Adaptive Variation of Reliability

Robin Kravets, Ken Calvert, P. Krishnan and Karsten Schwan*
*Georgia Institute of Technology, Bell Labs**
College of Computing
Georgia Institute of Technology
Atlanta, Georgia, USA
email: {robink,calvert,schwan}@cc.gatech.edu
Bell Labs
Holmdel, New Jersey, USA
email: pk@research.bell-labs.com

Abstract
This paper presents the specification and implementation of a variable reliability framework. By using this framework, applications can explore interesting performance enhancing techniques. Some of these techniques include the use of quality-based partitioning of data and the use of adaptation of reliability based on the current state of the application and the network. We present results from an initial implementation and demonstrate the usefulness of adaptive reliability variation for sample distributed applications.

Keywords
adaptive communication, variable reliability, sliding window protocol, data partitioning

1 INTRODUCTION

Historically, applications had to chose 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 in the unnecessary retransmission of messages by the sender and in the extended buffering of messages by the sender and the receiver. Alternatively, applications may use unreliable protocols, which may result in unacceptable losses.

The research contribution described in this paper 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.

This research is motivated by multimedia applications such as medical imaging, distributed virtual environments, and distributed simulations. The service requirements of such applications range from offering transmission times matching human perceptual needs (as in distributed virtual environments) to providing suitable reliability at acceptable speeds (as in medical imaging). In addition, any single application is likely to communicate via multiple data types, resulting in the simultaneous specification of multiple service requirements.

The remainder of this paper is organized as follows. Section 2 describes the variable reliability framework, and it presents a simple yet powerful way in which applications can specify their diverse reliability requirements. Section 3 demonstrates the use of intelligent partitioning of application data in conjunction with the use of variable reliability. We next present a generalized cost framework in which the application can specify service tradeoffs as payoff functions. This generalization is studied in Section 4, in which we show how these payoff functions may be used to adapt communication strategies to changes in communication patterns. Section 4.3 presents simple adaptation algorithms and demonstrates experimentally that the use of a variable reliability protocol allows applications to effectively adapt to changing communication needs. Concluding remarks appear in Section 5.

2 A VARIABLE RELIABILITY FRAMEWORK

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 (Marasli *et al.* 1996) quantify the cost of "too much" or "too little" reliability within the specification of their protocol. Similarly, Delgrossi, et al. (Delgrossi *et al.* 1993), 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 the notion of "reliability" it needs.

Implicit in our design is the concept of *application layer framing* (ALF) (Clark *et al.* 1984). Namely, we assume that an application can deal independently with the data units that it sends. 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 to deliver data sent from a sender to a receiver without duplication, loss, or messages delivered out of order. Unreliable data transfer makes no guarantees as to duplication, loss, or order. However, it is not obvious how to define a service "between" these extremes. The following sections discuss the issues involved in defining such a service and describe a specific implementation of a variable reliability protocol.

2.1 Specification of Variable Reliability

Variable reliability mechanisms can be applied to both realtime and non-realtime data. This paper focuses on the issues for non-realtime data (i.e., data that is time-sensitive, but not time-critical). Work by Delgrossi, et al. (Delgrossi *et al.* 1993) and Papadopoulos and Parulkar (Papadopoulos *et al.* 1996) addresses reliability for realtime continuous traffic streams. By concentrating on non-realtime data, we can ignore lifetime issues for individual messages, and concentrate on issues about 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 around frame transmission time, not individual message transmission time. This type of specification can be particularly useful for applications running over low-bandwidth and/or high error rate networks (e.g., wireless networks).

The scheme we specify below defines a service based on simple loss measurement and retransmission policies. The protocol specified by Marasli, Amer and Conrad (Marasli *et al.* 1996) 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 sections describe some parameters of our variable reliability protocol.

(a) 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. 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 (Gong *et al.* 1992).

In Figure 1, 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 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.

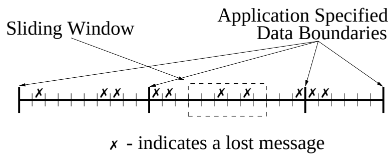


Figure 1 Example of Allowable Loss in a Sliding Window and in a Data Frame

(b) 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.

2.2 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 2 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.

