Network Programming – Beyond Sockets

Hugh Smith
Cal Poly Computer Engineering Program

Abstract

In this paper we present a methodology for teaching computer network programming. In typical Computer Networking textbooks used in networking courses the only coverage of network programming is a discussion of the sockets API. This approach seems logical since many times the students taking a Computer Networking course are upper division students who already know how to program. The problem with this approach is most students have not written asynchronous programs that need to work together to solve a problem as required in network programming. The methodology we present includes a process for breaking down the problem, developing state diagrams and then implementing state machines in C. This methodology consists of a number of modules that walk the students through this process. Our observations are that this approach has improved the completion rate and quality of the student programming assignment. We present the results of a student survey that indicates that this process is very helpful in implementing their final programming solution.

Keywords

Computer Networks, State Machines, Socket Programming

Introduction

This paper addresses an issue experienced with teaching students to write client-server based computer networking programs. While the students in the class are upper division Computer Science (CSC), Computer Engineering (CPE) or Software Engineering (SE) students they lack experience in writing asynchronous programs as required in client-server systems. While the students have a good understanding of how to use the socket API[3] to allow their programs to communicate across the network, they have not been taught a methodology to implement this type of system. In addition, we have observed that the above average programmer can successfully implement a complex client-server program, many of the average programmers take three to four times as long to implement a much weaker solution. These solutions tend to be an organically written (program grows as they encounter new conditions) and are almost impossible to read and debug.

In many computer networking courses, network programming (typically called socket programming or client-serve programming) is glossed over. Many textbooks[1,2] used in these courses cover the basics of using the sockets API to implement network programs. These textbooks also cover concepts such client-server systems and peer-to-peer communications. The issue with this approach is that while it covers the technical components of network programming (e.g. using the sockets API) it does not teach the students any techniques or process for actually creating (designing and implementing) networked systems.

The mythology I present in this paper consists of a number of modules. These modules work together to teach the students how to conceptualize the networking system they are implementing, how to document the system flows and how to develop state machines that meet the program specifications. These assignments consist of multiple design assignments and the implementation of a complex client-server system.

The overall process consists of developing packet flow diagrams (figure 1) based on the program specification and then using these packet flow diagrams to generate state diagrams. The process forces the students to create the state diagrams prior to implementing their programs. While state machines are well known in networking and digital design, we have found that typical CPE students do not carry over their state machine skills into their network programs. In addition, CSC and SE majors may not have any experience in state diagrams and state machines. Therefore, as part of the process the concept of state diagrams and how they may be applied to networking programs is presented to the students.

As shown in our results below, this approach has improved the completion rate and quality of the students' programming assignment. In addition, I present the results of two student surveys that address using this methodology in developing programs.

In the remainder of this paper we present an overview of the course, the steps used to teach a formal process for networking programming and our results based on instructor observations, student grades and surveys of the students.

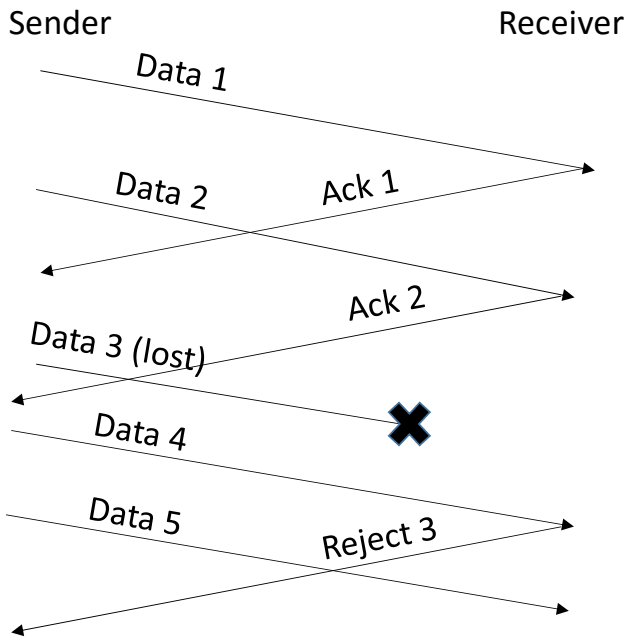

Figure 1: Packet Flow Diagram implementing a Sliding Window Flow Control Algorithm

Course Overview

The computer networking course is taken by senior level computing majors (CSC, CPE, SE). These students have had a minimum of four quarters of required programming courses. This includes CS1-CS3 where they learn C or Python and then Java. They are also required to take a Systems Programming course where they learn to use Unix systems calls similar to the Sockets API.

