

Payoff Adaptation of Communication for Distributed Interactive Applications

Robin Kravets*

Ken Calvert[†]

Karsten Schwan[‡]

Abstract

Present distributed applications are not typically designed to adapt themselves to changes in network conditions. In addition, network-based adaptation does not typically consider the requirements of specific applications and, therefore, may take actions contradictory to the application's needs. This paper presents a cooperative solution in which a configurable communication layer is used to adapt communication in response to both application requirements and network resource availability. Adaptation decisions are based on (1) *payoff functions* which capture the requirements of the application in a functional form, and (2) *service availability curves* which represent network resource availability. Experimentation with multimedia applications and a variable reliability protocol demonstrates the benefit of using payoff-based adaptation.

1 Motivation and Research Goals

Interactive applications will continue to push the limits of available network bandwidths, processing speeds, and other computing resources. One such application is a distributed virtual environment in which many users interact in a virtual world populated with rich graphical objects, live sensor feeds and image streams, voice and video streams, and potentially costly data operations or data generation. This application and many others like it [SES⁺97, ZBS97, MST94] present challenges concerning the specification of service requirements for their multiple types of data transfer and the online management of service quality. For instance, maintaining service quality may require runtime tradeoffs in communication reliability vs. timeliness offered across competing audio, video, or image streams [DKS90, CCWP95]. This paper explores service adaptation in the context of a single data stream, in which maintaining specific service quality may necessitate application-specific formulations of different quality levels within the packets of a data stream.

Most solutions to guaranteeing services across distributed systems rely on resource reservation [ZDE⁺93, JRR97]. Complementing such research, the objective of our work is to allow applications to operate within some acceptable service specification despite potentially insufficient networking resources. Namely, we formulate a dynamic interface between the application and the communication layer that permits the communication layer to make application-specific tradeoffs in certain service parameters given its detailed knowledge of current application requirements and communication resources. Such knowledge is kept current by use of dynamic resource monitoring. As a result, application needs and effective communication resource utilization may be optimized jointly.

The principal application used to evaluate our research is a distributed virtual environment in which multiple users simultaneously explore a virtual world. The world is composed of objects that may be moved, updated, created, and deleted. More importantly, objects are not confined to simple or complex geometric models, but

*College of Computing, Georgia Institute of Technology, Atlanta, Georgia, USA, robink@cc.gatech.edu. Support for this work is sponsored in part by an AT&T/Lucent Technologies PhD Fellowship.

[†]College of Computing, Georgia Institute of Technology, Atlanta, Georgia, USA, calvert@cc.gatech.edu

[‡]College of Computing, Georgia Institute of Technology, Atlanta, Georgia, USA, schwan@cc.gatech.edu

they may also incorporate continuous data visualization, complex streams of graphical images, and audio or video streams. Audio channels may be used for communication between users or for sound associated with objects. Video channels carry video streams of representations of the users themselves as well as video streams fed from other sources such as conference rooms or animations of data visualizations.

The dynamic nature of distributed virtual environments makes it difficult to quantify a priori the network services required during execution. For example, pre-allocation of the communication resources for a video channel would not be appropriate if the displayed video stream is not always visible to users operating in the virtual environment. Similarly, dynamic establishment of a video channel as the display becomes visible to the user may lead to poor resource utilization. This happens if a video channel is established at a high quality level, despite the fact that the user may not be “close enough” to require such a high level. Another example is the establishment of multiple audio channels for communication between users. Resources for such channels should be reserved only when those channels are in use, and such a reservation should take into account dynamic variations in resource needs per channel due to application-specific tradeoffs in channel usage (e.g., when using a channel to establish different levels of awareness [HS96] among end users). Tradeoffs for audio quality vs. bandwidth can be seen as audio quality ranges from 8KHz/64Kbps voice quality transmission to 44KHz/1.4Mbps CD quality transmission. The problem of inefficient network resource utilization due to inflexible resource specifications leads us to investigate the use of dynamic application resource specification.

When a virtual environment is distributed across a network shared with other applications, it is difficult for the environment to predict the level of service it will receive from the network. For example, if the application has objects that are being periodically updated, then an increased packet loss rate may introduce an unacceptable lag between updates. If the application has some knowledge of this increase in network loss rate, the application may be able to compensate by use of domain-specific error control techniques (e.g., a reduction in image quality [CCWP95]). Problems like these lead us to investigate the use of runtime resource monitoring in conjunction with communication configuration.

The key innovative contribution of our work is a cooperative solution to dynamic service management: the communication layer uses both application resource requirements and network resource availability to determine how to best configure the application’s communications. Specifically, this paper presents an adaptive communication layer that configures the operation of its communication protocols based on the perceived benefits to the applications using it. By permitting applications to share dynamic “service quality” requirements with the communication layer, in the form of *payoff functions*, the communication layer is able to make informed decisions about the communication configuration that best suits the current state of the application. Similarly, by dynamically monitoring resource availability from the network and then describing such availability as *service availability curves*, the communication layer is able to make suitable decisions about the communication configuration that best matches the current availability of network resources.

The dynamic techniques for service management proposed in this research address a large class of distributed multimedia applications. These applications share the characteristic that they can tolerate some degree of service degradation, and are often willing to trade the reduction of certain service requirements in order to improve others. This class of applications includes video conferencing systems [AMZ95], web applications [BHD⁺96], distributed interactive simulations [HW96] and distributed virtual environments [OSF⁺97].

The remainder of this paper addresses the issues involved in communication adaptation, focusing on incorporating expanded network resource availability specification into our communication layer. Section 2 describes the related research in the area of, interactive applications operating in dynamic network environments. Section 3 describes our communication layer and the application targeted in this paper. This section also defines *payoff functions*, a mechanism for specifying value-based application service requirements, and *service availability curves*, a technique for specifying dynamic network resource availability. We then present our techniques for payoff-based adaptation of communication configurations. Section 4 describes a variable reliability protocol and the results of experiments using this protocol and the techniques described in Section 3.

2 Approach

A variety of techniques have been employed for resource specification and management in multimedia or real-time applications. Typically, these techniques assume that applications state resource needs as specific desired levels of service (e.g., as in RSVP [ZDE⁺93]) or as feasible service ranges [BG96] or regions [ZBS97]. Such specifications are then used by connection-time resource management or reservation techniques, including bandwidth reservation or transmission scheduling [DKS90] algorithms. However, as with complex real-time applications [SZ92], a distributed multimedia application can rarely accurately predict its resource requirements for extended periods of time. This results in either inadequate reservations or in poor resource utilization throughout the application's execution. These problems may be addressed with dynamic resource management or adaptation techniques, which attempt to adapt resource allocations and/or usage to the changing requirements of the application. Sample techniques include runtime task scheduling in real-time systems [SZ92], load balancing or migration for real-time [RSYJ97] or scientific [SE⁺95] applications, and application-level adaptation in distributed [NSN⁺97] or parallel [BS91] programs.

Static, reservation-based schemes differ from adaptive resource management in that static schemes attempt to guarantee the availability of resources when they are needed. Unfortunately, the cumulative worst case needs of complex dynamic applications often result either in infeasible statically computed reservation requests or in oversized reservations that exhibit poor average runtime utilization of reserved resources. Additionally, once reserved, resources become unavailable to other applications that may need them. The dynamic use of reservation-based schemes, by acquiring and releasing resources throughout an application's execution, also cannot solve these problems, since it is impossible to guarantee that once released, resources will again be available when they are needed. In addition, reservations tend to be costly, so that their frequency of use must be minimized, as evidenced by the connection-based reservations prevalent in protocols like RSVP. Finally, a reservation-based approach may be too costly or infeasible for highly shared and large-scale systems like the Internet.

A number of researchers address the issue of resource management through the dynamic adjustment of bandwidth requirements. In [FGBA96], bandwidth levels are modified through the use of on-demand data specific loss compression or distillation, with the goal of reducing bandwidth requirements at the application level. Similarly, [MJS97] describes the use of filters, synthesizers and harmonizers, and schedulers to manipulate the application level bandwidth being used. In comparison, our techniques combine application level adaptations with network level communication adaptations to determine the effectiveness of the suggested configuration given the current state of the network. Additionally, we provide a mechanism that allows the application to give feedback about the benefits of different adaptation choices. Although it has been suggested that decisions about the benefit of adaptation as well as when adaptations should occur should be left to the user [LSD98], we claim that providing the application, and hence the user, with the correct tools to indicate such preferences can reduce the complexity on the application and ease the burden on the user.

The adaptive solutions promoted in our work may be used throughout the execution of a distributed, interactive application. Consequently, the decisions being made are based on current information on resource availability and on application needs. Our implementations of these solutions exhibit small runtime overheads since they make and enact adaptation decisions locally for each communication channel used by the distributed application. As a result, adaptation decisions may be made at rates that match the frequency at which an application changes its communication behavior. Furthermore, by separating the capture of the resource information from the decision making and adaptation processes, we may perform resource monitoring at a rate that captures major changes in resource availability but may differ from the adaptation rate. By stating application requirements and resource availability in functional forms, we can describe trends in resource availability and tradeoffs in application requirements, both of which tend to change more slowly than actual current values for available resources or current requirements. The resulting "rich" interfaces for specification of application requirements and resource availability also contribute to the low adaptation overheads experienced in our solutions.

