Coscheduling of Computation and Communication Resources in Push-Pull Communications to provide End-to-End QoS guarantees

Kanaka Juvva

August, 1999
$CMU - CS - 99 - 166$

School Of Computer Science
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213-3890

This research was supported by the Defense Advanced Research Project Agency in part under agreement E30602-97-2-0287 and in part under agreement F30602-96-1-0160. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing policies , either expressed or implied of U.S.Government.

Keywords: Coscheduling, QoS Guarantees, End-to-End delay, Proxy, Topology, Push Communication

Abstract

In this paper, we extend the Push-Pull Communication Model [4] to provide end-to-end quality of service (QoS) for clients located in distributed and heterogeneous nodes. Push-pull communications is a middleware service that has been implemented on top of a Resource Kernel [9]. It is a many-to-many communication model, which can easily and quickly disseminate information across heterogeneous nodes with flexible communication patterns. It supports both "push" (data transfer initiated by a sender) and "pull" (data transfer initiated by a receiver) communications. Nodes with widely differing processing power and networking bandwidth can coordinate and co-exist by the provision of appropriate and automatic support for transformation on data communication frequencies. In particular, different information sources and sinks can operate at *different* frequencies and also can choose another (intermediate) node to act as their *proxy* and deliver data at the desired frequency.

In this paper, we specifically address the timeliness and bandwidth guarantees of the push-pull model. The location of a proxy, the network topology and the underlying network support can impact the timeliness of data. We formally analyze the problem of choosing an optimal proxy location within a network. We obtain the somewhat counter-intuitive result that if slightly longer end-to-end latencies can be tolerated and unicast protocols are used, locating the proxy at the publisher node is the best. The situation turns complex if multicast protocols are used. We show that this problem of optimal proxy allocation can be formulated as a mixed integer programming problem that can be solved efficiently. As an example, we solve the proxy location problem for a high-speed vBNS network configuration. We obtain our end-to-end timeliness and bandwidth guarantees by using a resource kernel offering CPU guarantees at the end-points and the use of a guaranteed bandwidth network between push-pull clients. We discuss our implementation of this system and carry out a detailed performance evaluation on an integrated RT-Mach - Darwin[3] testbed at Carnegie Mellon. Our results open up interesting research directions for the QoS scheduling of applications which require both computation and communication resources.

1 Introduction

The advent of high-performance networks such as ATM and 100 Mbps networks in conjunction with significant advances in hardware technologies are spawning several distributed real-time and multimedia applications. Distributed multimedia systems, in particular, are becoming more prevalent and effective in making widespread information accessible in real-time. Examples include video-conferencing over the Internet, multi-party collaborations systems, and Internet telephony. Data communications in these systems take place among geographically distributed participants, whose computing and networking resources can vary considerably. A service infrastructure which supports such distributed communications should be scalable, flexible and cater to different CPU/network bandwidths while providing real-time guarantees. Real-time push-pull communications [4] is a middleware service which provides many-to-many communications to clients operating in the above mentioned conditions. Providing QoS guarantees in Push-Pull communications is a major challenge. Because there can be wide variation in each subscriber's resources and QoS requirements. For example, in a multimedia multi-party collaboration application an audio subscriber has stringent end-to-end delay requirements, a video subscriber needs good frame rate, a white-board subscriber requires reliable delivery and a text subscriber may prefer security. This paper focuses on providing bandwidth and delay guarantees.

1.1 Motivation

The real-time publisher/subscriber communication model [6, 7, 8] can be considered to represent "push communications" where data is "pushed" by information sources to information sinks. As a result, subscribers can obtain information only at the rate at which the data is being pushed. This model is appropriate and efficient for periodic and synchronous updates between sources and sinks which are operating at the same frequencies[1]. Unfortunately, this can be very limiting in many cases where different clients have different processing power and/or widely varying communication bandwidths (because of connectivity to a low bandwidth network such as a telephone modem or an encrypted satellite link). If consumers did not have the same processor power or network bandwidth, a publisher must either falsely assume that they all have the same capability or publish two (or more) streams to satisfy consumers with different capabilities.

It would be very desirable if a client with a relatively low processing power and/or communication bandwidth is able to consume published data at *its* own preferred rate. In other words, the data reaching this client depends on its own needs, and not that of the publishing volume/rate of the publisher. Also, the real-time publisher/subscriber model is completely *synchronous*: subscribers normally block on a "channel" (represented by a *distribution tag*) waiting for data to arrive. Publishers produce data at the rates that they determine, and the published data are immediately sent to the subscribers on that distribution tag. Also, the publisher/subscriber model does not have support for QoS guarantees. The Push-Pull Communications model addresses these issues.

The objectives of the Push-Pull model are listed below:

Synchronous Communication: Synchronous Communication allows consumers on the same data streams to receive *and* process data at (locally determined) rates, which are independent of those used at the information sources. As a result, clients with high or low processing power and/or high or low network bandwidth can still usefully consume data on a stream. In addition, this can happen without the knowledge of the data producers who do not have to distinguish among the capabilities of the receiving consumers. In our approach we use temporal scaling and spatial scaling to implement synchronous communication.

- *Temporal Scaling*: A source publishes at a frequency of $\mathbf{f}$ and the sink consumes it at $\mathbf{f/n}$ where n is normally greater than or equal to 1, and typically an integer. The transformation can, for example, 'drop' intermediate data to achieve the desired frequency transformation which correspondingly decreases the load [14, 15, 19] on the receiver's networking and processing capabilities. Our design and implementation currently supports only integer-based scaling owing to its simplicity and application-independence.
- *Spatial Scaling*: Multimedia data types such as audio and video are inherently scalable. Applications like video telephony will benefit if there is a provision to transform a picture of size $\mathbf{WxH}$ to $\mathbf{W'xH'}$ based on the processing power of the consumer.

Asynchronous Communication: In asynchronous communication, data can be "pulled" by an information consumer on demand. A "pulling" consumer can choose to consume data at a rate lower than the data production rate. In the extreme, a pulling consumer can choose to only consume data *asynchronously*. Pull buffers as explained below can provide asynchronous communication.

- *Pull Buffers*: A finite sequence of a real-time activity can be buffered at a location and can be pulled by a subscriber on demand at a later point of time. We use *pull communications* to support "history

[1] A subscriber may choose to operate at a different lower frequency by, for example, skipping every other published datum on a subscribed tag. However, for this to happen, the subscriber must still receive and "consume" the datum albeit in a trivial "drop-it" fashion.

buffers" and pulling of older messages on demand. Pull buffers are similar to Proxy Caches in Web-based systems.

Transparent Data Communication : In our approach, we use a *proxy* to perform data transformation transparent to the data source and the data sink. Proxy is a logical entity that can reside in the client library or in a daemon. The proxy performs data transformations without affecting the functionality of producers and consumers (but potentially increasing the end-to-end delay between the two). Spatial Scaling might also require that the proxy be aware of the semantics of the data (for example, that it is a raw video stream). A proxy is used only when necessary, and two (or more) clients can be receiving at two (or more) rates from the same data channel.

QoS Guarantees: Temporal scaling as explained above provides the sampling rate required for individual clients. Similarly, spatial scaling provides resolution control. Clients also may have different timeliness requirements. The proxy location, load on the proxy node, the distance (number of hops) between publisher and subscriber, the network topology, link capacities and link delays impact the end-to-end QoS. For the clients who do not require spatial/temporal scaling, proxies can be still used to provide BW and delay guarantees.

Protection and Enforcement: Since multiple communication channels can be in use simultaneously at different rates, there should be support for spatial and temporal protection among the various channels. As an example, increasing the sampling rate of a video stream should ideally not adversely affect the worst-case end-to-end delays of a audio stream. The scheduling/dispatching layers should use primitives which support enforcement when possible. For instance, a reservation-based scheduling scheme [9, 11, 3] provides temporal protection and guaranteed timeliness, while priority-based schemes can provide predictable timeliness under worst-case assumptions that do not *enforce* protection.

In summary, the real-time push-pull communications model supports predictable, efficient and synchronous as well as asynchronous communications among heterogeneous nodes.

1.2 Organization of the Paper

The rest of the paper is organized as follows. In Section 2, we provide a detailed analysis of proxy location. In Section 2.2, we compute predicted end-to-end delays. Section 2.4 describes proxy allocation to multiple subscribers and proxy allocation as an optimization problem. In Section 2.3 and Section 2.5 we describe performance impact of a proxy node with Unicast protocols and multicast protocols. In Section 3, we describe the architecture of the real-time push-pull communication model and describe its primary components. Section 3.3 gives a detailed performance evaluation of the model using end-to-end reservation in several configurations. Finally, we present our concluding remarks in Section 4.

1.3 Comparison with Related Work

In this section, we compare our approach with that of other related systems. The push-pull model is built on top of a Resource Kernel[9] and uses resource kernel primitives real-time priorities, real-time threads, RT-IPC and basic priority inheritance mechanisms at all levels (client and daemons) and focuses on QoS guarantees. It interacts with Darwin for network QoS support.

The model fits well in the context of both hard and soft real-time systems and particularly it is very promising to distributed multimedia applications like videoconferencing. We have successfully built a Multimedia mutli-party collaboration application on top of the model [4].

Maestro[1] is a middleware support tool for distributed multimedia and collaborative computing applications. Salamander[5] is a push-based distribution substrate designed to accommodate the large variation in Internet connectivity and client resources through the use of application specific plug-in modules. However [5] does not address real-time guarantees and temporal protection among different virtual data channels.

