seem to be achieved when all adaptation mechanisms were turned off; but this is because a significant portion of the users were being declined by the system, which is highly undesirable. Figure 1.16 (c) shows that alternative $3yCyR$, where all adaptation mechanisms are on indeed achieves the best proxy utilization (where 1.0 denotes the preferred utilization rate). This increased utilization shows itself in the reduction in the number of proxies needed to serve the multicast trees (Figure 1.16(d)). Similarly, the number of network links needed is the lowest when all



(a) total proxy tension

(b) total link tension

(c) average proxy utilization

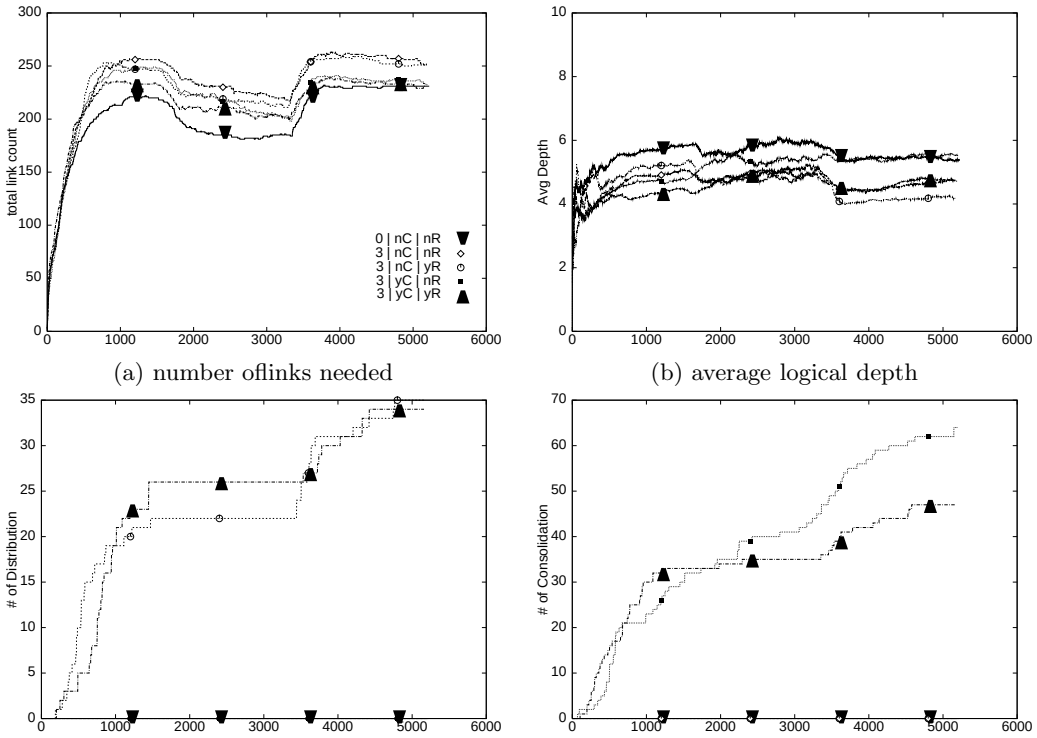
(d) number of proxies needed

Figure 1.16. Results: (a) total proxy tension, (b) total link tension, (c) average proxy utilization, (d) number of proxies needed, and (e) number of links needed **(*no-adaptation, 0nCnR*, and *all-adaptations, 3yCyR,* cases have been highlighted more dominantly than the others)**

adaptation mechanisms are applied together (Figure 1.17(a)), except for option $0nCnR$ (without any adaptation) which results in too many denials to be a feasible option (we obviously do not want to cause client denials to reduce the number of network links used).

Figure 1.17(b) shows that consolidations and distributions also reduce the average depth (hence, the delay and possible QoS degradations) of the multicast trees. The figure also shows that, although consolidations enable a smaller depth, redistributions are essential to make sure that mSON is able to scale the depth of the tree in the case of sudden injections of a large number of sources.

