Finding Service Paths in a Media Service Proxy Network

Dongyan Xu[a] and Klara Nahrstedt[b]

[a]Dept. of Computer Sciences, Purdue University, West Lafayette, IN, USA
[b]Dept. of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL, USA

ABSTRACT

With the deployment of multimedia service proxies at different locations in the networks, it is possible to create an *application-level media service proxy network*. Examples of media services offered by this network include media data adaptation, transformation, protection, enhancement, recovery, and different combinations of them. Multimedia sources and clients will then connect to this network and create customized, value-added, and composite media service delivered by one or more proxies in the media service proxy network.

In this paper, we focus on the problem of finding multimedia *service path* in the media service proxy network. A service path is a chain of media service proxies between a media source and a media client. With dynamic changes in proxy capacity and connection bandwidth between the proxies, our goal is to find the 'best' path with respect to end-to-end resource availability for each service path request. Our solution includes (1) a mechanism to monitor and propagate resource availability information in the media service proxy network and (2) an algorithm to find the best service path based on the resource monitoring results. Its main features include: (1) the resource monitoring mechanism provides reasonable accuracy and stability, while incurring controlled overhead; (2) the service path finding algorithm finds the best path for each service path request, and achieves high overall success rate among all requests; and (3) it is an application-level solution, and does not require changes to the lower-level network infrastructure.

Keywords: overlay networks, multimedia service, proxy, application-level routing

1. INTRODUCTION

To enable customized delivery or wide distribution of multimedia data, media service providers and media content carriers deploy media service proxies in local-area, metropolitan-area, or wide-area networks. A media Service ProxY (or *SPY* for the rest of the paper) accepts a media stream, performs certain application-level processing on the media data, and forwards the stream. For example, a SPY may be capable of: transcoding media data from one format to another, repairing or enhancing the poor image quality in a media stream, adding background music to a mute video, summarizing the content of a video stream, tracking the image of an object in the media stream, performing error correction coding/decoding, and performing real-time water-marking in a media stream for copyright protection. The media data processing often requires a non-trivial portion of a SPY's total capacity. In addition, the SPY may *increase* or *decrease* the data rate of a media stream after the media processing.

With SPYs deployed at different locations in the networks, an *application-level* media Service ProxY Network or *SPY-Net*, is formed. Media sources and clients can then connect to the SPY-Net to get customized, value-added, and composite media services (Figure 1). Furthermore, besides media data processing, SPYs also serve as relays: simply accepting a media stream and forwarding it. With this, the SPY-Net may also help media streams getting round some poor-quality network links - even if the lower-level network cannot do so. Although a SPY-Net has many potential capabilities such as individual service delivery between single source and single client, as well as large-scale media content distribution to multiple clients,[1, 2] we focus on media service delivery to an individual client in this paper. We also assume a relatively *small size* of a SPY-Net (less than a hundred SPYs).

Further author information: (Send correspondence to D. Xu)
D. Xu: E-mail: xu@cs.purdue.edu, Telephone: 1 765 494 6182

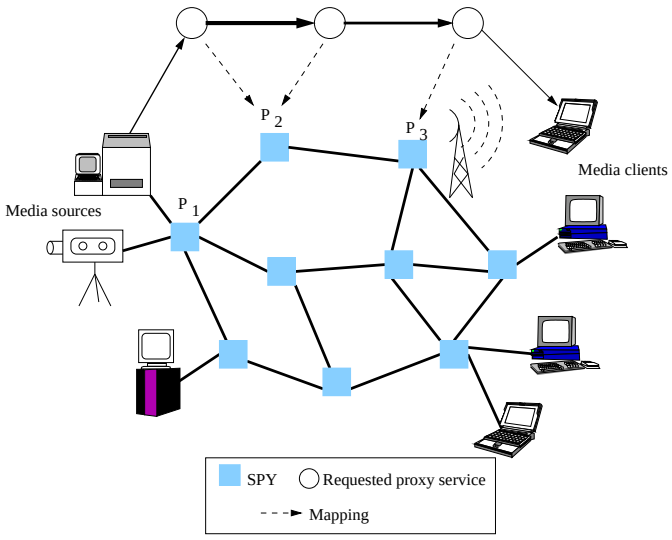


Figure 1: SPY-Net and SPY mapping for service paths

More specifically, we focus on the case of finding *service paths* in a SPY-Net. A service path consists of a chain of SPYs between a media source and a media client*. The SPYs perform *multiple* proxy services - in a certain order, on the media stream from the source to the destination. If working with existing media capturing, querying, or distribution systems (such as the e-Seminar System[4]), service paths will make it possible to create more composite and value-added media services. Some examples of service paths are: (1) A video stream is first water-marked by a copyright protection proxy service, then it is summarized by a video summarization service, and finally, an object tracking service tracks the image of an object of interest (by generating a rectangle around that image, for example). (2) A low-quality video stream from an outdoor mobile environment (for example, during a field trip) is first enhanced by an image repair and enhancement service, and then mixed with background music by a music adaptation service. (3) A media stream is compressed by a data compression service before going on a low bandwidth network link such as a trans-pacific link; on the other side, the corresponding decompression service decompresses the media stream before forwarding it to the destination.

Challenges in finding service paths in a SPY-Net come from the dynamic changes in resource availability, including both SPY capacity and connection bandwidth between SPYs. Specifically, the following problems have to be studied: (1) how to obtain and propagate resource availability information without incurring excessive overhead in the SPY-Net; (2) for a service path request, since the proxy services may change the data rate of the media stream, the service path may require *different* amounts of bandwidth in different segments of the path; (3) on the service path, how to map the requested proxy services to the actual SPYs: a SPY may perform multiple proxy services (for example, P_2 in Figure 1), if it has the required capacity and functionality; on the other hand, a SPY may just serve as a relay without performing any proxy service (for example, P_1 in Figure 1).

Existing solutions to network-level unicast QoS routing may not be applicable to the above challenges. More specifically, they do not solve problems (2) and (3) which arise only in service path computation. First, network-level unicast QoS routing computes an end-to-end path with *uniform* bandwidth on each link; while a service path may require different bandwidth in different segments. Second, network-level QoS routing does not involve end hosts; while a service path involves one or more SPYs. Therefore, network-level QoS routing does not perform the dual task of SPY mapping and path finding, which is more complicated than finding an end-to-end network path of uniform bandwidth.

*The generic concept of service path was first introduced in the Ninja Project.[3]

In this paper, we propose our solution to finding the 'best' service path with respect to end-to-end resource availability for each service path request. Our solution includes (1) a mechanism to monitor and propagate the resource availability information in the SPY-Net and (2) an algorithm to find the best service path based on the resource monitoring results. Its main features include: (1) the resource monitoring mechanism provides reasonable accuracy and stability, while incurring controlled overhead; (2) the service path finding algorithm finds the best path for each service path request, with respect to a generic resource evaluation function; (3) the service path finding algorithm achieves high overall success rate among all service path requests; and (4) it is an application-level solution, requiring no changes to the lower-level network infrastructure.

The rest of the paper is organized as follows. Section 2 provides a brief overview of our solution. Section 3 describes the SPY-Net resource availability monitoring mechanism. Section 4 presents the algorithm for finding service paths. Section 5 presents performance evaluation. Section 6 compares our work with related work. Finally, Section 7 concludes this paper.