As an argument for the utility of functional interfaces for specification of requirements and resource avail-

ability, consider a resource reservation scheme that provides the application with a simple yes/no answer as to whether some desired level of service may be met. In this example, the application has to probe the reservation agent multiple times to determine some feasible level of service. By increasing the richness of this service interface, a more detailed response may be given, possibly involving feasible service ranges or even tradeoffs in the effective service the network can provide to the application based on the network-level resources utilized. For example, a resource reservation scheme based on statistical reservation would be able to indicate to the application that it could provide a lower bandwidth service with a low probability of any loss all the way up to a high bandwidth service with a high probability of loss. This type of information would provide the application with a more useful picture of resource availability and would allow the application to determine what kind of tradeoffs it can make and the benefits that result. These tradeoffs become the key to effectively using the resources available to the application. By providing mechanisms that will allow applications to evaluate current resource availability, the applications can adapt to suit their requirements.

3 Problem Domain and Description

Our work addresses applications with dynamic service requirements that operate in environments with unpredictable resource behavior. We have developed techniques for quality of service specification which can be used as the basis for dynamic communication adaptation. Using these techniques, we show how configurable communication protocols can be used to adapt communication to changes in application requirements and network resource availability, trading, for example, reliability for delay. By considering dynamically configured applications, we are adopting the typical application model used by a growing set of researchers and developers concerned with attaining high levels of service quality in the presence of resource limitations.

This section first introduces our communication layer and one of the target applications for which it is intended. Next, we present *payoff functions*, a service specification technique which enables the application to place its own quantitative *value* on selected communication parameters. By using payoff functions in conjunction with dynamic resource information described by *service availability curves*, tradeoffs may be made across competing communication parameters. Benefits are realized through runtime reconfiguration of application communication parameters.

3.1 Communication Layer

Our approach defines the architecture of an end-to-end *communication layer* which mediates between an application and a network service, as depicted in Figure 1. The communication layer provides a mapping from the services available from the network to the services required by the application. The application submits data units to the end-to-end communication layer for transmission; the communication layer processes them and then interacts with the network service to transfer the information through the network. The communication layer may then process the transferred information further on the receiving side before ultimately passing data units up to the application. Thus, the communication layer implements a channel using the given network service, and makes it available to the application. The characteristics of this channel — as quantified by measures such as throughput, delay, delay jitter, loss rate, and cost, and by semantic characteristics such as order preservation — determine its *value* to the application.

Our goal is to design the communication layer to adapt its behavior to maximize service value in the face of changing application needs and network characteristics. We can view this as an optimization problem, the structure of which is depicted in Figure 1. The communication layer has access to several sets of interrelated variables, some of which it can control. The relevant sets of variables are:

- The application's *offered behavior*, which traditionally is quantified by parameters such as rate and burstiness of data generation, but may also be characterized by information about the type and relative importance of data units being transferred. The application controls the offered behavior.

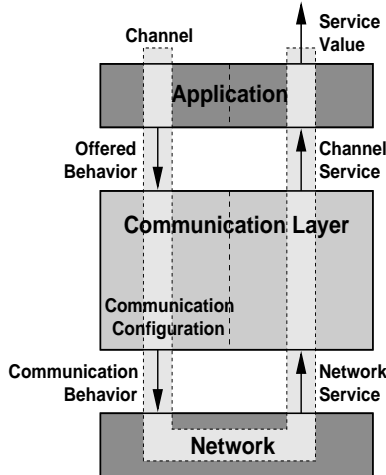


Figure 1: Communication Architecture

- The communication layer’s *communication configuration*, which captures the set of protocols and mechanisms used by the communication layer to enhance the service received from the network. For example, this variable captures whether forward error control or retransmission is used for error recovery. The communication layer controls the communication configuration directly.
- The communication layer’s *communication behavior*, i.e. what the network “sees” from the communication layer. This includes traditional QoS dimensions such as peak and mean transmission rate, burst length, etc. The communication layer controls the communication behavior directly.
- The *network service*, i.e. what the communication layer “sees” from the network on the receiving side. This is quantified by parameters including loss, delay, jitter, and cost. The parameters for network service are determined by the network, but are affected directly by the communication layer’s communication behavior. For example, an increased transmission rate over some interval may result in an increase in cost over that interval.
- The *channel service*, i.e. the behavior actually seen by the application. This will include quantities similar to network service, but measured at the interface between the communication layer and the application. The channel service is a function of communication configuration, communication behavior and network service.

In general these parameters will represent system behavior over some time interval, rather than instantaneous values. For the purposes of this discussion, we consider them single-valued (i.e. points in some multidimensional space).

The channel’s *service value* is a function of the channel service parameters. In order to solve the optimization problem of maximizing service value, the communication layer examines three relationships:

- The relationship between channel service and service value. We assume this is defined by a function provided by the application. This function defines the quantity to be maximized.
- The relationship between network service and communication behavior. In general, this can be defined by a curve (or plane) giving the value of the communication behavior that can be expected for each value of network service. We assume that the network service provides this curve.
- The relationship between communication configuration, communication behavior, network service and channel service, i.e. how the channel service delivered to the application is affected by the communication layer’s protocol configuration and communication characteristics and the network service. For

example, the use of retransmission as an error control strategy can decrease loss rate, but may increase delay. The use of forward error control, on the other hand, can decrease loss at the expense of increased bandwidth. (The assumption is that increasing delay or bandwidth decreases value.)

The characterization of this last relationship is a major challenge in the design and implementation of the communication layer. It might be realized through analysis, simulation, measurement, or (most likely) some combination of all three.

After discussing the target application, the remainder of this section describes an empirical approach to the implementation of such a communication layer. Section 3.3 discusses our approach for specifying the value of the channel service to the application. In Section 3.4, we address some of the issues involved in network service specification. And in Section 3.5, we present our techniques for using the information provided by these relationships to adaptively configure the communication.

3.2 Target Application

One specific application driving our research is a distributed virtual environment operating across the Internet. The environment was designed as a collaboration tool that supports concurrent interactions among multiple users. The environment permits users to navigate through a virtual world organized as multiple rooms. The virtual world as well as the objects in the world are represented using graphical, audio, video, and data visualization techniques. Users interact with objects in the world, with the world itself, and with the other users in the world. Rooms are separated by walls, but there may also be doors and windows present that allow users to see parts of other rooms or hear pieces of conversations being conducted elsewhere. Objects themselves may be stationary, movable, or they may even move on their own (i.e. other users or autonomous agents). Objects may be simple or complex graphical representations, still or motion images, or data visualizations, and may have associated audio.

The dynamic nature of the virtual world coupled with runtime changes in users' interests result in frequent and dynamic changes in users' service requirements. For example, while a user is relatively "far" from a video monitor visible in the environment, there is no need to offer high resolution real-time images for display on this monitor. However, both display rate and image quality must be improved to realistically model the fact that the user can see the image more clearly when approaching approaches the monitor. In [OSF⁺97], the idea of *Importance of Presence* is introduced to help determine the service requirements of objects in a distributed virtual environment. Due to the fact that each user in the virtual world may have different interest levels regarding different objects in the world, it is not beneficial to multicast all of the data to all of the users. One suggested solution to this problem is the use of layered multicast techniques [AMK97], where data is split into multiple layers and receivers can determine what layers they want to receive. Our approach looks at this problem from a different perspective, by allowing receivers to specify two additional pieces of information. First, individual receivers can specify different layerings of the data. This can correspond to different focus areas in the transmitted data. Second, we allow receiver-specific specification of communication protocols, as well as the configuration of the specific protocols, to be used for each such layer. For example, different receivers may receive the same data at different reliability levels or transmission rates. We take this approach due to the expected diversity of user's interests and requirements.

The communications in our distributed virtual environment fall into three categories. The first category includes data that must be transmitted reliably, including control messages and the one time transmission of some object descriptions. This type of data is very sensitive to packet loss and its transmission reliability must not be sacrificed. The second category is composed of data streams, where data must be segmented into multiple messages in order to be transmitted. Examples include changing texture maps, sensor inputs or continuously generated scientific data. This type of data often offers some level of redundancy. At the application level, it may be possible to tolerate data loss or compensate for it by inferring the lost data from other messages that are "spatially" close to it. For example, a lost message in an image can often be compensated for by examining the image surrounding the lost message and performing some type of

averaging function. Such recovery has limitations in the number of “nearby” or consecutive data losses it can tolerate. The third category consists of continuous streams of relatively small messages, each of which is an update to some specific application data. Examples include object position updates or tracker updates, which have the property that, if a message is lost, the next message will provide the newest information for the data [BS88]. Due to this property, there is little tolerance for waiting for retransmissions, since much of the lost data can be approximated from the next message much faster than waiting for the retransmission. Although continuously updated, this type of data is very sensitive to loss due to the effect on the lag perceived by the user.

Some types of data cover multiple categories. One specific example is real-time video. Each video frame is a large data set and needs to be segmented into multiple messages. New frames are continually being transmitted and the loss of some data from one frame may be tolerated due to the fact that the next frame will provide an updated image. Real-time video transmission also introduces many issues involving timing constraints that we do not address in this paper. Instead, this paper focuses on issues involving the transmission of large data sets that may have soft timing issues, but are not time critical. This aspect of data transmission is still applicable when considering the transmission of the individual frames of a video stream. Our distributed virtual environment currently supports video objects, where images are captured from a video camera in real time and transmitted to interested users. We are currently investigating the effect of using our payoff-based techniques to determine appropriate update rates for the video streams in our distributed virtual environment.

The techniques described in the remainder of this section address the transmission of large data sets, thereby targeting a large class of common multimedia applications. The third class of data, continuously transmitted updates, will be addressed in our future research.

