NetCp - A Project Environment for an Undergraduate Computer Networks Course

David Finkel
Department of Computer Science
Worcester Polytechnic Institute
Worcester, MA 01609
dfinkel@cs.wpi.edu

Surendar Chandra
Department of Computer Science
Duke University
Durham, NC 27708
surendar@cs.duke.edu

1 Introduction

Worcester Polytechnic Institute is a technologically oriented university, with a strong major in computer science. Since the inception of its project-centered WPI Plan in 1970 [WORC93], both the undergraduate curriculum and many individual courses have been based on large-scale projects. This paper describes software to support a large-scale project in an undergraduate Computer Networks course, and our experience using that software.

The idea of basing undergraduate systems courses on large scale projects is well-established. Both the MINIX operating system [TANE87] and the OSP project environment [KIFE91], among others, provide a framework for a project-based undergraduate operating systems course.

The undergraduate Computer Networks course is taken primarily by third- and fourth-year students, mostly computer science and electrical engineering majors. The students have taken a minimum of four previous courses in computer science, including a first operating systems course, covering roughly Chapters 1 - 6 of [TANE87]. Typically, the students have had courses in data structures, file structures, and a second course in operating systems before taking the Computer Networks. Thus the students are well-prepared to handle large programming assignments.

2 Netcp

2.1 Introduction

The package we used for our course is called Netcp. Netcp is modeled after the UNIX[1] command *rsh*. Netcp copies a file, specified in the command line, to a remote machine. The user first runs Netcp as a server on the receiver machine using the command line switch *-S* before data can be sent. Unlike rsh, Netcp does not worry about user authentication.

Netcp uses the network model developed by the International Standards Organization (ISO) called the ISO OSI (Open Systems Interconnection) Reference Model [TANE89]. The ISO model envisions 7 layers of protocol:

1. **Physical layer** deals with sending raw bits over a physical communication medium.
2. **Data link layer** takes this raw transmission facility and transforms it into a line that appears free of transmission errors.
3. **Network layer** is concerned with controlling the operation of a subnet and with routing packets.
4. **Transport layer** accepts data from Session layer, splits it up into smaller units if need be, and ensures that all pieces arrive correctly at the other end.
5. **Session layer** allows users on different machines to establish connections between them.
6. **Presentation layer** deals with syntax and semantics of information transmitted.

[1] UNIX is a registered trademark of USL

7. **Application Layer** contains a variety of user level protocols, such as support for a network virtual terminal.

We felt that by working with the *data link layer*, we would expose the students to the crucial issues involved in understanding computer networks. The data link layer is concerned with converting a physical medium to a form useful for sending messages. The lower layer deals with hardware intricacies and higher layers provide user services to effectively use a network.

Thus, Netcp uses only 3 layers: the physical layer, the data link layer and a hybrid layer which encapsulates all the functions of the higher level layers (network, transport, session, presentation and application layers).

The routines for the hybrid layer, the physical layer along with some support routines for the data link layer were provided in a library to the students. The students' assignment was to provide the remaining functionality of the data link layer to develop a working Netcp.

Netcp provides built-in functionality to set error rates individually on the server and client side, to provide a high enough error rate to test the students' code's ability to handle errors. NetCp also provides functions to print useful statistics about the data transfer. Extensive support is provided for dumping binary packets as hexadecimal numbers in any layer. This functionality is very useful in debugging packet loss and error correction routines.

2.2 System Requirements

Netcp uses UNIX facilities such as sockets and shared memory for internal communications and synchronization. Netcp was used in a cluster of Dec 5000, Dec 3100, Dec 2100 running Ultrix 4.3 and on Sun Sparc IPC running SunOS 4.1.1. In addition, Netcp has been ported to Silicon Graphics workstations and HPUX machines.

3 Project Ideas

Netcp, as given to the students, provides a stub routine that implements a rudimentary data link layer to get packets from either the physical or the hybrid layer and send it across to the hybrid or physical layer, respectively. Any of the following functionalities can be added as a student assignment.

- **Error free channel** Transmission errors are a fact of life. The data link layer must take a error prone physical layer and presents a error free channel to the higher layers. Netcp supports a command line option to increase the error rate artificially introduced by the physical layer. With increased error rates, the packets can be completely dropped by the physical layer.