2. SOLUTION OVERVIEW

Our solution can be summarized as follows: each SPY in the SPY-Net monitors (1) its own capacity and (2) the connection bandwidth from itself to a selected *subset* of other SPYs. The monitoring results of proxy capacity and connection bandwidth are periodically propagated to every other SPY in the SPY-Net. Therefore, each SPY is able to maintain a global view of the SPY-Net: a SPY-Net Monitoring Graph or *SNMG*. Based on the SNMG, a SPY executes the service path finding algorithm *locally*, for each service path request submitted to it. Our solution avoids the distributed service path probing upon each request, at the cost of periodic propagation of resource monitoring results. In addition, our simulation results (Section 5) show that the service path finding algorithm achieves good performance even with long propagation period.

Note that our solution takes advantage of the facts that (1) the size of a SPY-Net is relatively small and (2) all operations are performed at application-level instead of in the lower-level network infrastructure, where the implementation of these operations could have introduced unacceptable overhead.

3. MONITORING SPY-NET RESOURCES

To keep track of resource availability in the SPY-Net, each SPY maintains a *SPY-Net Monitoring Graph (SNMG)*. A node P in the SNMG represents a SPY in the SPY-Net. $C(P)$ represents its current capacity, and is monitored by P. An edge (P, P') in the SNMG represents the connection from SPY P to P'. $B(P, P')$ represents its current bandwidth. We assume that $B(P, P')$ is monitored by P. Each SPY periodically propagates its resource monitoring results to other SPYs, while it updates its SNMG with results from itself and other SPYs[†].

3.1. Proxy Capacity and Connection Bandwidth

Proxy capacity can be estimated with the help of the operating system running on the SPY. With recent advances in resource management frameworks for CPU,[5, 6] disk I/O bandwidth,[7] and buffer space, it becomes possible to estimate the current resource availability via APIs provided by these resource management schemes. Furthermore, by using the 'ticket and currency' based model,[8] it will be possible to 'collapse' multiple resources into a single 'abstract' resource with uniform denomination. However, this is outside the scope of this paper. For simplicity, we in this paper assume the availability of a single resource (such as CPU) as the proxy capacity.

Connection bandwidth between SPYs is more difficult to measure. Instead of measuring current *available* bandwidth, we suggest the probing of current *feasible* bandwidth between the SPYs as follows: when SPY P probes $B(P, P')$, P sends to P' a series of fixed-size pseudo-frames using different levels of bandwidth (for example, 28.8Kbps, 64Kbps, 128Kbps... 2Mbps), while P' responds with ACKs or NACKs indicating if the frames arrive on time. P progressively raises the bandwidth level if certain number of consecutive ACKs are received, but immediately drops to the previous level if one NACK is received. The probing continues until the highest stable level is reached. The probing is conservative and only indicates that $B(P, P')$ is feasible with high probability.

[†]We assume that each SPY knows every other SPY in the SPY-Net via some pre-configuration.

Note that if the proxy capacity and connection bandwidth are reservation-enabled, our solution will provide end-to-end resource availability guarantee for each service path request. However, even if the resources are not reservation-enabled, our service path finding algorithm will still improve the overall success rate of service path requests based on the resource monitoring information.

3.2. SNMG Construction and Maintenance

As implied in Section 3.1, probing connection bandwidth incurs both network traffic and SPY overhead. Therefore, it is necessary to control the number of connections being actively monitored by each SPY. However, since the monitored connections are candidates to be selected to form media service paths, it is important that good connections be dynamically discovered and added to the SNMG, while degraded connections be identified and purged.

Similar to the application-level multicast systems,[2, 9, 10] we put constraints on both the number of outbound connections a SPY actively probes and the number of inbound connections (probed by other SPYs) the SPY accepts. For each SPY P, we denote $d_{out}(P)$ and $d_{in}(P)$ as the upper bounds for the numbers of outbound and inbound monitored connections, respectively. We also denote $ADJ_{out}(P)$ and $ADJ_{in}(P)$ as the 'neighbor' SPYs associated with the outbound and inbound connections, respectively. The elements in $ADJ_{out}(P)$ and $ADJ_{in}(P)$ will be dynamically decided by P at runtime.

However, the method used by P to decide the elements in $ADJ_{out}(P)$ is different from the methods in the application-level multicast systems.[2, 9, 10] We describe our method as follows.

Initialization During SPY-Net deployment, each SPY P is assigned an initial $ADJ_{out}(P)$ and $ADJ_{in}(P)$, which satisfies the $(d_{out}(P), d_{in}(P))$ constraint. We assume that the initial assignments lead to a *connected* SNMG.

Monitoring and propagation At runtime, each SPY P periodically probes each connection from itself to a SPY $\in ADJ_{out}(P)$. The probing period T_P is a configurable parameter, and each probing starts randomly during the period. In addition, the current proxy capacity of P is obtained in each period. At the end of each period, P propagates the resource monitoring results to other SPYs in the SPY-Net[‡].

In addition, at the end of each period, P executes Dijkstra's algorithm to compute the following (the purpose will soon be explained): based on the currently monitored connections (edges) in the SNMG, what is the maximum end-to-end concatenated bandwidth from P to every other SPY. This can be computed by Dijkstra's algorithm, if we replace the '+' operator with 'min', and '>' operator with '<' in the algorithm. We denote the resultant maximum end-to-end bandwidth from P to P' as $B_{max}(P \longrightarrow P')$.

Trying and update Meanwhile, each SPY P also probes the connections to SPYs *outside* $ADJ_{out}(P)$ - at a much lower frequency. The probing period for these connections can be multiples of T_P, for example $10T_P$. After such a connection (P, P_x) is probed, P decides if P_x should be added to $ADJ_{out}(P)$:

- P compares $B(P, P_x)$ with $B_{max}(P \longrightarrow P_x)$ computed by Dijkstra's algorithm. If $B(P, P_x) \leq B_{max}(P \longrightarrow P_x)$, P_x will *not* be added to $ADJ_{out}(P)$.
- Otherwise, if the number of elements in $ADJ_{out}(P)$ has not reached $d_{out}(P)$, P_x will be added to $ADJ_{out}(P)$ immediately. Here, we assume that when P_x was probed, P_x already checked if its $d_{in}(P_x)$ limit has been reached. The probing will be performed only if the limit has *not* been reached.
- If $d_{out}(P)$ has been reached, a procedure (Figure 2) will be executed to find at least one 'victim' SPY $P_v \in ADJ_{out}(P) \cup \{P_x\}$ to be removed (or kept away - if it is P_x) from $ADJ_{out}(P)$. Specifically, the procedure identifies P_v such that: in graph $SNMG^+$, which is the current SNMG *plus* an additional edge (P, P_x), if edge (P, P_v) is removed from $SNMG^+$, the consequent end-to-end bandwidth from P to P_v ($B_{csq}(P \longrightarrow P_v)$ in Figure 2) will drop by the *lowest percentage*, compared with the current $B(P, P_v)$.

[‡]To save propagation bandwidth, each SPY may not propagate the items with only minor changes since last probing period.