Cal Poly is on a 10-week quarter system. The computer networking course consists of 3-hours of lecture per week and 3-hours of lab. In the lecture we cover the major aspects of the protocol stack layers 1-4. The lab component of the course is similar to an Electrical Engineering lab. The lab is held in the Cal Poly Advanced Networking Laboratory. This laboratory has 15 workbenches. Each workbench consists of multiple PCs, switches and routers. The students are given a lab procedure and have 3-hours to complete the lab assignment. The students are then required to turn in a lab report.

In addition to the lecture and lab components of the course, the students are required to implement multiple programs in C. The final programming assignment is the focus of our work on function decomposition and includes teaching a process for implementing client-server programs. This programming assignment is an implementation of a reliable file transfer using UDP. The students are required to design their own reliable communications protocol using sliding windows. Their implementations must use the internet checksum for bit error detection and selective repeat ARQ for error recover and flow control. The students link in a library that provides a modified version of the socket API function sendto(). This modified version of sendto() is able to introduce errors into the data transmitted by the application by modifying bits and dropping packets based on a configurable error rate.

Issues and Observations

We have been assigning this file transfer assignment for over ten years and have identified a number of issues:

1) Lack of functional decomposition skills – The students do not follow any formal method for functionally decomposing their larger assignments into meaningful subtasks. There seems to be a number of reasons for this. This includes the fact that many of their assignments in earlier classes could be "designed" successful in their head and hacked together. In addition, the students have learned that it is the final product (e.g. a working program) that counts and not the process.

2) It is difficult to successful hack together client-server programs – While many of the programs in earlier programming courses can be finished by applying more time and effort, this is not true for a complex asynchronous set of programs like this programming assignment. Informal observations indicate that students spending 30+ hours on this assignment are less likely to finish the assignment than students spending 15-20 hours. The more they hack their code to handle special cases that were not considered during the design process the less likely they are to successfully finish the assignment.

3) Asynchronous programming is new to most students. For many students their past programming experience is a based on implementing single linear programs. Even in our Operating Systems course where students work with multiple processes the program flow tends to be linear.

For many of these students, switching into an asynchronous programming paradigm (e.g. found in distributed systems, parallel process, and client-server systems) is not natural and they struggle with how to begin their assignment. Even though state diagrams and similar concepts are covered in our digital design and architecture courses, the students fail to see how these concepts can be applied to networking programs.

4) Many of the popular textbooks used for teaching computer networks[1,2] only cover socket programming from the API perspective. A methodology for going from a problem specification to a client-server implementation is not discussed.

Our approach

As discussed earlier the students are implementing a reliable UDP based file transfer client-server system. In order to walk the students through the process of designing this assignment we use a five-part process.

1) Packet flow diagrams – during discussion of different protocols (e.g. ARP, TCP) the students are show how to draw packet flow diagrams. Figure 1 shows an example of a packet flow diagram for a simple sliding window flow control algorithm. In this diagram the sender is sending data to the receiver and then processing acknowledgements and rejections as they asynchronously arrive from the receiver. The students actual design would require many different packet flow diagrams cover every possible packet flow. These diagram allow the students to depict the flow of information between multiple machines and to document possible error conditions and edge cases.

2) Mealy state diagrams/machines – The students are taught how to create state diagrams and implement mealy based state machines. Figure 2 shows a simplified state diagram for the sender of the data as shown in figure 1. The students are show how to use their packet flow diagrams to generate and validate their state diagrams prior to implementing the code for their state machine.

3) Stop and Wait design assignment – The students are then given a program specification for a reliable UDP based file transfer using a simpler stop and wait protocol for flow control and error recover. They are also given a series of design questions that walk them through the development of their packet flow diagrams. The final output of this assignment is a state diagram for both the client and the server that implements this reliable file transfer. This is only a design assignment and the students are not required to write programs to implement their state machines.

