RUBEN: A Technique for Scheduling Multimedia Applications in Overlay Networks

Fang Chen and Vana Kalogeraki
Department of Computer Science and Engineering
University of California at Riverside
Riverside, CA 92521
E-mail: {fchen,vana}@cs.ucr.edu

***Abstract*— In this paper we propose a technique for scheduling soft real-time multimedia applications in overlay networks. Our scheduling technique consists of a novel distributed and dynamic resource utilization based urgency scheduling algorithm, which exploits the urgency of the tasks and uses monitoring and feedback mechanisms to determine an efficient schedule for the tasks in the system. The algorithm is entirely distributed, uses only local knowledge and scales well with the size of the system. Extensive empirical results validate the performance of our mechanism as a function of the number of tasks, the frequency of feedback propagation and the load on the peers.**

I. Introduction

Overlay networks have gained much attention recently due to their attractive features of self-organization, scalability and decentralized control. Overlay networks (such as peer-to-peer networks) create virtual networks composed of heterogeneous processors with different resource capabilities and variable computation and communication latencies. Nodes in this network employ their own location and routing mechanisms and maintain soft state information about other nodes, in the form of next hop neighbors and resources available at those neighbors. The majority of the applications deployed on peer-to-peer (P2P) networks focus on file sharing or the exchange of objects of small size (such as video objects) [12], [6], [5].

Although such overlay networks have the potential to improve scalability and availability, they introduce an important challenge: *how can we provide Quality of Service (QoS) guarantees to the applications, such as timeliness*. For example, distributed multimedia applications consist of a number of distributed tasks such as delivery of discrete data (such as text, images and control information) or continuous data (such as audio and video streams) among multiple participants. In addition, the audio and video streams often need to be customized such as transcoded to different formats or presentations (e.g., lower resolution) to bring the data to different devices or to meet the particular requirements of the individual users. These pose end-to-end soft real-time and QoS requirements on data transmission, including fast and reliable transfer, and substantial throughput. To support the QoS demands of the distributed applications, the overlay network must be flexible, predictable and adaptable.

In this paper we propose an overlay (P2P) network consisting of service peers (or peers) [1] that are able to provide real-time multimedia services. An example of a service is a transcoding operation. In this network, a node becomes a service peer by establishing a connection to other service peers currently in the network. Key to our approach, and in accordance to the P2P paradigm, is that each service peer has only a partial view of the network. Clients (users) can connect to any of the peers to request multimedia services. The service peer is responsible to accept the multimedia request, perform local computations and propagate a multimedia stream. Our goal is not to find an end-to-end path. Instead we want to schedule the multimedia requests while providing end-to-end soft real-time guarantees to the requests and balancing the load across multiple peers.

The system implements a novel distributed and dynamic Resource Utilization Based urgENcy (RUBEN) scheduling algorithm to meet the end-to-end timing requirements of the tasks. Our scheduling algorithm is an extended version of the Least Laxity Scheduling (LLS) algorithm [7], [11] that assigns priorities to the tasks based on their urgencies. However, in P2P networks, multimedia applications can easily suffer from high-latencies and long communication delays caused by slow machines or wide area networks. Therefore, to meet the end-to-end timing requirements of the tasks, nodes generate and propagate resource utilization feedback to their peers to estimate the queueing latencies of the tasks at remote nodes and distribute the scheduling requests to the peers with the least load, and thus, have the best probability of meeting the tasks' deadlines. RUBEN allows task priorities to be adjusted dynamically as the tasks execute and therefore has the advantage (over other scheduling algorithms that use only a fixed parameter such as the deadline) that it captures the variable delays experienced by the tasks at the peer queues and the underlying communication system.

The importance of providing multimedia applications on overlay networks has been recognized by recent efforts [2], [13], [14], [1]. These follow two main approaches: (1) application-level multicast for content distribution in a group of peers [2], [13], or (2) building overlay proxy networks [16]

[1] In the remaining paper, the term peer refers to the service peer. We will use the terms peers, service peers and nodes interchangeably.

in which proxies provide application-level services. For example, Peercast [14] builds an application level multicast network to broadcast real-time audio and video streams among peers in an unstructured Gnutella-based overlay network. Our approach is different from application-level multicast techniques because our multimedia applications require both communication and processing (e.g., transcoding) at various nodes in the overlay network. Our approach shares similar goals with overlay proxy networks, but it differs in two important ways: (1) it eliminates the need to deploy dedicated hardware for the proxies and is therefore more cost effective, and (2) peers in our overlay network have more dynamic behavior. The advantage is that the overlay network can easily grow into a large population. The challenge is that in such a large-scale network, exact load and resource measurements are difficult to attain at all times, since the state of the network changes much faster than it can be communicated to the peers. Our approach is also more generic than layered streaming schemes [3] which only send partial streaming data for congestion control in network constrained peers.