3.3 Application Service Specification

Service requirements in virtual environments vary for each user and across multiple users. For instance, in the virtual world organized as a set of rooms, at any one time a user may only be interested in one specific room, perhaps with cursory interests in surrounding rooms. We believe that it is important that knowledge about application-level service requirements be available when configuring the communication for the application. Toward this end, we support the user-level specifications of the quantitative *value* of parameters of the communication service to the application. To clarify, consider a channel defined by the transmission of an image of interest to the user. Clearly, the quality of the image and, therefore, the resolution at which it must be transmitted is directly related to the user’s current level of interest in the image’s contents. In this case, the actual “service value” ascribed to the channel may be computed with some application-specific “value function” that uses parameters quantifying the user’s current level of interest (e.g., distance from the image in the virtual world) and quantifying the actual level of service at which the image is transmitted by the network. The value of the service to the application may dynamically change, so each value function represents the value to the application at a specific time or over a specific time period.

With a “value function”-based specification of desired service quality, it is possible to capture the tradeoffs the user is willing to make to realize certain levels of channel service. Typically, such tradeoffs represent relationships between conflicting service parameters. For example, in a lossy network, transfer rate may be sacrificed for high reliability data transfer. The remainder of this paper uses particular value functions, called *payoff functions* to study the utility of value functions in making such tradeoffs. Payoff functions are associated with specific channels, and each function states the value to the application of receiving a certain level of service via this channel. Specifically, from the application’s point of view, the payoff function evaluated at a specific channel service represents the benefit gained by the application of achieving that specific channel service. Many such functions exist (often referred to as utility functions[She95]) and may be formulated for the diverse multimedia applications considered in our work. In [JLT85] and [DKT+93], such functions are formulated in the context of real-time application deadlines. [AS98] uses “reward” functions to capture similar application-specified service valuations, but limits these specification to “poor”, “good”

and “excellent”. In comparison, our approach provides a more general interface for service valuation.

Figure 2(a) depicts two sample payoff functions addressing transmission reliability. The plot for Image 1 represents a payoff function for an image that has been requested at a range from 100%-75% reliability. The shape of the curve within this range represents the relative value to the application of achieving that level of reliability. In this example, the application prefers 100% reliability and will still be somewhat satisfied at 90%, but will be less satisfied as the reliability approaches 75%, as expressed by the rapidly diminishing payoff values. Payoff is zero at the lower limit of acceptable reliability. In comparison, if the image can be accepted with reliability ranging from 40%-80%, then the maximum payoff will be at the 80% reliability level as shown in the plot for Image 2 in Figure 2(a). If the image is transmitted with more than 80% reliability, there is no additional payoff. Payoff diminishes until it reaches the limit of acceptable quality, in this case 40%. Payoff values are normalized to the range of 0 to 1 to permit comparisons between payoffs stated for different service parameters.

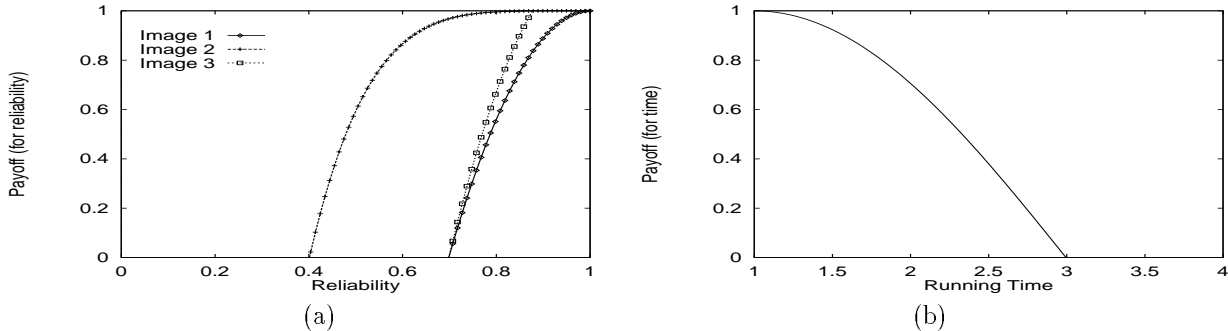


Figure 2: Sample Payoff Functions for Reliability and Time

The specification of payoff for one service parameter (e.g., for reliability in Figure 2(a)) does not capture the tradeoffs the application is willing to make between that parameter and others. To state such tradeoffs, the application must supply payoff information for a service parameter which represents some notion of “cost” to the application. For illustration, again consider transfer time vs. reliability. In lossy networks, an increase in reliability will “cost” an increase in transfer time due to retransmissions. Figure 2(b) represents a payoff function for transfer time, where the application has requested a transfer rate from 1-3 time units, and has now specified payoff functions for two competing service parameters. Unrecovered losses will cause the payoff for reliability to decrease, but recovering those losses will increase the transfer time and decrease the payoff for transfer time. By supplying payoff functions for both parameters, the application provides the information necessary to determine how to find optimal operating points.

Payoff functions may also be used to represent the comparative value between data units. Consider two images of interest to the user, where the payoff functions for reliability for both of these images are those depicted by the plot for Image 1 in Figure 2(a). Such a formulation is adequate if the interest level of the user is the same for both images. In the case where one image is of less interest, payoff for that image may be adjusted, as shown in the plot for Image 3. For this image, the payoff function expresses that maximum payoff is achieved for a lower reliability level as compared to Image 1, but the function still captures the shape of the original curve and also maintains the original minimum acceptable reliability level. Using these functions, the negative effects of transmitting Image 3 at a low reliability level are less than those experienced for Image 1.

The utility of payoff functions should be clear from the examples presented above: they may be used to impart to the providers of communication services application-level semantics with which differences in requested vs. delivered levels of service may be evaluated. In general, there is a payoff function, $P_d^p(l)$, for each service parameter, p , associated with each data, d , transferred at a service level, l . In this paper we focus on the service parameters *observed reliability* and *transfer time*. Given such service parameters and the payoff functions formulated in this section, we next discuss how to describe network resource availability

appropriately so that payoff may be maximized for these services.

3.4 Communication Resource Specification

If the communication layer does not have knowledge of the available network service, it cannot adjust communication behavior appropriately. For example, a user may be receiving both data updates and continuous video. When the available network bandwidth is reduced, both communication streams will be negatively affected, one by dropping frames, the other by losing or delaying data. Such behavior is undesirable, especially if it can be remedied by lowering the quality of the video stream in order to reduce bandwidth needs. Similarly, when increased network bandwidth becomes available, the communication layer should be able to respond by improving video quality. In general, then, opportunities for optimization are created by exposing network service information to effect adaptation of channel service.

The optimizations considered in our work explore the manner in which the communication layer can realize benefits for the application using knowledge about the relationship between requested vs. experienced channel service. For example, the communication layer may be supplied with a set of transfer rates and their respective loss rates, which jointly describe the current behavior of the available network service. In general, this implies that the communication layer has knowledge about what quality of service to expect for a range of different service requests. This relationship can be represented with *service availability curves*, as proposed in the form of Loss-Load Curves [Wil96], which characterize the dynamic service a network can provide to its clients. Given a specific transmission rate for a sender, loss-load curves provide the sender with an expected loss rate. This information allows the communication layer itself to determine the tradeoff between higher output rates and higher loss rates. When service availability curves are not available, communication resource availability may be determined by the communication layer itself, perhaps by gathering statistics based on recent communication history [BMR97] and by projecting such behavior into the near future [Bo].

The sample loss-load curve depicted in Figure 3 exhibits an increasing loss rate as the communication layer increases its transfer rate. In this example, the network is given a set of possible transfer rates and supplies the respective loss rates. A 10% loss is imposed for a transmission rate of 400KBps and increases 5% for each 100KBps increase in transmission rate.

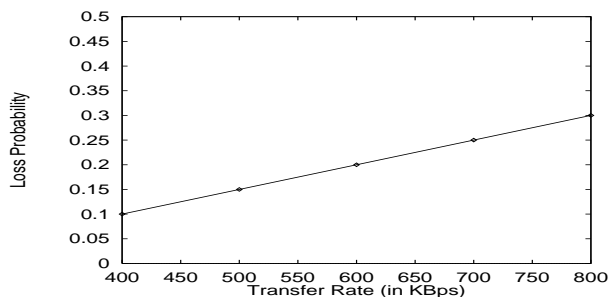


Figure 3: Example Loss-Load Curve

The specification of communication resources spans many parameters, including but not limited to delay, loss, jitter, available bandwidth and eventually some measure of monetary cost. The techniques described in this section provide a general approach to communication resource specification that is flexible and abstract enough to express current known and potential future communication service parameters.

3.5 Communication Adaptation

Payoff and service availability functions provide the communication framework with extensive information about an application’s service requirements and the network’s service availability. Now we consider the

use of such information to configure communications during application execution. Specifically, we employ *payoff adaptation* to optimize the use of available communication resources. The payoff adaptation method employed in this work is reactive. Specifically, it responds to changes in application requirements and network resources rather than attempting to anticipate them, as in predictive [BS91, CKV93] and learning-theory based adaptation methods [KLV95, KLP+95]. Payoff adaptation techniques allow us to balance application requirements like quality level and running time against resource availability. Resource information may be defined as execution time, available bandwidth, or message loss rate.

Two stages of adaptation. Payoff adaptation involves two stages. In the first stage, the communication layer determines a communication configuration for a specific point in time, given current application requirements and network service availability. In the second stage, the communication layer monitors changes in application requirements and network services. The effects of such changes on the service value are considered to determine a new configuration.

1. *Using payoff functions to determine resource requirements.* This first stage of payoff adaptation uses the application-supplied payoff functions to evaluate application resource requirements. Given a fixed application resource specification and a fixed network resource specification, payoff adaptation considers a range of possible operating parameters and chooses those that maximize the service value to the application.