4) Instructor thought process and solution – After the students turn in their stop and wait design assignment, the instructor goes over his/her solution to the stop and wait design assignment. The instructor discusses mistakes made during his/her design process and design tradeoffs. The development of the state diagram is presented on the whiteboard with an on-going

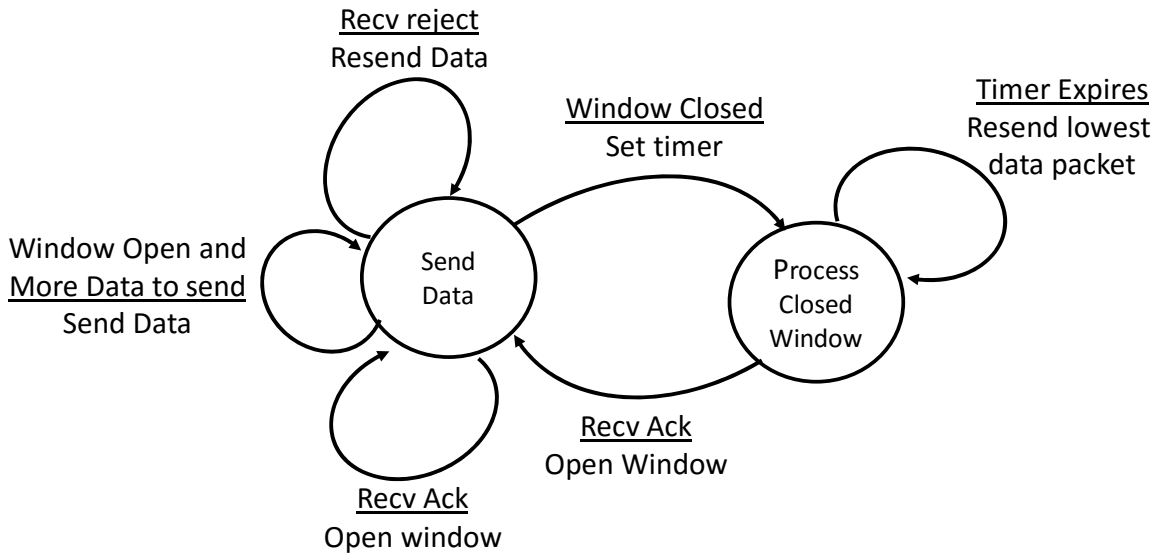

Figure 2: Simplified (incomplete) State Diagram for Sending Data

(Diagram for the Sender in the packet flow diagram of figure 1)

commentary on the thought process being used by the instructor. In addition to sharing the state diagram a solution in C that implements the state machines for the state diagrams is presented.

The students' design assignments for the stop and wait protocol are not graded in detail. It is up to the student to analyze issues with their solution based on the solution presented by the instructor. This allows the students to learn from their mistakes before they are required to actually implement a program (which is graded).

5) Programming assignment – The students are given a programming assignment to implement a reliable file transfer over UPD using selective repeat for flow control and error correction. This assignment is given in two parts.

a. Design assignment – Similar to the earlier stop and wait design assignment, the students develop packet flow diagrams and state machines that solve the file transfer problem. Similar to how TCP works, the students designed is required to include a connection establishment process, a reliable data transfer with flow control and a connection termination process.

While this design assignment is collected it is not graded and no solution is presented by the instructor. No feedback from the instructor is provided to the students prior to their implementation. The students learn from mistakes in their design as they code and debug their selective repeat based reliable file transfer program.

Prior to Implementing Design Methodology		After Implementing Design Methodology	
Fall 2009	50%	Winter 2016	85%
Spring 2009	48%	Spring 2016	65%
Fall 2008	56%	Fall 2015	56%
Winter 2008	40%	Winter2015	78%

Table 1: Percentage of students receiving a non-zero grade for Selective Repeat assignment (higher is better)

b. Working file transfer program – The students are then required to implement a working version of the reliable file transfer over UDP using sliding windows with selective repeat for flow control and error recover. The students are not required to implement their state diagrams and instead are allowed to choose how they want to implement their program.

Instructor Observations

For a subjective point of view, we are very pleased with the results of this process. There have been numerous comments from the students about using this technique in other courses (e.g. our parallel processing course and their senior project). Also a number of alumni have reported using this technique in the jobs after graduation.

