Computer Science Education
2001, Vol. 11, No. 1, pp. 1–14

0899-3408/01/1101-0007$16.00

The Network Workbench and Constructivism: Learning Protocols by Programming

J. Mark Pullen
Department of Computer Science, George Mason University, Fairfax, USA

ABSTRACT

The Network Workbench (NW) is a software environment for academic investigation of network protocol concepts. The structure and operation of NW has been described in detail elsewhere. This paper addresses the instructional design of NW with the intention of establishing that it represents 'natural constructivism' in that it is strongly aligned with the constructivist approach. We begin with synopsis of constructivist educational philosophy. After this the architecture of NW and the facilities it provides are described, as are the 12 associated protocol programming projects. Based on these descriptions, the design of NW and its projects are evaluated in light of principles of the constructivist educational philosophy and shown to be in very strong alignment with them. We conclude that, although its development did not proceed from a formal grounding in constructivism, NW represents successful implementation of constructivist philosophy as a natural approach to teaching protocols for computer networking.

INTRODUCTION

Computer science is a very young discipline. As educators in this discipline, we must find ways to teach concepts and technologies that did not exist as little as one decade ago. Study of formal educational philosophy can help us to do this, as can investigation of successful educational tools and practices. This paper examines a tool for computer science education called the Network Workbench (NW), along with the successful practices associated with that tool, in light of the educational philosophy called constructivism. By highlighting common factors between the successful use of the tool and the principles for success espoused by the philosophy, we can confirm the applicability of constructivism as a promising basis for developing new practices in computer science education.

Correspondence: J. Mark Pullen, Department of Computer Science, George Mason University, Fairfax, VA 22032, USA. E-mail: mpullen@gmu.edu

The Network Workbench

The NW is a software environment for academic investigation of network protocol concepts that has been developed at George Mason University (GMU). NW consists of a collection of software modules written in the C++ programming language, implementing a simplified but complete protocol stack modeled after the primary protocols of the Internet. A discrete event simulation (DES) system provides a means for the protocols to be executed in an observable, repeatable way by students. The students are able to create their own versions of each of the protocols in the stack and test their implementations either as individual protocols or in an integrated stack. NW has been described in detail by Pullen (2000a,b). The purpose of this paper is to demonstrate that the philosophy, implementation and observed benefits of NW represent 'natural constructivism' in that they conform to constructivist learning theory, even though NW was developed with little formal awareness of the details of that theory. In this sense, constructivism is a natural and effective approach to the educational area addressed by NW: supporting the process whereby students come to understand the basic principles and processes involved with Internet protocols.

Constructivism

Gruender (1996) provides a lucid description of the constructivist philosophy, which has its roots in the writings of 18th century Italian philosopher Giambattista Vico who held that human learning consists of constructing ideas based on observations. According to Gruender, modern constructivists believe that ''when one constructs for oneself a solution to a problem, the bits of knowledge that one's resulting ideas consist in and which yield their own satisfactions in one's life become a part of oneself.'' In other words, by puzzling out the solution to a problem the student internalizes the knowledge involved and develops an affinity for it due to the intellectual reward of solving the problem. Gruender goes on to compare and contrast the contributions of several other educational philosophies with those of constructivism. He concludes that effective instructional design should integrate the lessons of each of the philosophies.

> ''What we need... is a knowledge of effective techniques for helping students harness their own vitality, energy, curiosity, excitement, and wonder in the difficult and sometimes painful tasks of learning what they will need to know.''

This statement captures the motivation behind NW quite well. By the time they study network protocols, students of Computer Science generally are skilled at programming and find it to be enjoyable. They enjoy the challenge of starting with the algorithm for a protocol and converting it to a working solution. In the process of solving the programming problem they construct their internal knowledge of how it works.

Lebow (1993) points out that there is ''a tendency for theorists to treat contructivism as a method when it is actually a philosophy.'' He articulates five principles that respond to the question ''For what problems is contructivist philosophy the solution?'' His five principles of constructivism are the following.

- Maintain a buffer between the learner and the potentially damaging effects of instructional practices.
- Provide a context for learning that supports both autonomy and relatedness.
- Embed the reasons for learning into the learning activity itself.
- Support self-regulated learning by promoting skills and attitudes that enable a learner to assume increasing responsibility for the developmental restructuring process.
- Strengthen the learner's tendency to engage in intentional learning processes, especially by encouraging the strategic explanation of errors.

