An Adaptive Protocol for Locating Programmable Media Gateways

Wei Tsang Ooi
Department of Computer Science
Cornell University
weitsang@cs.cornell.edu

Robbert van Renesse
Department of Computer Science
Cornell University
rvr@cs.cornell.edu

ABSTRACT

We describe a new control protocol called Adaptive Gateway Location Protocol (AGLP). In this protocol, a client requests a computation on a multimedia stream. AGLP discovers programmable Internet servers that process multimedia streams, and assigns the computation to one of these so-called *gateways*. AGLP continuously searches for alternate gateways, and, transparent to users, migrates computations between them to improve efficiency. The AGLP protocol uses soft-states for robustness and scale. Simulation results support that our protocol quickly locates gateways and migrates computations while keeping the load on the network low. We also outline planned enhancements to AGLP.

1. INTRODUCTION

There is a growing interest in adding multimedia processing capabilities into the network. For example, active services such as MeGa [1] allow an application-level gateway to transcode multimedia streams into lower-bandwidth streams suitable for slow links, while still allowing the senders to send high quality high bandwidth streams to other well-connected receivers. Here at Cornell University, the Degas project [10] extends the model of MeGa, by allowing receivers to upload a program into a gateway to customize the processing of RTP media streams. Examples include creating a picture in picture effect by merging two video streams, or switching between different streams automatically based on audio signals. With multiple gateways running in the network the question arises as to which gateway should be chosen to run such a program.

Running the program in a gateway that is strategically located in the network could use network bandwidth more efficiently. For example, if the output video stream has lower bandwidth than the input stream, then the program should be run on a gateway that is close to the sender. On the other hand, if the program outputs a higher bandwidth stream, the program should be run close to the receiver.

The problem of determining the best gateway is an optimization problem. However, the dynamic nature of the network prevents us from solving the problem using a centralized, combinatoric algorithm. Senders and receivers may join and leave video sessions, new gateways may be added and deleted, and the underlying network behavior changes continuously. Therefore, we opt for a distributed, adaptive algorithm in Degas.

In this paper, we present the Adaptive Gateway Location Protocol (AGLP) used in Degas for choosing a gateway that efficiently utilizes bandwidth. Although we design AGLP to work with Degas, we believe that it can be modified to suit other applications as well. AGLP is a soft-state protocol based on the announce-listen model widely used in MBone tools. The simplicity of the model allows us to build a scalable, robust protocol that is resilient to crashes and message loss. AGLP adapts to changing network conditions, as well as the birth and death of gateways, senders, and receivers, by migrating computations (also called *services*) between gateways. An additional requirement on our protocol is that it assigns a new service to a gateway rapidly.

We designed our protocol to be compatible with existing MBone tools. No changes are required at the senders. This means that traditional MBone tools such as vic [7] and ivs [13] can be used as the video sending application. This makes it possible to deploy our protocol without affecting the existing MBone community.

Our simulations support that AGLP achieves its goals of rapid assignment and adaptive placement, while keeping the load on the network low. The rate of migrations is small, and a good gateway for such a migration can be selected within a minute.

The rest of this paper is organized as follows. We describe the AGLP protocol in Section 2. We analyze the performance in Section 3. In Section 4, we discuss improvements to AGLP we plan to make. Related work is described in Section 5 and we conclude our paper in Section 6. Please note that an in-depth discussion about Degas is out of the scope of this paper. Therefore certain details about the gateways have been omitted to simplify the presentation. Interested readers should refer to [10] for a full description of the Degas system.

2. PROTOCOL DESCRIPTION

Before we describe our protocol, we present the symbols and terminology used in our description:

- g is a well-known multicast channel used for exchanging control messages among the gateways, and between the gateways and client. Every gateway and client listens to g.
- s is a multicast session.
- P is a program that specifies an input session s and the processing to be done on video streams from s.
- C is the client that requests some processing to be done on video streams.
- $G_0, G_1, .., G_m$ are gateways available for running a program requested by C. One of the gateway will be selected to service C. Without loss of generality, we let G_0 be the current gateway servicing C.
- $S_0, S_2, .., S_k$ are video senders participating in video session s. These senders can be normal MBone video sources. They need not be aware of the existence of the gateways or C.

For simplicity, we assume that each client can submit only one program at a time, and each program can read from only one session. We also assume that all participants run the network time protocol NTP [8], which we rely on to measure the propagation delay of a packet.

Our protocol consists of two phases (see Figure 1). The first phase, *Quick-Start Phase*, chooses a gateway G_0 that is close to C, without worrying about optimizing bandwidth utilization. The second phase, the *Adapting Phase*, optimizes the bandwidth utilization by migrating services to a better gateway. We describe these two phases in Section 2.1 and Section 2.2 respectively.

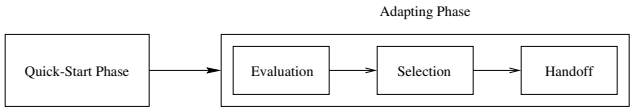

Figure 1: Different phases in the AGLP Protocol.