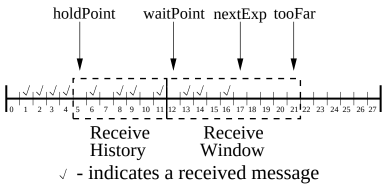


Figure 2 Window Parameters for a Variable Reliability Protocol

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.

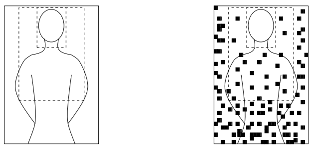
3 QUALITY-BASED PARTITIONING OF DATA

Applications may handle data that can be grouped into different levels of quality. Quality may refer to running time, reliability or any such service parameters. For this discussion, we equate quality with reliability. In section 4, we will investigate the effects of multiple quality parameters. Essentially, quality is defined specifically for each application. In some situations, quality may even differ among the multiple messages of an application's single data frame. In order to ensure that the requirements of the data with the strictest requirements are met, the application must pay the price for processing all of the data at that quality level. By allowing the application to more precisely define the quality requirements of its data, the communication can be adapted accordingly.

We illustrate the concept of quality-based partitioning of data and its interaction with our variable reliability mechanism using two distinct examples. Our first example explores static quality-based partitioning with an image transfer application that requires different reliability levels for fixed regions of the image. Our second example explores dynamic quality-based partitioning with a distributed simulation that dynamically changes the reliability requirements of regions of its world.

3.1 Image Transfer Application

Image transfer provides an excellent avenue for data partitioning. 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 3(a)). This section of the image is defined with a strict quality, while the quality of the rest of the image is allowed to decrease as we move further away from the face.



(a) (b)

Figure 3 Image with Application Specified Quality Regions

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 one hundred percent reliability, while the reliability levels of the rest of the image decrease as we move away from the face. (see figure 3(b), black spaces indicate lost messages.)

For our experiments, we use an application that transfers a 6.5Mbyte image across a 10Mb Ethernet connection. The application runs under three different partitioning configurations. 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 percentages. We simulate random loss of messages.

Figure 4 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 percentage 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 loses. 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.

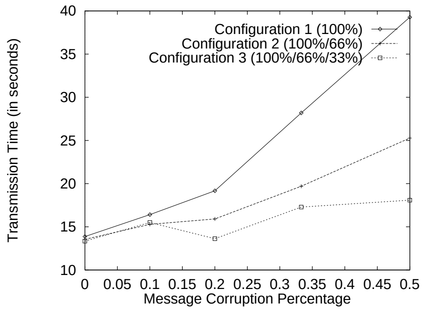


Figure 4 Transmission Time for Quality-Based Partitioned Image

3.2 Distributed Simulation Application

The second application used for protocol evaluation is a distributed game, simulating multiple robots acting on a joint playing field and sharing a world view. The world view in this application is processed as an image and passed back and forth between the robots 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 robots. As each robot moves through the world, it updates the data in the world and receives updates from other robots.

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 robot only cares about its immediate surroundings. In other words, the reliability of an update is determined by the update's proximity to the robot receiving the update. The sections that are out of the robot's view may only require periodic, unreliable updates, or no updates at all. The application can dynamically change the assignment of data to a specific reliability level. This allows the application to choose what data is most important to it, and pay the overhead of reliability for that data.

For this distributed simulation application, the world is divided into concentric "rings" around the location of the robot. Figure 5 shows an example from the viewpoint of robot 0. Each of these rings corresponds to a reliability level. The loss tolerance in the distant rings is higher than that of the closer rings. In other words, as the updates come from further away, the reliability of the update is relaxed.

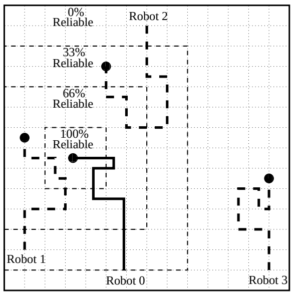


Figure 5 Snapshot of Distributed Robot Game with Variable Reliability Rings

The goal of this experiment is to determine the performance impact of allowing the robots to change the reliability of updates. Experiments monitor the number of retransmissions required and the number of allowed losses. Results are computed based on the number of updates received and the number of messages retransmitted. These numbers are averaged over four runs. Table 1 shows results for robot 0 from a configuration with 100% reliability and from a configuration with the type of partitioning described above. In runs using a totally reliable world, all losses triggered a retransmission request. In runs using quality-based partitioning, only 7% to 22% of the losses trigger retransmission requests. This percentage increases as the amount of reliability requested is increased. For a more detailed explanation of this experiment and the results, see (Kravets *et al.* 1996).