In the paper we also present an extensive empirical evaluation of the proposed scheduling algorithm. We investigate the accuracy and performance of RUBEN as a function of the number of tasks in the system, the frequency of propagating resource utilization feedback and the load on the peers.

II. The Proposed System

A service peer becomes a member of the overlay network by establishing a direct connection with at least one service peer currently in the network. Each node p is characterized by a $horizon_p$ and keeps a small number of connections to other peers. Each peer consists of a Connection Manager, a Resource Manager and a Local Scheduler. The Local Scheduler implements the RUBEN scheduling algorithm. The RUBEN algorithm is a distributed scheduling algorithm, therefore the local Scheduler uses information from remote peers. This information to the local Scheduler is provided by the Connection Manager and the Resource Manager. The Local Resource Manager monitors the usage on the node's local resources (CPU and bandwidth) at runtime and profiles the behavior of the tasks as they execute. (In Section II-B we describe the local Resource Manager in more detail). The Connection Manager is responsible for managing the connections at the node and maintaining current peer resource utilization information (discussed in Section II-A). Transcoding services require both communication and processing time and therefore can execute only on peers that have enough available computation and communication resources. For example, if the Resource Manager reports that the processor at the peer is overloaded (*e.g.,* over 70% CPU load), then the Connection Manager will propagate the request to other peers.

We model a soft real-time multimedia application as one or more sequential tasks: $T_1, T_2, ..., T_n$. The execution of a task is triggered by a client thread (such as a user requesting a transcoding service) and completes when the client thread finishes execution. A task includes both local computation and communication times. Tasks have *soft* deadlines, where missing a deadline is not catastrophic for the system. Each task t is associated with a $laxity_t$ value that represents a measure of urgency for the task and determines the order with which the task will be executed in the system; the highest priority goes to the task with the smallest laxity value. Tasks are aperiodic, are generated concurrently and asynchronously from multiple client threads and compete for the same computing resources. They carry scheduling parameters from one peer to another and therefore enable a wide area scheduling strategy.

A. Resource Utilization Based Urgency Scheduling (RUBEN)

We propose a novel distributed scheduling algorithm called *Resource Utilization Based urgENcy scheduling (RUBEN)*.

The goal of our scheduling algorithm is to *maximize the number of soft real-time multimedia tasks that meet their end-to-end deadlines.*

To do that, the algorithm distributes the scheduling load to those peers that have the least load and therefore the highest probability of meeting the tasks' deadlines. This is done in a distributed fashion by the nodes generating and propagating resource utilization feedback to their peers. Nodes in our algorithm propagate feedback within a limited range, which essentially serves as hints for the remote peers. This information is used by the nodes to estimate the utilization load at remote peers.

1) Operation of the LLS algorithm: To meet the deadlines of the tasks, RUBEN is based on an extended version of the Least Laxity Scheduling (LLS) algorithm that assigns priorities to the tasks based on their laxity values. RUBEN associates a laxity value with each task which is carried along with the request messages between the peers. The laxity value for task t is computed as $laxity_t = deadline_t - remaining\_computation\_time_t$ and represents a measure of urgency of the task. The computation time includes both the local processing time of the task in each node (which consists of the actual execution times of the task and the waiting time in the Schedulers' queues) and the communication time to invoke a request on a remote node. The task with the smallest laxity value has the highest priority. The initial laxity value of the task is computed as a difference between the deadline and the estimated time for the task to complete and is updated dynamically during the execution.