2.1 Quick-Start Phase

There are two reasons why the Quick-Start Phase is necessary. First, we want to reduce the start-up latency experienced by the user. Secondly, we do not have any knowledge about the behavior of the program requested by the client, nor do we know anything about the session (such as the identity of the senders, and bandwidth of incoming video streams). The gateway we select at the Quick-Start Phase serves as a temporary gateway. This gateway collects information so that further optimization can be done. The Quick-Start Phase works as follows (see Figure 2).

The client C who wants to request some processing to be done on the gateway first multicast a request message onto the common multicast channel g. A gateway G_i that receives the request message and is available to serve C replies

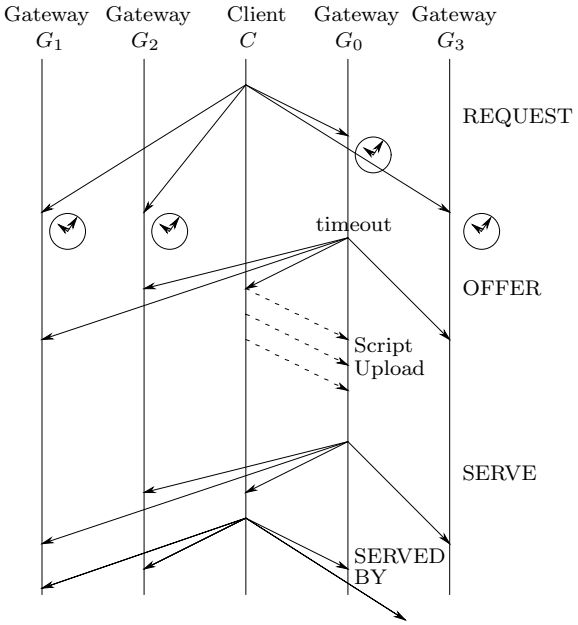

Figure 2: The Quick-Start phase of AGLP.

with a offer(C) message. However, instead of replying immediately, we employ a technique commonly known as Multicast Damping to reduce the number of offer messages received by C. Each G_i waits for time $T_{\text{offer,i}}$ before multicasting the offer onto g. Moreover, a gateway will suppress its offer(C) message if it has received an offer(C) message from another gateway while waiting.

Client C listens to g and accepts the first offer that it receives. Without loss of generality, let the first offer that C receives be from gateway G_0. C subsequently creates a TCP connection with G_0 at port p, where p is a port number embedded in the offer(C) message. Subsequent offers from other gateways will be ignored by C.

C sends the necessary information needed for processing to G_0 using the TCP connection. This includes the multicast address of the input session, s, the multicast address of the output session, s', and a program that specifies how to process the incoming video streams.

After G_0 has received all the necessary information, G_0 joins the session s, processes incoming video streams, and multicasts the output onto channel s' (see Figure 3). C listens to channel s' to receive the post-processed video it requested. At this point, we enter a state where gateway G_0 is serving client C. G_0 and C periodically announce this relationship onto g. Every T_{serve} seconds, G_0 announces a serve(C) message onto g. Similarly, C sends a served-by(G_0) message to G_0 every $T_{\text{served-by}}$ seconds.

The receipt of serve(C) message by C indicates that the Quick-Start Phase has completed successfully. If C does not receive any serve(C) message in a period of length T_{request}, C will restart the whole process by sending another re-

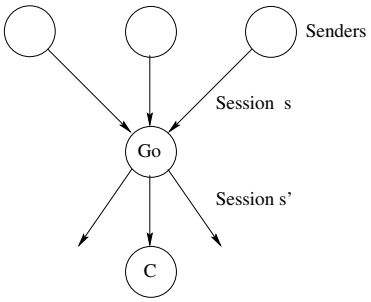

Figure 3: Gateway G_0 listens to session s and receives video streams from the senders. G_0 processes those streams and sends the result out onto session s', on which C is listening.

quest message. Otherwise, the Quick-Start is successful, and AGLP proceeds to the next phase.

2.2 Adapting Phase

During the adapting phase, a service for C may be migrated from the current gateway G_0, to another more suitable gateway, as more information about the session is discovered and changes in the environment are detected. The adapting phase consists of three stages: evaluation, selection and replacement (see Figure 4 for an example). In the evaluation stage, each gateway evaluates itself against G_0 to check if it is more suitable than G_0 for serving C. Once a gateway determine that it can serve G_0 better, it will notify G_0. G_0 periodically runs a selection process, to select the best alternate gateway. Once a replacement G_r is chosen, G_0 hands-off the service for C to G_r. We explain these three stages in greater detail in the following subsections.

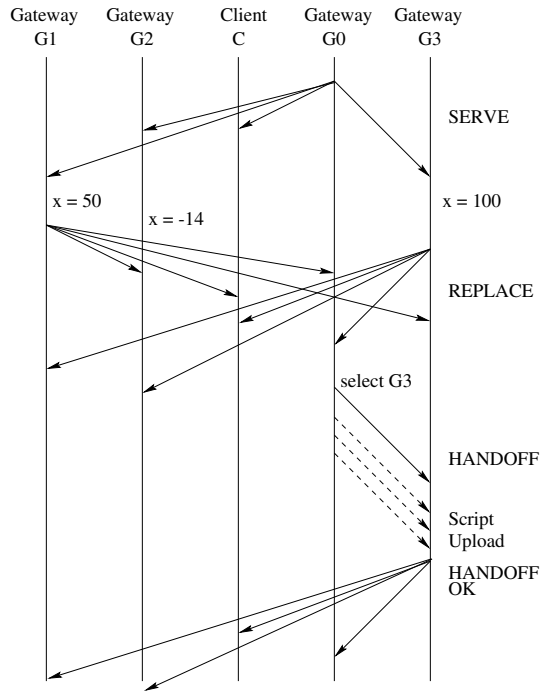

Figure 4: The Adapting phase of AGLP.