In [20], Fox et. al. propose a general proxy architecture for dynamic distillation of data based on client variation. It does not address temporal dependencies which impose tight timing constraints on the distillation process, which affects the architecture of the distiller. The work in [17] addresses adding group communications support to CORBA. Some work is going on in RT-CORBA [10, 13] to provide QoS and minimize end-to-end latencies in CORBA based systems. Work in [16, 17, 18] address group communication protocols and fault-tolerance. There is some research on QoS which deals with global optimum allocation of resources to provide the specified QoS [21].

At a higher level, while Web browsing has evolved from pull communications to push communications, our model has evolved from push communications (the publisher-subscriber model) to the current one which includes pull communications. The primary difference is that we focus on relatively closed systems with explicit resource management on the end-points and bandwidth guarantees, and are able to provide end-to-end timing guarantees.

2 Analysis of Proxy Location

We first define some terms used in the rest of this paper:

- *Publisher:* A Publisher produces information on a communication channel.
- *Subscriber:* A Subscriber consumes information from the communication channel. Publishers and subscribers are also called push-pull clients.
- *Distribution Tag:* Publisher/Subscriber uses a distribution tag as a logical handle for a communication channel. Tag Table in each daemon serves as a repository of all the distribution tags.
- *Push-Pull Daemon:* The Push/Pull IPC daemon resides on every node involved in the push-pull communications. IPC Daemons on various nodes communicate with one another keeping them all appraised of changes in distribution tags and publication/subscription status.
- *Pull Buffers:* Pull buffers store data samples, which can later be pulled by the pull subscribers.

2.1 Proxy

A consumer upon subscription to a tag (either as a push-client or a pull-client) can specify frequency scaling that must be done by the middleware service on the data stream it is subscribing to. A "proxy" is used to accomplish this frequency transformation, and once created, the proxy is transparent to the data source as well as the data sink. A proxy can exist in one of three configurations:

1. `Proxy at Subscriber:`
   The transformation and scaling takes place on the subscriber node and is useful when the subscriber node is powerful but the application is not interested in the higher frequency. The Proxy executes in the subscriber's address space.
2. `Proxy at Publisher:`
   The transformation and scaling takes place on the publisher node. This is useful when the publisher node is not loaded and frequency transformation benefits the subscriber. The Proxy executes in the publisher's address space.
3. `Proxy on remote/Intermediate node:`
   The transformation and scaling takes place on an intermediate node (potentially) specified by the subscriber. This is useful when both the publisher and the subscriber do not have the slack to perform the scaling themselves and the subscriber benefits from the scaling. In practice, for low-bandwidth clients across a modem link, nodes which act as the gateway to the wired network are good candidates for being proxies. The Proxy executes in a daemon's address space.

In the rest of this section, we will analyze the impact on the system resources (CPU cycles and network bandwidth) due to the assignment of proxies at different nodes. We first study the impact of a proxy assignment on the end-to-end delay encountered by a subscriber receiving a message stream.

2.2 Computing End-To-End Delay

In our analysis of the real-time push-pull model, we assume that one period T delay is allocated for processing and re-transmission at *each* node in the datapath between a publisher and a subscriber. This is a normal assumption in the use of rate-monotonic analysis [22, 23, 24] as it applies to distributed real-time systems. While other assumptions can also be analyzed by the framework, this is a convenient assumption that simplifies presentation.

Hence, if one node is located on the path between a publisher and a subscriber for a message stream with period T, we assume that one period T will be used on the publisher node to transmit each message and another period T will be used on the intermediate node to receive and re-transmit that message to the subscriber. This results in a net end-to-end latency of $2T$ for the message stream in this case.

An interesting side-effect of this assumption results in the presence of a proxy which temporally scales from a published period of T to a (longer) period[2] of T'. When the proxy is on the publisher node, it transmits every T' units of time and *not* T units of time. Hence, if there are 3 nodes in the path to the subscriber, the end-to-end delay is $3T'$. In contrast, if the proxy is on the subscriber node, the publisher and intermediate nodes transmit data every T units of time, resulting in a net end-to-end delay of $3T$, which is shorter than $3T'$! End-to-end delays in between $3T$ and $3T'$ arise if the proxy is on an intermediate remote node. The delay is longer if the proxy is closer to the publisher and vice-versa.

[2] Recall that by our assumption, $T'/T =$ an integer.

	Proxy On Intermediate Node			Proxy at Publisher Node			Proxy at Subscriber Node		
	CPU Load	Network Bandwidth	End-to-End Delay	CPU Load	Network Bandwidth	End-to-End Delay	CPU Load	Network Bandwidth	End-to-End Delay
Publisher Node	High	High	-	Low	Low	-	High	High	-
Subscriber Node	Low	Low	Medium	Low	Low	High	High	High	Low
Proxy Node	High	Medium	-	-	-	-	-	-	-

Table 1: Adversity of Performance Impact due to Proxy Location Choice using Unicast Protocols

	Proxy On Intermediate Node			Proxy at Publisher Node			Proxy at Subscriber Node		
	CPU Load	Network Bandwidth	End-to-End Delay	CPU Load	Network Bandwidth	End-to-End Delay	CPU Load	Network Bandwidth	End-to-End Delay
Publisher Node	Low	Low	-	High	High	-	Low	Low	-
Subscriber Node	Low	Low	Medium	Low	Low	High	High	High	Low
Proxy Node	High	Medium	-	-	-	-	-	-	-

Table 2: Adversity of Performance Impact due to Proxy Location Choice using Multicast Protocols

2.3 Performance Impact of a Proxy Node

We now discuss the performance impact of locating proxies in different nodes. The introduction of a proxy at any node can introduce additional load on the CPU as well as impact the need for network bandwidth on that node. The CPU load has two components: additional communication protocol processing of incoming and outgoing network packets and computational processing to scale published data. In the case of the integral temporal scaling that we perform, the computational processing time tends to be small since packets are just dropped or re-directed forward to the subscriber destination(s). However, with several subscribers, the CPU load can turn out to be significant. Similarly, the bandwidth demand increases with multiple proxy requests from subscribers. As a result, the processing power and network bandwidth resources available on a proxy node can limit the number of proxy requests that it can handle.

In summary, the addition of one proxy to a node catering to one or more subscribers consists of (or impacts) the following:

- *CPU load*: Computation load for communication protocol processing of incoming and outgoing packets, and processing cycles for data scaling.
- *Network bandwidth*: Bandwidth on the network links of this node must be utilized to receive and send data packets.
- *End-to-end latency*: The end-to-end latency from a publisher to its subscriber changes with the choice of this node as the proxy (as per the discussion in Section 2.2).

2.3.1 Impact under the Use of Unicast Protocols

In this section, we assume that the transmission of a message from one publisher to one or more subscribers must be accomplished using unicast protocols.

Consider a distribution tag with one publisher, one subscriber and a proxy. The CPU load and network bandwidth requirements at the publisher node, the subscriber node and the proxy node vary depending upon where the proxy is located. Since the proxy can be located at the publisher node, the subscriber node or an (intermediate) remote node, the impact it has changes. This impact of the choice of the proxy location on the publisher node, the subscriber node or an intermediate (remote) node is summarized in Table 1. The impact at the publisher node, subscriber node or the proxy node form the rows of the table. The choice of the proxy location forms the columns. The end-to-end delay metric is valid only at the subscriber node. For example, consider the impact at the subscriber node (second row). When the proxy is at a remote node (1st column), the subscriber node's CPU load is the lowest possible, its bandwidth requirement at its link is the lowest possible, but its end-to-end latency is neither the highest possible nor the lowest possible (as per Section 2.2).

Table 1 leads to some interesting observations. If end-to-end latencies are ignored, both CPU and network bandwidth requirements are minimal when the proxy is at the publisher node. This is true both at the publisher and subscriber nodes. The trade-off is that if there are k hops between the publisher and the subscriber, this would result in an end-to-end delay of kT' instead of the best-case kT which occurs when the proxy is located at the subscriber node. If this longer delay is acceptable[3], locating the proxy at the publisher node is the right thing to do. Finally, as seen in Section 3.3, the computational processing overhead for temporal scaling at any node adds negligible overhead.

We now provide an algorithm and an example to illustrate this behavior.

2.4 Allocation of a Single Proxy to Multiple Subscribers

Consider the allocation of a single proxy for n subscribers to receive data from a single publisher. Proxy receives data and performs temporal scaling for all the n subscribers and transmits to them. Publisher and subscribers are situated in different nodes in the system. The best location of proxy can be the one which guarantees end-to-end delays for all the subscribers while minimizing the total cost (computation + communication). The required definitions are given below.