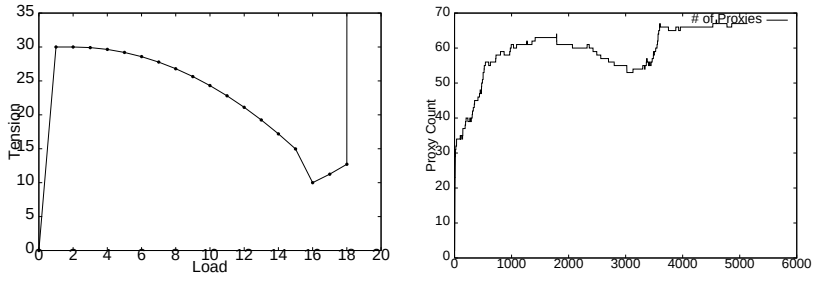
Note that, in these experiments, each logical link between proxies was on the average 2 to 2.5 physical links; in general consolidations caused increases in this number as they led to sparser multicast trees with less number of proxies. However, overall, the total number of links



(a) number oflinks needed (b) average logical depth

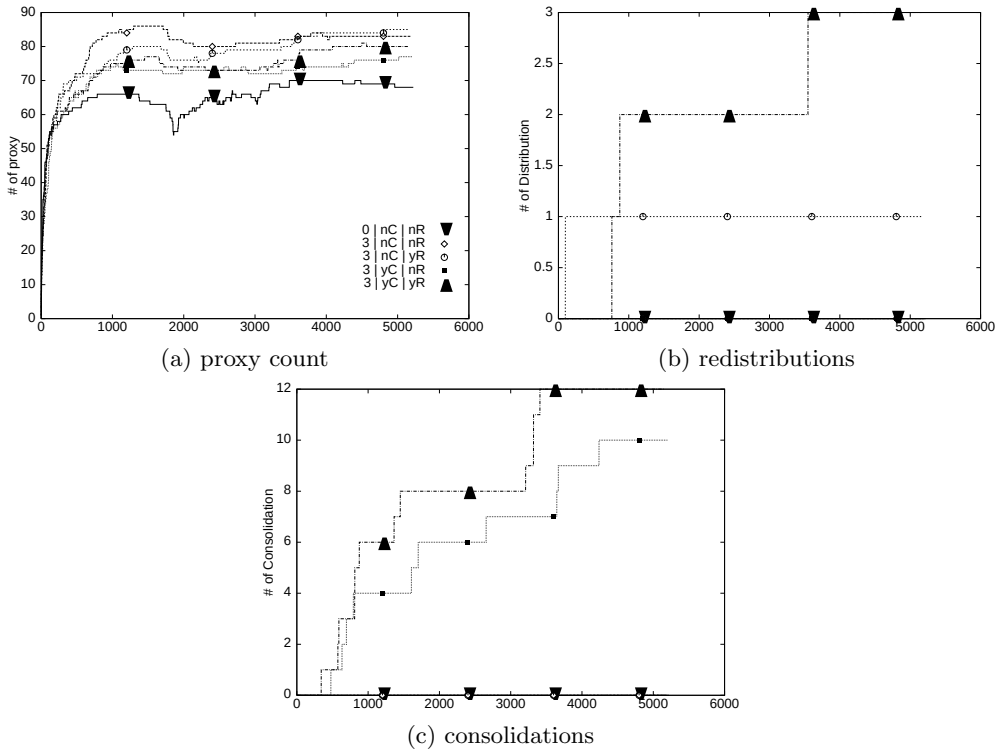
(c) number ofredistributions (d) number of consolidations

Figure 1.17. Results: (a) number of links needed, (b) average logical depth of the leaves, (c) number of redistributions, and (d) number of consolidations **(*no-adaptation, 0nCnR*, and *all-adaptations, 3yCyR*, cases have been highlighted more dominantly than the others)**



(a) (b)

Figure 1.18. An experiment with (a) an alternative policy graph and (b) the corresponding proxy need



(a) proxy count (b) redistributions

(c) consolidations

Figure 1.19. Result on GT-ITM topology: (a) proxy count, (b) redistributions, and (c) consolidations

used by the system was minimized by consolidations and redirections as shown earlier in Figure 1.17(a). Finally, Figures 1.17(c) and (d) show how mSON use redistributions and consolidations to tackle the sudden changes in the structure. Although not shown in this figure, redirections are also heavily used, especially by alternative $3yCyR$ which uses all adaptations.

We have also experimented with other policy graphs to verify that policy graph shapes have the expected impacts on the behavior of the system. Figure 1.18 shows an alternative policy graph and the resulting proxy utilization for the same experimental setup (with all adaptations allowed). This graph favors utilizations closer to the optimal. Consequently, when the number of sources drop, as expected, the number of proxies used by mSON drops more significantly. Furthermore, since this policy graph favors consolidations over redistributions, as expected it does not scale as quickly large fluctuations of sources.

Results on the GT-ITM Topology

We also studied the performance of the mSON architecture using a 100 node Transit-Stub network topology generated with the GT-ITM topology generator [Zegura et al., 1996]. The resulting network had relatively longer shortest-paths compared to three-layered model.