In their classic paper relating constructivist theory for instructional design and teaching practices, Savery and Duffy (1995) describe the following tenets for the constructivist approach.

- Anchor all learning activities to a larger task or problem.
- Support the learner in developing ownership for the overall problem or task.
- Design an authentic task.
- Design the task and the learning environment to reflect the complexity of the environment they should be able to function in at the end of learning.
- Give the learner ownership of the process used to develop the solution.
- Encourage testing ideas against alternative views and alternative contexts.
- Provide opportunity for and support reflection on both the content learned and the learning process.

Taken together, these two sets of principles create a clear picture providing a basis to determine whether an educational tool such as NW is in alignment with the constructivist philosophy. The remainder of this paper is aimed at

establishing that NW is so aligned. Because of the clear emphasis in constructivism on activities that support learning, such a determination requires an examination of how NW is intended to be used. For this purpose, the remainder of the paper is divided into two expository Sections and a concluding summary. The first Section describes NW in sufficient detail to enable the reader to understand how it works and how students are able to use it to develop their own authentic implementations of a working protocol stack and to do so without spending excessive time in tangential issues that are not central to protocol operation. The following Section provides a detailed comparison between the educational process associated with NW and the tenets of constructivism.

Network Workbench Motivation and Principles

Teaching introductory networking poses a challenge. We know that students learn best those concepts that are reinforced by activities requiring them to use the concepts. However, networking is a complex, interdisciplinary subject. The simplest meaningful project could bog students down for long periods of time in irrelevant details of socket-level programming. Other difficulties for students would include debugging programs under constantly changing conditions due to other students' activities on the LAN and difficulty in understanding the relationships of the numerous protocols from available programming tools such as memory dumps. Furthermore, creating a running protocol at the lower layers of the stack typically requires modification of the computer's operating system. This is a complex, arcane activity that is not suitable for an introductory course.

Further complexity arises from the need to understand the most important collection of protocols in use today, namely the core protocols of the Internet. There is great value in the interoperability made possible by these protocols. However, to achieve this interoperability requires adherence to a particular set of rules for information transfer. It is important that students learn how interoperability it facilitated by building a protocol stack that conforms to the protocol definitions. Therefore, implementing the protocols means both solving a technical problem (programming a protocol that will communicate with another instance of itself) and adhering to a techno-social compact (programming a protocol such that it will interoperate with a different implementation).

Given these difficulties, a network simulation environment represents the best compromise between the problems associated with student network

programming and the need for students to reinforce classroom learning by doing a real project. Simulation is widely used to abstract away non-essential physical details (which also are likely to be non-repeatable in testing) while exercising the critical aspects of a protocol. Simulations run as user programs in the computer, avoiding problems associated with modifying the operating system.

Following this line of reasoning to its logical conclusion led to a 5-year effort that produced the NW. Network Workbench (NW) is available to the academic community under no-cost license to support investigation of network protocol concepts. It contains a complete protocol stack, abstracted from the Internet stack, and a set of exercises that focus on critical protocol algorithms used in the Internet stack. It also has proved sufficiently powerful, comprehensive, and extensible to serve as a basis for advanced student research projects. Examination scores on protocol questions have shown that the principles associated with NW projects are learned well, while comments on course-end critiques have shown that students value the projects. About 300 students per year now complete NW projects as part of GMU networking courses. Other schools have begun to adopt NW for the same purpose.

Over the past 5 years NW has matured from a programming assignment given to Master's-level classes into a network simulator that has proved useful for a variety of academic projects from undergraduate through doctoral level. The following are the guiding principles in this development.

- Abstract away unnecessary details, while ensuring that the student must grapple with the central algorithms of the protocols by programming them in C++.
- Provide as much infrastructure as possible to support the learning process; in particular, provide working interfaces among all layers in the protocol stack and a complete working version of the protocol stack in executable form.
- Provide code of good quality, with all details visible except the actual program code that solves graded exercises. As a side effect of this project students gain valuable experience in working with code written by others, a task many can expect to take on after graduation.
- Provide complete visibility into all aspects of the network being simulated, including the ability to observe protocol data units as they move between any two layers in the stack and complete statistics showing performance metrics for each simulation run.