- Number of subscribers = n ; $n > 0$.
- Number of nodes in the system = m ; $m > 0$. One of the m nodes will be picked to host the proxy for all the subscribers.
- Publisher produces data at a rate of f (with period T) and the proxy scales these data for the subscriber for consumption at a lower frequency of f' (with period T', which is an integral multiple of T). Recall that with integer temporal scaling, f / f' = an integer ≥ 1, such that $f \geq f'$ and $T \leq T'$.
- Let Δ_1^r, $\Delta_2^r \cdots \Delta_n^r$ be the required end-to-end delays of the n subscribers respectively.
- L = Network Bandwidth required for a subscriber without temporal scaling of data. L' = Network Bandwidth required for a subscriber with temporal scaling of data. $L' \leq L$.
- d_j^{pub} = distance (number of hops) of a proxy node j from publisher. $d_{i,j}^{sub}$ = distance (number of hops) of a node j from subscriber i. $d_j^{pub} \geq 0$; $1 \leq$ j $\leq m$; $d_{i,j}^{sub} \geq 0$; $1 \leq$ i $\leq n$; $1 \leq$ j $\leq m$; $d_j^{pub} + d_{i,j}^{sub} \geq 1$; $1 \leq$ i $\leq n$; $1 \leq$ j $\leq$ m
- $\delta_{i,j}$: end-to-end delay from the publisher to subscriber i with node j acting as proxy; Where $\delta_{i,j} = d_j^{pub}T + d_{i,j}^{sub}T'$
- $cost_{i,j}$ = Communication cost for the $path_{ij}$, directly proportional to the volume of data being processed; $cost_{i,j} = K * (d_j^{pub}$f $+ d_{i,j}^{sub}f')$, where K is a constant. $TotalCost_j = \sum_{i=1}^{n} cost_{ij}$ is total communication cost for *all* subscribers.
- The next step is to determine the proxy node on a least cost path that satisfies end-to-end delay for all the subscribers. This can be carried out by just finding the node j with the least value of $TotalCost_j$ and has $\delta_{i,j} <= \Delta^r$ for all n subscribers. An example system is given in Appendix A which describes how to determine a proxy node for a simple topology.

2.4.1 Proxy Allocation as an Optimization Problem

The proxy allocation problem can be formulated as a mixed-integer programming problem with linear constraints for a given topology of networked nodes. This mixed-integer programming problem can be solved using standard optimization packages such as CPLEX.

The general formulation can be used with no limits on the number of tags, and any number of publishers/subscribers on each tag. Due to space and presentation considerations, we will illustrate the formulation assuming one proxy per tag.

The steps involved in the formulation are as follows:

1. Pick binary variables to represent proxy location
2. State end-to-end delay constraints in terms of these binary variables and distances from nodes
3. State bandwidth constraints on each network link
4. State processing limits (constraints) on each processing node
5. Minimize a Cost Function which can be any one of
   - Total Network Bandwidth
   - Total CPU cycles used
   - Total Network BW used + (weight * Total CPU cycles)

[3]Recall from the discussion of Section 2.2 that this end-to-end delay computation is based on a one-period deadline allocation and that this assumption can be easily relaxed.

2.4.2 Proxy Allocation in the High-Speed vBNS Network

We apply the bandwidth minimization cost function to the vBNS network topology shown in Figure 1. vBNS is a high-speed high-bandwidth network deployed across the US and is extensively used for research. Assume a publisher at CMU and a subscriber at San Diego. The datapath between the publisher and the subscriber is shown with a dotted line. Nodes are numbered from $\{0 - 6\}$ and communication links are labeled as $\{a - f\}$. A Mixed Integer Programming formulation for a proxy allocation for the vBNS topology is given below.

Let *Bw(a), BW(b), BW(c), BW(d), BW(e)* and *BW(f)* be the bandwidth capacities of the links a,b,c,d,e and f respectively. Let L and L' be the unscaled and scaled bandwidths for the CMU - San Diego flow, and T and T' be the time periods of the publisher and the subscriber. X, Y and Z are integer binary variables that determine the proxy location. For example, $XYZ = 001$ implies that the proxy is at Node *1*. Then, the mixed integer programming problem to determine the proxy location is as follows:

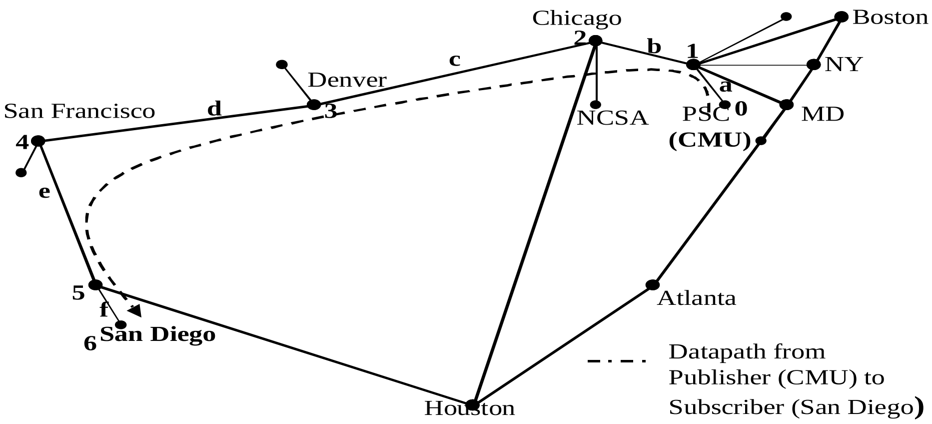


Figure 1: *vBNS Topology*

Objective function:

Minimize

$$(1-X)(1-Y)(1-Z)6L' + (1-X)(1-Y)Z(L+5L') + (1-X)Y(1-Z)(2L+4L') \\ +(1-X)YZ(3L+3L') + X(1-Y)(1-Z)(4L+2L') + X(1-Y)Z(5L+L') + XY(1-Z)6L$$

Subject to:

Delay constraint

$$(1-X)(1-Y)(1-Z)6T' + (1-X)(1-Y)Z(T+5T') + (1-X)Y(1-Z)(2T+4T') \\ +(1-X)YZ(3T+3T') + X(1-Y)(1-Z)(4T+2T') + X(1-Y)Z(5T+T') + XY(1-Z)6T <= \Delta \text{ (delay constraint)}$$

and

Network bandwidth constraints

Link a:

$$(1-X)(1-Y)(1-Z)L' + (1-X)(1-Y)ZL + (1-X)(Y(1-Z)L \\ +(1-X)YZL + X(1-Y)(1-Z)L + X(1-Y)(1-Z)L + XY(1-Z)L \leq BW(a)$$

Link b:

$$(1-X)(1-Y)(1-Z)L' + (1-X)(1-Y)ZL + (1-X)(Y(1-Z)L' \\ +(1-X)YZL + X(1-Y)(1-Z)L + X(1-Y)(1-Z)L + XY(1-Z)L \leq BW(b)$$

Link c:

$$(1-X)(1-Y)(1-Z)L' + (1-X)(1-Y)ZL' + (1-X)(Y(1-Z)L' \\ +(1-X)YZL + X(1-Y)(1-Z)L + X(1-Y)(1-Z)L + XY(1-Z)L \leq BW(c)$$

Link d:

$$
\begin{aligned}
&(1-X)(1-Y)(1-Z)L' + (1-X)(1-Y)ZL' + (1-X)(Y(1-Z)L' \\
&\quad +(1-X)YZL' + X(1-Y)(1-Z)L' + X(1-Y)(1-Z)L + XY(1-Z)L \leq BW(d)
\end{aligned}
$$

Link e:

$$
\begin{aligned}
&(1-X)(1-Y)(1-Z)L' + (1-X)(1-Y)ZL' + (1-X)(Y(1-Z)L' \\
&\quad +(1-X)YZL' + X(1-Y)(1-Z)L' + X(1-Y)(1-Z)L + XY(1-Z)L \leq BW(e)
\end{aligned}
$$

Link f:

$$
\begin{aligned}
&(1-X)(1-Y)(1-Z)L' + (1-X)(1-Y)ZL' + (1-X)(Y(1-Z)L' \\
&\quad +(1-X)YZL' + X(1-Y)(1-Z)L' + X(1-Y)(1-Z)L' + XY(1-Z)L \leq BW(f)
\end{aligned}
$$

The objective function and the resource constraints are non-linear in X, Y and Z containing terms of the form XY, YZ, XYZ and XZ. These constraints can be linearized by introducing substitutions and adding extra (linear) constraints to the problem. We use the following substitutions:

$$XY = a$$

$$YZ = b$$

$$ZX = c$$

$$aZ = d$$

Note that $XYZ = aZ = d$. For example, the reduced objective function now becomes

$$
\begin{aligned}
&(1+a+b+c-X-Y-Z-d)6L' + (Z+d-c-b)(L+5L') + (y+d-a-b)(2L+4L') \\
&\quad +(b-d)(3L+3L') + (X-a-c+d)(4L+2L') + (c-d)(5L+L') + (a+d)6L
\end{aligned}
$$

which is linear in the new variables, and we insert the following additional constraints:

$a \geq 0$	$b \geq 0$	$c \geq 0$	$d \geq 0$
$d - x \leq 0$	$a - y \leq 0$	$-a + x + y \leq 1$	$b - z \leq 0$
$c - x \leq 0$	$-b + y + z \leq 1$	$c - z \leq 0$	
$d - z \leq 0$	$-c + x + z \leq 1$	$d - a \leq 0$	$-d + z + a \leq 1$

Similarly, delay and BW constraints become linear in the new variables. If there are multiple subscribers (or proxies) per tag and/or multiple tags, the above sequence will be repeated with different sets of binary variables for each proxy for each tag. The end-result will be that the number of terms in the constraints and the objective function will increase correspondingly. But each new term will be a simple addend to the sum and the complexity of the formulation remains the same.

The above formulation problem has been solved using CPLEX optimization package [27] for different topologies and subscribers. CPLEX always found a solution to the problem for all cases and run time was in the order of milliseconds. We present the optimization running times for the vBNS topology in Table 3.