2.2.1 Stage 1: Evaluation

We first introduce a few variables that corresponds to the criteria used to perform evaluation:

- b_i: the bandwidth of video stream from sender S_i
- b_C: the bandwidth of the output video stream
- $d_{i,j}$: the distance between gateway G_i and sender S_j
- $d_{i,C}$: the distance between gateway G_i and client C

We now describe how this information is collected and how the evaluation is performed.

After joining session s, G_0 starts to collect information about the current session. This information includes the identity of the senders in the session, and bandwidth of the input streams and the output stream, and the distance (or latency) $d_{0,j}$ from each sender S_j. The identities and distances can be learned from RTCP [12] packets, while the bandwidth information can be gathered by simply counting the packets as they are being processed. This information is included in the serve messages and multicast onto group g.

Each gateway G_i, that is available to serve C, maintains a table of distances to itself from the sources, $D_{self} = d_{i,0}..d_{i,k}$. This table is maintained as soft-states, and is refreshed by periodically joining session s, and listening for RTCP packets. A distance can be calculated by subtracting the NTP timestamp of a sender's report from the arrival time.

Each gateway, upon receiving a serve(C, s) message from G_0, starts the evaluation test to compare the suitability of serving client C. The test produces a *score*, x_i. This score is calculated as follows. First, let U_i be

$$U_i = \sum_j (b_j \times d_{i,j}) + b_C \times d_{i,C}$$

Intuitively, U_i corresponds to the bandwidth utilization. We calculate x_i as

$$x_i = U_0 - U_i$$

A score $x_i > 0$ indicates that G_i is better than G_0 for serving C. Each gateway with a score larger than ϵ will try to replace the current gateway. We choose a threshold ϵ instead of 0 for two reasons. First, a score between 0 and ϵ indicates that the gateway is only slightly better than G_0. The small improvement that we gain is not worthy of the overhead caused by the replacement process. Second, by ignoring gateways that are only slightly better, we can avoid unnecessary oscillation caused by small changes in network conditions. In the next subsection, we discuss how a replacement is selected by G_0.

2.2.2 Stage 2: Gateway Replacement

After evaluation, each gateway with a score larger than ϵ will notify the current gateway, and wait for a reply. This process is similar to the Quick-Start Phase. Again, we use Multicast Damping for scalability reasons. The gateways start a timer and wait for $T_{\text{replace},i}$ seconds. When the timer expires, gateway G_i multicasts a replace(C, x_i) message onto

g. If G_i receives another replace(C,x_j) message from another gateway G_j and $x_j > x_i$, then G_i suppresses its own replace message.

The current gateway keep tracks of the gateway with the lowest score so far, which we call the replacement gateway G_r. T_{adapt} seconds after G_0 receives the first replace message, gateway G_0 unicasts a message handoff(C,p), to G_r. G_0 then establishes a TCP connection to G_r at port p through which G_0 sends the program and input session address s to G_r. G_r subsequently starts the service, and multicasts a handoff-ok(C,G_0,s'') announcement, where s'' is a new multicast address where the processed media stream is going to be sent.

2.2.3 Stage 3: Service Handoff

G_r joins session s, starts processing the input video streams, and sends the output onto session s''. G_r also begins the periodical announcement of serve(C, s'') messages.

At this stage, both G_r and G_0 are providing service for C. Upon receiving both handoff-ok(C,G_0,s'') and serve(C,s''), C knows that another more suitable gateway has been found and this new gateway is ready to serve C. C can now switch from group s' to group s''. C stops announcing served-by(C,G_0) and starts announcing served-by(C,G_r). G_0 stops processing video streams from s eventually after no served-by(C,G_0) is received for T_{bye} seconds.

We provide a summary list of messages involved in this protocol in Table 1.

3. ANALYSIS AND SIMULATION

In this section we evaluate our protocol. In particular, we want to confirm that our protocol satisfies two desirable properties:

- robustness:
  - a gateway eventually runs the service requested by a client;
  - all services are eventually terminated when no client is listening;
  - the service is eventually moved to the optimal gateway.
- scalability:
  - as the number of gateways increases, the number of states maintained and the number of messages exchanged does not increase significantly.

3.1 Robustness

We achieve robustness by maintaining only soft-states which are periodically forgotten and need to be refreshed. Soft-state protocols are used in many light-weight protocols in MBone applications such as SDP [6] and RTCP [12]. Failure recovery is automatic in soft-state protocols, since the failure of a gateway or network link will cause refresh messages to be lost and states to be forgotten. Refresh messages in AGLP include serve and served-by—we illustrate how they support failure recovery by describing two scenarios below.