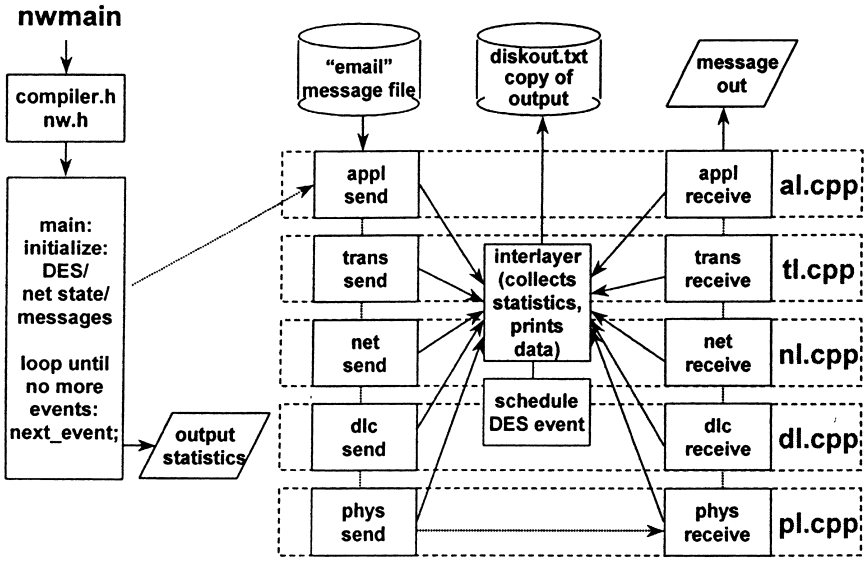

Fig. 1. Network Workbench (NW) system architecture.

- Keep the whole simulation system simple, understandable, and independent of physical presence in the lab where the simulation software is maintained. Include an option for running the simulation remotely using a text interface and also an option for running the simulation on the student's own computer.

Network Workbench Architecture and Projects

Figure 1 shows the general structure of the Workbench. The following are the top-level system blocks.

Header Files

A small, compiler-specific header file establishes any code definitions that differ among the supported compilers. Currently these are Unix compilers of Sun C++ under Solaris and Gnu G++ under Linux, and Windows compilers Borland C++ Builder and Microsoft Visual C++. The compiler header is followed by the main header file, nw.h, which defines base classes as well as data types, data structures, and state enumerations for the entire Workbench.

Main Program
The main execution sequence demonstrates the basic simplicity of the Workbench, in that it consists of only 12 lines of C++ code that succinctly describe the overall system logic of NW. The core of the simulation is a single line of code that repeatedly invokes the DES function *next_event*. This in turn invokes the C++ function corresponding to the next event on the DES event list, a concept that has proved easy for students to understand.

Interlayer Module
This component is responsible for much that is unique in NW. Whereas in a normal protocol stack each layer invokes the next lower layer to obtain service, in the Workbench this invocation passes through an intermediate function contained in the *interlayer* module. This module collects statistics about the operation of the simulated network, prints an optional trace that can be turned on/off at any point in the simulation, and invokes the DES subsystem to create a future event that will cause the next lower layer to be invoked. This method of execution facilitates exposing or hiding actions of various layers as appropriate to the project at hand, using a profile array that can be reset dynamically during the simulation. It also provides a mechanism whereby execution times of the individual protocol layers can be represented explicitly in the simulation by adding a simulated time between layers.

Input Files
The 'e-mail' application consists of a collection of files generated by the Unix fortune program that contain tidbits of wit and wisdom. Each file holds a number of messages appropriate to the assignment. The application layer adapts itself to the simulation to be performed. It can represent a wide area network (WAN), local area network (LAN), or a system of internetted LANs. The network topology input comes from a set of four text files. The input files are accessible to be edited by students, but generally they are used as distributed with NW, unedited.

Protocol Stack
The Workbench stack is abstracted from the Internet (TCP/IP) stack. Like the Internet stack it contains five layers. Much of the detailed C++ code in NW is found in modules for the application layer (al), transport layer (tl), network layer (nl), datalink control layer (dl), and physical layer (pl).