Topology	*# Nodes*	*(# Publishers, # Subscribers)*	*Pre-Solution time* ms	*Solution time* ms
vBNS	7	(1,1)	0	10
vBNS	7	(1,2)	10	10
vBNS	7	(1,5)	10	20
vBNS	7	(1,10)	20	40

Table 3: The Run-Times Efficiency of Solving Proxy Allocation problem for vBNS Topology

2.5 Impact with the Use of Multicast Protocols

Suppose that, instead of using unicast protocols to transmit data to multiple subscribers, the publication of a message on a tag is carried out on a multicast address. The table, corresponding to the one of Section 2.5 (Table 1), for this case is given in Table 2. Suppose that a publisher is publishing to two or more subscribers with different scaling factors (as shown in Figure). Suppose that the proxy is now located on the publisher node itself. Each scaled message stream must now therefore be transmitted on a different multicast address. This increases the CPU load on the publisher node. Specifically, its protocol processing overhead increases as does its network bandwidth requirement. Hence, the entries in the first row, second column of Table 2 list {High, High} when they were {Low, Low} in the case of Table 1. As can be seen in Table 2, there is no single column that yields all "Lows" with the first column of having a remote proxy coming close.

In this case, the best choice of a proxy location is not at all obvious. One would actually expect that the problem of picking the optimal locations of proxies given a set of distribution tags, publishers and subscribers to be computationally expensive.

3 The Push-Pull Communication Implementation and Performance Evaluation

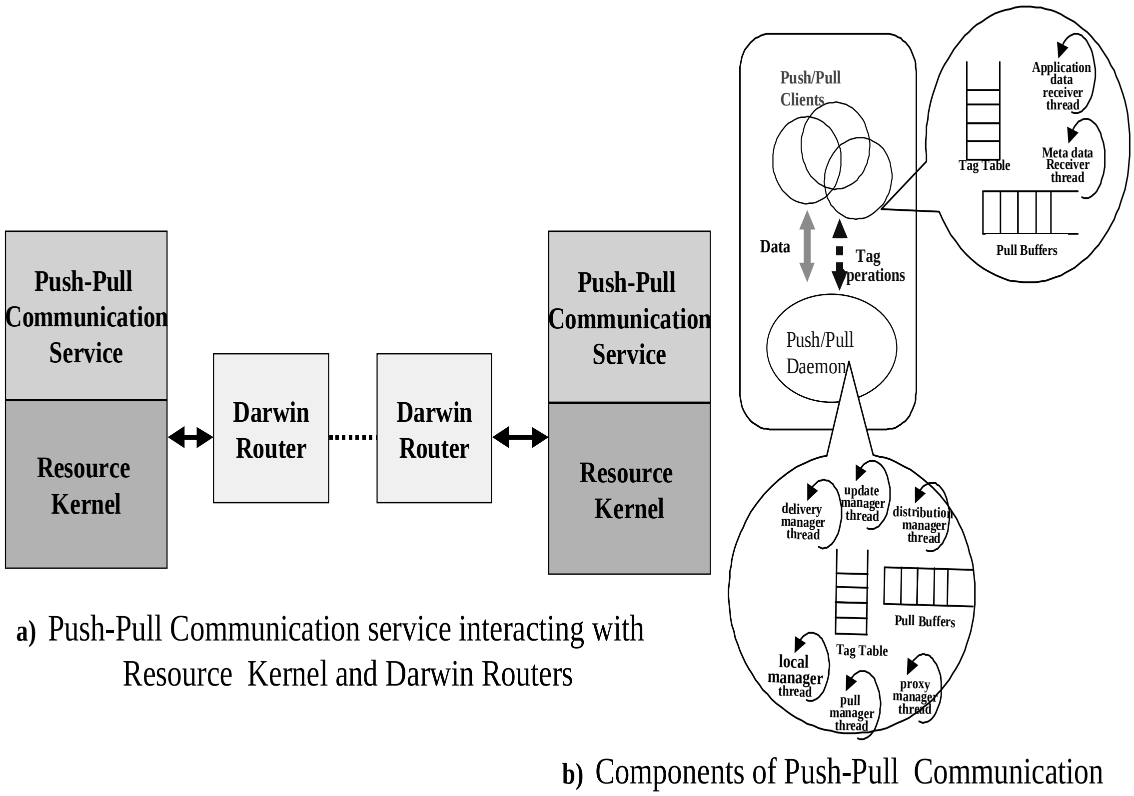


Figure 2: *The Architectural Components of the Real-Time Push-Pull Communications Model.*

The architecture of the real-time push-pull communications model is illustrated in Figure 2 (a). The communication service is built on top of a Resource Kernel[9] which provides QoS guarantees. Nodes in the system are assumed to be interconnected using DARWIN [3], a network fabric that provides bandwidth reservations. The Darwin network is explained in the next section. As shown in Figure 2 (b) every node runs a daemon comprising multiple threads. A client wishing to communicate using the real-time push-pull service interfaces to a library in its own address space. The client library consists of several threads as shown in Figure 2 (b), maintains local information from the real-time push-pull service and its own local buffers. This information maintained in the

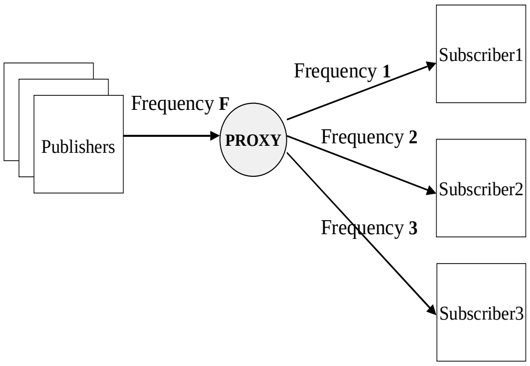


Figure 3: *An Intermediate Location of the Proxy performing temporal scaling to 3 different subscribers.*

client is structured such that any damage to this data will affect only this client. The threads within the daemon and the client are shown in the figure.

3.1 Proxy

The basics of the Proxy were already described in Section 2.1. The existence (or non-existence) of the proxy is retained in the attributes of the distribution tags (channels) maintained by the daemons and client libraries, and used appropriately when data is published or pulled. This is illustrated in Figure 3. The proxy location is chosen by a subscriber by setting the proxy attributes of the particular channel on the distribution tag. The proxy attributes are **{{**`Mode, IP address`**}**`, Scaling Factor`**}**.

- `Mode` is either `Proxy at subscriber`, `Proxy at publisher` or `Proxy at intermediate node`. The `IP address` field is used only in `Proxy-Mode-3` and identifies the remote node.
- The `Scaling Factor` is used for temporal scaling. It is the number by which the source frequency field is divided to get the desired frequency. For example a scaling factor of n implies for every 'n' samples of publisher data,the subscriber receives 1 sample.

A push-pull daemon is run on every node of the system using our middleware service, and the proxy agent when needed resides typically within the daemon of the node where it is located. It can reside in a client library if the proxy is at either the publisher or the subscriber node. Data published to a subscriber who has requested a proxy is frequency-transformed as it is being transmitted (by dropping as necessary). The critical aspect is that multiple subscribers on the same channel may want to have their proxy in distinct modes. The performance impact of the proxy on the system in different modes is illustrated in Section 2.3.

3.2 Pull Communications

"Pull" communications is used by a subscriber to "pull" specific data samples asynchronously on demand. For instance, a subscriber may want to pull dynamically a particular recent sequence of an ongoing video conference. An aircraft control system may want to pull weather information about a particular region in the past two hours. As a result, it is desirable that the middleware service support a **history buffer** which stores recent versions of data samples published on a channel. A customer can then request on demand the most recent copy of a data sample, the n^{th}-most recent version or a specific absolute sample from this history.[4]

The "Pull attributes" we support are **{{**`Publisher, Mode, IP address`**}**`, Number of Messages`**}**, where

- `Publisher` is the source from which messages are to be pulled.
- `Mode` can be one of the following:
  1. *Buffering at Subscriber:*
     The daemon on the subscriber node buffers the messages. This mode is useful when there are no networking constraints at the client side, and the requirements are unique to this client.

[4]Our current implementation supports a circular buffer which stores a fixed number of the most recent samples published.

2. *Buffering at Publisher:*
   The publisher node itself maintains the buffer. This is a logical place if multiple clients need the capability to access recent history on a demand basis.
3. *Buffering at Remote/Intermediate node:*
   Buffering is done at an intermediate node. This mode is used when the publisher is loaded and multiple clients can benefit or when one client has constraints on networking, processing and storage capabilities.

- `IP address` is the address of the remote node for *Pull-Mode-3*.
- `Number of Messages` is the desired number of messages to be stored in the buffer.

A subscriber issues pull requests to pull the messages. Time-stamping and/or versioning of data is required to indicate a specific message. In our current implementation, all the messages carry a sequence number and the sequence number is used to pull a specific message.

3.2.1 Resource Kernel

The Push-Pull Communications executes as a middleware service on top of a Resource Kernel. The Resource Kernel provides timely and protected access to machine resources namely CPU, disk bandwidth and network bandwidth [9, 11, 12]. Push-Pull uses the Resource kernel primitives to provide timely and efficient resource utilization.

3.2.2 Darwin: The Guaranteed Bandwidth Network

Darwin [3] is a network fabric deployed at Carnegie Mellon University, which provides guaranteed access to network bandwidth for end-hosts. The Darwin architecture shown in Figure 4 is similar in many ways to traditional resource management structures. For example, the resource management mechanisms for the Internet defined by the IETF in the last few years rely on QoS routing (service brokers), RSVP, and local resource managers that set up packet classifiers and schedulers. A key component is the hierarchical fair service curve and (HFSC) scheduler [26] that manages link bandwidth and that can implement a broad range of sharing policies, including fair sharing and guaranteed services. Moreover, it supports the hierarchical resource management that is needed for hierarchical deployment, and it allows controlled sharing of resource between sibling nodes without violating guarantees inside the subtrees, so subtrees can be managed independently.

