TESTING ENVIRONMENT FOR INNOVATIVE TRANSPORT PROTOCOLS

Phillip T. Conrad

Computer and Information Science Department
Temple University, Philadelphia, PA 19122 USA
Email: conrad@acm.org

Paul D. Amer
Mason Taube
Gul Sezen
Sami Iren
Armando Caro

Computer and Information Science Department
University of Delaware, Newark, DE 19716 USA
Email: {amer,taube,sezen,iren,acaro}@cis.udel.edu

ABSTRACT

This paper describes the development of a test environment for innovative transport protocols. Central to this work is the development of a Universal Transport Library (UTL). UTL is a library of transport protocols that provides application programmers the ability to write to a single Application Programming Interface (API), then test their application with many different transport protocols. UTL also allows for rapid prototyping of transport protocols at user level. UTL has been incoporated into two multimedia communication systems designed to provide better performance over lossy networks by using innovative transport protocol features: NETCICATS (a Network-Conscious Image Compression and Transmission System) and ReMDoR (a Remote Multimedia Document Retrieval system). These three tools facilitate the evaluation of flexible transport protocols and compression techniques for multimedia communications over lossy battlefield networks. [1]

1 INTRODUCTION

Traditional transport protocols that operate over unreliable battlefield networks provide an "all-or-nothing" choice for transport Quality of Service (QoS); either total order and reliability (e.g., TCP) or no guarantees at all (e.g., UDP). The Partial Order/Partial Reliability (PO/PR) hypothesis is that since, for ARQ-type protocols, improvements in reliability and order come at a cost of worsening delay and throughput, some applications may benefit from a service in between that provided by TCP and that provided by UDP. A PO/PR service allows the application to make tradeoffs between order/reliability and delay/throughput on a message-by-message basis (see for example, RFC1693) [1, 4, 6].

Our investigation of this hypothesis has two goals: (1) to understand and characterize the nature of these tradeoffs via analysis, simulation and experimentation, and (2) to implement a transport protocol that provides a service user with as much flexibility as possible in selecting the best tradeoff for a particular application.

This hypothesis must be investigated in the context of specific applications. Two applications that may be part of a digital battlefield scenario have been developed within the University of Delaware's Protocol Engineering Lab to serve as this testbed: (1) a Network-Conscious Image Compression and Transmission System (NETCICATS), and (2) a Remote Multimedia Document Retrieval (ReMDoR) system. Section 2 overviews these two systems.

Specifically, this article describes the development of the Universal Transport Library (UTL) in Section 3. UTL allows developers to write applications that can take advantage of the innovative protocol features, and allows protocol designers to rapidly prototype new transport services.

[1] Prepared through collaborative participation in the Advanced Telecommunications/Information Distribution Research Program (ATIRP) Consortium sponsored by the U.S. Army Research Laboratory under the Federated Laboratory Program, Cooperative Agreement DAAL01-96-2-0002. This work also supported, in part, by the US Army Research Office (ARO) (DAAH04-94-G-0093). The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied of the Army Research Laboratory or the U.S. Government.

2 OVERVIEW OF NETCICATS AND ReMDoR GOALS

2.1 NETCICATS

NETCICATS is an approach to image compression that seeks not solely to maximize compression, but rather to optimize *overall* performance when compressed images are transmitted over a lossy packet-switched network such as a battlefield network [2, 9, 10, 11]. Using an Application Level Framing philosophy, an image is compressed into path MTU-size Application Data Units (ADUs) at the application layer. Each ADU "carries its semantics"; that is, each ADU is a self-contained unit possessing all information necessary for a receiver to decode and display the information within that packet. Therefore, each piece of independent information can be delivered out-of-order to the receiving application, thereby enabling faster progressive display of images.

2.2 ReMDoR

ReMDoR is a multimedia document retrieval system that allows authors to specify synchronization requirements and varying degrees of reliability for multimedia elements [7]. The basic model is similar to that of the World Wide Web; documents are available on a server and are retrieved via a browser. Unlike Web documents, however, ReMDoR documents are *temporal*—they have a time dimension requiring synchronization of elements such as audio, video, still-images, text, pauses, and interactions.

The ReMDoR browser is similar to traditional web browsers, making experiments convenient. It has capabilities that support experimentation with innovative protocols and data compression techniques, such as (1) the ability to easily incorporate new image formats, and (2) the ability to record the "response-time" of the application on an object-by-object basis [3].