	Average Number of Updates Received	Average Number of Messages Retransmitted
100% Reliability	15377.00	5058.25
Quality-Based Reliability	11838.00	1728.00

Table 1 Average Updates Received and Messages Retransmitted (Robot 0)

4 ADAPTATION OF COMMUNICATION REQUIREMENTS

One of the most demanding requirements of distributed applications is that they continue working under unpredictable network conditions. Not only must they continue working, the applications must also maintain certain characteristics. Examples of these characteristics are response time, image quality, or speech comprehension. This may be difficult if the application considers the communication to be a static entity. The ability to adapt to current network conditions can make the difference between a usable and an unusable application (Diot *et al.* 1995). In this section, we consider the concepts involved in communication adaptation and study the effects of different adaptation methods, focusing on reliability. Our ongoing work deals with intelligent variation of other parameters like flow control, acceptable compression, etc.

Communication parameters are monitored and updated by a communication controller. The design of our experiments allows for differing controllers. One model may have the control centered in the application, thereby localizing the control to changes in that application. Another model includes an external controller, which may be able to control multiple applications. Our goal is to allow each application to determine what type of control model best suits it. In our applications, the controller is implemented as a library of functions used by the application.

4.1 Adaptation Measurements

As network service characteristics change, applications can exploit the variable reliability mechanism to adapt to these changes. In order to determine the benefit of using adaptation techniques, we need to quantify the benefits. By introducing the use of payoff functions, the application can assign priorities to different types and levels of quality. In general there is a payoff function, P_v, associated with each "variable". These functions are specified by the application. The net payoff, P_{total}, is a function, Q, of all the defined P_v. For our current measurements, we have two variables: transfer time and reliability. Their payoff functions are $P_{\text{time}}(\text{TransferTime})$ and $P_{\text{reliability}}(\text{Reliability})$. Net payoff is defined as $P_{\text{total}} = Q(P_{\text{time}}(\text{TransferTime}), P_{\text{reliability}}(\text{Reliability}))$.

Williamson and Cheriton (Williamson *et al.* 1991) define the concept of loss-load curves to determine the action of the application in the face of congestion. Given an application's transmission rate and the current state of the network, the loss-load curve provides the application with the percentage of packets that may be dropped. This technique allows the application to determine the cost of high transmission rates with higher loss rates, but places the responsibility of dealing with losses in the hands of the application. In contrast, we supply a mechanism for handling losses and allow the application to specify its parameters. A controller can make use of information provided by loss-load curves to help it determine the best course of action.

Sample payoff functions appear in figure 6. As we would expect, the payoff for reliability is highest when one hundred percent of the data is received. This slowly drops off until it reaches the limit of acceptable reliability, and at this point drops to zero. Similarly, the payoff for time is highest for shorter transmission times, and drops to zero for some maximum acceptable transmission time.