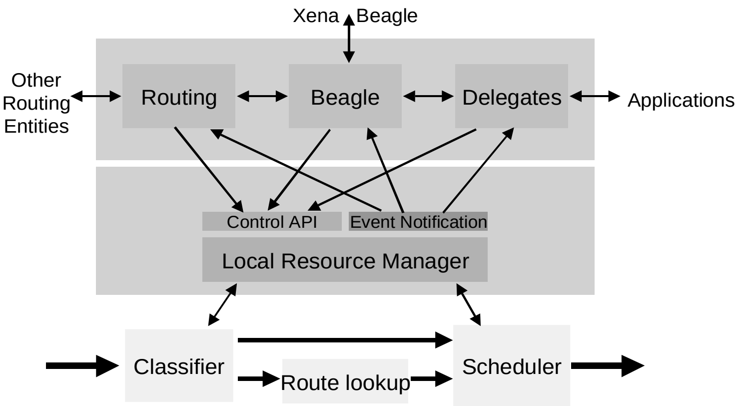


Figure 4: *Darwin node architecture.*

3.3 Performance Evaluation

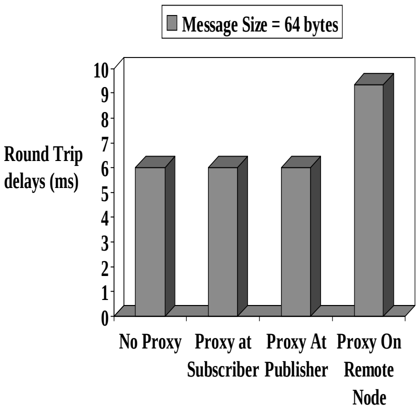
The real-time push-pull model has been successfully designed and implemented on a testbed consisting of endpoints running RT-Mach and the Darwin network at Carnegie Mellon which provides guaranteed bandwidth between specific end-points. This section describes a set of measurements obtained on a network of three Pentium-120 MHz PCs with 32MB RAM running RT-Mach version RK97a. The network was 10Mbps ethernet, a Subscriber and a Publisher were run on same and different nodes, with proxy in several configurations. Each of the Darwin routers was a Pentium-II 256 MHz with 128MB RAM.

3.3.1 The Goals and Use of Our Performance Evaluation

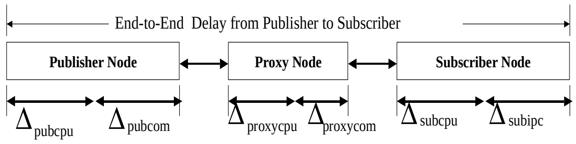
The goals of our performance evaluations were two-fold. One, we strive to fully understand the performance impact of the use of a proxy. As will be seen, having a proxy on a remote (intermediate) node can lead to additional delays. Our second goal was to understand the impact on end-to-end delays with multiple hops through a real network. These measurements can be directly plugged into standard rate-monotonic analysis techniques to obtain predicted worst-case end-to-end delays between publishers and subscribers. In addition, explicit resource management on the end-points and the use of a guaranteed bandwidth network like Darwin can be used to provide guaranteed access to required bandwidth between push-pull clients.

3.3.2 The Performance Impact of a Proxy

The experiment we conducted to measure the performance impact of a proxy is as follows. A publisher transmits a 64-byte message which is received by a subscriber, which in turn re-transmits that message by publishing on a separate tag. The original publisher receives this message and the time taken for this sequence to complete at the first publisher node corresponds to a *Round-trip Delay.* We calculated the average of this round-trip delay after 100 messages. These measurements based on an unoptimized implementation[5] are summarized in Figure 5.



(a) Round Trip Delays



(b) Delay Components

Figure 5: *Round-Trip delays for Different Proxy Locations and corresponding End-to-End Components*

We repeated the experiment in 2 configurations: without a proxy, and with a proxy in between the first publisher/subscriber pair. In addition, the proxy if used could be located on the publisher site, the subscriber site or an intermediate site. The measurements of round-trip delays are shown in Figure 5. The key (non-network-related) components of the end-to-end delay illustrated in Figure 5 are given in Table 4. When proxy is located at a publisher or a subscriber, $\Delta_{proxycpu}$ and $\Delta_{proxycom}$ are included in $\Delta_{pubcpu}/\Delta_{subcpu}$ and Δ_{pubcom} [6] respectively. So they are not measured separately.

Delay Components in ms	*Proxy at Subscriber*	*Proxy at Proxy at Publisher*	*Proxy on Remote Node*
Δ_{pubcpu} : computational processing time	0.2816	.2871	0.2807
Δ_{pubcom} : communication processing time	0.4920	0.498	0.494
Δ_{subipc} : ipc overhead	0.3601	0.364	0.3626
Δ_{subcpu} :computational processing time	0.33	0.328	0.3329
$\Delta_{proxycpu}$: computational processing time	-	-	0.042
$\Delta_{proxycom}$: communication processing time	-	-	0.49664

Table 4: Individual Components for End-to-End delays, Message Size = 64 Bytes

As can be seen, the presence of a proxy at a subscriber node or a publisher node adds very little overhead compared to the case of having no proxy at all. In this case, the proxy was scaling the data stream by a factor

[5]The system measured uses an ISA bus 8-bit Ethernet card, and we expect significantly better absolute performance numbers on a 32-bit PCI card.

[6]The communication processing time in push-pul daemon and doesn't include network protocol processing time

of one, i.e. passing the data straight through. When the proxy is on a remote node, a latency of about $3ms$ [7] is added to the round-trip path. Several other experiments were conducted varying the distance (with Darwin nodes routing the messages) between a publisher, a subscriber through a proxy and with different message lengths.

3.3.3 Experiment #1: The Impact of Message Lengths when Publisher/Subscriber on a single node

In this experiment the publisher and the subscriber are located in the same node as shown in Figure 6 (b), and the proxy is at either the publisher, the subscriber or the remote node. We vary the message lengths and the round-trip delays encountered are as shown in Figure 6 (a). As one can see, delays increase linearly as message length increases and longer delays are encountered when the proxy is located on a remote node. When the publisher and subscriber are on the same node, locating the proxy on a remote node may not appear logical. This situation can potentially occur for proxy caches in departmental servers.

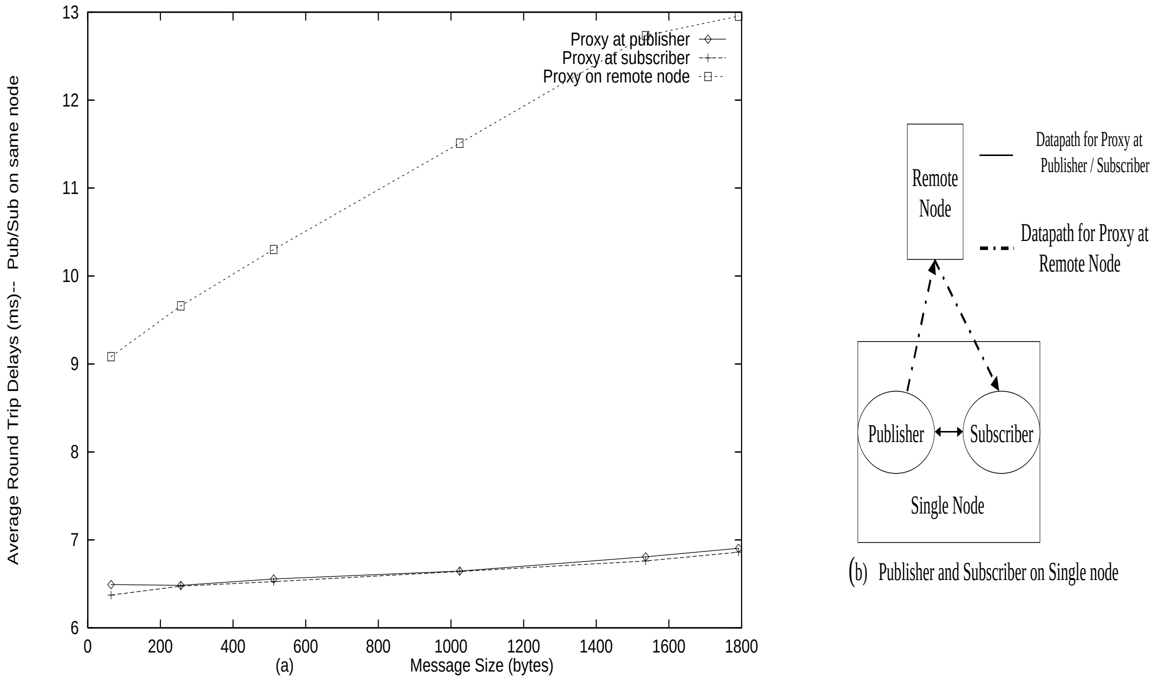


Figure 6: *The Round-Trip Delays and Publisher, Subscriber and Proxy configuration for* Experiment #1.

3.3.4 Experiment #2: The Impact of Message Lengths when Publisher and Subscriber are at one hop distance

In this experiment, the publisher and the subscriber are located in different nodes and the proxy is on one of the three nodes as shown in Figure 7 (b). Round-trip delays encountered for all the three configurations as message length is varied are shown in Figure 7 (a). The delays increase almost linearly when the proxy is at the publisher or the subscriber. With the proxy on a remote node, an additional delay of about 2.5ms is introduced.