As an example, again consider the transfer of an image in the virtual environment. For this example, the application has specified two service parameters: ObservedReliability (OR), which captures the amount of the original data transmitted that is actually received, and TransferTime (TT), which captures the end-to-end transfer time for application data. To determine the maximum payoff, the communication layer executes the following steps:

- (1) *for* ($i = \text{each ReliabilityLevel}$) *do*
- (2) *for* ($j = \text{each TransferRate}$) *do*
- (3) $\text{LossRate}_j = F_{\text{LLCurve}}(\text{TransferRate}_j)$
- (4) $\text{ObservedReliability}_{i,j} = F_{\text{or}}(\text{ReliabilityLevel}_i, \text{LossRate}_j)$
- (5) $\text{TransferTime}_{i,j} = F_{\text{tt}}(\text{ReliabilityLevel}_i, \text{LossRate}_j, \text{TransferRate}_j)$
- (6) $\text{TotalPayoff}_{i,j} = Q(P^{\text{or}}(\text{ObservedReliability}_{i,j}), P^{\text{tt}}(\text{TransferTime}_{i,j}))$

To evaluate individual service parameters, we need information about the relationships between multiple communication variables as discussed in Section 3.1. This relationship may be obtained via a function which maps communication configuration, communication behavior and network service into channel service. In general, such interrelationships may be collected using profiling techniques prior to running the application, from an analytical model, or they may be gathered by the application during execution (e.g., based on recent history). In the context of our example, the information supplied by the loss-load curve in Figure 3 provides the communication layer with a mapping from transmission rate to loss rate, as shown in step (3) above. The functions F_{or} (step (4)) and F_{tt} (step(5)) supply the mapping from reliability level (communication configuration), transmission rate (communication behavior) and loss rate (network service) to the channel service parameters. These functions calculate $\text{OR}_{i,j}$ and $\text{TT}_{i,j}$, representing the service parameters evaluated at the specific reliability level, i , transmission rate, j , and loss rate.

In the following step, the communication layer evaluates the benefit to the application of operating with these service parameters. As discussed in Section 3.3, there is a payoff function, $P_d^p(l)$, for each service parameter associated with a data and service level pair. Since the application indicates multiple service parameters, the payoff for each of these parameters must be considered to determine a net payoff (step (6)). In other words, the net payoff, P_{total} , is a function, Q , over all the defined $P_d^p(l)$, where p represents the service parameter, l represents the service level and d represents the specific data in question. In the case of an image with two payoff functions, one for reliability and one for transmission time, the payoff for the transmission of the image is determined by evaluating the payoff functions, $P^{\text{or}}(\text{OR}_{i,j})$ and $P^{\text{tt}}(\text{TT}_{i,j})$, at a specific reliability level, i , and transmission rate, j . The function Q , which defines the composition of payoffs for multiple application parameters, is supplied by the application. The application may also use a default

composition function, such as multiplication. Multiplication has the advantages of retaining payoff values between 0 and 1, as well as causing total payoff to go to 0 if any individual payoff is 0.

These steps are repeated to determine the payoff for each possible operating point (as indicated by the loops in steps (1) and (2)). From these operating points, the communication layer chooses the reliability level and transmission rate that maximizes total payoff for the application. Our approach to choosing an operating point given payoff and resource availability functions has the following advantages:

- It is extremely flexible because it makes no assumptions about the form of the payoff and resource availability functions, treating them as "black boxes". The functions might be analytical formulas, graphs (sets of ordered pairs), or even programs in a suitable language.
- The cost of computing the operating point is simply determined by the cost of evaluating all the functions at a single point, times the number of potential operating points considered. Thus, it should be relatively easy to predict the cost of determining an operating point.
- There is an obvious tradeoff between the optimality of the chosen operating point and the granularity of approximation, i.e. the number of operating points considered.

Given payoff and/or resource functions that have certain characteristics (e.g. continuous and monotonic), it may be possible to eliminate potential operating points from consideration.

2. Adapting to parameter changes. The second stage of payoff adaptation involves adapting to changes in both application requirements and network resources. As application requirements change, the application may provide the communication layer with new payoff curves for its data, or it may simply change the characteristics of its communication. At this point, the communication layer must reevaluate the communication configuration and determine the need for reconfiguration. Reconfiguration decisions may be caused by changes in network resource parameters, which may be provided by the network in the form of a new loss-load curve, or derived from observing the network.

The benefits of using payoff-based adaptation are derived from the richness of the information provided by payoff functions to the communication layer. The communication layer is given a wide range of resource allocations that will satisfy the application and has the information necessary to determine what inside this range would best benefit the application. This freedom allows the communication layer to base adaptations on the requirements of the application. In comparison, adaptation schemes that use specification ranges or regions have the freedom to work within those areas, but have no guidance as to what would benefit the application. We can mimic such functionality by setting the service levels within the range or region to 1 and the payoff for the area outside the range to 0.

4 Evaluation With A Variable Reliability Framework

Historically, applications had to choose between completely reliable message transfer (as with TCP) and "best effort", non-guaranteed service (as in UDP). This presents little choice to applications able to handle relaxed levels of reliability. Instead, applications may have to pay the price of using a reliable protocol, which can be measured as the unnecessary retransmission of messages by the sender and as the amount of buffer space for buffering messages at the sender and the receiver. Alternatively, applications may use unreliable protocols, which may result in unacceptable losses.

Our research contribution is a framework for implementing variable reliability communication. The use of this framework provides applications with opportunities to realize performance gains by exploiting tradeoffs with respect to the reliability of message communication. Specifically, with the framework, an application can express its data's reliability requirements more precisely. This information can then be combined with feedback from the network resources being used. The application can use such feedback to adjust its requirements during execution, thereby improving its use of available network resources. Preliminary experimental

results demonstrate that this approach leads to improved application performance compared to using reliable communications.

Our techniques are being evaluated in the context of a distributed virtual environment. The experiments discussed in this section address the reliability requirements of large data sets within such an application. Specifically, some of the graphical objects in the environment are “continuous objects” in the sense that their display requires continuous updates to images or texture maps or even motion JPEG. This section first defines the problem within the context of the reliability of data transfers. We then specify and explain the details of our variable reliability protocol. Finally, we present the results from three sets of experiments which show the effects of using the variable reliability protocol within our communication resource framework in conjunction with payoff adaptation.

4.1 Reliability and Communication

It is well-known that for multimedia applications the two types of reliability provided by TCP and UDP are insufficient. This is due to the fact that most such applications are able to tolerate some application-specific losses. Given a lack of choice, such applications must use reliability even when they do not need it or implement their own versions of reliability on top of an existing unreliable protocol. Marasli, Amer and Conrad [MAC96] quantify the costs of “too much” or “too little” reliability within the specification of their protocol. Similarly, Delgrossi, et al. [DHH⁺93], evaluate the cost of using standard reliable sliding window protocols like go-back- n and selective retransmission in situations where reliability could be relaxed. In comparison, our approach is to provide a framework in which each application can define its own notion of reliability. Additionally, we allow the application to define different levels of reliability for data with different requirements.

Implicit in our design is the concept of *application layer framing* (ALF)[CT90]. Namely, we assume that an application can deal independently with the data units it is sending. In other words, each unit of data sent by the application contains sufficient identifying information to allow the application to process it. Among other things, this means that data need not be held up for ordering constraints (beyond those defined by application framing boundaries – as explained below).

Reliable data transfer can be defined as a service that guarantees the delivery of data sent from a sender to a receiver without duplication, loss, or out of order messages. Unreliable data transfer makes no guarantees as to duplication, loss, or order. However, it is not obvious how to define a service “between” these two extremes. The following sections discuss the issues involved in defining such a service and describe a specific implementation of a variable reliability protocol.

4.2 Specification of Variable Reliability

Variable reliability mechanisms can be applied to realtime and non-realtime data. This section addresses non-realtime data (i.e., data that is time-sensitive, but not time-critical). Delgrossi, et al. [DHH⁺93] and Papadopoulos and Parulkar [PP96] address reliability for realtime continuous traffic streams. This difference in focus permits us to ignore lifetime issues for individual messages and concentrate on issues concerning the amount of data received and the spacing of message losses. We target applications where the structure of the data is important and where timing issues are based on frame rather than individual message transmission times. This type of specification can be particularly useful for applications running over low-bandwidth and/or high error rate networks (e.g., wireless networks or the Internet).

The scheme we specify below defines a service based on simple loss measurement and retransmission policies. The protocol specified by Marasli, Amer and Conrad [MAC96] provides probabilistic reliability guarantees based on the number of retransmissions of a message. In contrast, our protocol provides applications with hard guarantees about reliability based on specified loss allowances. The following describes some parameters

of our variable reliability protocol.

Loss Tolerance: Two mechanisms govern loss tolerance for our variable reliability protocol. The first is the definition of *application-specified data boundaries* or *data frames*. A data frame consists of data messages, each of which is an *application data unit* (ADU). The messages from different frames are handled separately, thereby limiting the effects of problems in the transmission of one frame on the transmission of others. Since a data frame can also be considered an ADU, this partitioning of data frames into data messages provides a two level hierarchy of ADUs, where the ADUs in each level can still be process independently of other ADUS of that level. The second mechanism is a simple *sliding window* within the messages of the data frame. The use of these mechanisms is controlled by the specification of suitable loss tolerance parameters.

Two parameters may be specified by the application at the data frame level. The first parameter is the *maximum number of consecutive losses*, which essentially defines the tolerable ADU loss burst size. The second parameter is the *high level loss percentage*, which indicates how much loss in the total data frame the application can tolerate. At the sliding window level, the application can define the *maximum number of losses allowed in a window*. These parameters are similar to those specified by Gong and Parulkar [GP92].