Discrete Event Simulation Subsystem
This small and highly robust part of the Workbench is responsible for maintaining a linked event list, removing the latest event from the list when *next_event* is invoked, and invoking the function corresponding to that event.

Output Files
The text outputs of the Workbench are the initial network description, the results of interlayer traces, and the summary statistics. The statistics are selectable according to the layer from which they were collected. All text outputs are also written to a disk file so that the results of the simulation can be reviewed.

Project Code Stubs
Each code module to be written by students is provided in the NW library in the form of a stub module with correct C++ interfaces but containing no protocol code. For each project there is a module that establishes default conditions appropriate to that project. It also presets the layers at which protocol data transfer will generate trace output and selects statistics to be output at the end of the simulation. These presets can be overridden dynamically in student-written code for selective tracing. A setup process is available to move all required data files and code files to a working directory so that initial use of the Workbench is simple for the student.

Executable Module Library
The basic functions of the Workbench are collected in a single compiled executable module. Executables for all other code modules are available in a library file. The library includes solution modules for all projects. Solution modules are the only ones for which C++ code is not made available to the student. Available source code includes other functions a student might want to replace, for example, the bit error random number generator for network links.

Figure 2 shows the NW protocol stack. Version 4.1 of the Workbench offers 12 structured projects featuring these protocols. It also provides a generic capability to study any arbitrary protocol that fits in the five-layer context. The projects are described in detail in Pullen (2000b), including the mechanisms of the protocols and the algorithm for each project. The projects are presented in a sequence that proceeds from simpler to more challenging. The projects are listed below.

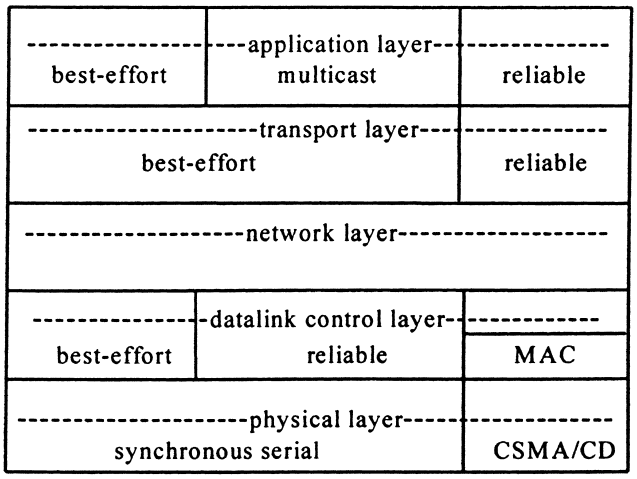

Fig. 2. Network Workbench (NW) protocol stack.

For the datalink control (DLC) layer, studies of the following are given.

- DLC frame formatting, in which the student programs functions for bit stuffing/unstuffing.
- DLC error detection, in which the student programs cyclic redundancy check calculations.
- DLC flow and error control, in which the student programs the decision logic for Automatic Request for Retransmission.

For the Medium Access Control (MAC) sublayer, studies of the following are given.

- CSMA/CD local area networks, in which the student programs a function for binary exponential backoff as used in Ethernet for resolution of collisions.
- Token passing local area networks, in which the student programs the function that deals with a received token.

For the network layer, studies of the following are given.

- WAN topology, in which the student programs a function that generates the matrices used by NW to describe the WAN.

- Network layer routing, in which the student programs functions for route optimization and packet forwarding.
- Network layer routing information distribution, in which the student programs the function that updates Link State Advertisements.
- Multicast networking, in which the student programs a function that determines the set of WAN interfaces participating in the multicast tree.
- Network security, in which the student programs a filter function for a firewall.

For the transport layer, a study of reliable transport, in which the student programs a function that implements the core logic of the sending end in a protocol abstracted from TCP.

For the application layer, a study in which the student programs the message handling function for an email list server.

For the entire protocol stack, a study of internetworking where all previous project parts are combined with the result that reliable transport of e-mail messages is interspersed with multicast stream transmission.