3.3.5 Experiment#3: The Impact of Message Lengths when Publisher and Subscriber are at two hops distance

This experiment measures round-trip delays encountered when the Publisher and Subscriber are separated by one Darwin router and are at two hops distance as shown in Figure 8 (b), and the proxy is at the publisher node or the subscriber node. As shown in Figure 8 (a), the router introduces less delays for smaller messages. This can be useful for small data packets such as audio. Audio data needs smaller delays and keeping audio packets as small as 256 or 512 bytes can result in delays similar to that of Experiment #2.

The delays introduced by the Darwin routers will be discussed shortly.

[7] We expect all the overheads to come down by a factor of 4 or more with faster PCs. Experiments had to be carried out on a 90 Mhz PC.

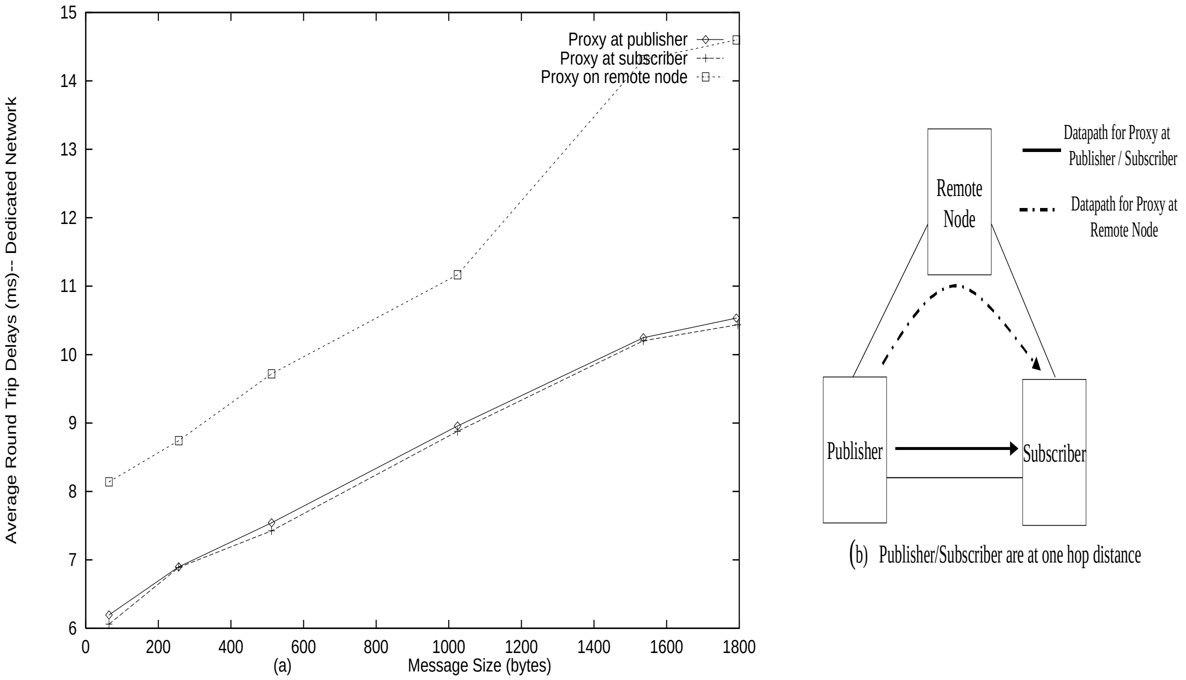


Figure 7: *The Round-Trip Delays and Publisher, Subscriber and Proxy configuration for* Experiment #2

3.3.6 Experiment #4: The Impact of Message Lengths and when Publisher and Subscriber are at three hops distance

This experiment measures round-trip delays when the publisher and the subscriber are at three hops distance and separated by two Darwin routers as shown in Figure 9 (b). As one would expect again, with 3 hops distance, smaller messages have less routing delays than that of larger messages as shown in Figure 9 (a).

3.3.7 Delays Introduced by Darwin Routers

Based on the round-trip delays from Experiments 2, 3 and 4, the delays introduced by the Darwin routers can be computed and are plotted in Figure 10. As can be seen, the delays introduced by a Darwin router increases (roughly) linearly with message size. This is as expected. Figure 10 also indicates that the delays increase faster for messages longer than 256 bytes. We attribute this effect to the use of multiple buffers within the router to accommodate longer messages.

3.3.8 Experiment #5: End-to-End Delays w/o reservation and w/ CPU and Network competition

Here, we repeat Experiment #3 with competing applications and traffic on both the end-points and within the network but without any CPU and bandwidth reservation. The round-trip delays measured are shown in Figure 11. As one would expect in the absence of any guaranteed bandwidth, the end-to-end delays are high (upto 5 to 10 times more than in Figure 8 (a)). This provides our motivation for integrated end-point and network QoS management.

3.3.9 Other Lessons Learned

A multimedia multi-party collaborative conferencing system has been implemented on top of the push-pull communication and its details can be found in [4]. We learnt several lessons during the design and implementation of the real-time push-pull layer and the *RT-Conference* system. We summarize them below:

- *Push-Pull*: The push-pull communications service made the programming of the distributed portions of the system rather easy enabling seamless communication of the various streams. Actually, the (correct) use of UDP/IP in the underlying communication layer of the push-pull model even had an unexpected side benefit. Recently, during a demonstration where *RT-Conference* was run over a real-time network which offered bandwidth guarantees, an operator error brought down the Darwin network. When the network

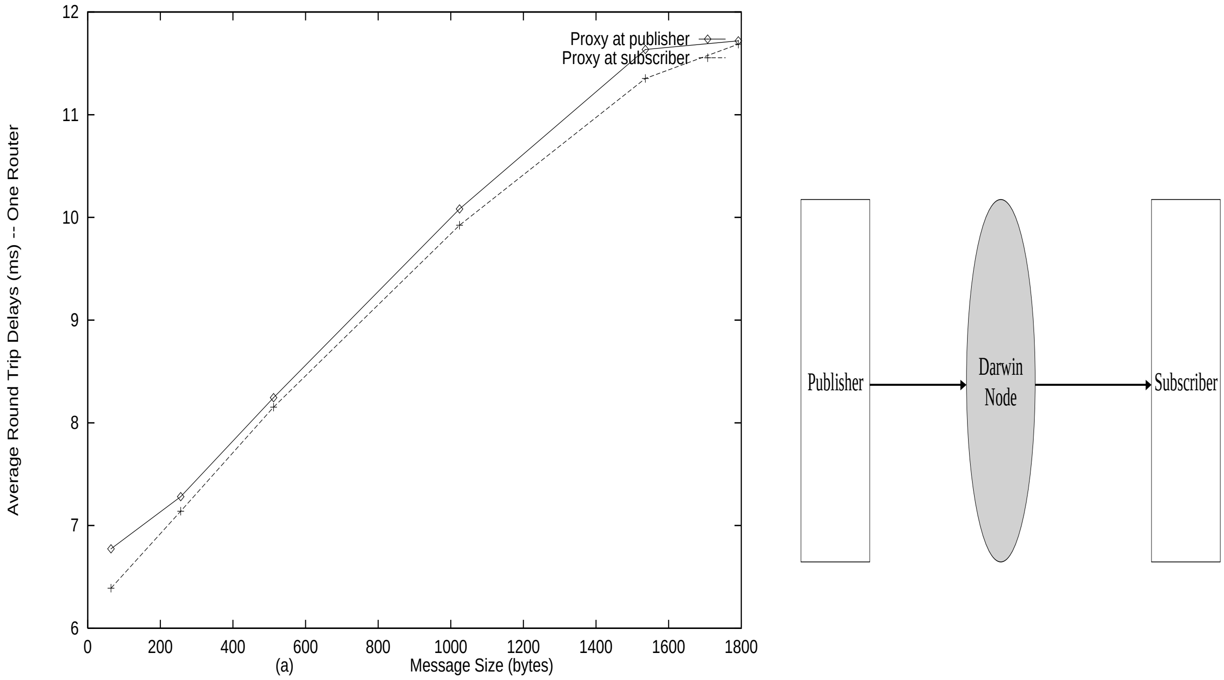


Figure 8: *The Round-Trip Delays and Publisher, Subscriber and Proxy configuration for* Experiment #3

bandwidth[3] got re-established, all video and audio streams recovered without any intervention and *RT-Conference* resumed its functioning. Thus this middleware service can avail of the underlying resource management schemes.

- *Flexibility:* The model is very flexible and can support both hard real-time and soft real-time applications. Several applications with different QoS requirements can coexist in the model at the same time. The model clearly distinguishes non real-time actions from real-time actions.

- *Priority and reservation management*: In a system such as *RT-Conference*, the priorities (or the choice of reservation periods which in turn dictates the priorities) of the various threads play a critical role. Each data type in this system has different semantics to the user, and different timing characteristics. Audio is very sensitive to jitter and is significant for interactive communications. It is therefore relatively easy to assign audio the highest priority in the system. The real-time data stream of the video game was assigned the next highest priority. The video thread was assigned the next highest priority followed by the white-board and the chat window. However, other combinations of priorities may also work depending upon available system resources and the expected frequency of usage of some data types. Hence, these parameters may actually need to be offered as options to the end-user(s).

  In addition to the threads within the client application, the underlying communication service threads must also cooperate with the application threads with appropriate priorities. It is useful to note here that daemon threads and client threads must coordinate their priority levels, else one or the other would suffer. Such situations are the logical candidates for abstractions like processor reserve, disk and network reserves [9], which provide timeliness guarantees *and* temporal protection from misbehaving threads.