FIND-VICTIM ()
$B_{csq}(P \longrightarrow P_x) = B_{max}(P \longrightarrow P_x)$;
$percent(P_x) = \frac{B(P,P_x) - B_{csq}(P \longrightarrow P_x)}{B(P,P_x)} * 100\%$;
for each $P_i \in ADJ_{out}(P)$ {
$B_{csq}(P \longrightarrow P_i) = max_{P_j \in ADJ_{out}(P) \cup \{P_x\}, P_j \neq P_i}(min(B(P,P_j), B_{max}(P_j \longrightarrow P_i)))$;
$percent(P_i) = \frac{B(P,P_i) - B_{csq}(P \longrightarrow P_i)}{B(P,P_i)} * 100\%$;
}
return $\{P_v | P_v \in ADJ_{out}(P) \cup \{P_x\}$, and $percent(P_v)$ is the lowest$\}$

Figure 2: The procedure to find victim(s)

We note that in the procedure, $B_{max}(P_j \longrightarrow P_i)$ is needed. However, Dijkstra algorithm executed by P does *not* compute this value. The problem is solved as follows: each $P_j \in ADJ_{out}(P) \cup \{P_x\}$ piggy-backs the values of $B_{max}(P_j \longrightarrow P_i)$ ($P_i \in ADJ_{out}(P)$ and $P_i \neq P_j$) with an ACK back to P, during the bandwidth probing for $B(P,P_j)$. P can then use these values.

- After the procedure, if P_x is the only victim selected, then nothing happens. Otherwise, P removes the 'victim' SPY(s) from $ADJ_{out}(P)$ and adds P_x to it. This change will be known by all other SPYs after P's next round of propagation.

The intuition behind our method is: between any pair of SPYs P and P', if $B_{max}(P \longrightarrow P')$ in the current SNMG is equal to or better than the direct connection bandwidth $B(P,P')$, then connection (P,P') will not be monitored. Here we prefer multiple 'short' connections to one 'long-haul' connection from P to P'. The former may provide more SPY mapping (from requested proxy services to actual SPYs) options during service path finding, and it may better reflect the routing topology of the lower-level networks. Meanwhile, to dynamically discover improving/degrading connections, we also probe connections not in the SNMG. To replace a monitored connection due to the d_{out} constraint, our policy in the procedure is to purge such a connection whose removal will lead to the lowest percent of end-to-end bandwidth drop between any pair of SPYs. Note that the removal will not affect any on-going service paths which involve the victim connection. The only consequence is that the victim connection will not appear in service paths computed after the removal.

In addition, P will keep the probing result of a connection *not* added to SNMG for a period of T_P. The result will be leveraged to create 'short-cuts' *during* media service sessions. This will be presented in Section 4.3.

Finally, SNMG partition may occur due to SPY failure or transient inconsistency of SNMGs at different SPYs. Fortunately, the application-level multicast systems[2, 10] have proposed methods to detect and repair 'mesh partitions', which can be readily applied to the SPY-Net. Details of these methods are omitted due to the space limitation of this paper.

4. FINDING SERVICE PATHS IN SPY-NET

After describing the SPY-Net resource monitoring mechanism, we proceed to present the service path finding algorithm. Each SPY executes this algorithm *locally* on the SNMG. We first formally define the problem of finding the best media service path.

4.1. Problem Definition

Suppose in each service path request, there are K requested media proxy services. We call each requested media proxy service a *virtual SPY (VSPY)*, and the service path a *Virtual service path*. Let $vP_k (1 \leq k \leq K)$ represent the VSPYs and their logical service order. For denotation convenience, let vP_0 and vP_{K+1} be the media source and destination hosts (they are not SPYs), respectively. Let $C(vP_k)$ be the proxy capacity requirement of vP_k $(1 \leq k \leq K)$, and $B(vP_k, vP_{k+1})$ be the bandwidth requirement from vP_k to vP_{k+1} $(0 \leq k \leq K)$.

Let P_s and P_d be the SPY-Net 'access points' (i.e. the nearest SPYs) of the media source and destination hosts, respectively. We assume that the service path request is sender-initiated. Therefore, a service path request will be submitted to P_s, which will execute the service path finding algorithm.

The objective of the algorithm is to find both (1) a path $\mathbf{P}$ in SNMG from P_s to P_d: let $|\mathbf{P}|$ be the number of SPYs on $\mathbf{P}$, and P_i $(1 \leq i \leq |\mathbf{P}|)$ be the SPYs. Especially, $P_1 = P_s$ and $P_{|\mathbf{P}|} = P_d$; and (2) a mapping $\mathbf{M}$ from vP_k $(1 \leq k \leq K)$ to a SPY on $\mathbf{P}$[§]: if vP_i and vP_j $(1 \leq i < j \leq K)$ are mapped to SPYs P_x and P_y, then either $P_x = P_y$ or P_y is reachable from P_x on $\mathbf{P}$ (which we denote as $P_x \leq P_y$). Among all possible paths and mappings, $\mathbf{P}$ and $\mathbf{M}$ achieve the *minimum* value of a generic evaluation function F. We define F as follows:

$$F(\mathbf{P}, \mathbf{M}) = G(f_C(\mathbf{P}, \mathbf{M}), f_B(\mathbf{P}, \mathbf{M})), \tag{1}$$

$G(x, y)$ is a *non-decreasing* function specific to a SPY-Net (simple examples such as $G(x, y) = max(\alpha x, \beta y)$ or $G(x, y) = \alpha x + \beta y$, $\alpha > 0$, $\beta > 0$).

$f_C(\mathbf{P}, \mathbf{M})$ is the proxy capacity evaluation function defined as:

$$f_C(\mathbf{P}, \mathbf{M}) = \sum_{i=1}^{|\mathbf{P}|} \frac{\sum_{\mathbf{M}(vP_k)=P_i} C(vP_k)}{C(P_i)} \tag{2}$$

$f_B(\mathbf{P}, \mathbf{M})$ is the connection bandwidth evaluation function defined as:

$$f_B(\mathbf{P}, \mathbf{M}) = \sum_{i=1}^{|\mathbf{P}|-1} \frac{B(vP_j, vP_{j+1})_{\mathbf{M}(vP_j) \leq P_i \text{ and } P_{i+1} \leq \mathbf{M}(vP_{j+1})}}{B(P_i, P_{i+1})} \tag{3}$$

Intuitively, function f_C computes the accumulative 'risk' (or 'cost') with respect to proxy capacity 'requirement-to-availability' ratio on $\mathbf{P}$: the higher the f_C value, the more costly the accumulative proxy capacity requirement is. Meanwhile, function f_B characterizes the accumulative risk or cost with respect to bandwidth 'requirement-to-availability' ratio on $\mathbf{P}$: the higher the f_B value, the more costly the accumulative bandwidth requirement is. To combine the evaluation of f_C and f_B, the SPY-Net specific and configurable function G is introduced. The reason is that the values of f_C and f_B are not directly comparable, due to their differences in resource types and probing methods. Function G is therefore used to 'normalize the price' of both proxy capacity and connection bandwidth. For example, if $G(x, y) = c_1 x + c_2 y$, we can intuitive think of $c_1 * f_C(\mathbf{P}, \mathbf{M})$ and $c_2 * f_B(\mathbf{P}, \mathbf{M})$ as the 'cost' of proxy capacity and connection bandwidth under some uniform 'currency', respectively.