In Figure 4, the maximum number of consecutive losses is 2, which means that there will never be more than two consecutive losses; the high level loss percentage is 50%, which means that the receiver is guaranteed to receive at least 50% of the ADUs in each frame; and the maximum number of allowable losses in a window of size 6 is 3. The combination of the maximum number of allowable losses and the maximum burst size ensures that the losses are spread out through the messages of the data frame and not grouped together in one portion of the frame.

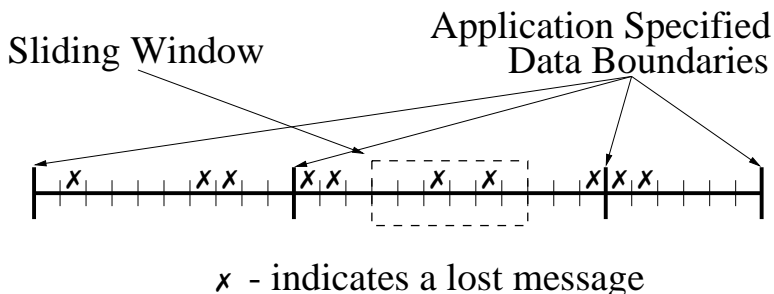


Figure 4: Example of Allowable Loss in a Sliding Window and in a Data Frame

Application-Specified Data Boundaries: The application can specify how many data frames it can handle simultaneously. The protocol will then guarantee that the data being passed to the application will always belong to appropriate frames. Given that some number, n , of frames may be transmitted simultaneously, we define a sliding window on data frames. If the sliding window size is n , at most n frames will ever have ADUs in transmission at any given point in time. The sender will not start to send the $n + 1$ st frame until the first frame has been “successfully” transmitted, where “successfully” is defined in accordance with the specified reliability. Messages from old frames will be discarded on the receiving side.

4.3 Variable Reliability Protocol

The use of a sliding window mechanism drives the protocol’s implementation. In a sliding window protocol, decisions on when to ask for specific retransmissions are made by the receiver. These decisions are made based on the policy defined for the specific protocol. For example, the policy for a reliable protocol would be to ask for a retransmission of all messages determined as lost by the receiver. In such a protocol, an ACK indicates that the message has been received successfully by the receiver. But this definition is a policy decision. For our purposes, we would like to say that an ACK indicates that the sender no longer needs to buffer or retransmit the ACKed message; thus, depending on the policy used in a given protocol, the sender

may receive ACKs for messages that were never received. The key is that the receiver controls the ACK policy.

In our variable reliability protocol, the receiver tracks messages using two information sources: the *receive window* and the *receive history*. The *receive window* indicates the range of *active* messages, where an active message is a message that may still be accepted and buffered by the receiver. Information about the reception of messages is maintained in this window in order to prevent the passing of duplicate messages to the application. The *receive history* is used to determine if a loss in the receive window can be allowed; messages in the receive history are no longer active, and they will not be accepted by the receiver. This is because message losses in the receive history have already been “allowed”. Additionally, the receiver assumes that messages received out of order are lost. This assumption permits simplification of the protocol at the cost of some unnecessary losses and retransmissions.

The protocol uses cumulative ACKs and NAKs. When the receiver detects a lost message, it checks to see if that loss violates any loss tolerance parameters. If so, the receiver sends a NAK for that message. If not, the message will be acknowledged in the next timer-generated ACK. Due to the fact that the loss of the last message in a window or frame may go unnoticed by the receiver, the sender maintains a timer for each message, which, on expiration, causes the message to be retransmitted. A number of techniques may improve protocol performance for this kind of occurrence, including indicating the number of messages in a data frame or sending markers after the last message in a data frame.

Figure 5 shows an example of the receive window and the receive history during execution. In this example, the receive window size is 10 and at most 3 messages may be lost out of any 10 consecutive messages. The maximum number of consecutive losses is 1. The variable *nextExp* is the sequence number following the highest-numbered received message. The variables *waitPoint* and *tooFar* delimit the receive window. The value of *waitPoint* indicates the message that the receiver is currently waiting for. If the receiver is not waiting for any outstanding messages, then *waitPoint* is set to *nextExp*. *tooFar* is always *waitPoint* plus the receive window size. In the case of an unacceptable loss, *waitPoint* will be set to the sequence number of that message. The variable *holdPoint* indicates the oldest lost message in the receive history. This and any other later allowed losses in the receive history will determine if a new loss can be allowed. *holdPoint* is at least *waitPoint* minus the receive window size.

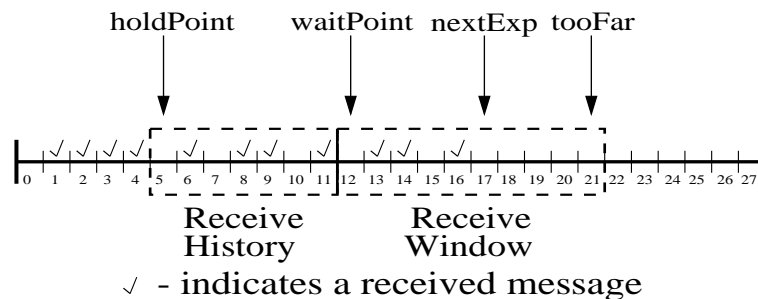


Figure 5: Window Parameters for a Variable Reliability Protocol

This snapshot shows the receiving protocol waiting for message number 12. In this example, the receive history indicates that messages 5, 7 and 10 were allowed to be lost. On the receipt of message 13, the receiver determines that it cannot allow message 12 to be lost, because 3 messages from the “window” (i.e., 3-12) have already been lost. The receiver sends a NAK for message 12. While it is waiting for the retransmission, it continues to process the new incoming messages, and so advancing *nextExp* to 14, 15 and then 17. Once the retransmission is received, *waitPoint* is advanced to message 15, and *holdPoint* is advanced to message 7.

This section has described the definition for a specific reliability stream. In order to enable applications to switch between different reliability levels, the variable reliability framework can maintain multiple reliability streams in parallel. The sender can specify any one of these streams for each data message it sends. In this

way, the sender determines what reliability is required for specific messages, and the receiver implements the policies for each reliability stream. For a specific example, consider the different types of frames in an MPEG video stream. Each type of frame can be transmitted via a different reliability stream, thus allowing I frames to use the highest reliability stream while still allowing P and B frames to use middle and lower reliability streams.

4.4 Results

Image transfer is the basis for many multimedia applications and often tolerates some loss. This tolerance makes it a suitable candidate for taking advantage of adaptable communication. Turner and Peterson describe a retransmission-free protocol for image transfer [TP92]. They make the assumption that applications will be willing to give up some degree of image quality for the decrease in delay. The application is given the choice of receiving all or none of the retransmissions. We believe that applications like ours can benefit from the ability to make more precise tradeoffs between delay and reliability. This section presents the results from three sets of experiments designed to show the benefit of payoff adaptation in conjunction with the use of a variable reliability protocol. The first set of experiments demonstrates the benefits of allowing applications to change resource requirements dynamically. The second set of experiments explores the utility of using observation-based network resource information. The third set of experiments considers richer resource information in the form of loss-load curves.

4.4.1 Adaptive Quality Specification

Data manipulated by applications can often be grouped into different levels of quality. Quality may refer to running time, reliability or any other similar service parameters. In any case, the meaning of “quality” is specific to each application, and quality values may be equated to application data at many levels of granularity, ranging from data sets to frames to individual messages.

We use two distinct examples to illustrate these concepts of adaptive quality specification and its interaction with our variable reliability mechanism. The first example explores static quality specification with an image transfer application that requires different reliability levels for different fixed regions of the image. The second example explores dynamic quality specification in a distributed virtual environment that dynamically changes the reliability requirements of regions of its world. For both of these applications, simple payoff functions are used. Payoff functions for quality correspond to reliability levels in the variable reliability protocol. As reliability requirements change, the application changes the payoff function for that data. Each payoff function has a maximum payoff at the respective quality level, and is zero below that level. These payoff functions cause the communication layer to transfer data at the reliability level that provides the appropriate quality level.

Locality-Based Adaptive Quality Specification: Image transfer provides an excellent avenue for statically partitioning data into multiple quality regions. In image transfer, each image is a data frame. If we consider the transfer of an image of a person, the most important part of that image might be the face (see Figure 6(a)). This section of the image must be of high quality, while the quality of the rest of the image is allowed to decrease as we move further away from the face. This may be expressed by specification of a separate payoff function for each section.

We can now combine the concept of the variable reliability protocol with quality-based data partitioning. Through this combination, we can ensure that the face is received at 100% reliability, while the reliability levels of the rest of the image decrease as we move away from the face. (see Figure 6(b), black spaces indicate lost messages.)

For our locality-based experiments, we use an application that transfers a 6.5Mbyte image across a 10Mb Ethernet connection. The application runs under three different partitioning configurations.

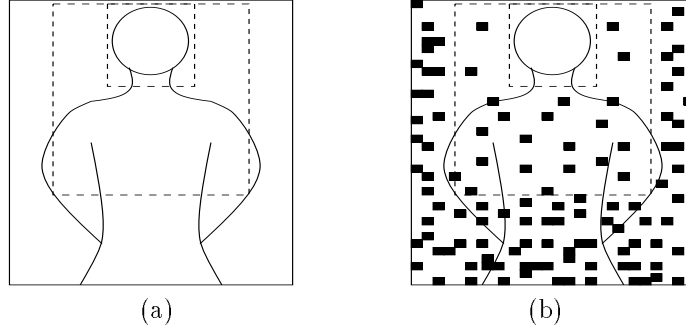


Figure 6: Image with Application-Specified Quality Regions

Configuration 1 imposes 100% reliability on the entire image. Configuration 2 partitions the image into two regions: 11% of the image (e.g., the face) requires 100% reliability; 89% requires 66% reliability. Configuration 3 partitions the image into three regions: 11% requires 100% reliability; 28% requires 66% reliability; and 61% requires 33% reliability. Each of these configurations is run at progressively higher message loss probabilities. We simulate random loss of messages.