- Suppose that the gateway that is serving C crashes. The periodic serve message will cease and C will eventually forget that some gateway is servicing it. C will start requesting service again by entering the Quick-Start phase.
- Suppose that the message handoff-ok is lost on its way to C. C will not switch to the new gateway. Even though the new gateway has started serving C, it will not receive a served-by message from C. The new gateway will eventually timeout after T_{bye} seconds, and end its service.

We simulated AGLP in networks with up to 50% loss rate. Although this caused somewhat longer start-up/handoff latencies and redundant requests, the protocol still worked correctly.

3.2 Scalability — Memory Requirements

We envision that the number of gateways running in the network $|G|$ will be large (up to thousands), and the number of clients requesting service to be in the same range. The number of senders per client, $|S|$, however, is expected to be small (say, less than 10). Similarly, because the processing requested by client could be computation intensive, we expect the maximum number of clients that can be served by each gateway, $|C|$, to be small as well.

Each gateway maintains the following soft-states:

- A list of clients it is currently serving;
- The gateway with the best score so far;
- A table that records the distance to all senders for each client it serves;
- A table that records the bandwidth of all input streams and output streams for each client it serves.

On the client side, the only soft-states that are maintained are the sessions to listen to, and the gateway currently serving the client.

The size of the state maintained in the gateway is thus $O(|S| \times |C|)$, and is $O(1)$ for the client. Since a gateway does not keep state for every other gateway, and both $|S|$ and $|C|$ are expected to be small, our protocol is scalable in terms of memory size.

3.3 Scalability — Networking

Multicast Damping is a widely used technique to improve scalability in one-to-many protocols (*e.g.*, it is used in IGMP [4] and SRM [5]). As described in Section 2, we use Multicast Damping for the request-offer and serve-replace message exchanges to avoid implosion of messages. The effectiveness of this technique, however, depends heavily on the time-out values chosen, T_{offer} and T_{replace}. Even though there is extensive work done in analyzing the effect of timers in Multicast Damping (see, for example, [5] and [9]), there are some unique requirements for our timers. T_{offer} should be proportional to the distance from the client, so that the first

request() A request for service by a client.

offer(C, p) A response to a request message from client C. Indicates that the sending gateway is available to serve C. C should contact this gateway at port p for details.

serve(C, S, D) The sending gateway is currently running a service for C. S is the list of session members, D is a vector containing distances from each member in S as well as the distance from C.

served-by(G) Response to the gateway serving C to notify that C is still listening to output from G.

replace(C, x) Notify others that the sending gateway is more suitable for serving C. x indicates how much better the sending gateway is.

handoff(C, p) Message from the current gateway to G' to indicate that G' has been chosen to replace the current gateway for serving C. G' should listen to port p for service specification.

handoff-ok(C, G, s') Announcement from a new gateway G' that it is ready to replace G to serve C. s' is the new multicast address where the output from the service will be sent.

Table 1: A summary of message types and their contents in AGLP

reply received by the client comes from the gateway that is closest to the client. For T_{replace}, the timer value should be inversely proportional to the score of a gateway. We discuss these two parameters in this section.

In order to evaluate the performance of AGLP under these parameters, we simulate our protocol using the ns2 network simulator and run it on a 500-node topology generated using the gt-itm toolkit [2]. We place gateways and the client at random locations in the generated network.

In AGLP, we set the value of T_{offer} to $k \times d$, where k is a constant and d is the propagation delay between gateway and client, measured using an NTP timestamp embedded in the request message. A small value of k results in a lower start-up latency, but a larger number of duplicates. The number of duplicates also depends on the distribution of gateways in the network. If gateways are sparsely distributed, then the number of duplicates increases.

We tried different values of k in our simulations. In Figure 5 we show the average number of duplicate offer messages received by the client for different values of k in cases where the number of gateways G is either 50, 100, or 200. A value of $k \geq 2$ causes the number of duplicates to stay below 3 even as the number of gateways increases up to 200. Figure 6 shows the latencies that the client experiences.

We conclude that $k = 2$ works well in reducing the number of duplicates while keeping the start-up latency within a reasonable time. In Figures 7 and 8 we show the behavior of Multicast Damping as a function of the number of gateways in more detail, along with a 95% confidence interval for each measurement. Our experiments indicate that AGLP scales well for $k = 2$. In the remaining experiments we are using this value for k.

We set the value of T_{replace} to k'/x, where x is the score. In Figure 9 we show the number of duplicate replace responses as a function of k'. We see that for $k' > 500$ the number of duplicates is under 10, which we consider acceptable. Figure

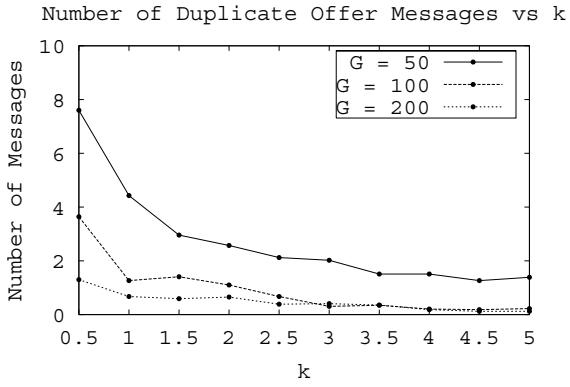

Figure 5: Duplicate offer messages for different values of k and G (the number of gateways).