The LLS algorithm is a dynamic algorithm and therefore can have a high scheduling overhead because of two reasons: (1) the laxity values of the tasks may need to be updated continuously during execution, and (2) the tasks in the Scheduler queues must be reordered when new tasks arrive. In our implementation, to minimize the scheduling overhead, we chose to update the laxity values only when a new task arrives or when the currently executing task finishes or is propagated to a remote peer. This significantly reduces the scheduling overhead with the penalty of delaying the execution of some tasks. Since the execution time of the transcoding tasks is about 120ms (for the transcoding of a GOP of an average size of 60kbytes on a PIII 600MHZ processor), the delay penalty is

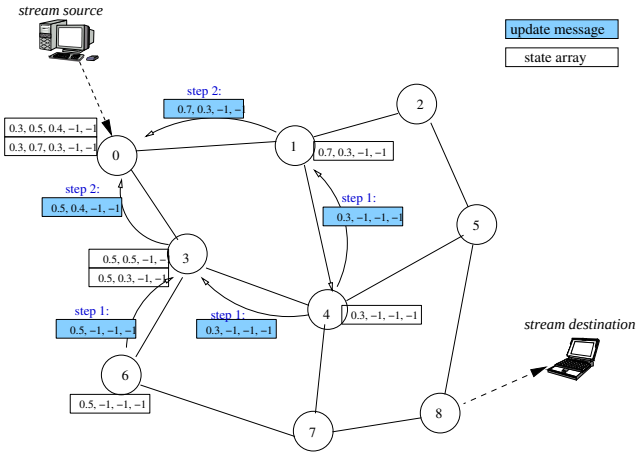

Fig. 1. Example of an overlay network and operation of the resource utilization feedback mechanism.

relatively small. We also measured the overhead of reordering the tasks in the Scheduler's queue. Our experiments indicate that for a task queue size of 10, the average overhead is 691 microseconds and the maximum overhead is 1.5 milliseconds. Both of these values are acceptable for streaming applications. To avoid task starvation, tasks with missed deadlines could be dropped from the system. Otherwise, they could continue execution only if no tasks with positive laxity values are in the local Scheduler's queue.

2) Resource Utilization Feedback: The resource utilization feedback propagation mechanism at each peer works as follows (shown in Figure 1):

- The Local Resource Manager for node p maintains current resource utilization measurements about its peers in a local data structure called $state_p$. This is an array of length l for each of m immediate peers. Essentially, the $state$ array captures resource information up to l hops away (horizon = l). Initially, when a node has no information about its peers, the array is set to infinity.
  The Connection Manager uses an $update$ message to propagate resource feedback to its immediate peers. The first entry in the $update$ array corresponds to the node's current resource measurement (CPU, bandwidth). The $i-th$ entry corresponds to the estimated resource utilization for a horizon i. When the node sends an update to peer k, it excludes the resource measurements that correspond to that particular direction. This is computed as:
  $$update_k[0] = CurrResourceMeasurement$$
  $$update_k[i] = \frac{1}{m-1} \sum_{q=1,..,m\&q!=k} state_q[i-1]$$
  where the $CurrResourceMeasurement$ corresponds to the average CPU utilization and bandwidth estimation and $i = 1, ...l-1$. The node periodically updates its measurements and propagates it to its peers. The resource measurement at horizon i is computed as the average of the measurements of all peers that are i hops away. If an entry is infinity, that entry is disregarded.
- Upon the receipt of an update message from an immediate peer k, node p updates its local $state_p$ data structure with the most recent resource information, as:
  $$state_p[i] = update_k[i]; i = 0, 1, .., l-1$$
- When RUBEN makes a peer scheduling decision, it chooses to send the request at the peer with the least resource utilization, which is indicated by the corresponding $state$ array values. When several peers report similar utilization, choosing any of the peers will suffice.

There are two importance observations. First, the accuracy of the information decreases as the number of hops between the peers increases. To remedy this, we introduce levels of confidence through weights $(w_0, w_2, ..., w_{l-1}), \sum_{i=0,..,l-1} w_i = 1$, where the higher confidence goes to peers one hop away. Assuming p peers, RUBEN estimates the utilization for peer p as: $load_p = \sum_{i=0,...l-1} w_i * state_p[i]$.

Second, there is a trade off between the accuracy of the resource utilization information maintained by the Resource Manager peers and the overhead of the feedback propagation. Sending feedback information too frequently increases the number of messages in the system and can easily lead to link congestion, especially when the resource utilization is estimated too frequently. We therefore introduce a measurement threshold R_t, in which an update message is generated only when $|load_p(t) - load_p(t-1)| > R_t$.