Figure 7 depicts the running times for the transfer of a partitioned image in a lossy network under each of these conditions. The results are an average of 10 runs at each loss probability rate. The closer we come to specifying the reliability requirements of the application, the better we can judge the necessity for message retransmission. As the loss increases, the running time for configuration 1 increases to 280% of the running time for a run with no losses. In comparison, configuration 2's running time increases to 180% and configuration 3's running time to only 136%. The improved transmission time in configuration 3 is gained while remaining within the reliability requirements of the application.

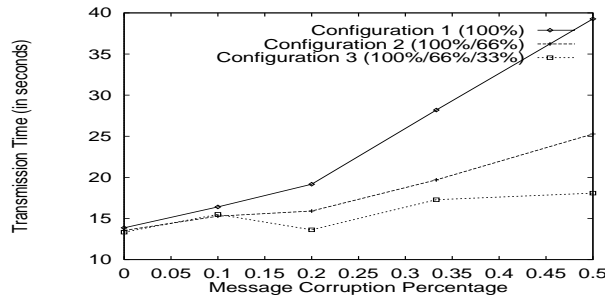


Figure 7: Transmission Time for Quality-Based Partitioned Image

Distance-Based Adaptive Quality Specification: The second application used for evaluation is a simulation of a distributed virtual environment, where multiple users operate in a joint world and share a world view. The world view in this application is processed as an image and passed back and forth between the users as it is being updated. The world is decomposed into blocks, where each block has an owner that is responsible for coordinating reads and writes as well as sending updates to the other users. As each user moves through the world, it updates the data in the world and receives updates from other users.

The quality-based partitioning of data and the variable reliability protocol lend themselves well to being used for this application, since each piece of data potentially has a different reliability requirement, and these requirements may change as the simulation evolves. Intuitively, each user only cares about the immediate surroundings. In other words, the reliability of an update is determined by the update's proximity to the user receiving the update. The sections that are out of the user's view may only require periodic unreliable updates, or no updates at all. The application can dynamically change the

4.4.2 Payoff Adaptation with Observed Network Behavior

This section describes the results of two different experiments designed to explore the effects of end-to-end adaptation. Both experiments use a simplified version of the image transfer application described in Section 4.4.1. In this version, the entire image is set to be at one reliability level. These experiments compare the effects of two different adaptation algorithms.

For both experiments, the sender determines a course of action using local information and information collected by the receiver. Specifically, in response to changes in the error rate, the sender can change the reliability level being used in order to achieve an acceptable transmission time. Our current implementation gives the sender the ability to dynamically switch between four distinct reliability streams: 100% reliability allows no losses; 66% reliability allows 1 consecutive loss and 34% loss in a window; 33% reliability allows 3 consecutive losses and 67% loss in a window; and 0% reliability allows any loss it notices.

In order to determine the benefits of using payoff adaptation, we compare it to the results of using simpler, threshold-based adaptation and of using fixed reliability levels. The sender using payoff adaptation employs the techniques described in Section 3.5. For threshold adaptation, the sender uses a fixed scheme for placing values on resource information. Specifically, whenever a predefined threshold is crossed, the sender automatically changes the communication to a lower or higher reliability level. In this example, the sender monitors the message loss, which is quantized into four ranges. Each range is associated with a reliability level, as specified above. If the observed message loss crosses a threshold from one range to the next, then the sender adjusts to the reliability associated with the new range. In our application, if the sender has been experiencing 1% message loss, which then jumps to 10%, the sender would reduce the reliability level from 100% to 66%. The placement of the thresholds within the range of message loss determines the performance of the application.

For this application, two service parameters are specified: *ObservedReliability* and *TransferTime*, as defined in Section 3.5. For this experiment, we focus on the effect of adapting the reliability level and leave the transfer rate fixed at 650KBps. The operating points for the functions for observed reliability and transfer time, F_{or} and F_{tt} , were collected prior to the final experiment. Total payoff is defined as $P_{total} = P^{or}(ObservedReliability) * P^{tt}(TransferTime)$. P_{total} is evaluated over the available reliability levels, given the currently observed loss rate. By using the steps described in Section 3.5, the communication layer can determine the reliability level that maximizes total payoff for the application. The function for determining total payoff for this experiment was chosen to be multiplication. Multiplication retains the payoff scale of 0-1 and implies that the total payoff will be zero if any of the individual payoffs are zero.

The payoff functions that we use are shown in Figure 9. These payoff functions represent some features matching the application’s behavior; in particular, the reliability drops off at a relatively constant rate up to 50% reliability and drops quickly after that. The payoff function for time allows for some minimal delay at a high payoff, but drops off sharply after two times the expected transfer time.

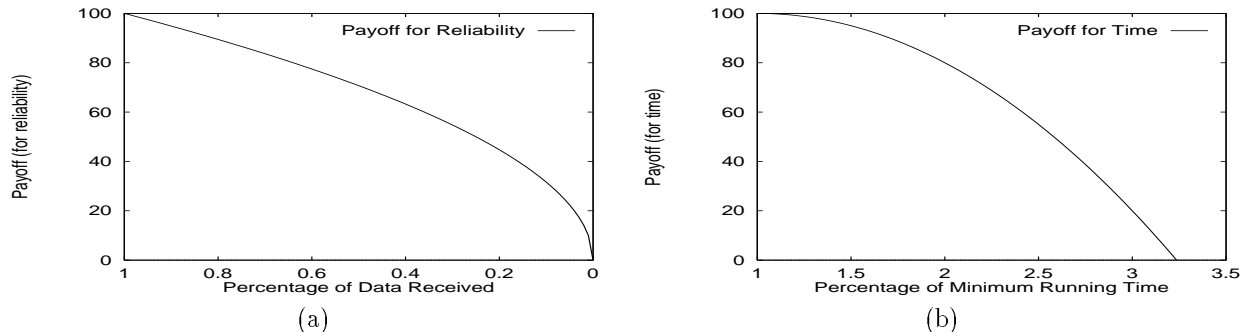


Figure 9: Payoff Functions for Reliability and Time

Figure 10 shows the final payoff for each of the adaptation algorithms as well as for two fixed reliability levels. The goal of the payoff adaptation method is to adaptively choose the best running point for the application. We can see in Figure 10 that the curve for the payoff adaptation method is essentially an upper envelope for the other curves and outperforms them at various points. This illustrates two points. First, a simple adaptive algorithm can exploit the benefits provided by the variable reliability protocol infrastructure, and it can choose the correct operating points efficiently. In other words, it can correctly choose the reliability level at which to run. Second, end-to-end adaptation has the potential to provide the communication layer with sufficient feedback to adjust reliability.

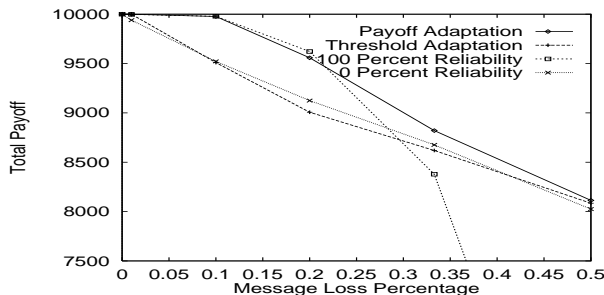


Figure 10: Final Payoff

It is illustrative to look at the raw running times of adaptation algorithms and to compare those times to their effectiveness in terms of the total number of bytes transferred. Figure 11(a) shows the running times for the two adaptation methods and two fixed reliability runs. The results show that for most of the experiment, the payoff adaptation method runs slower than the threshold adaptation method. If we consider the total amount of data that is received, we can see in Figure 11(b) that the payoff adaptation method also receives more of the transmitted data than the threshold adaptation method. These results should be expected, since the payoff adaptation method adapts to optimize payoff for both the amount of data received and the running time.

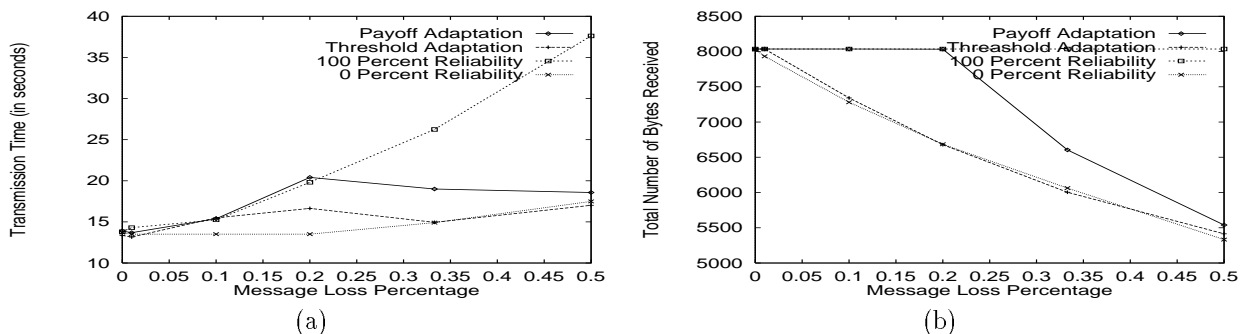


Figure 11: Time and Data Statistics

The results in this section demonstrate that communication configuration based purely on application requirements does not provide efficient service to the application. On the other hand, given some knowledge of the current behavior of the network, the communication layer can adapt the configuration to effect more suitable service for the application. Additionally, combining application specification and network resource information, the communication layer can maximize the payoff to the application for that service. The final experiment presented in this paper addresses situations where more detailed network information is available.