Figure 1.19 shows the results obtained using this model. As shown in Figure 1.19(a) the proxy count needed to support the system was lowest in alternatives $3yCnR$ and $3yCyR$ where consolidations were on (again excluding the case without adaptations, $0nCnR$, where there were significantly high number of service requests that were denied). This observation is also supported by Figures 1.19(b) and (c), which show that since in this case the system performs significantly less number of redistributions, a redistribution-only mechanism would not be enough to support adaptation to extremely high fluctuations (especially quick drops) of sources. Thus a redistribution+consolidation based scheme is the most desirable approach to adaptations to drastic network changes.

3. Related Work

Recently there has been considerable interest in using proxies and application level overlays for streaming data delivery [Shi and Turner, 2002, Banerjee et al., 2003, Banerjee et al., 2002, Castro et al., 2003b, Castro et al., 2002, Acharya and Smith, 2000, Rajaie et al., 2000, Jin et al., 2002, Wang et al., 1998, Sen et al., 1999, Miao and Ortega, 1999]. Su and Yemini [Su and Yemini, 2001] 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) [Passmore and Freeman, 1997] and virtual private networks (VPNs) [Scott et al., 1998].

The X-Bone [Touch, 2001] is a system for dynamic deployment and management of Internet overlay networks. Current overlay networks include commercial VPNs [Scott et al., 1998] and IP tunnelled networks M-Bone [Eriksson, 1994]. 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, [Lim et al., 2001] 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 program-

mable interface [Takahashi et al., 1999]. The RON project [Andersen et al., 2001] 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.

Xu *et al.* introduced peer-to-peer streaming in [Xu et al., 2002]. Chawathe, McCanne and Brewer [Chawathe et al., 2002] 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 [Chu et al., 2000] have proposed end system multicast to support small-sized applications such as audio or video conferencing. This approach is not as scalable since it requires that every member maintain a complete list of every other member in the group.

Shi and Turner [Shi and Turner, 2002] have proposed a centralized greedy heuristic, called the Compact Tree algorithm to minimize the maximum latency from the source to a Multicast Service Node (equivalently, a proxy). Similarly, Banerjee et al. [Banerjee et al., 2003] have proposed a multicast overlay tree construction algorithm to minimize average-latency from the source to leaf level multimedia service nodes or proxies. Unlike mSON, these efforts are focused on the objective of optimization from the perspective of individual end-users. In contrast, our goal is to enable efficient distribution of streaming media by minimizing network and proxy resources needed.

SpreadIt [Deshpande et al., 2001] is an application level multicast architecture for delivering streaming live media over a network of clients, using the resources of the clients themselves. ZIGZAG [Tran et al., 2003] also takes a similar approach and create media stream hierarchies over clients themselves. Other cooperative peer-to-peer structures based on clients include, CoopNet [Padmanabhan et al., 2002]. In particular, the Bayeux 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. It relies on probabilistic approaches to minimize bandwidth usage. Kim, Lam, and Lee [M.S.Kim et al., 2003] presented a distributed algorithm that builds a multicast tree in which the average receiving rate, computed over all receivers in the tree, is maximized. Recently, Yang [Yang, 2003] considered a more elaborate model for streaming media approach where multiple delivery path(s) are established be-

tween source and destination(s) to improve service. SplitStream [Castro et al., 2003a] splits data is divided into disjoint sections, stripes, and delivers each stripe through a separate tree. In Chainsaw [Pai et al., 2005], instead of creating trees, the peer-to-peer system operates by an explicit notification-request mechanism between the peers. Nemo [Birrer and Bustamante, 2005] aims to add resilience to peer-to-peer multicasting systems, where peers are autonomous, unpredictable. FatNemo [Birrer et al., 2004], is one of the few overlay multicasting systems which recognizes the challenges associated with multi-source multicasting. It creates with larger clusters closer to the root, to eliminate bottlenecks closer to the root. Unlike mSON architecture presented here, oStream [Cui et al., 2004, Cui and Nahrstedt, 2003] focuses on the problem of asynchrony prevalent in systems which serve stored on-demand media.

In addition to these research activities, many CDN vendors provide various solutions for streaming media delivery on the Internet. In the solutions provided by Akamai[Akamai Technology, 2005], 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[RealNetworks, 2005] is software installed on a network or ISP gateway that aggregates and handles client requests for media streamed. In contrast mSON's dynamic adjustment of delivery network's structure to user population and distribution as well as network conditions, RealProxy requires manual specification and adjustment of multiplexing structure. [C-StarOne, 2005, Nguyen and Zakhor, 2002] propose streaming video from multiple sources to a single receiver to exploit path diversity and to reduce packet loss.