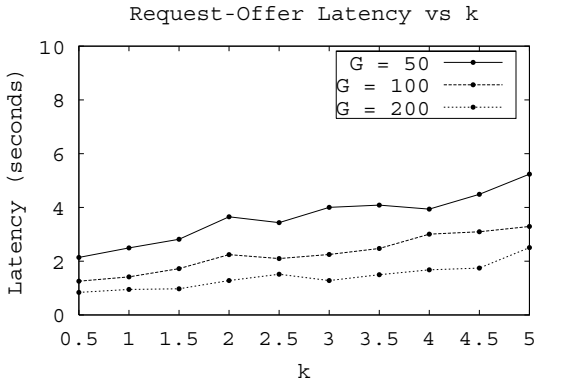

Figure 6: The delay between sending a request and receiving the first offer.

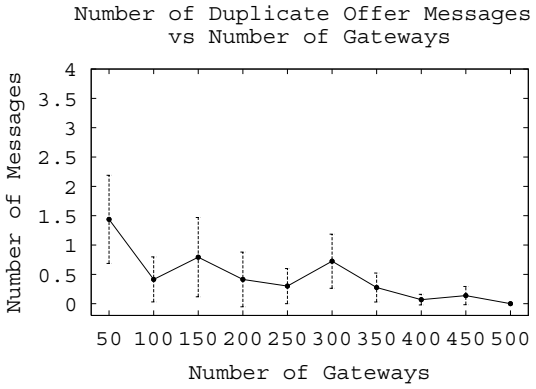

Figure 7: Duplicate offer messages for $k = 2$.

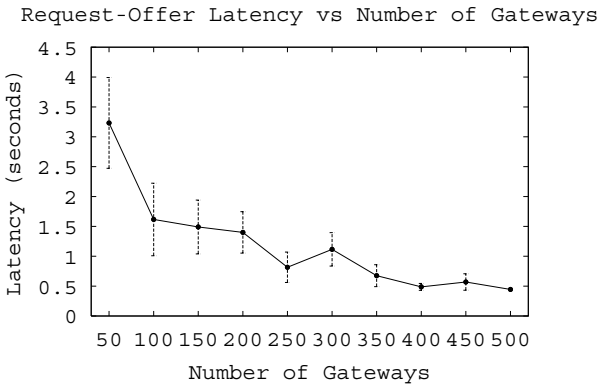

Figure 8: The delay between sending a request and receiving the first offer for $k = 2$.

10 shows that the average number of migrations before a service reaches the optimal gateway goes down with k'. We were surprised by this result. After all, as k' goes up, it becomes less likely that the client will receive a response from the optimal gateway within T_{adapt}. However, after further consideration we are able to explain this.

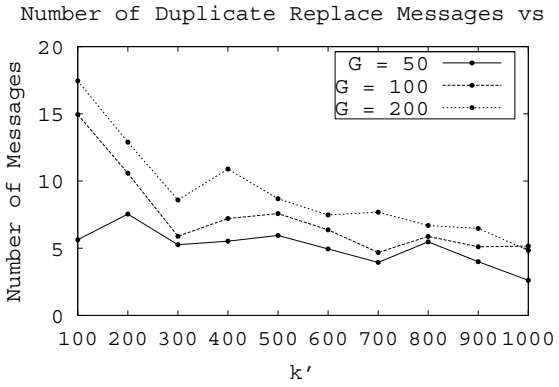

Figure 9: Duplicate replace messages.

After the current gateway gets the first replace response, it waits T_{adapt} seconds before selecting a gateway to hand-off to. That is, after sending the last serve message, it waits

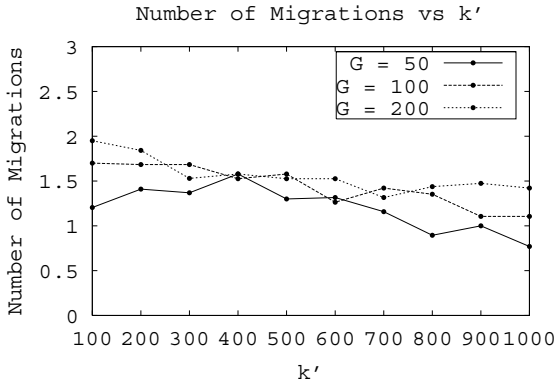

Figure 10: Number of migrations needed to migrate to an optimal gateway.

a total of $RTT_1 + k'/x_1 + T_{\text{adapt}}$ seconds, where RTT_1 is the round-trip time to the first responding gateway, and x_1 is the score at that gateway. In order for the optimal gateway's response to be received in time, we need to have the following condition (see Figure 11):

$$RTT_{optimal} + \frac{k'}{x_{optimal}} < RTT_1 + \frac{k'}{x_1} + T_{\text{adapt}}$$

We can rewrite this as:

$$\frac{RTT_{optimal} - RTT_1 - T_{\text{adapt}}}{(\frac{1}{x_1} - \frac{1}{x_{optimal}})} < k'$$

Thus, the larger k', the more likely that the optimal gateway responds in time, as reflected in Figure 10.