With G combining f_C and f_B, function F evaluates how risky or costly the choices of $\mathbf{P}$ and $\mathbf{M}$ are, with respect to overall resource usage: the lower the $F(\mathbf{P}, \mathbf{M})$ value, the less risky or costly the choices for a service path request. Therefore, the goal of our algorithm is to find $\mathbf{P}$ and $\mathbf{M}$ that minimize $F(\mathbf{P}, \mathbf{M})$.

Although the evaluation function F is generic, our service path finding algorithm works as long as function G is non-decreasing. This claim will be shown in Section 4.2. For now, we first have the following lemma (proof can be found in a technical report[12]):

Lemma 1 *Let $G(x, y)$ be a non-decreasing function. For a service path $\mathbf{P}$ and VSPY-to-SPY mapping $\mathbf{M}$, if $\mathbf{P}'$ is a sub-path of $\mathbf{P}$ which starts from the same P_s, and $\mathbf{M}_k$ $(1 \leq k \leq K)$ is an incomplete mapping from the first k VSPYs to SPYs on $\mathbf{P}'$, such that $\mathbf{M}(vP_i) = \mathbf{M}_k(vP_i)$ $(1 \leq i \leq k)$, then we have $F(\mathbf{P}', \mathbf{M}_k) \leq F(\mathbf{P}, \mathbf{M})$.*

[§]We assume in this paper that a SPY has the functionality to perform every type of proxy service requested. This is practical using the *active service* technique.[11]

4.2. Media Service Path Finding Algorithm

Our media service path finding algorithm extends Dijkstra's algorithm.[13] More specifically, we extend Dijkstra's algorithm to accommodate the 'try-out' of different VSPY-to-SPY mapping options, while exploring the different paths from P_s to P_d.

Extended multi-stage estimates for functions F, f_C, and f_B In Dijkstra's algorithm, each node P in the graph is associated with a shortest-path estimate $d[P]$. For each node P in SNMG, we extend this estimate to $3*(K+1)$ estimates: $F^k[P]$, $f_C^k[P]$, and $f_B^k[P]$ $(0 \leq k \leq K)$. $F^k[P]$ estimates the minimum value of function F - for any path from P_s to P and for any *incomplete* mapping from the first k VSPYs to SPYs on the path. $f_C^k[P]$ and $f_B^k[P]$ maintain the corresponding f_C and f_B values leading to $F^k[P]$, respectively.

Extended pointers for tracing service paths In Dijkstra's algorithm, $\pi[P]$ represents the predecessor of P. We extend $\pi[P]$ to $\pi^k[P]$, $\psi^k[P]$, and $path^k[P]$ $(0 \leq k \leq K)$: $\pi^k[P]$ represents the predecessor of P, in the case when the first k VSPYs have been resolved (mapped) on the path from P_s to P. $\psi^k[P]$ represents the number of VSPYs that have been resolved on the path from P_s to $\pi^k[P]$. Therefore, there are exactly $(k - \psi^k[P])$ number of VSPYs mapped to SPY P. We will use both $\pi^k[P]$ and $\psi^k[P]$ as 'pointers' to trace the best service path when the algorithm terminates. Finally, $path^k[P]$ is for our presentation convenience only. It is a linked list recording the path from P_s to P with the first k VSPYs resolved.

Extended RELAX procedure We extend the RELAX procedure to implement the progressive try-out of different VSPY-to-SPY mappings. The procedure is for two nodes P_u and P_v in the SNMG with an edge from P_u to P_v, as shown in Figure 3. Each iteration in RELAX tries out the following VSPY-to-SPY mapping: given that the first k VSPYs have been mapped to SPYs from P_s to P_u, can we achieve a smaller value of $F^i[P_v]$, if we map the following $(i-k)$ VSPYs to P_v?

```
RELAX(P_u, P_v, k, G)
    for i = k to K {
        x = B(vP_k, vP_{k+1}) / B(P_u, P_v);
        y = (sum_{j=k+1}^{i} C(vP_j)) / C(P_v);
        if x ≤ 1 and y ≤ 1 { //resource availability checking
            b = f_B^k[P_u] + x; c = f_C^k[P_u] + y;
            if G(c, b) < F^i[P_v] {
                F^i[P_v] = G(c, b);
                f_B^i P_v = b; f_C^i P_v = c;
                π^i[P_v] = P_u; ψ^i[P_v] = k;
                path^i[P_v] = path^i[P_u] + P_u;
            }
        }
    }
```

Figure 3: Extended RELAX procedure

The computational complexity of the extended RELAX procedure is $O(K)$, assuming that $\sum_{j=k+1}^{i}(C(vP_j))$ is computed once and re-used by subsequent RELAX procedure calls. Fortunately, the number of VSPYs in a service path request is typically very small (less than five VSPYs). Therefore, RELAX can practically be considered as a constant time procedure.

Extended INITIALIZE procedure We also extend the INITIALIZE procedure to be called at the beginning of the algorithm (complete pseudo-code can be found in a technical report[12]). Especially, the value of $F^k[P_s]$ $(0 \leq k \leq K)$ is initialized as follows:

$$F^k[P_s] = G(\frac{\sum_{j=1}^{k}(C(vP_j))}{C(P_s)}, 0) \tag{4}$$

Extended Dijkstra's algorithm We are now ready to extend Dijkstra's algorithm (Figure 4). The major extension is one more layer of loop ('for k =...') outside the original algorithm. At the end of each iteration of this loop, we will get a path from P_s to P_d (in fact, to other SPYs too) and a mapping $\mathbf{M}_k$ for the first k VSPYs, which achieve the minimum value of $F(\mathbf{P}, \mathbf{M}_k)$. Therefore, the VSPY-to-SPY mapping progresses by one VSPY after each iteration. In addition, due to the iterative nature of the VSPY-to-SPY mapping, loop may be formed across different iterations. Therefore, we introduce a loop checking before performing RELAX. This will prevent any loop in resultant service paths.

```
EXTENDED-DIJKSTRA(SNMG, P_s)
    INITIALIZE (SNMG, P_s);
    for k = 0 to K {
        S^k = φ; Q^k = V[SNMG];
    }
    for k = 0 to K
        while Q^k ≠ φ do {
            P_u = EXTRACT-MIN(Q^k); S^k = S^k ∪ {P_u};
            for each P_v ∈ ADJ_out(P_u)
                if P_v is not on path^k[P_u] and P_v ∈ Q^k //loop checking
                    RELAX(P_u, P_v, k, G);
        }
```

Figure 4: Extended DIJKSTRA's algorithm

After the execution of EXTENDED-DIJKSTRA, the best service path can be identified by traversing from P_d back to P_s: the predecessor of P_d is $P_{pred} = \pi^K[P_d]$, and the VSPYs which are mapped to P_d can be determined as the VSPYs with index numbers $\psi^K[P_d] + 1$ to K. Next, if we let $k = \psi^K[P_d]$, then the predecessor of P_{pred} is $\pi^k[P_{pred}]$, and the VSPYs mapped to P_{pred} are the VSPYs with index numbers from $\psi^k[P_{pred}] + 1$ to k, and so on, until we reach P_s.