  Students can develop routines to do simple error checking using standard CRC polynomials such as CRC-12, CRC-16 or CRC-CCITT. A successful packet should be acknowledged. However, this option requires substantial bit manipulation operations and hence a good understanding of C language is required.

- **Timeout Control** Once a error packet is identified, remedial action needs to be taken. Errors can be detected by the receiver as a CRC mismatch or the sender as a timeout. In either case, a retransmission has to be effected in some way.

  Students can develop the timer routines for timeout. A simple flow control mechanism can be developed that operates on a hand shake mode of operation (a packet is sent, and until a acknowledgment is received, the sender keeps sending the packet).

- **Data Framing** In order to provide service to the network layer, the data link layer must use the service provided to it by the physical layer. The physical layer can be constrained to accept packet of a size (say 64 bytes) which is smaller than the hybrid layer frame size (say 128 bytes).

  Students can use either character stuffing or bit stuffing to fit their frames into this 64 byte boundary.

- **Flow Control** Simple hand shake protocols support very low throughput, especially on a noisy line. In [TANE89], six protocols are presented, which are in increasing order of complexity and provide better throughput. These involve using a sliding windows protocol, piggybacking of acknowledgments, using negative acknowledgments, and using a selective repeat or go back n strategy for packet pipelining.

- **Protocol Performance** At the end of a successful run, Netcp displays statistics about the run, such as the time taken in user space, system space etc. Using some of these measures, the students can compare the performance of the various flow control protocols. We think that this helps students in appreciating the importance of performance models.

4 Conclusion

4.1 The Projects in our course

The NetCp package was used in our course to support a sequence of two assignments, covering approximately the first half of the course. The overall goal of the two assignments was to develop an increasingly sophisticated working model of the data link layer.

In both assignments, the overall goal was to transfer files between machines using the NetCp package.

The first assignment concentrated on the issues of framing, check sums, and a simple timeout/retransmission protocol. The students were provided with code from NetCp to handle the command to transfer a file, to simulate the upper layers of the protocol (the hybrid layer) and to simulate the physical layer, as well as functions FromHybridLayer, ToHybridLayer, FromPhysicalLayer, and ToPhysicalLayer to receive or send data to the other layers.

The hybrid layer provide a stream of bytes to the student's data link layer. The student's program needed to manage the framing and checksums, and then start a timer when a packet was sent to the physical layer. On receiving a packet from the physical layer, the student's program needed to verify the checksum, take appropriate action if the frame was an acknowledgment, send an acknowledgment if necessary, and then pass the data to the hybrid layer. The student's program also had to retransmit packets if a timer expired without an acknowledgment. The students used a send-and-wait protocol, so that after a packet was transmitted, no additional data packets were transmitted until an acknowledgment was received or the timer expired. The physical layer was configured both to lose packets and to introduce errors in packets.

Assignment 1 ran entirely on a single machine, and no data was actually transmitted across the network. Instead, the simulated physical layer merely passed the data (after appropriately introducing delays and errors) from one copy of the student's data link layer to another copy.

The second assignment required the students to introduce considerably more complexity into their model data link layer programs. The primary complication was to introduce a sliding window protocol, so that several packets might be sent before an acknowledgment was received. This made the management of timers much more complicated. In addition, the students' programs used piggybacking for acknowledgments in the second assignment. Under this scheme, when possible, acknowledgments are not sent as separate packets, but are included in data packets.

4.2 Student reaction to the project assignments

In past years, one of us has taught a senior-level operating systems course in which the students had a single project that they implemented in stages throughout the term. Student reaction to that project has been strongly favorable, especially from alumni. In fact, the experience with that operating systems course encouraged us to design this computer networks course around a large-scale project.

The project in this computer networks course received generally favorable comments from the students in the end-of-term course evaluations. However, a substantial number of students were critical about the project. One criticism was that the amount of coding required was too large. The lesson seemed to be that in previous courses, many students were able to complete programming assignments with a minimum of planning – the assignments were small enough to write from scratch. These assignments, however, were significantly larger, and we observed that the successful students started early (of course!) and had a clear design for their program before they began to code. Perhaps the presentation of the assignments needs to be modified to encourage or require the students to go through a more formal design process for their assignments.