4. Conclusion

In this chapter, we presented a policy-driven peer-to-peer *media streaming overlay network* (mSON) for streaming live media from media sources to end-users. mSON is specifically designed to efficiently support multiple media streams with different sources and user populations at the same time. mSON minimizes its footprint on the overlay network to use its resources most effectively. For this purpose, mSON represents resource usage policies at the proxies (ASs - or Multicast Service Nodes, MSNs) as well as at the network peering points between them. The redirection, redistribution, and consolidation based negotiations between proxies prevent resources from being underutilized or overutilized, while maintaining small delays. This way, while maintaining a high resource utilization, they also prevent congestions, service rejections, and jitters

in the streams users receive. We reported experimental results that show that mSON achieves these tasks effectively under a variety of conditions.

References

Acharya, S. and Smith, B. (2000). Middleman: A video caching proxy server. In *NOSSDAV*.

Akamai Technology (2005). *Information available at* `http://www.akamai.com/`.

Albrightson, R., Garcia-Luna-Aceves, J.J., and Boyle, J. (1994). Eigrp-a fast routing protocol based on distance vectors. In *Networld/Interop*.

Amir, Yair, Awerbuch, Baruch, Danilov, Claudiu, and Stanton, Jonathan (2002). Global flow control for wide area overlay networks: a cost-benefit approach. In *OPENARCH*, pages 155–166.

Andersen, David G., Balakrishnan, Hari, Kaashoek, M. Frans, and Morris, Robert (2001). Resilient overlay networks. In *Proc. SOSP 2001, Banff, Canada*.

Banerjee, S., Kommareddy, C., Kar, K., Bhattacharjee, B., and Khuller, S. (2003). Construction of an efficient overlay multicast infrastructure for real-time applications. In *IEEE INFOCOM*, pages 1521–1531.

Banerjee, Suman, Bhattacharjee, Bobby, and Kommareddy, Christopher (2002). Scalable application layer multicast. *SIGCOMM Comput. Commun. Rev.*, 32(4):205–217.

Birrer, S. and Bustamante, F.E. (2005). Nemo- resilient peer-to-peer multicast without the cost. In *MMCN*.

Birrer, S., Lu, D., Bustamante, F.E., Y.Qiao, and Dinda, P. (2004). Fatnemo: building a resilient multi-source multicast fat-tree. In *Ninth International Workshop on Web Content Caching and Distribution*.

Blum, Avrim, Chalasani, Prasad, Coppersmith, Don, Pulleyblank, Bill, Raghavan, Prabhakar, and Sudan, Madhu (1994). The minimum latency problem. In *STOC '94: Proceedings of the twenty-sixth annual ACM symposium on Theory of computing*, pages 163–171.

C-StarOne (2005). *Information available at* `http://www.centerspan.com/`.

Castro, M., Druschel, P., Kermarrec, A., and Rowstron, A. (2002). SCRIBE: A large-scale and decentralized application-level multicast infrastructure. *IEEE Journal on Selected Areas in communications (JSAC)*, 20(8).

Castro, M., Druschel, P., Kermarrec, A.-M., Nandi, A., Rowstron, A.I.T., and Singh, A. (2003a). Splitstream: high-bandwidth multicast in cooperative environments. In *SOSP*, pages 298–313.

Castro, M., Jones, M.B., Kermarrec, A.-M., Rowstron, A., Theimer, M., Wang, H., and Wolman, A. (2003b). An evaluation of scalable

application-level multicast built using peer-to-peer overlay networks. In *IEEE INFOCOM*.

Chawathe, Y., McCane, S., and Brewer, E. (2002). An architecture for Internet content distribution as an infrastructure service. http://yatin.chawathe.com/ yatin/papers/scattercast.ps.

Chu, Y.-H., Rao, S.G., and Zhang, H. (2000). A case for end system multicast. In *Measurement and modeling of computer systems*, pages 1–12.

Cui, Y., Li, B., and Nahrstedt, K. (2004). ostream: asynchronous streaming multicast in application-layer overlay networks. *IEEE Journal on Selected Areas in Communications*, 22(1).

Cui, Y. and Nahrstedt, K. (2003). Layered peer-to-peer streaming. In *NOSSDAV '03: Proceedings of the 13th international workshop on Network and operating systems support for digital audio and video*, pages 162–171.

Cui, Y., Xue, Y., and Nahrstedt, K. (2003). Optimal resource allocation in overlay multicast. In *ICNP*, pages 71–83.