The computational complexity of our service path finding algorithm is $O(KV^2 + K^2EV)$. The increased complexity comes from the loop checking (Figure 4), which takes $O(V)$ time. To proof the correctness of our algorithm, we first have the following lemma:

Lemma 2 *There is no loop on the best service path computed by EXTENDED-DIJKSTRA.*

With Lemma 1 and Lemma 2, we have the following theorem about the correctness of our service path finding algorithm (proof can be found in a technical report[12]):

Theorem 1 *If EXTENDED-DIJKSTRA algorithm is run on the SNMG. At the termination, for any P in SNMG, let $\mathbf{P}$ and $\mathbf{M}$ be the computed service path and VSPY-to-SPY mapping, respectively. Then $\mathbf{P}$ is loop-free; and $F(\mathbf{P}, \mathbf{M}) = F^K[P]$ is minimum among all paths from P_s to P and all mappings of the K VSPYs.*

4.3. In-session Service Path Pruning

After P_s computes the service path, the media service session will start. We assume that the media data are packetized into application-level packets and transmitted along the SPYs on the service path. The application-level packets are of uniform format which is understood by every SPY in the SPY-Net. We further assume that the header of each application-level packet contains (1) the addresses of SPYs ($P_i, 1 \leq i \leq |\mathbf{P}|$) on service path $\mathbf{P}$, (2) the VSPY-to-SPY mapping $\mathbf{M}$; and (3) the bandwidth requirement $B(vP_k, vP_{k+1})$ $(0 \leq k \leq K)$ on the virtual service path.

However, the service path is not pinned during the media service session. In this section, we propose an in-session service path pruning technique, which dynamically changes the service path during the service session. The goal is to reduce the number of SPYs on $\mathbf{P}$. Recall in Section 3.2, to control the number of monitored connections and to expose more VSPY-to-SPY mapping opportunities, the SPY-Net monitoring mechanism

prefers multiple 'short' connections to one long-haul connection between two SPYs, if the concatenated end-to-end bandwidth of the former is no less than the connection bandwidth of the latter. However, this may lead to service paths with consecutive 'forward-only' SPYs - SPYs to which no VSPYs are mapped, serving only as relays. For example, in Figure 5, between P_2 to P_6, there are three forward-only SPYs P_3, P_4, and P_5. Consecutive forward-only SPYs incur additional service session delay and points of failure.

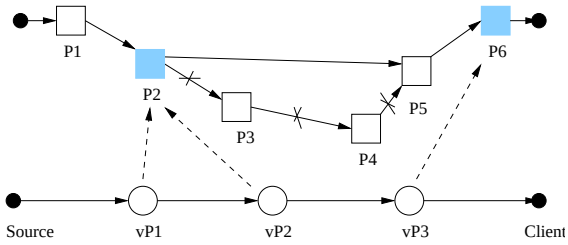


Figure 5: In-session service path pruning

Our service path pruning makes use of *recent* probing results of connections *not* added to the SNMG. Recall that in Section 3.2, such a result will be kept by the probing SPY P for a period of T_P. In Figure 5 for example, suppose P_2 recently probed connection (P_2, P_5), which is not in the SNMG. However, if $B(P_2, P_5)$ is equal to the concatenated bandwidth from P_2 to P_5 via P_3 and P_4, P_5 will not be added to $ADJ_{out}(P_2)$. During the service session, when P_2 sends out application-level media data packets, P_2 can in fact send them directly to P_5, because P_2 knows that $B(P_2, P_5)$ is equal to the bandwidth along $(P_2 \rightarrow P_3 \rightarrow P_4 \rightarrow P_5)$. Therefore, the service path is dynamically pruned to $P_1 \rightarrow P_2 \rightarrow P_5 \rightarrow P_6$.

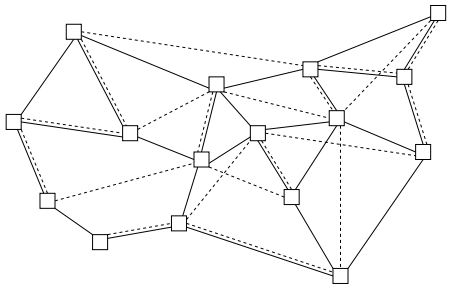
5. EXPERIMENTS

5.1. Simulation Setup

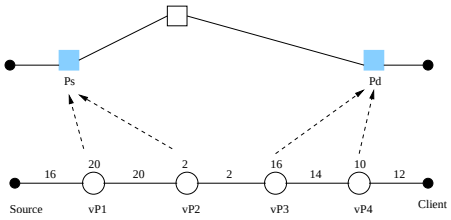
In this section, we present simulation results to evaluate the performance of the SPY-Net architecture and service path finding algorithm. Using our own simulation program, we simulate a SPY-Net of 16 SPYs. In Figure 6(a), the solid lines represent the actual links in the lower-level network, while the dotted lines represent the *initial assignment* of the SNMG edges. To determine the actual route between a pair of SPYs, the lower-level network uses a *non-QoS* routing mechanism based on hop count only. Each SPY has an initial total capacity of C_{init} units. Each physical connection in Figure 6(a) has an initial total bandwidth of $B_{init} = 1000$ units. When a SPY P probes the *application-level* connection bandwidth from P to P', it will get the concatenated (minimum) bandwidth along the lower-level route from P to P'. Function G in the service path finding algorithm is defined as $G(x, y) = x + y$.

Service path requests are generated according to a Poisson process. In each run of the simulation, we use a different average request arrival rate, ranging from 100 req/min to 400 req/min. The range of request arrival rate is chosen to reflect different workload of the SPY-Net (from under-utilized to overloaded). Each run lasts 3 hours. For each request, the source SPY P_s is randomly selected among all SPYs, while the destination SPY P_d is selected from all SPYs - each with a different and time-varying probability. This is to create non-uniform and changing load across the SPY-Net. The session duration of each requested service path is also heterogeneous, ranging randomly between 1 minute and 45 minutes. The virtual service path associated with each service path request is generated as follows: it contains 4 VSPYs[¶]. For both proxy capacity of a VSPY and connection bandwidth between any consecutive VSPYs, the minimum and maximum requirements are r (r is randomly set among 1, 2, and 3) units and $r * M (M > 1)$ units, respectively. Other capacity and bandwidth requirements are randomly set between the minimum and maximum. The lower portion of Figure 6(b) shows an example of virtual service path, in which $M = 10$ and $r = 2$.

[¶]We assume that the number of VSPYs in a service path request is typically very small.