- *Co-Scheduling of computation and communication resources:* Distributed applications like push-pull need both CPU and Network resources. Providing end-to-end QoS requires the correct scheduling of both the resources. Push-Pull uses Resource Kernel's CPU reservations and Darwin's HFSC scheduler to manage CPU load and link bandwidth. Without such resource management mechanisms, unbounded delays can occur as CPU/Network utilization becomes 100%. Due to limitations on paper length these results are not shown in the paper and some of them can be found in [9].

  The Push-Pull communication model makes the assumption that the network resources i.e. nodes, topology are identified *a priori*. Integrating push-pull with network service brokers to identify available resources online would be useful.

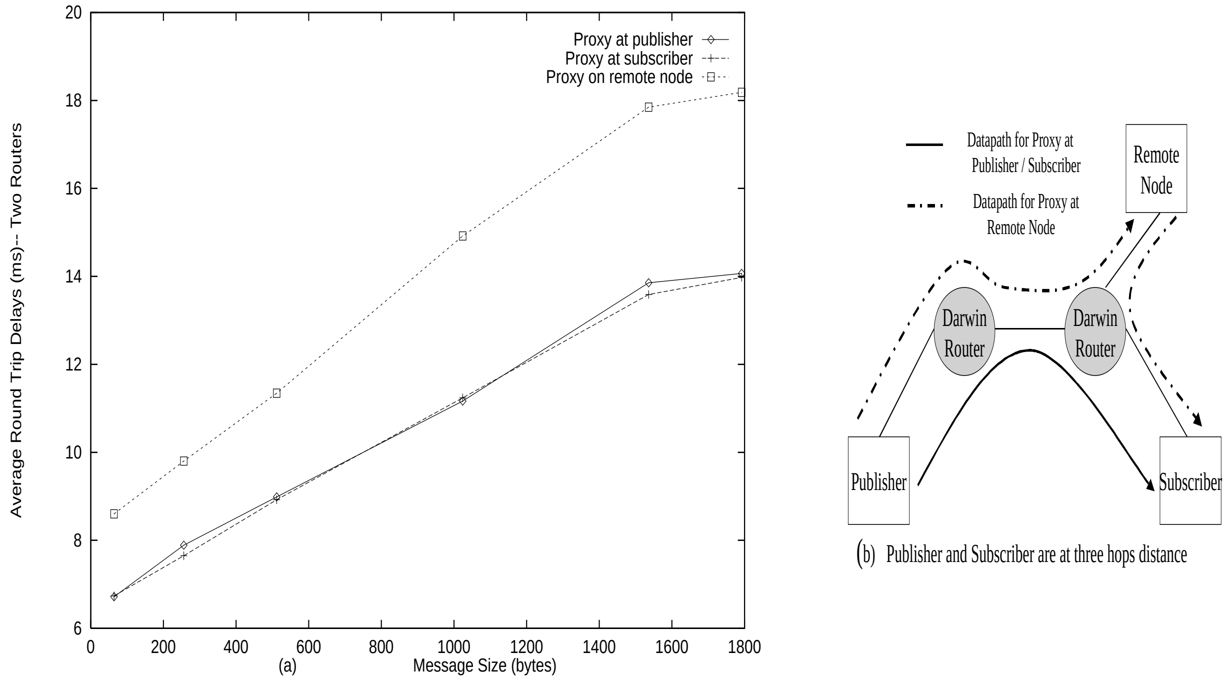


Figure 9: *The Round-Trip Delays and Publisher, Subscriber and Proxy configuration for* Experiment #4 .

4 Concluding Remarks

In this paper, we have addressed the problem of providing end-to-end QoS guarantees to subscribers operating in distributed, heterogeneous environments. In specific, we focus on providing bandwidth and delay guarantees to data subscribers using a real-time push-pull model of communications. A proxy is used to perform data transformation required for the heterogeneous subscribers. The proxy architecture plays an important role in supporting heterogeneous clients. The proxy location, the network topology and the distance between a publisher and a subscriber all directly affect the end-to-end latency obtained. We have therefore analyzed the impact (bandwidth, computation and delay) of a proxy on the publisher node and subscriber node. We have also analyzed the choice of proxy location and formulated a mixed-integer programming problem with linear constraints to obtain the optimal choice of proxy locations. Run-times for optimization in the context of the high-speed vBNS network are of the order of tens of milliseconds.

Both OS *and* network support are clearly required to provide end-to-end bandwidth and delay guarantees. Our push-pull communications service has therefore been successfully implemented as a middleware layer on top of RT-Mach, a resource-centric kernel which provides guaranteed and timely access to processor and disk bandwidth. The Darwin network provides end-to-end bandwidth guarantees. We have carried out detailed end-to-end measurements for different proxy configurations and different message sizes. The results of these detailed measurements can be used by techniques such as rate-monotonic analysis to predict end-to-end delays in real-time push-pull systems. Providing more flexible QoS guarantees for each subscriber and embedding upgradable proxies in network elements for custom data scaling are some possible future directions for this work.

Appendix A

An Exhaustive Algorithm for Proxy Allocation

In this appendix, we present a brute-force algorithm to determine the optimal proxy allocation using an exhaustive search technique and an exmaple problem problem using the algorithm. Some of the variables used here are defined in Section 2.4.

Proxy_Alloc_Alg()

for $j = 1 \cdots m$ *nodes*

  for $\bar{i} = 1 \cdots n$ *subscribers*

  compute $\delta_{i,j}$ *and* $cost_{i,j}$

  if $(\Delta_i^r > \delta_{i,j})$

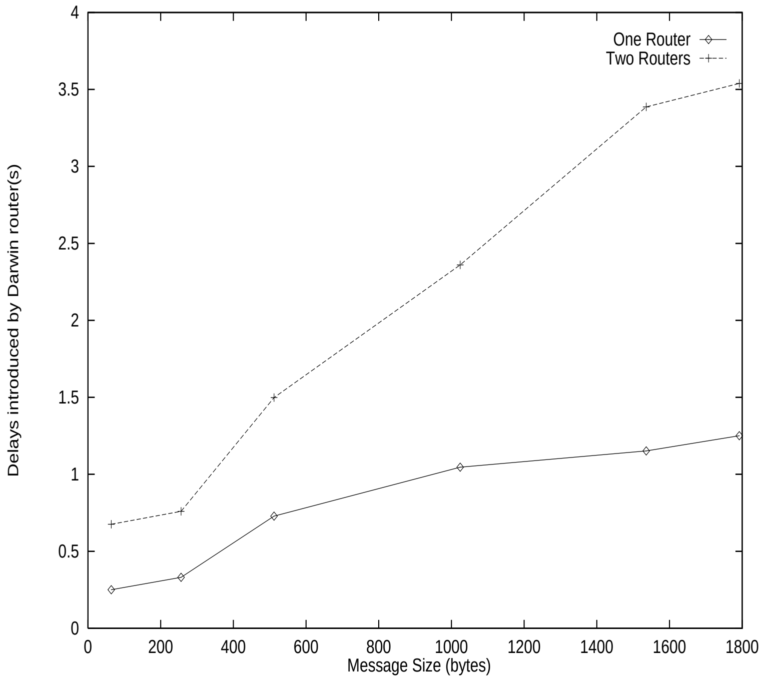


Figure 10: *Delays introduced by Darwin router(s)*

```
        TotalCost_j = ∞ /* This path is bad for subscriber i */
        path_j is infeasible. Delay constraint is violated for the path_j.
       break
     else
        TotalCost_j += cost_{i,j}
     end if
   end for; /* for loop */
end /* for loop */
```

The next step is to determine the proxy node on a least cost path that satisfies end-to-end delay for all the subscribers. This can be carried out by just finding the node j with the least value of $TotalCost_j$.

Example System We apply the algorithm described above to an example configuration consisting of 5 nodes connected in a star topology shown in Figure 12.

Let:

Number of publishers = 1; Number of subscribers = 2

Desired end-to-end delay = $2T$, where T is period of the publisher

Suppose that the one publisher is on node 1, and the subscribers are on nodes 3 and 4. From Figure 12, we have

$$d_1^{pub} = 0, d_2^{pub} = 2, d_3^{pub} = 2, d_4^{pub} = 2, d_5^{pub} = 1$$

$$d_{1,1}^{sub} = 2, d_{1,2}^{sub} = 2, d_{1,3}^{sub} = 2, d_{1,4}^{sub} = 0, d_{1,5}^{sub} = 1$$

$$d_{2,1}^{sub} = 2, d_{2,2}^{sub} = 2, d_{2,3}^{sub} = 0, d_{2,4}^{sub} = 2, d_{2,5}^{sub} = 1$$

Based on these distances between the nodes, the push delay and cost is computed as follows. Suppose the proxy is located on node 1. Since node 1 transmits at a scaled period of T' and there are two processing elements on the path (as per Section 2.2), we have the end-to-end delay for subscriber 1 as

$$\delta_{1,1} = 2T'$$

Similarly, we have

$$\delta_{2,1} = 2T'$$

Since data proportional to f' is transmitted by the publisher for *each* subscriber, the total communication costs on *all* links with the proxy on node 1 is given by[8]

$$TotalCost_1 = K * 4f'$$

[8] The link between nodes 1 and 5 will be used twice to send data to both subscribers.