4.4.3 Payoff Adaptation with Loss-Load Curves

This section addresses the use of service availability curves in conjunction with payoff adaptation. Network resource information is represented in a fashion similar to the loss-load curves described in Section 3.4. By supplying the communication layer with a specific loss rate for each transfer rate, the communication layer can use this information to determine expected transfer time experienced at each rate. Note that in this example, transfer rates are no longer fixed. Additionally, since we are using loss-load curves, the loss rate is implied from the specific transmission rate. The operating points for the functions for observed reliability and transfer time, F_{or} and F_{tt} , were determined from baseline timing runs. Data points were gathered for reliability levels of 70%, 80%, 90% and 100%; transfer rates of 400KBps, 500KBps, 600KBps, 700KBps and 800KBps. Each of these runs are subjected to loss probabilities varying from 0-50% in increments of 5. From loss-load curves in the form $F_{LLCurve}(\text{TransferRate}) = \text{LossRate}$, the communication layer can determine the estimated transfer time for each transmission rate and reliability level. Next, transfer time and end-to-end loss are used as parameters to the application-supplied payoff functions P_{OR}^{rel} and P_{TT}^{time} . Finally, by evaluating the payoff functions to maximize P_{total} , the communication layer can determine the transfer rate/reliability pair that it considers the best fit for the application's requirements. The payoff function for reliability is shown in Figure 12(a), while the payoff function for time is the one shown in Figure 2(b). Adaptation is performed in response to changes in network resource availability, as specified by changing loss-load curves. As the communication layer is informed about a change in the loss-load curve, it reevaluates the payoff functions to determine whether it should reconfigure the communication parameters.

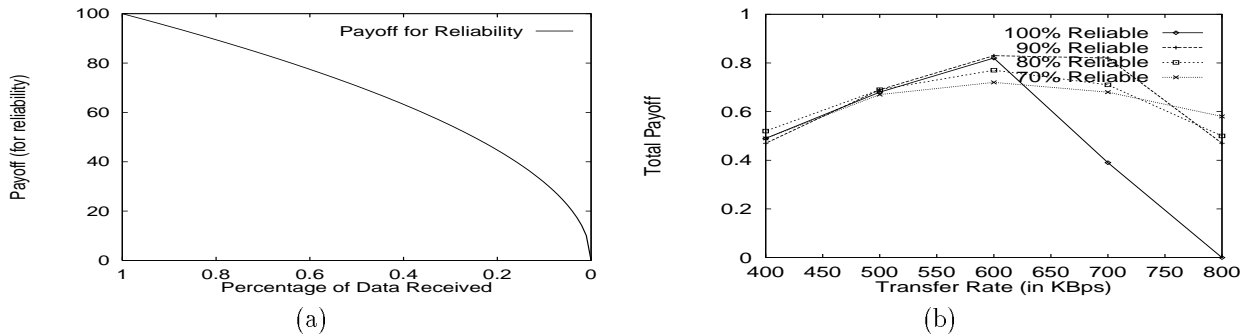


Figure 12: Payoff Function for Reliability and Net Payoff Curves

For a specific example, consider Figure 3 which shows a loss-load curve with simultaneous increases in loss and transfer rates. Figure 12(b) depicts the payoff values obtained for this specific loss-load curve over varying transfer rates. The net payoff is calculated as the product of P^{rel} and P^{time} . Each line in Figure 12(b) represents the transfer of the image at the given reliability for the set of transfer rates. Using these curves, we can determine the optimal transfer rate and reliability for this specific loss-load curve and the requirements of the application by finding a maximum for P_{total} . In this example, the communication layer should choose to transfer data at a speed of 600KBps and a reliability level of 90%.

For these experiments, we use an application that transfers a 6.5Mbyte image 30 times across a 10Mb Ethernet connection, where the communication layer monitors the loss-load curves supplied by the network. The communication layer has the ability to change two parameters: reliability and transfer rate. For reliability, the communication layer can change the acceptable loss-burst size and allowable loss percentage in a window. In theory, the communication layer could allow a continuous choice for reliability. Our current implementation permits the choice between four distinct reliability levels: 100%, 90%, 80% and 70%. For the transfer rate, rates from 400KBps to 800KBps in increments of 100 are chosen.

The experimental results in this section demonstrate the effects of changes in loss-load curves over time, which model changes in network service (e.g., the network becoming more and less congested). The loss-load curves used are based on the one in Figure 3. This linear loss-load curve imposes a 5% increase in loss for each 100KBps increase in requested bandwidth. Changes in the loss-load curve are represented by shifting

the entire curve up or down along the y -axis, thus imposing more or less loss depending on the way in which it is shifted. In this experiment, the loss-load curve is shifted every 15 seconds.

The experiment compares the results of two runs of the application. In the first run, the communication is static. The application transmits at 800KBps and requires 100% reliability. In the second run, the communication is dynamically controlled through payoff-based adaptation. The communication layer monitors the loss-load curves and recalculates the communication parameters as the loss-load curves change. The solid line in Figure 13 represents the loss over time seen by an application that is transmitting at 800KBps. The dashed lines in Figure 14 show the dynamic choices made by the communication layer as to reliability and offered load. Due to the choice of offered load, the loss rate experienced over time is represented by the dashed line in Figure 13. These decisions are based on the payoff calculations made by the communication layer at each change in the loss-load curves. The total payoff for both runs at each change is shown in Figure 15. The solid line represents the total payoff for the first run and the dashed line represents the total payoff for the second run.

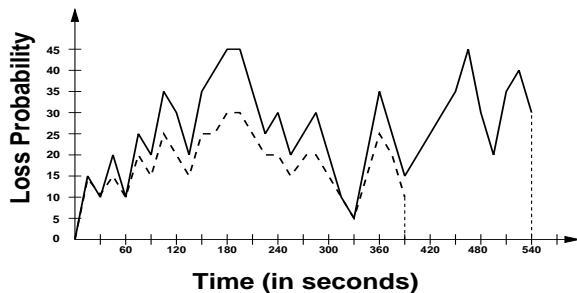


Figure 13: Observed Loss over Time

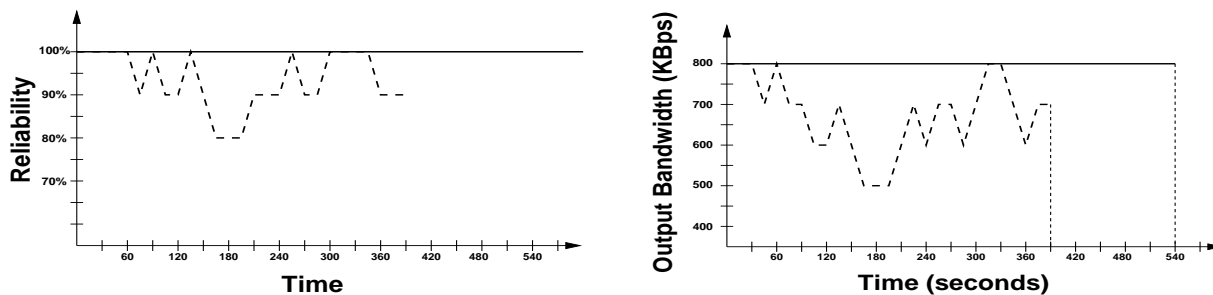


Figure 14: Application Reliability and Bandwidth over Time

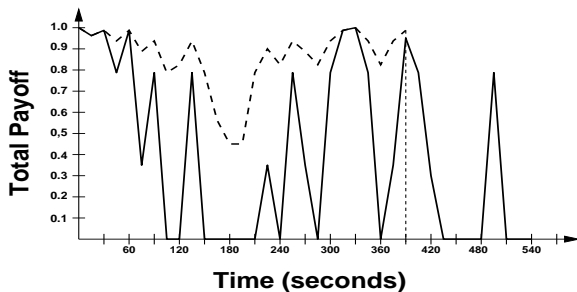


Figure 15: Total Payoff over Time

Figure 15 shows that, given the ability to choose communications parameters, the application's payoff can be maximized at discrete points in time. In addition, final payoff is shown in Table 2, which depicts the

total amounts of data received and the total transfer times for both runs of the application. As expected, the static run receives the maximum payoff for the amount of data received. In comparison, the adaptive run has relatively high payoff for the amount of data received. The biggest difference is in the payoff for the transmission time. The product of these two payoff values gives us the total payoff for the transfer. The static run pays off at 0.8165, while the adaptive run pays off at 0.9627. As expected, the application that adapts to fluctuations in network resources is able to make intelligent decisions based on its willingness to trade off reliability for transfer time.

	Data Received	Fraction of Total Data Sent	Payoff for Data	Transmission Time	Percent of Minimum Time	Payoff for Time
Static	109350 msgsg	1.00	1.000	537 sec	177	0.8165
Adaptive	104874 msgsg	0.959	0.9832	381 sec	126	0.9784

Table 2: Results for Amount of Data Received and Transfer Time

The experiments in this section demonstrate that service availability curves, in the form of loss-load curves, can provide useful information when determining an acceptable communication configuration. By using the information provided in the loss-load curves, the communication layer can determine the effect of a range of transmission rates on the loss rate seen by the application. This foreknowledge provides the communication layer with more fine-grained information than it could get from observational techniques, and so allows the communication layer to make more informed decisions during adaptation.