(a) Topology of lower-level network (solid) and initial SNMG (dotted)



(b) Example virtual service path and illustration of *Intuit* algorithm

Figure 6: Simulation setup

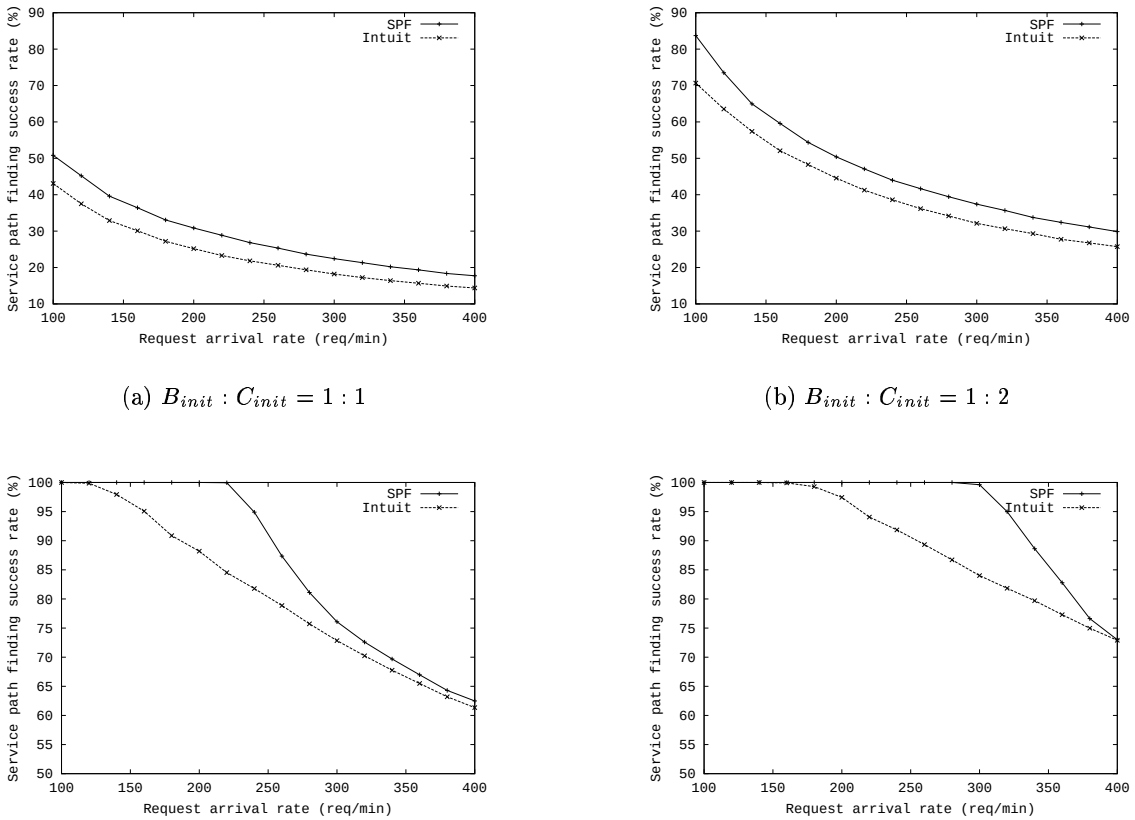
5.2. Simulation Results

(1) Overall success rate of finding service paths We first evaluate the success rate achieved by our service path finding algorithm, which we denote as *SPF* in the rest of this section. A service path is successfully found, if and only if during the *entire* service session, its resource requirements are always satisfied by the resource availability on the path.

For comparison, we also simulate another service path finding algorithm which we call *Intuit*. Algorithm *Intuit* maps the VSPYs on a virtual service path to either P_s or P_d, and exposes only the 'thinnest' connection to the network. Therefore, *Intuit* results in finding an end-to-end path (without any SPY) from P_s to P_d, with the lowest bandwidth requirement on the virtual service path. We use *Intuit* as a comparison, because it aligns well with the intuition to reduce the service path finding problem to the unicast QoS routing problem. *Intuit* works as follows: (1) it identifies the minimum bandwidth requirement B_{min} on the virtual service path, for example, the bandwidth between vP_2 and vP_3 in Figure 6(b); (2) it maps the VSPYs *before* this 'thinnest' connection to P_s, for example, vP_1 and vP_2 are mapped to P_s in Figure 6(b); (3) it maps the VSPYs *after* the 'thinnest' connection to P_d, for example, vP_3 and vP_4 are mapped to P_d; (4) it finds a best end-to-end path $\mathbf{P}$ from P_s to P_d which can offer bandwidth B_{min}.

Figure 7 shows the path finding success rate achieved by *SPF* and *Intuit*, under different request arrival rates. Results in each sub-figure are obtained under a different ratio of $B_{init} : C_{init}$ (with B_{init} fixed at 1000 units). M (see Section 5.1) is fixed at 10. We observe that the success rate of *SPF* is constantly *no* lower than that of *Intuit*. The former may be higher than the latter by as much as 20%. The reason is that *SPF* dynamically identifies insufficient resources and discovers available resources - either proxy capacity or connection bandwidth. On the other hand, *Intuit* only tries to minimize bandwidth usage. In Figures 7(a) and 7(b), C_{init} is not significantly higher than B_{init}. In this case, both proxy capacity and connection bandwidth can become scarce at different times. We observe that the success rate of *SPF* is always higher than that of *Intuit*. In Figures 7(c) and 7(d), C_{init} is 6 and 8 times of B_{init}, respectively. In this case, proxy capacity tends to be more plentiful than bandwidth. We notice that although *SPF* is still the winner, the success rates of both algorithms gradually converge, with the increase of request arrival rate. This is because when bandwidth is relatively scarce and request arrival rate is high, *SPF* tends to find the same path as *Intuit* does for each request.

We also study the impact of M on service path finding success rate. M is the ratio of maximum to minimum resource requirement on a virtual service path. To focus on bandwidth requirement, we use the same initial resource availability B_{init} and C_{init} as in Figure 7(d) where $B_{init} : C_{init} = 1 : 8$, so that proxy capacity is relatively plentiful. However, we increase the value of M from 10 to 15 and 20, respectively, and the results are shown in Figure 8. Comparing Figures 7(d), 8(a), and 8(b), we are interested in the convergence of *SPF* and *Intuit* success rates. We observe that the higher the M, the lower the request arrival rate at which *SPF* loses its lead (at 400 req/min, 280 req/min, and 220 req/min when $M = 10, 15, 20$, respectively).



(a) $B_{init} : C_{init} = 1 : 1$

(b) $B_{init} : C_{init} = 1 : 2$

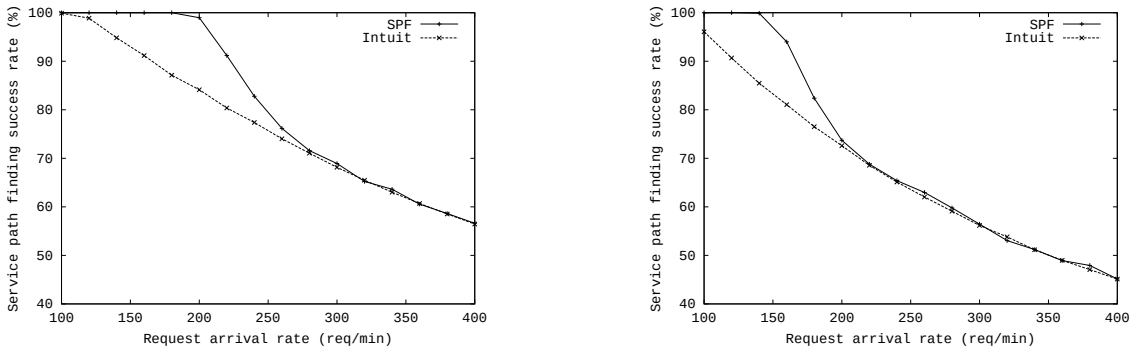
(c) $B_{init} : C_{init} = 1 : 6$

(d) $B_{init} : C_{init} = 1 : 8$

Figure 7: Service path finding success rate under different $B_{init} : C_{init}$ ratio