2.3 Shared Requirements

The underlying hypothesis of both NETCICATS and ReMDoR is that PO/PR transport protocols allow the application to deliver more information to the user sooner than would be the case with traditional protocols. As such, the experiments planned for both systems involve some common elements.

In both cases it will be necessary to run the applications over more than one transport protocol—perhaps several protocols offering a variety of levels of order and reliability, perhaps a variety of flow control techniques, etc. The ideal test environment is one where each application can switch transport protocols by selecting from a pull-down menu, or passing a different command line option. Such an environment would facilitate comparisons between the performance of some protocol "A" vs. another protocol "B". Section 3 describes how the Universal Transport Library helps an application developer achieve this flexibility.

Another requirement is to be able to automate the running of experiments over alternate transport protocols and network conditions; i.e., to control the entire experimental framework (application settings, network loss rates, transport protocol) from an automated script, and automatically collect data (such as response-time, network utilization, etc.). This capability would allow experiments to be repeated without human intervention.

3 UNIVERSAL TRANSPORT LIBRARY (UTL)

3.1 Introduction

UTL is a library of transport protocols that provides flexible QoS tradeoffs. UTL was developed at the University of Delaware in support of transport protocol research—particularly ReMDoR and NETCICATS.

UTL serves two functions:

- It provides a common API wrapper around a range of transport protocols. This wrapper allows application programmers to write code that is independent of the underlying transport protocol.
- It provides a framework for development of innovative transport protocols by allowing protocols to be composed from smaller units of processing.

3.2 Motivation

Testing the PO/PR hypothesis for ReMDoR and NETCICATS applications involves gathering data on the performance of these applications running over various transport protocols. UTL was proposed as a solution to two problems that arose in the course of this project.

The first of these problems arose in the development of the initial ReMDoR prototype. The various protocols required different programming styles, and had different Application Programming Interfaces (APIs). Writing special case code to handle each protocol was causing the application programming to become unreasonably complex. Knowing that NETCICATS development would have to handle the same issues, we determined it would be wise to develop a a *common API for various transport protocols.*

The second problem arose in considering how to compare PO/PR service to ordered/reliable service. The most common example of an ordered/reliable transport service is TCP. It is interesting to compare PO/PR service with TCP, since many current applications use TCP. Important features of TCP include its facilities for flow control and congestion avoidance; these features enable applications to fairly share available bandwidth on IP networks. However, TCP congestion avoidance features slow it down relative to protocols lacking such features. This presents an "apples vs. oranges" type of problem: if one is trying to design an experiment to compare, say, the partially-ordered/reliable service of POCv2, against the ordered/reliable service of TCP, a POCv2 vs. TCP comparison is unfair if POCv2 does not implement a flow control scheme similar to that of TCP.

We are pursuing two solutions to this problem:

- short-term, we plan to develop an in-house ordered/reliable protocol without TCP flow control and compare this protocol with POCv2, thus allowing a fair "Apples and Apples" comparison,
- long-term, we plan to add TCP-compatible congestion avoidance features into the protocols developed at UD (k-XP, POCv2, TRUMP [8]), and compare these protocols directly with TCP.

We determined that the best way to approach both of these projects was to first develop a *framework for the rapid prototyping and efficient development of usable transport protocols.* Hence UTL is an attempt to address both the need for a common Transport API, and a framework for transport protocol development.

3.3 How UTL Is Used

UTL is a library of C functions that a programmer can link in with an application to enable that application to access several connection-oriented transport services through a single API. The application can then vary the transport protocol used by altering a single parameter on the "listen" call (passive open) or the "connect" call (active open).

By allowing easy experimentation with different transport protocols, the experimenter can "isolate" particular aspects of transport services to better understand the effects of each in isolation. Figure 1 illustrates how UTL is used by client/server applications, such as ReMDoR and NETCICATS. The various services available to applications via the UTL API are presented in Table 1.

The UTL API is similar to the Berkeley Sockets described in many texts on Unix network programming [12]. What all UTL services have in common is that they are connection-oriented, message services. Where they differ is in order, reliability, and other features (see Table 1).

For each feature of UTL, there is a "fall-back" position to a reasonable default. The UTL API provides several innovative features, but the application only has to be bothered with the ones it finds useful; features that are not useful for a particular application can be safely ignored. The provision of *fallbacks* to default behaviors simplifies the use of the UTL API when its more advanced features are not required for a given application.