5 Conclusion

Distributed applications operating in dynamic network environments may experience unpredictable changes in resource availability. Our intent is to enable such applications to operate within acceptable parameters despite potentially insufficient network resources. Our solution approach is the provision of an adaptive communication layer that configures communication protocols based on the benefit the communication layer expects to realize for the application. To this end, we have presented our communication layer which provides a mapping between the services available from the network and the services needed by the application. Through this mapping, the communication layer monitors and adapts the service provided to the application using payoff-based adaptation techniques. These techniques use knowledge of application service requirements and network service availability to maximize the perceived benefit to the application.

In this paper, we demonstrate that, with information about the requirements of the application, the communication layer can improve the match of application requirements with network resources. Similarly, with information about the state of the network, the communication layer can compensate for such changes. Additionally, our work provides a cooperative solution in which the communication layer uses both application resource requirements and information about network resource availability to determine how to configure the communications. Through experimentation, we show that payoff-based adaptation using both types of information can provide communication services that are better suited to applications in situations where both application requirements and network resources are dynamically changing.

References

- [AMK97] Elan Amir, Steven McCanne, and Randy Katz. Receiver-driven bandwidth adaptation for light-weight session. In *ACM Multimedia '97*, November 1997.

- [AMZ95] Elan Amir, Steven McCanne, and Hui Zhang. An application level video gateway. In *ACM Multimedia '95*. 255-265, 1995.
- [AS98] Tarek Abdelzaher and Kang Shin. End-host architecture for QoS-adaptive communication. In *IEEE Real-Time Technology and Application Symposium (RTAS'98)*, June 1998.
- [BG96] R. Bettati and A. Gupta. Dynamic resource migration for multiparty real-time communication. In *IEEE International Conference on Distributed Computing Systems (ICDCS) '96*, May 1996.
- [BHD⁺96] B.B. Bederson, J.D. Hollan, A. Druin, D. Rogers, J. Stewart, and D. Vick. A zooming web browser. In *Multimedia Computing and Networking (MMCN) '96*, 1996.
- [BMR97] N. Brownlee, C. Mills, and G. Ruth. Traffic flow measurement: Architecture. Technical Report RFC 2063, Internet Engineering Task Force, January 1997.
- [Bol] J-C. Bolot. Cost-quality tradeoffs in the internet. *Computer Networks and ISDN Systems*.
- [BS88] T. Bihari and K. Schwan. A comparison of four adaptation algorithms for increasing the reliability of real-time software. Technical report, Department of Computer and Information Science, The Ohio State University, OSU-CISRC-4/88-TR13, April 1988.
- [BS91] Thomas Bihari and Karsten Schwan. Dynamic adaptation of real-time software. *ACM Transactions on Computer Systems*, 9(2):143–174, May 1991.
- [CCWP95] Crispin Cowan, Shanwei Cen, Jonathan Walpole, and Calton Pu. Adaptive methods for distributed video presentation. *Computing Surveys Symposium on Multimedia*, 27(4):580–585, December 1995.
- [CKV93] K. Curewitz, P. Krishnan, and J. S. Vitter. Practical prefetching via data compression. In *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, pages 257–266, May 1993.
- [CT90] David D. Clark and David L. Tennenhouse. Architectural considerations for a new generation of protocols. In *Proceedings of the SIGCOMM '90 Symposium*, pages 200–208, September 1990.
- [DHH⁺93] Luca Delgrossi, Christian Halstrick, Ralf G. Herrtwich, Frank Hoffmann, Jochen Sandvoss, and Barbara Twachtmann. Reliability issues in multimedia transport. In *First IEEE Workshop on the Architecture and Implementation of High Performance Communication Subsystems (HPCS) '93*, 1993.
- [DKS90] Alan Demers, Srinivasan Keshav, and Scott Shenker. Analysis and simulation of a fair-queueing algorithm. *Journal of Internetworking Research and Experience*, pages 3–26, October 1990.
- [DKT⁺93] Jayanta Dey, James Kurose, Don Towsley, C.M. Krishna, and Mehesh Girkar. Efficient on-line processor scheduling of iris (increasing reward with increasing service) real-time tasks. In *ACM Sigmetrics*, 1993.
- [FGBA96] Armando Fox, Steven D. Gribble, Eric A. Brewer, and Elan Amir. Adapting to network and client variability via on-demand dynamic distillation. In *Proceedings of the Seventh International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'96)*, October 1996.
- [GP92] Fengmin Gong and Guru Parulkar. An application-oriented error control scheme for high speed networks. Technical Report TR 92-37, Washington University, St. Louis, 1992.
- [HS96] Scott Hudson and Ian Smith. Techniques for addressing fundamental privacy and disruption tradeoffs in awareness support systems. In *Proceedings of the ACM Conference on Computer Supported Cooperative Work*, November 1996.
- [HW96] J. Huang and P.-J. Wan. On supporting mission-critical multimedia applications. In *Proceedings of the International Conference on Multimedia Computing and Systems '96*, 1996.
- [JLT85] E. Douglas Jensen, C. Douglass Locke, and Hideyuki Tokuda. A time-driven scheduling model for real-time operating systems. In *IEEE Real-Time Systems Symposium*, December 1985.
- [JRR97] Michael B. Jones, Daniela Rosu, and Marcel Rosu. CPU Reservations and Time Constraints: Efficient, Predictable Scheduling of Independent Activities. *16th ACM Symposium on Operating Systems Principles*, 1997.
- [KCS96] Robin Kravets, Ken Calvert, and Karsten Schwan. Dynamically configurable communication protocols and distributed applications: Motivation and experience. Technical Report GIT-CC-96-16, Georgia Institute of Technology, 1996.
- [KLP⁺95] S. Keshav, C. Lund, S. J. Phillips, N. Reingold, and H. Saran. An empirical evaluation of virtual circuit holding time policies in ip-over-atm networks. In *Proceedings of IEEE INFOCOM 95*, 1995.

- [KLV95] P. Krishnan, P. M. Long, and J. S. Vitter. Learning to make rent-to-buy decisions in probabilistic environments. In Armand Prieditis and Stuart Russell, editors, *Machine Learning: Proceedings of the Twelfth International Conference*. Morgan Kaufmann, 1995.
- [LSD98] Bjorn Landfeldt, Aruna Seneviratne, and Christophe Diot. User services assistant: An end-to-end reactive QoS architecture. In *International Workshop on Quality of Service (IQWoS'98)*, 1998.
- [MAC96] Rahmi Marasli, Paul Amer, and Phillip Conrad. Retransmission-based partially reliable services: An analytical model. In *Proceedings of IEEE INFOCOM 96*, 1996.
- [MJS97] G. Robert Malan, Farnam Jahanian, and Sushila Subramanian. Salamander: A push-based distribution substrate for internet applications. In *Proceedings of the USENIX Symposium on Internet Technologies and Systems*, December 1997.
- [MST94] Clifford W. Mercer, Stefan Savage, and Hideyuki Tokuda. Processor Capacity Reserves for Multimedia Operating Systems. "*IEEE Int. Conference on Multimedia Computing and Systems*", 1994.
- [NSN⁺97] Brian D. Noble, M. Satyanarayanan, Dushyanth Narayanan, James Eric Tilton, Jason Flinn, and R. Walker Kevin. Agile application-aware adaptation for mobility. In *Proceedings of the Sixteenth ACM Symposium on Operating Systems Principles (SOSP) '97*, October 1997.
- [OSF⁺97] Seiwoong Oh, Hiroyuki Sugano, Kazutoshi Fujikawa, Toshio Matsuura, Shinji Shimojo, Masatoshi Arikawa, and Hideo Miyahara. A dynamic QoS adaptation mechanism for networked virtual reality. In *Proceedings of Fifth IFIP International Workshop on Quality of Service*, May 1997.
- [PP96] Christos Papadopoulos and Gurudatta Parulkar. Retransmission-based error control for continuous media applications. In *The 6th International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV) '96*, 1996.
- [RSYJ97] Daniela Rosu, Karsten Schwan, Sudhakar Yalamanchili, and Rakesh Jha. On adaptive resource allocation for complex real-time applications. In *18th IEEE Real-Time Systems Symposium, San Francisco, CA*. IEEE, December 1997.
- [SE⁺95] J. Saltz, G. Edjlali, et al. Data Parallel Programming in an Adaptive Environment. *Proc. of the 9th International Parallel Processing Symposium*, pages 812–819, 1995.
- [SES⁺97] Beth Schroeder, Greg Eisenhauer, Karsten Schwan, Jeremy Heiner, Vernard Martin, and Jeffrey Vetter. From interactive applications to distributed laboratories. *IEEE Concurrency*, 1997.
- [She95] Scott Shenker. Fundamental design issues for the future internet. *IEEE Journal on Selected Areas in Communications*, 13(7), September 1995.
- [SZ92] Karsten Schwan and Hongyi Zhou. Dynamic scheduling of hard real-time tasks and real-time threads. *IEEE Transactions on Software Engineering*, 18(8):736–748, August 1992.
- [TP92] Charles J. Turner and Larry L. Peterson. Image transfer: An end-to-end design. In *Proceedings of the SIGCOMM '92 Symposium*, August 1992.
- [Wil96] Carey Williamson. Dynamic bandwidth allocation using loss-load curves. *IEEE/ACM Transactions on Networking*, 4(6):829–839, December 1996.
- [ZBS97] John A. Zinky, David E. Bakken, and Richard D. Schantz. Architectural Support for Quality of Service for CORBA Objects. *Theory and Practice of Object Systems*, 1997.
- [ZDE⁺93] L. Zhang, S. Deering, D. Estrin, S. Shenker, and D. Zappala. RSVP: A new resource reservation protocol. *IEEE Network*, pages 8–18, September 1993.