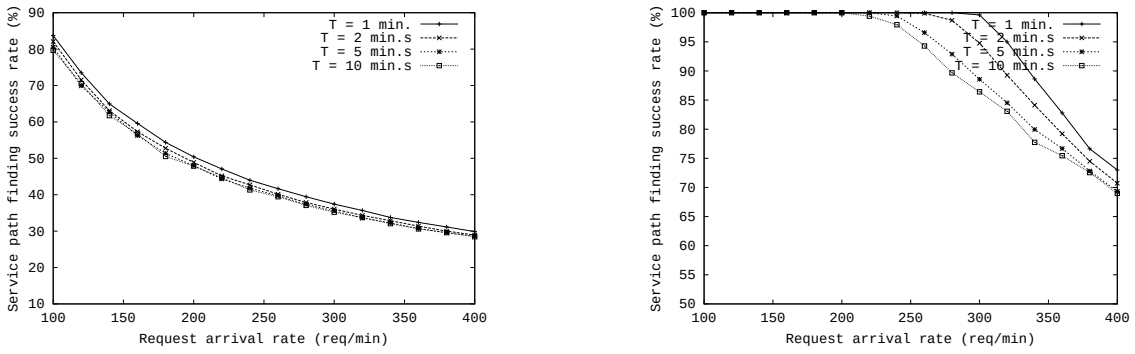
(2) Impact of propagation period on service path finding success rate The success rate of *SPF* is affected by the period length for resource monitoring result propagation. The shorter the period, the more updated the resource availability information a SPY can gather. However, too frequent propagation of resource monitoring results causes non-trivial network traffic, and will hurt the scalability of a SPY-Net with respect to the number of SPYs. In the experiments of **(1)**, we use a propagation period T of 1 minute, which may not be realistic in a real SPY-Net. Therefore, we increase T to 2, 5, and 10 minutes, respectively. The results are shown in Figure 9, and they are for the experiment configurations of $M = 10, B_{init} : C_{init} = 1 : 2$ and $M = 10, B_{init} : C_{init} = 1 : 8$, respectively. The results verify the negative impact of larger T on the success rate of *SPF*. Furthermore, the different degree of success rate degradation in Figures 9(a) and 9(b) suggests that the appropriate value of T may be different in SPY-Nets with different system parameters. An in-depth study on the relation between T and the system parameters (such as B_{init} and C_{init}) is our on-going work.

(3) Service path length reduction via in-session pruning In **(3)** and **(4)**, we evaluate the effect of our resource monitoring and SNMG maintenance mechanisms. We first evaluate the in-session pruning technique to reduce service path length. In this experiment, we compare the service path length before and after in-session pruning. We set T as 1 minute, which is also the duration a bandwidth probing result will be kept. We also set the $d_{out}(P)$ of each SPY P as 4, 6, 8, and 10, respectively. Figure 10(a) shows the result. The x-axis is the service path length *before* pruning, and the y-axis is the average length of the same paths after pruning. We observe the reduction of service path length. We also observe that with the increase of T (not shown in Figure 10(a)) and $d_{out}(P)$, the reduction becomes larger. This is because larger T and $d_{out}(P)$ introduces more



(a) $M = 15$ (b) $M = 20$

Figure 8: Service path finding success rate under different M



(a) $B_{init} : C_{init} = 1 : 2$ (b) $B_{init} : C_{init} = 1 : 8$

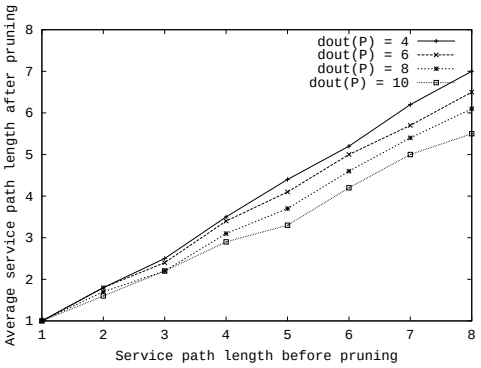
Figure 9: Service path finding success rate of *SPF* under different propagation period

pruning possibilities in the SNMG.

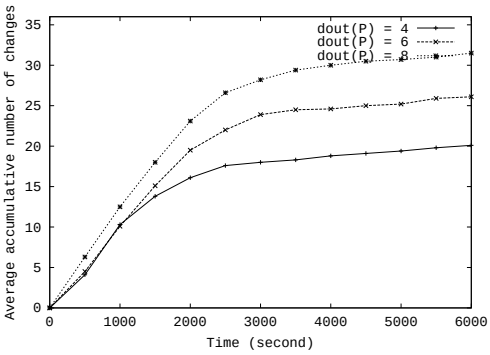
(4) Stability of SNMG topology Finally, we study the stability of the SNMG topology during the execution of our simulation. We set $d_{out}(P)$ for each SPY as 4, 6, and 8, respectively. We also set T as 1 minute. Under request arrival rate of 200 req/min, We record the average accumulative number of changes in $ADJ_{out}(P)$ with the elapse of time. The results are shown in Figure 10(b). We notice that under each $d_{out}(P)$, the topology of SNMG becomes stable after the first 5 minutes. Furthermore, we observe that (not shown in Figure 10(b)) more than 95% of the actual lower-level links are included in the stable SNMG (recall the difference between the initial SNMG assignment and the lower-level network topology in Section 5.1). This demonstrates that our resource monitoring and SNMG maintenance mechanisms achieve topology stability for the SNMG.

6. RELATED WORK

Service paths (and similar concepts) have been proposed, and their usefulness has been justified.[3, 14, 15] In the Ninja Project,[3] service path is defined as a sequence of application-level service operators and connectors. It focuses on path specification, instantiation, and protocol translation along the path. The CANS framework[14] is proposed for the composition of adaptive service delivery over the Internet. It highlights QoS adaptation along the service delivery path under different resource conditions. A programming framework[15] is presented for the



(a) Service path length reduction via pruning ($T = 1$ min)



(b) Average accumulative number of changes in $ADJ_{out}(P)$ for a SPY P ($T = 1$ min)

Figure 10: Effect of resource monitoring and SNMG maintenance mechanisms

construction of network services to access media data. Its focus is more on the programming model and open signaling between media-playing objects. However, these works do not address the problem of how to find a good service path in the networks. Finally, an integrated path and server selection technique[16] is proposed for networked multimedia environments. However, it does not consider multiple proxy services and the mapping to physical proxies.

Application-level multicast[1, 2, 9, 10, 17, 18] has recently been studied. An application-level overlay network is proposed to support multicast even without IP-level multicast capability. Application-level multicast architectures have been introduced in the context of large scale content distribution,[1, 2, 18] or small-to-medium scale group communications.[9, 10] Our SPY-Net architecture shares the same design principle of maintaining an overlay network, monitoring its resource and performance at application-level, and planning data transmission according to the monitored resource or performance condition.

However, the SPY-Net has the following differences from the proposed application-level multicast architectures: (1) it aims at enabling the dynamic combination of multiple media proxy services, and the composition of new and value-added media services from basic proxy services; and (2) when planning media service paths, it not only considers the performance of network connections, but also considers the end-system resource availability in media service proxies.