As an example of a fallback, consider the notion of a "service profile." The UTL API allows an application to specify a service profile, which is a data structure defining a partial order and reliability vector [7]. The service profile is only used when the underlying protocol supports partial order and partial reliability. If the application has no need to take advantage of such features, it can ignore the parts of the API which allow the service profile to be specified. In this case, a default partial order of a "chain" (total order) with all objects reliable is in effect. Thus an application programmer can make use of UTL without knowing about service profiles; if no service profile is defined by the application, protocols providing PO/PR service will default to ordered/reliable service.

Note that a particular application will not necessarily function over every UTL service. If an application depends on ordered data delivery to operate correctly, and it is run over an unordered service, the application will likely fail to perform its task correctly. For any given application, the designer may want to restrict the choices of UTL services available to the user to a reasonable subset. Within that subset, however, it is still useful to be able to flexibly choose from a range of options when running experiments. It can also be instructive to demonstrate exactly how a particular application fails when an unsuitable transport service is chosen.

The current definition of the UTL library describes six transport services (see Table 1), and one "pass-

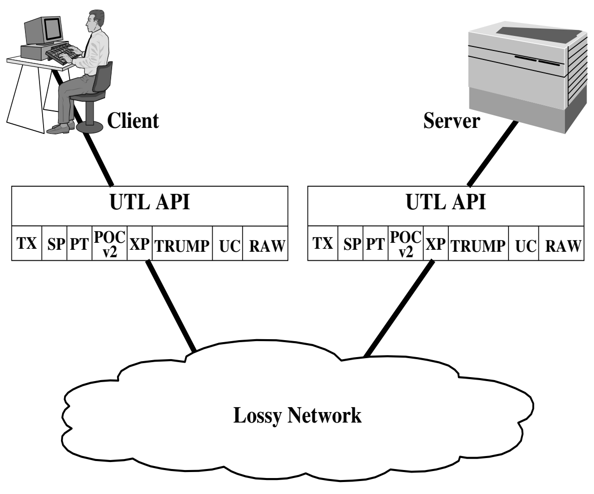

Figure 1: Client/Server Application over Internet via UTL

UTL API							
TX	**SP**	**PT**	**POCv2**	**XP**	**TRUMP**	**UC**	**RAW**
•Ordered •Reliable	•Ordered •Reliable •Access to buffered data	•Partially ordered •Reliable •Sync support	•Partially ordered •Partially reliable •Sync support •Access to buffered data	•Unordered •Partially reliable	•Unordered •Partially reliable (time-based; application provides staleness time for each message)	•Unordered •Unreliable	•Pass-thru service

Table 1: UTL API Layer and Transport Services Provided

thru" service to allow access to raw file descriptors. The pass-thru service is provided to allow applications to easily multiplex and demultiplex UTL style communication with regular access to Unix files and sockets using a `select()` style mechanism.[2]

The UTL components visible to a UTL application user are the header file (`#include <utl.h>`) and the object library of C functions (`libUTL.a`). The functions in `utl.h` are similar to those of the sockets API; familiar functions such as `listen()`, `connect()`, `accept()`, `read()`, `write()` become `utl_Listen()`, `utl_Connect()`, `utl_Accept()`, `utl_Read()`, and `utl_Write`. The parameters are slightly different from the sockets API counterparts; mainly in ways that facilitate reduction in data copying between the application and the user-level transport protocol implementations within the UTL library.

Another advantage of the UTL approach is that it allows for rapid prototyping of protocols. Within UTL, transport protocol development is done at the "user level", not in the kernel. New transport services are layered on top of UDP or TCP. This makes code easier to write and debug, and more portable. UTL's layered model allows for code reuse; for example, one can write "segmentation/reassembly" once, and then reuse this functionality in several protocols. One can also add new versions of protocols in parallel. For example, one could have two versions of partial reliability running side by side each with a different flow-control/congestion-avoidance technique. The two versions of partial reliability could then be incorporated into different transport services involving various kinds of reordering, segmentation, and so forth. This flexible framework for experimentation is at the heart of the UTL design philosophy.

3.4 Innovative Transport Protocol Features

In addition to allowing flexibility in terms of order and reliability, some additional special features have been incorporated into some of the protocols within the UTL framework:

[2] `select()` is part of most Unix implementations of the sockets API; see [12] for more details on the `select()` method of I/O multiplexing/demultiplexing.