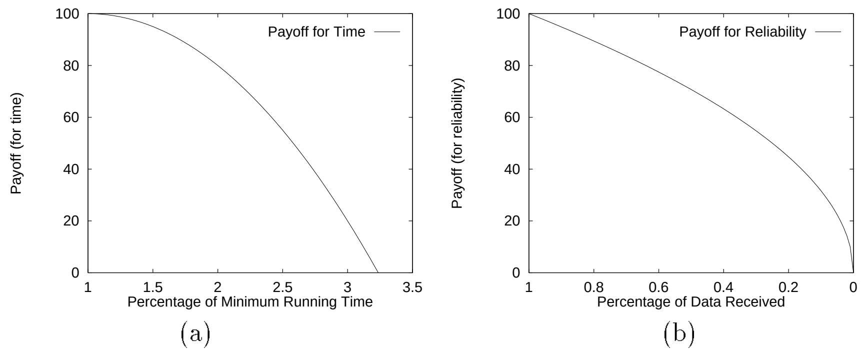


Figure 6 Actual Payoff Functions for Time and Reliability

4.2 Adaptation Locality

By assessment of available resources and of the quality requirements provided by the application, the communication can be adapted to better fit the application's needs and the available resources. Adaptation locality answers the question of where resource information comes from. A controller will have many different sources of information from which it can determine its current state. We define three levels of adaptation locality. A controller using *local adaptation* methods only considers information about the current running state of a piece of an application. For example, a controller may only knows about the sending part of the application. The applications in section 3 show the use of local adaptation methods. A controller using *end-to-end adaptation* considers information from endpoints of an application. For end-to-end adaptation, we would have communicating controllers at the sending and receiving sides of an application. The experiments in section 4.3 consider end-to-end adaptation. A controller using *global adaptation* methods considers information from local and external resources. In the case of global adaptation, many different controllers may exchange information. Our ongoing work considers the impact of global adaptation.

4.3 Experiments

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 3. 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 controller is on the sending side and determines a course of action using local information and information collected by the receiver and returned to the controller. Specifically, in response to changes in the error rate, the controller can change the reliability level being used. The current implementation gives the controller the ability to dynamically switch between four distinct reliability streams: 100% reliability allows no losses; 66% reliability allows 1 consecutive loss and 34% in a window; 33% reliability allows 3 consecutive losses and 67% loss in a window; and 0% reliability allows any loss it notices.

(a) Adaptation Algorithms

Adaptation algorithms define how resource information is used in determining the service provided to the application. We define two adaptation algorithms: threshold adaptation and payoff adaptation. This research investigates reactive adaptation methods. Future work will consider the use of predictive (Bihari *et al.* 1991) and learning-theory-based adaptation methods (Krishnan *et al.* 1995, Keshav *et al.* 1984).

Threshold Adaptation: A controller using threshold adaptation has a fixed scheme that it uses to value resource information. Whenever a predefined threshold is crossed, the controller automatically changes the communication. In this example, the controller monitors the message loss percentage, which is quantized in four ranges. Each range is associated with a reliability level. If the observed message loss percentage crosses a threshold from one range to the next, then the controller adjusts to the reliability associated with the new range. In our application, if the controller has been seeing 1 percent message loss, which then jumps to 10 percent, the controller would reduce the reliability level from 100 percent to 66 percent. The placement of the thresholds within the range of message loss determines the performance of the application.

Payoff Adaptation: A controller using payoff adaptation considers the effect of changes on application performance. Before making any changes, the controller compares the cost of running with the current state with the payoff of running in the changed state. For this purpose, the controller needs information about the inter-relationship between the variables (via a function f : Reliability $\rightarrow$ Time, for example). The payoff adaptation technique allows us to balance application requirements like quality level and running time with resource availability.

Reliability measurements are based on the total percentage of data received.

Total payoff is defined as $P_{\text{total}} = P_{\text{time}}(\text{TransferTime}) * P_{\text{reliability}}(\text{Reliability})$. The payoff functions that we used are shown in figure 6. 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. We are currently investigating other payoff functions, e.g., fast dropping exponential functions. The operating points for the controller were gathered prior to the final experiment.

(b) Observations

Figure 7 shows the final payoff for each of the adaptation algorithms considered in our work. The goal of the payoff adaptation method is to adaptively choose the best running point for the application. We can see in figure 7 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 applications with sufficient feedback to adjust their reliability. In our current implementation, controllers exchange their statistics periodically. We are currently investigating the effect of changing the rate of transfer of information between the controllers.

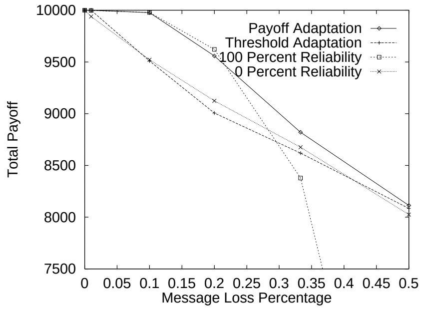


Figure 7 Final Payoff

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 8(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 8(b) that the payoff adaptation method also receives more of the transmitted data than the threshold adaptation method.