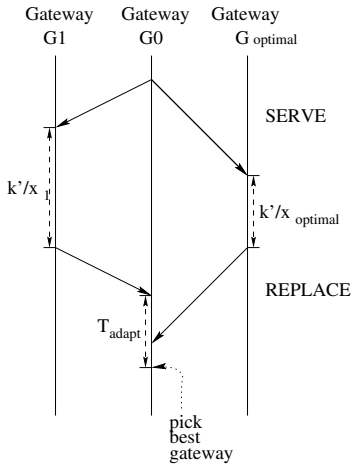

Figure 11: Exchanges of serve and replace messages.

In the following experiments, we use $k' = 1000$ as a conservative choice. For this value of k', we find that there are no more than 8 replace messages received (see Figure 12) even if we run a gateway on all 500 nodes in the network. There were at most two migrations in all runs of our simulations (see Figure 13 for averages and 95% confidence intervals). In Figure 14 we show how this translates into time. On average, all services were migrated to the optimal gateway within 60 seconds, which we find acceptable.

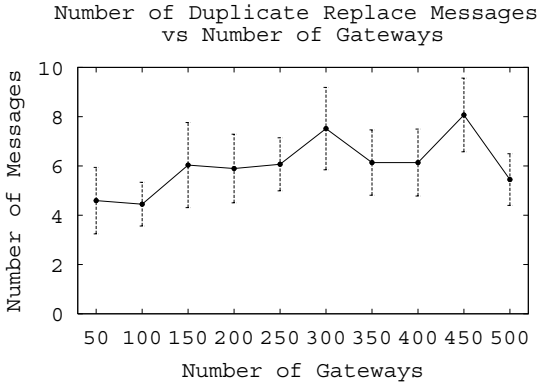

Figure 12: Duplicate replace messages received by a gateway for $k' = 1000$.

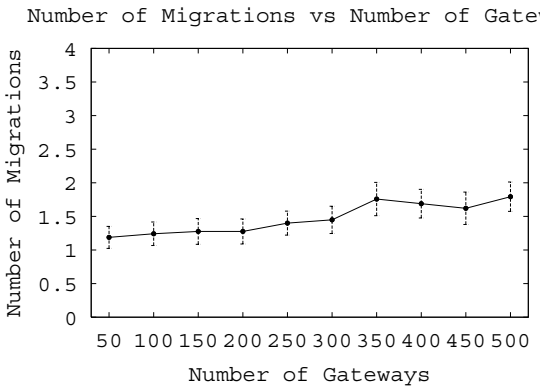

Figure 13: Number of migrations to migrate to an optimal gateway for $k' = 1000$.

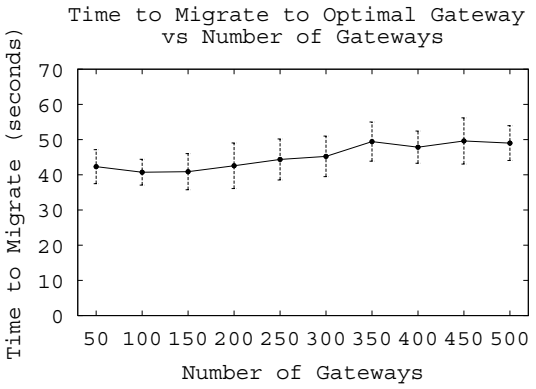

Figure 14: Time to migrate to an optimal gateway for $k' = 1000$.

4. ENHANCEMENTS TO AGLP

Our current implementation of AGLP, as described above, has many restrictions. For example, we assume that each service reads from only one multicast session, and outputs to another session. We also assume that each client can request one service at a time. As described below, we plan to relax these restrictions.

4.1 Multiple Receivers

Although so far we have assumed that the client is the only one who benefits from the service provided by the gateways, we can easily allow multiple clients to receive the post-processed streams from gateways. Since the post-processed video stream is multicast onto session s', any host that is interested in the post-processed stream can tune in to session s' to receive the stream. This can be done as follows.

We will augment the Session Description Protocol to include information about services currently provided by the gateways. A host can view the list of services available using a GUI front end, and join any session that it is interested in. The host will periodically announce served-by messages onto the common multicast channel g. The served-by messages from multiple receivers can be consolidated by using Multicast Damping: if a receiver R receives a served-by message, then R reschedules the announcement of its own served-by message. This reduces the total number of served-by message sent. If the original client C that initiated the service quits, the gateway will continue serving the other receivers as it is still receiving served-by messages.

Two problems arise. First, what if the gateway servicing the receivers fails after C quits? The other receivers do not have access to the original program submitted by C, and therefore cannot restart the service. One possible solution to the first problem is to have each receiver download the program from the gateway (if C permits it) as they join session s'. Another solution is to let the gateway periodically multicast the programs onto a separate channel.