At system initialization, where not enough information is known about the peers, the Scheduler can arbitrarily choose any of the peers. On the other hand, if the $state$ array maintains high utilization values, this is a strong indication that all the processors or network links are overloaded and therefore the scheduling direction would probably not make much difference. Although the end-to-end execution time is greatly influenced by the decisions made at the individual nodes, our scheduling algorithm chooses a path that has enough resources for the execution of the whole task and therefore maximizes the probability that the task timing requirements are met.

B. Resource Manager Measurements

CPU Monitoring. The Local Resource Manager monitors the current CPU load and the memory and disk bandwidth through system calls to the `/proc` interface. Our previous work [11] indicates that such profiling can be done at runtime with less than 1% overhead. Because `/proc` gives us only instantaneous load information, (1) we use exponentially weighted averaging to approximate the mean values of the measurements, with greater weight being given to more recent measurements, and (2) we tune the profiling frequency dynamically to adequately capture the load fluctuations at the peer. The CPU load information is propagated through our adaptive feedback mechanism to remote peers and stored in the Connection Managers' PeerList.

Bandwidth Estimation. The end-to-end execution times of the tasks are also affected by the communication latencies as they propagate across multiple nodes in the P2P system. Existing schemes employ low-level approaches that increase

accuracy at the expense of overhead [10] [4]. In our system, we use a simple scheme similar to the method used by Hicks et al [8] that measures how much data has been sent at a given link. The basic idea is that if all the data has been sent, then the link has at least this available capacity. Otherwise, if two continuous blocking send operations exist, the number of bytes, $datasent(i)$, sent out in the second send operation can be used to compute the current available bandwidth. Assuming a send interval $interval$, we estimate the available bandwidth on the link as $b(i) = \frac{datasent(i)}{interval}$. Note that the above estimation gives us only a lower bound of the available link capacity. There is a tradeoff between the send operation frequency and the accuracy of the estimation. The more frequently the calculation is performed, the faster the value of available bandwidth will converge.

III. Implementation and Experimental Results

To evaluate the performance of our scheduling algorithm we built a P2P system and performed both empirical and simulation experiments. Because of lack of space, we felt it was more important to include the results from our empirical implementation. We run our experiments in Emulab [15] in which nodes are geographically distributed in a wide-area environment. Our topology uses nine nodes that are configured with PIII 600MHZ or PIII 850MHZ processor with 512M memory running RedHat Linux 7.3. The link bandwidth between nodes is specified in the configuration script. Figure 1 shows the topology that we used for our empirical experiments.

A. The Transcoding Application

In the experiments, we used soft real-time transcoding tasks that transcode consecutive video stream units (GOPs in our MPEG streams) in the system [9]. Examples of tasks are changing the bit-rate or adjusting the resolution. The transcoding tasks have two important characteristics: (1) streams must be transcoded on the way from the source to the destination peer in order to be transformed to a suitable format for the destination peer's device; different streams may need different transformations, and (2) GOPs from the same stream must arrive at the destination peer in a timely manner. We used MPEG-1 streams with a playback length of 2 minutes. Each GOP contains 12-13 frames with a playback time of 0.5 second. The deadlines of the tasks were set to about 1.5sec and the laxity values were between 350ms and 850ms.

B. Results

We start to evaluate the effectiveness of our scheduling algorithm by measuring the miss ratio of RUBEN, calculated as the percentage of tasks that miss their deadlines. The miss ratio is primarily affected by: (1) the utilization of the nodes and (2) the accuracy of the feedback information propagated to the peers. As the load on the peers increases, the probability that the tasks miss their deadlines increases. In addition, if the update frequency is low, the peers may not estimate the task queueing latencies accurately.

Effect of CPU load on task miss ratio. In the first experiment, we evaluated how RUBEN compares to other well-known scheduling algorithms such as LLS (without the feedback information propagation) and the Earliest Deadline First (EDF assigns the highest priority to the task with the earliest deadline). It is important to mention that EDF is primarily being used in local scheduling environments, so it is not directly comparable with RUBEN. The reason we evaluated it is because we wanted to test how well EDF would provide real-time support to multimedia applications in the P2P system. Figure 2 demonstrates the miss ratio of RUBEN, EDF and LLS as a function of the number of tasks in the system. To concentrate on the CPU load, for this experiment we set each link bandwidth to be 100 Mbit. Clearly, RUBEN outperforms the others in a consistent manner. For example, for 24 number of tasks, EDF and LLS miss 30.3%, and 17.6% tasks respectively, while RUBEN has a much smaller miss ratio 8.6%. However, as the system becomes more and more overloaded, RUBEN reaches LLS's miss ratio because the feedback mechanism has a smaller influence on how the tasks should be distributed.