Deshpande, H., Bawa, M., and Garcia-Molina, H. (2001). Streaming live media over a peer-to-peer network. Technical Report 2001-30, Stanford University.

Eriksson, H. (1994). MBone: the multicast backbone. *Communications of the ACM*, pages 54–60.

Garcia-Lunes-Aceves, J. J. (1993). Loop-free routing using diffusing computations. *IEEE/ACM Trans. Netw.*, 1(1):130–141.

Jin, S., Bestavros, A., and Iyengar, A. (2002). Accelerating Internet streaming media delivery using network-aware partial caching. In *International Conference on Distributed Computing Systems*.

Kar, K., Sarkar, S., and Tassiulas, L. (2001). Optimization based rate control for multirate multicast sessions. In *INFOCOM*, pages 123–132.

Kelly, F., Maulloo, A., and Tan, D. (1998). Rate control in communication networks: shadow prices, proportional fairness and stability. *Journal of the Operational Research Society*, 49(3):237–252.

Lim, L. K., Gao, J., Ng, T. S. E., Chandra, P. R., Steenkiste, P., and Zhang, H. (2001). Customizable virtual private network service with QoS. *Computer Networks*, pages 137–151.

Miao, Z. and Ortega, A. (1999). Proxy caching for efficient video services over the Internet. In *PVW*.

M.S.Kim, Lam, S.S., and Lee, D.-Y. (2003). Optimal distribution tree for Internet streaming media. In *ICDCS*, pages 116–125.

Murthy, S. and J.J.Garcia-Luna-Aceves (1995). Dynamics of aloop-free path-finding algorithm. In *IEEE Globecom*, pages 1347–1351.

Nguyen, T. and Zakhor, A. (2002). Distributed video streaming over the Internet. In *Multimedia Computing and Networking (MMCN)*.

Padmanabhan, V.N., Wang, H.J., Chou, P.A., and Sripanidkulchai, K. (2002). Distributing streaming media content using cooperative networking. In *NOSSDAV*, pages 177–186.

Pai, V., Kumar, K., Tamilmani, K., Sambamurthy, V., and Mohr, A. E. (2005). Chainsaw: eliminating trees from overlay multicast. In *4th International Workshop on Peer-to-Peer Systems*.

Passmore, D. and Freeman, J. (1997). The virtual LAN technology. Technical Report 200374-001, 3COM.

Rajaie, R., Yu, H., Handley, M., and Estrin, D. (2000). Multimedia proxy caching mechanism for quality adaptive streaming applications on the Internet. In *INFOCOM*.

RealNetworks (2005). *Information available at* `http://www.real.com/`.

Schmid, Andreas and Steigner, Christoph (2002). Avoiding counting to infinity in distance vector routing. *Telecommunication Systems*, 19(3-4):497–514.

Scott, C., Wolfe, P., and Erwin, M. (1998). *Virtual private networks*. OReilly, Sebastopol.

Sen, S., Rexford, J., and Towsley, D. (1999). Proxy prefix caching for multimedia servers. In *INFOCOM*.

Shi, S. and Turner, J. (2002). Routing in overlay multicast networks. In *IEEE INFOCOM*, pages 1200–1208.

Su, G. and Yemini, Y. (2001). Virtual Active Networks: Towards multi-edged network computing. *Computer Networks*, pages 153–168.

Takahashi, E., Steenkiste, P., Gao, J., and fischer, A. (1999). A programming interface for network resource management. In *Proceedings of the 1999 IEEE Open Architectures and Network Programming*, pages 34–44.

Touch, J. (2001). Dynamic Internet overlay deployment and management using the X-Bone. *Computer Networks*, pages 117–135.

Tran, D.A., Hua, K.A., and Do, T.T. (2003). Zigzag: an efficient peer-to-peer scheme for media streaming. In *INFOCOM*.

Wang, Y., Zhang, Z. L., Du, D. H., and Su, D. (1998). A network conscious approach to end-to-end delivery over wide-area networks using proxy servers. In *INFOCOM*.

Xu, D., Hefeeda, M., Hambrusch, S.E., and Bhargava, B.K. (2002). On peer-to-peer media streaming. In *ICDCS*, pages 363–371.

Yang, J. (2003). Deliver multimedia streams with flexible qos via a multicast DAG. In *ICDCS*, pages 126–137.

Zegura, Ellen W., Calvert, Kenneth L., and Bhattacharjee, Samrat (1996). How to model an internetwork. In *INFOCOM*, pages 594–602.