The second problem concerns the calculation of scores during evaluation. Since the output from the gateway is now multicast to multiple receivers, how can we characterize the bandwidth utilization of the output stream? We can estimate the propagation delay from the gateway to all receivers by using receiver report RTCP packets, but since bandwidths are shared in the multicast tree, we cannot simply sum the products of the bandwidth and the distances. We plan to investigate both problems further.

4.2 Composable Services

So far we have tacitly assumed that each client requests service from only one gateway. We can extend AGLP to allow multiple services to be requested by a client. An interesting consequence of this is that the client can submit multiple programs that can be composed to perform a task.

For example, a client would like to create a "Quad Splitter" view of four video streams from some session s. One approach is to submit a program that says "take these four streams, scale each of them down by half, and arrange them to create a quad-splitter view." If the four video sources are located far from each other, this program is best run at a

gateway somewhere in the middle of the four sources. However, if a client can request multiple services, then a better way to create a quad-splitter view is to write five programs: each of the first four programs reads from one video sender, scales it by half, and sends the scaled video out to a new session. The fifth program reads from the output sessions of the first four programs and creates the quad-splitter view. Our adaptive protocol will cause the scaling processes to be performed near the senders, resulting in more efficient use of bandwidth (see Figure 15)

Several modifications to AGLP are needed to support this. First, a gateway G needs to know whether it is receiving data from another gateway G'. This is needed so that when a service on G' is migrated to another gateway, G can detect the handoff and switch its input session to the output session of the new gateway. The client can indicate this information to G in the program uploaded to G. G can then pay attention to any handoff message from G', and switch accordingly.

Secondly, a gateway must be able to receive and process streams from multiple sessions. We have been assuming that a service reads multiple streams from a single session, and output to one session. This is inadequate if we want to allow composable services. For instance, say gateway G is receiving streams from two gateways G' and G''. Initially G' and G'' can send their outputs to a shared session, which G can listen to. As services on G' and G'' migrate to other gateways, G will have to listen to two different sessions to receive its inputs.

The third modification needed concerns failure recovery when a gateway G fails. One way to recover from the failure is to restart only the service that ran on that gateway. However, the client needs to maintain consistent information about where each of the gateways receives its input from and where they send their outputs to, so that the client can modify its program to indicate the new input or output session. Maintaining consistent information is hard because our protocol uses soft-states. We believe that a better solution is to let the client restart all services from scratch. Even though this is inefficient as it will cause all living gateways to stop running their service for C, this does provide a quick recovery from failures.

4.3 Load Balancing

We have implemented a simple method to balance the loads on gateways. Only a gateway with load lower than a certain threshold is eligible to offer services to a client. Ideally, we should take the available resources of a gateway into consideration. We should include metrics such as available CPU time, available memory, or the availability of special multimedia hardware into our evaluation function. However, it is not clear how to integrate these different metrics into one variable in order to decide which gateway is more suitable to service a particular client.

5. RELATED WORK

Many techniques that we use in AGLP are already widely used in the network community, especially in the MBone tools. For example, announce-listen based soft-state protocols are used in the Active Service Control Protocol (ASCP) [1], the Internet Group Management Protocol (IGMP) [4], the Session Description Protocol (SDP) [6], and RTCP [12]. Among these protocols, ASCP is the closest to our work. ASCP is used to locate an active server in the network to perform a specified transcoding. However, ASCP does not adapt to network conditions, and the transcoding may not occur on a server that is strategically located. This may result in inefficient utilization of network bandwidth.

Other protocols for locating services existed. For example, DHCP [3] uses a centralized server at a known location to provide information about the location of the local DNS servers. DHCP is intended for local area networks only – DHCP does not scale, and its centralized design makes it vulnerable to crashes. SLP [14] uses another approach, where each server periodically announces the availability of services to a well-known multicast channel. A client who requires some service listens to the multicast channel to discover the services available. This approach is designed for local area networks, and suffers from a scalability problem when a large number of servers are available. [11] describes a wide-area version of SLP for locating Internet Telephony Gateway, but this work does not take network bandwidth into consideration.

MeGaDiP (Media Gateway Discovery Protocol) [15] uses centralized directory agents called *dealers* to find media gateways located along the end-to-end path between two end hosts. An end host contacts a local dealer to find a gateway. If no gateway is available, the dealer forwards the request to another dealer along the end-to-end path. List of dealers along the path are obtained using traceroute and modified DNS lookup. While both AGLP and MeGaDiP try to minimize network traffic, AGLP is distributed, does not require changes to DNS, and can support multiple end hosts.

The Conductor system [16] allows adaptors to be deployed at key locations in the network to adapt data flows to changing network conditions. Conductor uses a centralized algorithm to decide on deployments of adaptors into strategic locations in the network. The Conductor may produce a non-optimal plan because the complexity of calculating an optimal plan in a centralized location is prohibitive. We use a distributed, adaptive scheme that does not suffer from this problem. Conductor adapts TCP streams, but does not support the connection-less, RTP-based multicast packets that AGLP supports.

6. CONCLUSION

In this paper, we present an adaptive control protocol called AGLP for running services on media processing gateways in the Internet. Our protocol supports the following functionality:

- allowing the client to request a service, and submit media processing program to a gateway;
- deciding which gateway should be used to perform a service;
- migrating services to more suitable gateways (adaptability).