Effect of feedback frequency to task miss ratio. Since the task distribution is affected by the propagation of feedback information, we also explore how the different propagation frequencies affect the task miss ratio. In this, we propagate new feedback only when the difference in the resource utilization measurements is above or below a threshold R_t. In Figure 3, we show the results for three threshold values: $R_t = 0.05, 0.10, 0.20$. We observe that the $R_t = 0.05$ and $R_t = 0.10$ curves show similar results with a maximum difference of 4% (for 24 tasks). However, for the $R_t = 0.20$, the difference is higher. For example, for 24 tasks, $R_t = 0.05$ missed 8.6% of tasks, but $R_t = 0.20$ missed 18.3% of tasks, which is more than double of the former. The reason is that, as we expected, a smaller threshold value can better capture the CPU load change and enable the peers to keep more accurate information.

Effect of feedback frequency to message overhead. We also measured the message overhead as the average number of messages sent out by each node per minute. In Figure 4 we notice that the message overhead initially increases sharply and then increases more slowly. The reason is that when more tasks are generated, almost all the nodes equally contribute their resources to the tasks and as a result the system experiences fewer load fluctuations. In addition, the figure illustrates that smaller threshold values cause more messages: with $R_t = 0.05$, each node generates about 70 messages per minute on average, while the same numbers for $R_t = 0.10$ and $R_t = 0.20$ are around 60 and 40 messages per minute, respectively. Note, that these numbers may be too large for some nodes in a typical P2P system.

Effect of different processors to CPU load. During the experiments we also noticed that the relative gain of RUBEN depends on the types and speed of the processors used by the peers. For example, in Emulab, all nodes have similar speeds (PIII 600MHZ and 850MHZ). Therefore, they get overloaded

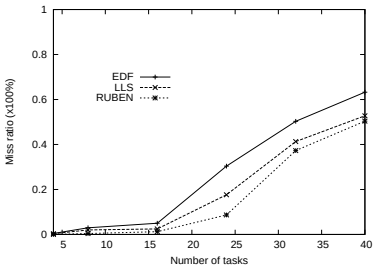

Fig. 2. Effect of CPU load on task miss ratio.

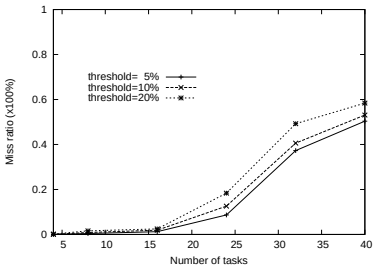

Fig. 3. Effect of feedback frequency on task miss ratio.

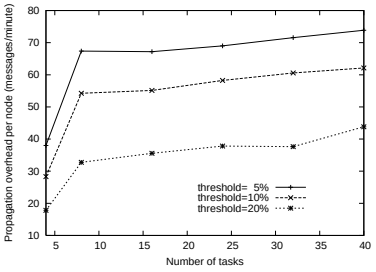

Fig. 4. Message overhead under different threshold values.

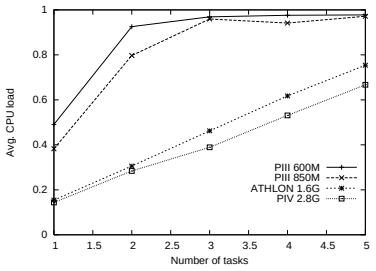

Fig. 5. Average CPU load of heterogeneous CPUs.

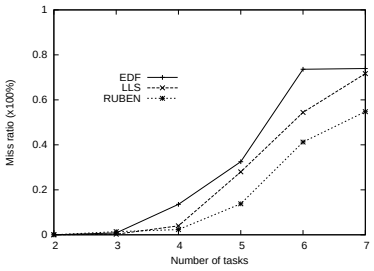

Fig. 6. Effect of bandwidth estimation on task miss ratio.

quickly with even a smaller number of tasks. If we assume heterogeneous CPUs as they exist in today's P2P systems, the gain is expected to be even larger. Figure 5 illustrates how different CPUs contribute to the average CPU load.