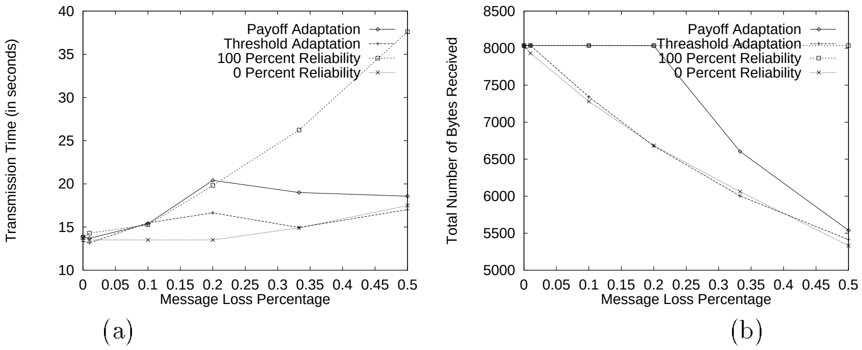


Figure 8 Time and Data Statistics

5 CONCLUSIONS AND FUTURE WORK

The premise of this work is that many multimedia applications will benefit from being able to choose between different reliability levels, rather than being forced to choose between full or no reliability. We have presented a mechanism by which applications can specify their varying reliability needs. We have implemented an infrastructure that provides these variable reliability options to applications. We have demonstrated that applications can exploit this framework to enhance performance in interesting ways; for example, by quality based partitioning of data and adaptively adjusting their reliability requirements to current network conditions. Our experiments show that significant improvements are possible by using the features of our variable reliability method.

Our ongoing work addresses other issues that fit into this framework. Specifically, we are looking at flow-control techniques, other adaptation methods via end-to-end communication of network information, allowing applications to choose other features (like compression level) and applicability on high-error/low-bandwidth networks, like wireless networks.

Acknowledgments

This work is sponsored in part by an AT&T PhD Fellowship and in part by NSF grants CDA-9501637, CDA-9422033, and ECS-9411846. Portions of this work were completed at Bell Labs. We'd like to thank Binay Sugla for several helpful discussions and comments.

REFERENCES

Bihari, T. and Schwan, K. (1991) Dynamic Adaptation of Real-Time Software. *ACM Transactions on Computer Systems*, **Vol. 9, No. 2**, 143-174, May 1992.

Clark, D.D. and Tennenhouse, D.L. (1984) Architectural Considerations for a New Generation of Protocols. *Proceedings of the SIGCOMM '90 Symposium*, 200-208.

Delgrossi, L., Halstrick, C., Herrtwich, R.G., Hoffmann, F., Sandvoss, J. and Twachtmann, B. (1993) The desirability of adjusting for residual effects in a crossover design. *First IEEE Workshop on the Architecture and Implementation of High Performance Communication Subsystems (HPCS) '93.*

Diot, C., Huitema, C. and Turletti, T. (1995) Multimedia Applications Should Be Adaptive. *Third IEEE Workshop on the Architecture and Implementation of High Performance Communication Subsystems (HPCS) '95.*

Gong, F. and Parulkar, G. (1992) An Application-Oriented Error Control Scheme for High Speed Networks. *Washington University, St. Louis*, **TR 92-37**.

Keshav, S., Lund, C., Phillips, S.J., Reingold, N. and Saran, H. (1984) An Empirical Evaluation of Virtual Circuit Holding Time Policies in IP-over-ATM Networks. *Proceedings of IEEE INFOCOM 95.*

Kravets, R., Calvert, K. and Schwan, K. (1996) Dynamically Configurable Communication Protocols and Distributed Applications: Motivation and Experience. *Georgia Institute of Technology*, **GIT-CC-96-16**.

Krishnan, P., Long, P.M. and Vitter, J.S. (1995) Learning to Make Rent-to-Buy Decisions in Probabilistic Environments. *Machine Learning: Proceedings of the Twelfth International Conference.*

Marasli, R., Amer, P. and Conrad, P. (1996) Retransmission-Based Partially Reliable Services: An Analytical Model. *Proceedings of IEEE INFOCOM 96.*

Papadopoulos, C. and Parulkar, G. (1996) Retransmission-Based Error Control for Continuous Media Applications. *The 6th International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV) '96.*

Williamson, C.L. and Cheriton, D.R. (1991) Loss-Load Curves: Support for Rate-Based Congestion Control in High-Speed Datagram Networks. *Proceedings of the SIGCOMM '91 Symposium.*