The instructor notes that this process has made a noticeable change in student participation when discussing the selective repeat assignment in lecture. During these discussion, student questions

Survey Question	Value (1-5) 5 = strongly agree with the statement
"I learned a lot from the two design assignments"	4.0/5
"I feel that the design assignments helped me to break down my implementation of program #3 into smaller/more manageable pieces (e.g. functions)"	4.18/5
"The design component of the course helped me feel more comfortable with the process for breaking down a large problem into manageable pieces."	4.1/5

Table 2 Computer Network Course End of Quarter Survey (higher is better)

are more insightful and they indicate an understanding of the issues involved in this assignment. Student questions during office hour are more directed and show a better understanding of the steps needed to implement their program.

Grade and Survey results

While it is very difficult to isolate and measure curriculum changes, we have collected metrics from a number of sources:

1) Program grades – The grading rubric for this assignment has changed over time (e.g. extra credit, changes in point deductions) and we feel a direct comparison of program grades before and after the change would not be accurate. What has not changed is what it takes to get a zero on this assignment. Table 1 shows the percent of non-zero grades for the selective reject programming assignment for four quarters before the change and four quarters after the change (higher is better). Consistent with the instructor's observations, the number of non-zero grades has increased since implementing this change.

2) During the spring quarter of 2016 the instructor performed a survey of the students that included questions regarding this design process. Table 2 shows the results of the three questions asked about the effectiveness of this process. This survey used a 5-point scale with 1 being strongly disagrees with the statement and 5 being strongly agrees with the statement (higher is better). The students' perception of the benefits of this change match the instructions observations.

Concerning our observations prior to making this change (see section on Issues and Observations) these survey shows that the students recognize the benefits of this approach when writing their network programs. At least for network programming these results suggest that the students believe that decomposing the problem using the packet flow diagrams and state diagrams is helpful in successfully completing their network programs.

Survey Question: For the following classes, how often did you perform a structured Top-down/Bottom-up Design when developing hardware and/or software.	
Course	**Value (1-5) 5 = Extensively**
Computer Networks	4.33/5
Operating Systems	3.53/5
Embedded Systems	3.60/5
Digital Design	3.13/5

Table 3: Senior Student Survey on Functional Decomposition (higher is better)

3) General Senior Survey – In an effort to address issues identified with our students' ability to perform function decompositions of problems prior to programming them, our CPE office performed a survey of our senior level students. In this survey the students were asked for a given course "how often did you perform a structured Top-down / Bottom-up Design when developing hardware and/or software". Table 3 shows the results of this survey for a number of upper division programming courses. This survey was also based on a 5-point (higher is better) scale. The results of this survey show that students recognize the focus on a methodology for network programming as presented in this course.

Conclusion

In this paper we present a methodology for teaching the design and implementation of asynchronous networking programs. This methodology walks the students through the process of understanding the problem through the use of packet flow diagrams. The students then create a state diagram that is able to process all of the possible scenarios identified in their packet flow diagrams. By first creating the packet flow diagrams the students are able to identify edge cases and verify that these conditions are handled correctly in their program.

We have found this approach to be helpful in teaching networking programming. The students recognize this benefit as shown through both the grades the students receive for the assignment and their subjective input based on class surveys.

In the future we plan on implementing this methodology earlier in the quarter and having the students use this methodology for multiple programming assignments. In addition, while its clear that the students see a benefit in designing their network programs, it is also clear that this is not translated into them designing in other courses. We are in the process of increasing the use of functional decomposition of problems in earlier programming courses. While the technique used in this course (packet flow and state diagrams) may be applicable to a subset of courses, the idea of breaking down their programming assignments into meaningful pieces prior to implementing their assignments will benefit the students in many courses.

References

1 Tanenbaum, Andrew, Computer Networks (5th edition), Pearson, ISBN-13:9780133506488, October 2010

2 Stallings, William, Data and Computer Communications, Pearson, ISBN-13: 9780133506488, September 2003

3 Toll, William, "Socket Programming in the data Communications Laboratory", ACM SIGSCE '95, Nashville TN, 1995.

Hugh Smith

Hugh Smith is an Assistant Professor in the Computer Science Department at the California Polytechnic State University (Cal Poly). He is also a member of the Cal Poly Computer Engineering program. He received his B.S. degree in Computer Science from Xavier University in Cincinnati Ohio. He received his M.S. and Ph.D. degrees in Computer Science from Michigan State University. His current interests are in the area of computer networks and embedded systems.