Effect of network bandwidth on task miss ratio. We also evaluated the effect of the estimated network bandwidth on the task miss ratio. For this experiment, we used the same topology but we set the link bandwidth between nodes to range from 1Mb to 4Mb. Figure 6 shows that the system can support a smaller number of tasks compared to the previous experiments, before it becomes overloaded and most of the tasks miss their deadlines. The benefit of RUBEN compared to EDF and LLS is more obvious in this case (as it estimates the available bandwidth) and increases with the number of tasks. For example, when the task number is 7, EDF has a miss ratio of 73.9%, LLS 71.6%, while RUBEN only 54.7%.

IV. Conclusion

In this paper we have proposed a novel scheduling architecture, *RUBEN*, that uses current resource utilization based measurements to provide real-time guarantees to multimedia tasks. RUBEN exploits the urgency of the tasks and the load on the peer resources to maximize the number of tasks that meet their timing requirements and determine an efficient schedule for the requests. Our mechanism is fully distributed, uses only local knowledge and scales well with the size of the system. Our empirical results validate our mechanism and show that it provides real-time guarantees to distributed applications.

Acknowledgments. We would like to thank Laxmi Bhuyan for many constructive discussions on this work. This work has been supported by NSF award 0330481.

References

[1] S. Banerjee, C. Kommareddy, K. Kar, B. Bhattacharjee, and S. Khuller. Construction of an efficient overlay multicast infrastructure for real-time applications. In *Proc. Infocom*, 2003.

[2] Y. Chu, S. Rao, S. Seshan, and H. Zhang. Enabling conferencing applications on the internet using an overlay multicast architecture. In *ACM SIGCOMM*, 2001.

[3] Y. Cui and K. Nahrstedt. Layered peer-to-peer streaming. In *13th International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV 2003)*, 2003.

[4] J. Curtis and A. McGregor. Review of bandwidth estimation techniques. In *New Zealand Computer Science Research Students' Conference*, 2003.

[5] F. Dabek, B. Zhan, P. Druschel, J. Kubiatowicz and I. Stoica. Towards a common api for structured peer-to-peer overlays. In *IPTPS*, 2003.

[6] A. Delis, V. Kalogeraki, and D. Gunopulos. Peer-to-peer architectures for scalable, efficient and reliable media services. In *IPDPS*, 2003.

[7] M. L. Dertouzos and A. K.-L. Mok. Multiprocessor on-line scheduling of hard real-time tasks. In *IEEE Transactions on Software Engineering, vol. 15, no. 12 (December 1989), pp. 1497-1506*.

[8] M. Hicks, A. Nagarajan, and R. V. Renesse. User-specified adaptive scheduling in a streaming media network. In *IEEE OPENARCH*, 2003.

[9] T. F. Homepage. http://ffmpeg.sourceforge.net/.

[10] M. Jain and C. Dovrolis. End-to-end available bandwidth: Measurement methodology, dynamics, and relation with tcp throughput. In *Proceedings of the ACM SIGCOMM Conference*, 2002.

[11] V. Kalogeraki. Resource management for real-time fault-tolerant distributed systems. In *Ph.D. Thesis, Department of Electrical and Computer Engineering, University of California, Santa Barbara*.

[12] D. Milojicic, V. Kalogeraki, R. Luksoe, K. Nagaraja, J. Pruyne, B. Richard, S. Rollins, and Z. Xu. Peer-to-peer computing. In *HP Technical Report, HPL-2002-57*.

[13] D. Pendarakis, S. Shi, D. Verma, and M. Waldvogel. Almi: An application level multicast infrastructure. In *USENIX Syposium on Internet Technologies and Systems*, 2001.

[14] The Peercast Project. http://www.percast.org.

[15] B. White, J. Lepreau, L. Stoller, R. Icci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar. An integrated experimental environment for distributed systems and networks. In *OSDI*, 2002.

[16] D. Xu and K. Nahrstedt. Finding service paths in a media service proxy network. In *SPIE/ACM MMCN*, 2002.