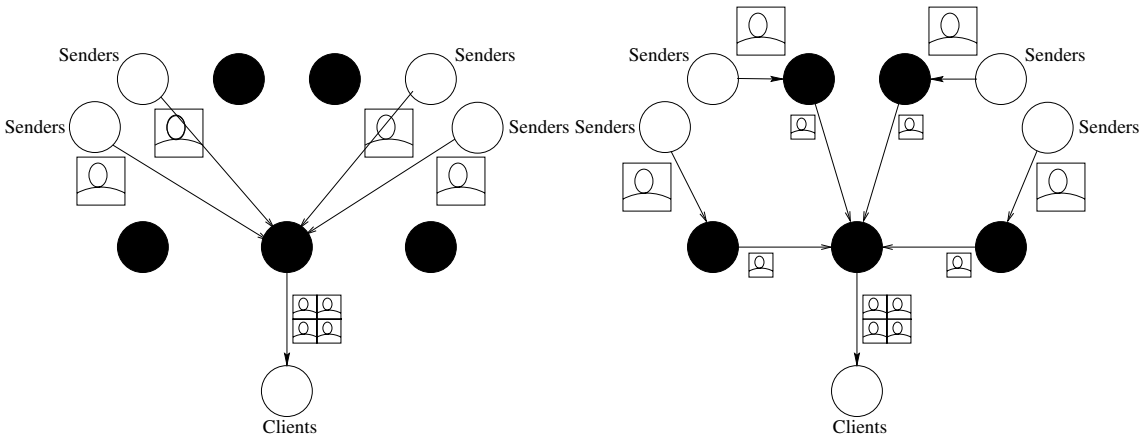

Figure 15: Composable Service is possible with AGLP. The left diagram shows a possible configuration when a client uses a single gateway to create a quad-splitter view. The right figure shows a possible configuration when multiple programs are used. Scaling the video near the sources may result in significant reduction in bandwidth usage.

AGLP builds on the announce-listen paradigm and uses soft-states to maintain information. As a result, our protocol is both scalable and robust. AGLP is compatible with existing MBone tools, so that no changes are required at the senders. Furthermore, the existence of gateways and clients is transparent to the senders.

Although AGLP was designed for the Degas system, the protocol can be modified for any application that needs to decide where to run certain services inside the network. With the increasing interest in the research community to move computation, traditionally performed at the edge of the network, into the network itself, we believe applications for AGLP will increase in the future.

7. REFERENCES

[1] E. Amir, S. McCanne, and Z. Hui. An application level video gateway. In *Proc. of 3rd ACM Intl. Multimedia Conf. and Exhibition*, pages 255–266, San Francisco, CA, November 1995.

[2] K. Calvert, M. Doar, and E. Zegura. Modeling internet topology. *IEEE Communications Magazine*, 36(6):160–163, June 1997.

[3] R. Droms. RFC 2131: Dynamic host configuration protocol, March 1997.

[4] W. Fenner. RFC 2236: Internet Group Management Protocol, version 2, November 1997.

[5] S. Floyd, V. Jacobson, C. G. Liu, S. McCanne, and L. Zhang. A reliable multicast framework for light-weight sessions and application level framing. *IEEE/ACM Transactions on Networking*, 5(6):784–803, December 1997.

[6] M. Handley and V. Jacobson. RFC 2327: SDP: Session description protocol, April 1998.

[7] S. McCanne and V. Jacobson. vic: A flexible framework for packet video. In *Proc. of 3rd ACM Intl. Multimedia Conf. and Exhibition*, pages 511–522, San Francisco, CA, November 1995.

[8] D. L. Mills. RFC 1305: Network time protocol (version 3) specification, implementation, March 1992.

[9] J. Nonnenmacher and E. W. Biersack. Scalable feedback for large groups. *IEEE/ACM Transactions on Networking 1999*, 7(3):375–386, June 1999.

[10] W. T. Ooi and B. Smith. The design and implementation of programmable media gateways. In *Proc. of 10th. Intl. Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV'00)*, Chapel Hill, North Carolina, June 2000.

[11] J. Rosenberg and H. Schulzrinne. Internet telephony gateway location. In *Proc. of IEEE INFOCOM*, March 1998.

[12] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. RFC 1889: RTP: A transport protocol for real-time applications, January 1996.

[13] T. Turletti. The INRIA videoconferencing system. *ConneXions - The Interoperability Report Journal*, 8(10):20–24, October 1994.

[14] J. Veizades, E. Guttman, C. Perkins, and S. Kaplan. RFC 2165: Service location protocol, June 1997.

[15] D. Xu, K. Nahrstedt, and D. Wichadakul. MeGaDiP: a wide-area media gateway discovery protocol. In *Proc. of IEEE Intl. Performance, Computing and Communications Conf.*, Pheonix, Arizona, February 2000.

[16] M. Yarvis, A. A. Wang, A. Rudenko, P. Reiher, , and G. J. Popek. Conductor: Distributed adaptation for complex networks. Technical Report CSD-TR-990042, University of California, Los Angeles, Los Angeles, California, August 1999.