In technical details, SPY-Net uses a link-state-like approach in finding service paths, while in the application-level multicast systems,[2, 10] a distance-vector-like approach is used. We do not choose the distance-vector approach because: different service path requests may have very different requested service types, proxy capacity requirement, and connection bandwidth requirement. Given our problem definition for service path finding (Section 4.1), it is not possible to adopt a single definition of 'distance' to be estimated and propagated. On the contrary, the 'distance' can only be determined *jointly* by SPY-Net resource availability and by individual service path request upon its arrival. This is also the reason why it may be difficult to use the service path *pre-computation* approach proposed by Shaikh et al[19] (though SPY-Net does share the same link-state mechanism). It is likely that SPY-Net will share the same overlay network with an application-level multicast architecture. Therefore, the different routing or path finding mechanisms will co-exist and complement each other.

Interestingly, we find that our approach can easily be merged into the ALMI infrastructure.[9] Our SNMG currently kept by each SPY will then become the *neighbor monitoring graph* kept by the *session controller*.[9] The session controller may then execute our service path finding algorithm. Also, given its relatively small size, the SPY-Net can also be plugged in to the Resilient Overlay Networks (RON) architecture,[20] which already provides the capability of network performance monitoring.

7. CONCLUSION

We envision the emergence of application-level media service proxy networks (SPY-Nets). Media sources and clients will be able to 'tap to' a SPY-Net to form customized, composite, and value-added media proxy services. In this paper, we study the problem of finding service paths in the SPY-Net. Our solution includes (1) a mechanism to monitor and propagate resource availability condition in the SPY-Net and (2) an algorithm to find the best service path based on the resource monitoring results. Our theoretical analysis and simulation results show that: (1) the resource monitoring mechanism achieves reasonable stability and accuracy for service path computation, while incurring controlled probing and propagation overhead; (2) the service path finding algorithm finds the best path for each service path request, with respect to a generic resource evaluation function; (3) the service path finding algorithm achieves high success rate. Especially, it performs no worse (and in many cases better) than a service path planning method which may align well with common intuition; and (4) it is an application-level solution, requiring no changes to the lower-level network infrastructure. Our on-going work includes the issue of interoperability with current application-level multicast architectures, and the issue of multi-dimensional resource condition (for example, with both network bandwidth and delay) in the resource evaluation function.

ACKNOWLEDGMENTS

We would like to thank the anonymous reviewers and Dr. Anees Shaikh, our 'shepherd', for their detailed and constructive comments which have helped us improve the final version of this paper. The work presented in this paper was supported by the National Science Foundation under contract number 98-70736, the Air Force Grant under contract number F30602-97-2-0121, National Science Foundation Career Grant under contract number NSF CCR 96-23867, NSF PACI grant under contract number NSF PACI 1-1-13006, NSF CISE Infrastructure grant under contract number NSF EIA 99-72884, NSF CISE Infrastructure grant under contract number NSF CDA 96-24396, and NASA grant under contract number NASA NAG 2-1250. However, Views and conclusions of this paper are those of the authors, and should not be interpreted as representing the official policies, either expressed or implied, of the funding agencies.

REFERENCES

1. J. Jannotti, D. Gifford, K. Johnson, F. Kaashoek, and J. J. O'Toole, "Overcast: Reliable Multicasting with an Overlay Network," *Proceedings of USENIX Symposium on OS Design and Implementation (OSDI2000)* , Oct. 2000.
2. Y. Chawathe, "Scattercast: an Architecture for Internet Broadcast Distribution as an Infrastructure Service," *PhD Thesis, University of California at Berkeley* , 2000.
3. S. Gribble, M. Welsh, R. von Behren, E. Brewer, D. Culler, N. Borisov, S. Czerwinski, R. Gummadi, J. Hill, A. Joseph, R. Katz, Z. Mao, S. Ross, and B. Zhao, "The Ninja Architecture for Robust Internet-Scale Systems and Services," *Computer Networks, Special Issue on Pervasive Computing* , 2001.
4. A. Steinmetz, "The e-Seminar Lecture Recording and Distribution System - a Bottom-up Approach from Video to Knowledge Streaming," *Proceedings of SPIE/ACM Multimedia Computing and Networking (MMCN'01)* , Jan. 2001.
5. D. Yau, "Performance Evaluation of CPU Isolation Mechanisms in a Multimedia OS Kernel," *Proceedings of SPIE/ACM Multimedia Computing and Networking (MMCN'01)* , Jan. 2001.
6. D. Steere, A. Goel, J. Gruenberg, D. McNamee, C. Pu, and J. Walpole, "A Feedback-driven Proportion Allocator for Real-Time Scheduling," *Proceedings of USENIX Symposium on OS Design and Implementation (OSDI'99)* , Feb. 1999.
7. P. Shenoy and H. Vin, "Cello: a Disk Scheduling Framework for Next Generation Operating Systems," *Real-Time Systems Journal, Special Issue on Flexible Scheduling of Real-Time Systems* , 2000.
8. T. Zhao and V. Karamcheti, "Expressing and Enforcing Distributed Resources Sharing Agreements," *Proceedings of SC'2000: High Performance Networking and Computing Conference* , Nov. 2000.
9. D. Pendarakis, S. Shi, D. Verma, and M. Waldvogel, "ALMI: An Application Level Multicast Infrastructure," *Proceedings of USENIX Symposium on Internet Technologies and Systems (USITS'01)* , Mar. 2001.

10. Y. Chu, S. Rao, and H. Zhang, "A Case for End System Multicast," *Proceedings of ACM SIGMETRICS 2000* , June 2000.
11. E. Amir, S. McCanne, and R. Katz, "An Active Service Framework and its Application to Real-Time Multimedia Transcoding," *Proceedings of ACM SIGCOMM'98* , Sept. 1998.
12. D. Xu and K. Nahrstedt, "Finding Service Paths in a Media Service Proxy Network," *Technical Report, Department of Computer Sciences, Purdue University* , Oct. 2001.
13. T. Cormen, C. Leiserson, and R. Rivest, "Introduction to Algorithms," *MIT Press/McGraw Hill* , 1990.
14. X. Fu, W. Shi, A. Akkerman, and V. Karamcheti, "CANS: Composable, Adaptive Network Services Infrastructure," *Proceedings of USENIX Symposium on Internet Technologies and Systems (USITS'01)* , Mar. 2001.
15. A. Nakao, A. Bavier, and L. Peterson, "Constructing End-to-End Paths for Playing Media Objects," *Proceedings of IEEE OPENARCH 2001* , Apr. 2001.
16. Z. Fu and N. Venkatasubramanian, "Combined Path and Server Selection in Dynamic Multimedia Environments," *Proceedings of ACM Multimedia'99* , Nov. 1999.
17. S. Shi, J. Turner, and M. Waldvogel, "Dimensioning Server Access Bandwidth and Multicast Routing in Overlay Network," *Proceedings of International Workshop on Network and OS Support for Digital Audio and Video (NOSSDAV'01)* , June 2001.
18. Inktomi, "Inktomi Content Networking Solutions," *http://www.inktomi.com/products/cns* .
19. A. Shaikh, J. Rexford, and K. Shin, "Efficient Precomputation of Quality of Service Routes," *Proceedings of International Workshop on Network and OS Support for Digital Audio and Video (NOSSDAV'98)* , July 1998.
20. D. Andersen, H. Balakrishnan, F. Kaashoek, and R. Morris, "The Case for Resilient Overlay Networks," *Proceedings of International Workshop on Hot Topics in Operating Systems (HotOS-VIII)* , May 2001.