The other major criticism revolved around some students' difficulty in writing program for which parts had already been written by someone else. Again, in previous courses students had typically written the entire programs by themselves, so this course provided some new experiences for them. We feel that learning to program in an environment in which parts of a program have been written by someone else is an important skill for our students, many of whom anticipate careers in software engineering. This observations suggests that we should make some changes in the presentation of the assignments to prepare students for this challenge.

4.3 Availability of the software

The NetCp package is available by anonymous ftp from wpi.wpi.edu (130.215.24.1) in the directory NetCp. After setting the transfer mode to binary, get the file netcp.tar.Z After retrieving the file, it must be uncompressed and un-tarred. The package includes a file giving instructions on how to compile and use NetCp. Comments from instructors using NetCp would be very much appreciated by the authors.

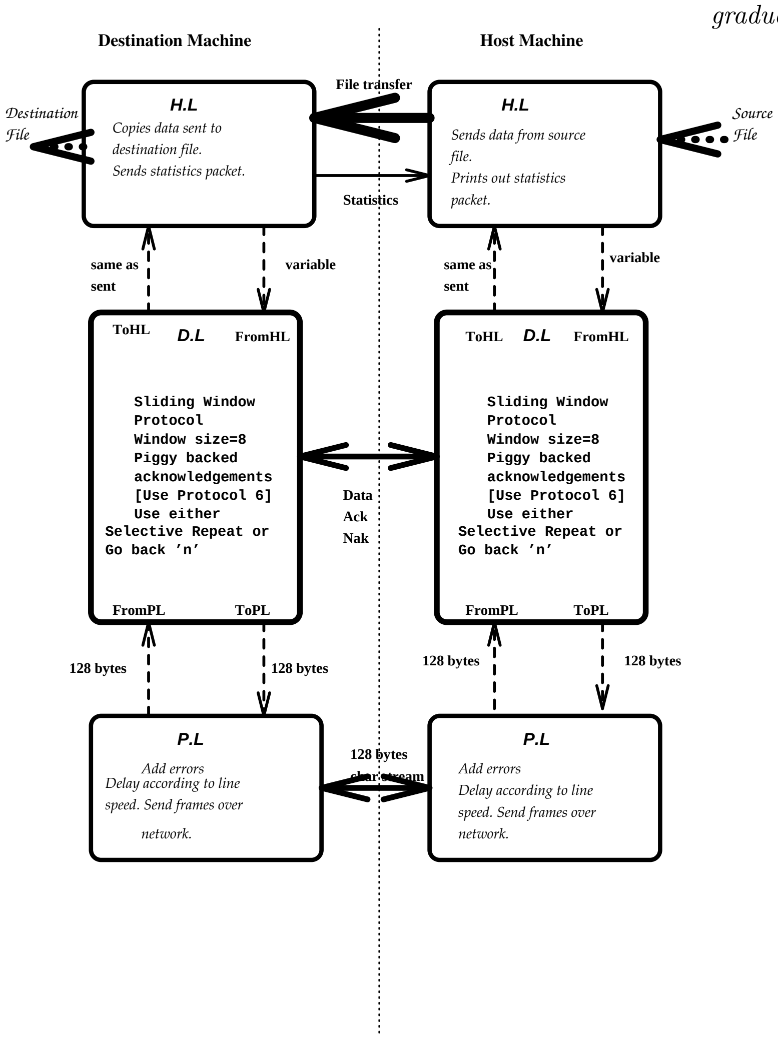

Figure 1: Overview of netcp.

References

[KIFE91] M. Kifer and S.A. Smolka, *OSP: An Environment for Operating System Projects*, Addison-Wesley Publishing Company, Reading, Mass., 1991.

[TANE87] A.S. Tanenbaum, *Operating Systems Design and Implementation*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1987.

[TANE89] A.S. Tanenbaum, *Computer Networks*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1989.

[WORC93] Worcester Polytechnic Institute, *Undergraduate Catalog, 1993-94*, Worcester, Mass.