- ADN-Cancel. This feature allows cancellation of messages that have already been submitted to the transport layer. The application specifies an Application Data Name (ADN) for each message and can cancel the transmission of any message (or group of messages) by specifying its (their) ADN. As an example of the use of this feature, consider a system for transmitting images: when the receiver has received enough data to make a decision regarding the image (e.g., for tele-medicine: "transport patient or do-not-transport", or in situational awareness: "friend vs. foe", "target vs. non-target"), the receiver can cancel the transmission of all additional packets carrying an ADN for the image in question, without severing the connection. This may be particularly useful in situations where multi-access communication make bandwidth a scarce commodity.
- Support for Application Level Framing and Integrated Layer Processing. The "Buffer-Access" feature provides access to out-of-order buffered data so the application can start presentation-layer conversions (such as decryption or decompression) earlier, even if there is a higher-level requirement for ordered data delivery.
- Explicit Release. This feature is used with partial-order to simulate petri-net based synchronization of multimedia documents [7].

4 CONCLUSIONS

We have described how the Universal Transport Library and improved ReMDoR browser will provide an effective test environment for the evaluation of innovative transport protocols for multimedia applications over lossy networks, such as battlefield networks and congested networks. Examples of some of the experiments that the UTL framework make possible are:

- comparing the overhead of PO/PR service vs. ordered/reliable, unordered/unreliable. This could be accomplished using UTL by running a test application over POCv2 with "total order, everything reliable" or "no order, everything unreliable" as the service profile, and then comparing the performance to that of the same application running over UC (no ordering layer), and SP (total ordering layer) [5].
- comparing PO/PR service (POCv2) to ordered/reliable service (TX, SP) in terms of delay, reliability trade-offs. It is easy to see that PO/PR service provides no benefit if the network has no loss or reordering. What is interesting to determine is the loss rate at which PO/PR service starts to provide a benefit that can be perceived by the user. This could be done using the ReMDoR browser to measure delay of presentation objects, over TX, SP and POCv2. For POCv2, one can also measure the rate at which POCv2 declares objects lost in the case of partially-reliable service.

5 REFERENCES

[1] P. Amer, C. Chassot, T. Connolly, M. Diaz, and P. Conrad. Partial order transport service for multimedia and other applications. *IEEE/ACM Trans on Networking*, 2(5):440–456, October 1994.

[2] P. Amer, S. Iren, G. Sezen, P. Conrad, M. Taube, and A. Caro. Network-conscious GIF image transmission over the Internet. In *4th International Workshop on High Performance Protocol Architectures (HIPPARCH'98)*, June 1998.

[3] A. Caro. Remdor 2.0: Remote multimedia document retrieval over partially-ordered, partially-reliable transport protocols, May 1998. BS Thesis, CIS Dept., University of Delaware.

[4] T. Connolly, P. Amer, and P. Conrad. An extension to TCP: Partial order service. RFC 1693, November 1994.

[5] P. Conrad. Order, reliability, and synchronization in transport layer protocols for multimedia document retrieval. PhD Dissertation, CIS Dept. University of Delaware, (in progress).

[6] P. Conrad, P. Amer, E. Golden, S. Iren, R. Marasli, and A. Caro. Transport qos over unreliable networks: No guarantees, no free lunch! In *IFIP Fifth International Workshop on Quality of Service (IWQOS '97)*, New York, NY, May 1997.

[7] P. Conrad, E. Golden, P. Amer, and R. Marasli. A multimedia document retrieval system using partially-ordered/partially-reliable transport service. In *Multimedia Computing and Networking 1996*, San Jose, CA, January 1996.

[8] E. Golden. TRUMP: Timed-reliability unordered message protocol, December 1997. MS Thesis, CIS Dept., University of Delaware.

[9] S. Iren, P. Amer, A. Caro, G. Sezen, M. Taube, and P. Conrad. Network-conscious compressed image transmission over battlefield networks. In *MILCOM '98*, Bedford, MA, October 1998.

[10] S. Iren, P. Amer, and P. Conrad. NETCICATS: network-conscious image compression and transmission system. In *Fourth International Workshop on Multimedia Information Systems (MIS'98)*, Istanbul, Turkey, September 1998.

[11] S. Iren, P. Amer, and P. Conrad. Network-conscious compressed images over wireless networks. In *5th International Workshop on Interactive Distributed Multimedia Systems and Telecommunication Services (IDMS'98)*, Oslo, Norway, September 1998.

[12] W. Stevens. *UNIX Network Programming*. Prentice-Hall, 1990.