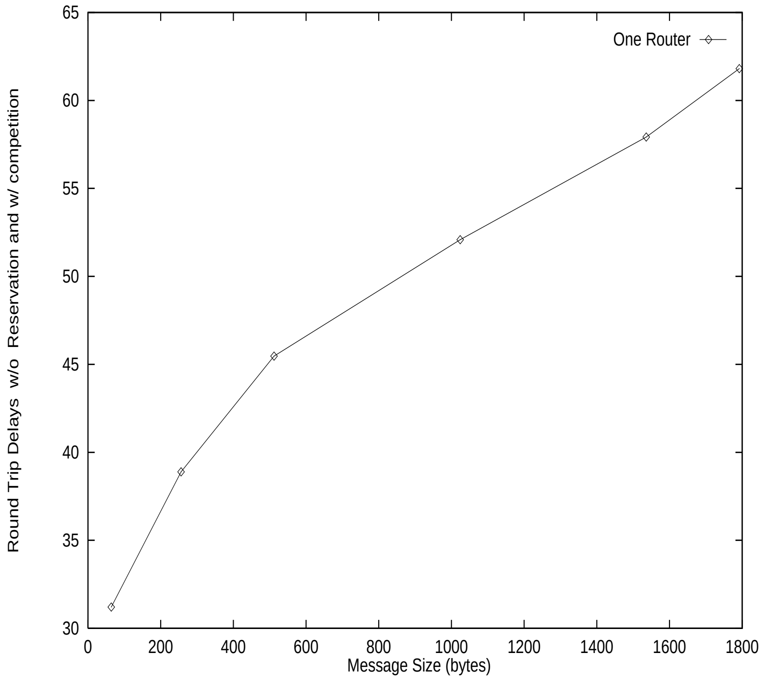


Figure 11: *Round-trip Delays w/o reservation and w/ competition*

Similarly, we have

$$\delta_{1,2} = 2T + 2T',\ \delta_{2,2} = 2T + 2T',\ TotalCost_2 = K * (4f + 4f')$$
$$\delta_{1,3} = 2T + 2T',\ \delta_{2,3} = 2T,\ TotalCost_3 = K * (4f + 2f')$$
$$\delta_{1,4} = 2T,\ \delta_{2,4} = 2T + 2T',\ TotalCost_4 = K * (4f + 2f')$$
$$\delta_{1,5} = T + T',\ \delta_{2,5} = T + T',\ TotalCost_5 = K * (2f + 2f')$$

The location of the proxy node is therefore chosen to be Node 1, as it satisfies the end-to-end delay requirements for both the subscribers and also has the least cost of $4Kf'$. As we also discussed earlier in Section 2.2, it can be seen that the end-to-end delays achieved by each of the subscribers in the case of proxy node 1 is actually higher ($2T'$) than that for node 5 ($T + T'$). A similar analysis can be applied to CPU processing costs as well. With the proxy at the publisher node, minimal additional CPU costs are incurred. With the proxy at an intermediate node, entire protocol processing costs of all published packets, and context-switching costs are incurred in addition to the normal costs of transmission. With the proxy at the subscriber node, additional protocol processing costs are involved due to the extra (unscaled) packets being received.

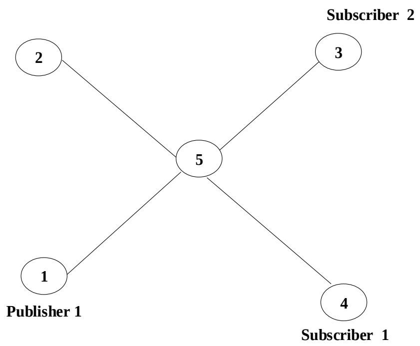


Figure 12: The Topology of the System Used in Appendix 'A'.

References

[1] Ken Birman, Roy Friedman, Mark Hayden and Injong Rhee. Middleware Support for Distributed Multimedia and Collaborative Computing. SPIE International conference on Multimedia computing and Networking,'98, San Jose, USA.

[2] Robert van Renesse, Ken Birman, Thorsten von Eicken and Keith Marzullo New Applications for Group Computing In Theory and Practice of Distributed Systems, Lecture Notes in Computer Science, Vol.938.

[3] Prashant Chandra, Allan Fisher, Coresy Cosak, T . S.Eugene Ng, Peter Steenkiste, Eduardo Takshashim Hui Zhang Darwin: Resource Management for Valud-Added Customizable Network Services Sixth IEEE International Conference on Network Protocols (ICNP'98), Austin, October 1998.

[4] Kanaka Juvva, Raj Rajkumar The Real-Time Push-Pull Communication Model for Distributed Real-Time and Multimedia Applications In Technical Report, CMU-CS-99107, Department of Computer Science, Carnegie Mellon University, Jan 1999.

[5] G. Robert Malan, Farnam Jahanian, and Sushila Subramanian Salamander: A Push-based Distribution Substrate for Internet Applications. Proceedings of the USENIX Symposium on Internet Technologies and Systems, December 1997; Monterey, California.

[6] Raj Rajkumar, Mike Gagliardi and Lui Sha The Real-Time Publisher/Subscriber Inter-Process Communication Model for Distributed Real-Time Systems: Design and Implementation In Proceedings of the IEEE Real-time Technology and Applications Symposium, June 1995.

[7] Mike Gagliardi, Raj Rajkumar and Lui Sha Designing for Evolvability: Building Blocks for Evolvable Real-Time Systems. In Proceedings of the IEEE Real-time Technology and Applications Symposium, June 1996.

[8] Raj Rajkumar and Mike Gagliardi. High Availability in The Real-Time Publisher/Subscriber Inter-Process Communication Model. In Proceedings of the IEEE Real-Time Systems Symposium, December 1996.

[9] Raj Rajkumar, Kanaka Juvva, Anastasio Molano and Shui Oikawa. Resource Kernels: A Resource Centric Approach to Real-Time Systems. In Proceedings of the SPIE/ACM Conference on Multimedia Computing and Networking, January 1998.

[10] Douglas C. Schmidt, David L. Levine, and Sumedh Mungee. The Design of the TAO Real-Time Object Request Broker, In Computer Communications Journal,Summer 1997.

[11] Anastasio Molano, Kanaka Juvva and Raj Rajkumar. Real-Time Filesystems:Guaranteeing Timing Constraints for Disk Accesses in RT-Mach. In Proceedings of the IEEE Real-Time Systems Symposium, December, 1997.

[12] Anastasio Molano, Raj Rajkumar, Kanaka Juvva. Dynamic Disk Bandwidth Management and Metadata Pre-fetching in a Reserved Real-Time Filesystem. In Proceedings of 10th Euromicro Real-Time Workshop.

[13] V. Fay Wolfe, L.C.DiPippo, R.Ginis, M.Squadrito, S. Wohlever, I. Zykh, and R. Johnston Real-Time CORBA. In Proceedings of the Third IEEE Real-TIme Technology and Applications Symposium, (Montreal Canada), June 1997.

[14] Talley, T.M., Jeffay, K. Two-Dimensional Scaling Techniques for Adaptive, Rate-Based Transmission Control of Live Audio and Video Streams. Proc. Second ACM Intl. Conference on Multimedia, San Francisco, CA, October 1994, pp. 247-254.

[15] Peter Nee, Kevin Jeffay, Gunner Danneels. The Performance of Two-Dimensional Media Scaling for Internet Videoconferencing. In Proceedings of the Seventh International Workshop on Network and Operating System Support for Digital Audio and Video, St. Louis, MO,May 1997.

[16] van Renesse, R., Birman, K.P., and Maffeis,S. Horus: A Flexible Group Communication System. Commun ACM 39, 4 (April 1996).

[17] Maeffis, S. Adding group communication and fault-tolerance to CORBA. In Proceedings of the 1995 USENIX Conference on Object- Oriented Technologies (Monterey,calif.,June 1995).

[18] L.E.Moser,P.M. Melliar-Smith, D.A. Agarwal, R.K. Budhia and C.A. Lingeley-Papadopoulos. Totem: A Fault-Tolerant Multicast Group Communication System. Commun ACM 39, 4 (April 1996).

[19] Amir, E., McCanne, S., and Katz, R. Receiver-driven Bandwidth Adaptation for Light-Weight Sessions. Usenix-97.

[20] A. Fox, S.D.Gribble,Y. Chawathe,E. Brewer, P. Gauthier. Cluster-Based Scalable Network Services. In Proceedings of the Sixteenth ACM Symposium on Operating Systems Principles(SOSP-16), Saint-Malo, France, October 1997.

[21] R. Rajkumar, C. Lee, J. P. Lehoczky, and D. P. Siewiorek. A QoS-based Resource Allocation Model In *Proceedings of the IEEE Real-Time Systems Symposium*. December, 1998.

[22] Lehoczky, J. P., Sha, L. and Ding, Y. The Rate Monotonic Scheduling Algorithm — Exact Characterization and Average-Case Behavior. Real-Time Systems Symposium, Dec, 1989.

[23] Joseph, M. and Pandya. Finding Response Times in a Real-Time System. The Computer Journal (British Computing Society,(29) 5:390-395, October, 1986.

[24] Tindell, K. An Extendible Approach for Analyzing Fixed Priority Hard Real-Time Tasks. Technical Report YCS189, Department of Computer Science, University of York, December, 1992.

[25] J. C. R. Bennett and H. Zhang WF^2Q: Worst-case Fair-Weighted Fair-Queueing. Proceeding of INFOCOM 96, March 1996.

[26] Ion Stoica, H. Zhang and T. S. Eugene Ng. A Hierarchical Fair Service Service Algorithm for Link-Sharing, Real-Time and Priority Service. Proceeding of SIGCOMM 97, September 1997.

[27] Using the CPLEX Callable Library. Using the CPLEX Base System with CPLEX Barrier and Mixed Integer Solver Options. CPLEX Division, ILOG Inc., 1997.

—oOo—