The Network Workbench (NW) and Constructivist Principles

The motivation behind NW and its projects began with the observation that students understand network protocols much better after they create a mental model of the protocol workings in the process of programming them. The specific context provided by NW is intended to support this process of puzzling in such a way that little of the student's time is devoted to issues that are not central to how the protocol works. For example, the interfaces between stack layers are provided by the Workbench, as are the mechanics of the discrete event simulation. This approach was based on practical observation, made over years of teaching networking, that classroom instruction was found to be less than satisfactory for imparting a real understanding of the protocols' workings. In response, NW was developed. Student response and examination results indicate that NW works very well for its intended purpose.

To understand the success of NW, it is necessary to examine the educational theory that explains why it works so well. The educational philosophy most closely aligned with NW is constructivism. This Section compares the philosophy and practices associated with NW to the principles of constructivism as set forth both by Lebow and by Savery and Duffy.

Lebow's Five Principles in the Network Workbench (NW)

1. *Maintain a buffer between the learner and the potentially damaging effects of instructional practices*. Lebow stresses this as taking precedence over all of the other principles. He believes that some educational practices can best be described as 'poisonous pedagogy' and urges that instructional design proceed from a basis of motivating the student through personal relevance of the method of study and by enabling the student to take charge of learning. It might seem that the high level of structure associated with a well-specified protocol, combined with the attention to detail necessary to program its operation, would prove confining rather than enabling to the student. However, experience with NW indicates that Computer Science students find motivation in solving these problems precisely because they are meaningful and are supported by a familiar, structured environment. The 'buffer' that Lebow calls for is represented by the supporting infrastructure that the student is *not* required to develop. This viewpoint is reinforced by the fact that, in the past, students displayed a tendency to become frustrated when using earlier versions of NW where this infrastructure was not available.
2. *Provide a context for learning that supports both autonomy and relatedness*. The Workbench conforms exactly to this principle: Each student develops an individual solution to each problem in a manner that is individually organized and paced. At the same time, the NW environment is common to the whole class of students and provides a unifying framework for the project.
3. *Embed the reasons for learning into the learning activity itself*. Successful completion of an NW project provides reinforcement on two levels: The fact that a protocol is implemented correctly means the simulated network also is able to work correctly. Thus, the student experiences again at every project the challenge of solving the puzzle, along with the importance of solving it correctly.
4. *Support self-regulated learning by promoting skills and attitudes that enable a learner to assume increasing responsibility for the developmental restructuring process*. The connection between NW and this principle admittedly is somewhat vague. In one sense the fact that the projects are presented in order of increasing difficulty and generality is a stimulus to broader learning and understanding. Still it must be admitted that most students do not move beyond the structure provided in the projects to

examine larger problems and principles on their own. This sort of growth generally is associated with mentorship by faculty in advanced courses, not with any particular stimulus arising from NW.

5. *Strengthen the learner's tendency to engage in intentional learning processes, especially by encouraging the strategic explanation of errors*. As applied to NW, this principle fits nicely. The process of debugging each protocol implementation by observing its behavior in a simulated environment is one that naturally provides feedback to correct errors. The student cannot produce a successful solution without observing whether or not it is correct and correcting the code as needed.

Savery and Duffy's Eight Instructional Design Principles in the Network Workbench

1. *Anchor all learning activities to a larger task or problem*. Each NW project takes place in the context of a complete working protocol stack, which is in turn presented in the context of understanding how the Internet works.
2. *Support the learner in developing ownership for the overall problem or task*. The student has complete charge of solving the problem (programming the protocol) while NW provides all supporting software. Supporting documentation in the project book by Pullen (2000b) provides a collection of useful programming and debugging techniques, presented without specific directions so that their employment is up to the student.
3. *Design an authentic task. The protocols are real, although simplified*. Moreover, they must function as part of a larger whole, an interoperating protocol stack.
4. *Design the task and the learning environment to reflect the complexity of the environment they should be able to function in at the end of learning*. Each protocol is simplified to the point where it has just the basic properties the student needs to understand at the completion of an introductory course. In an advanced graduate course, the students who use NW design their own protocols at whatever level of complexity is needed for projects which they propose themselves.
5. *Give the learner ownership of the process used to develop the solution*. The student programs the protocols and runs the solutions independently. Coding, debugging and testing are up to the student. NW provides a working solution in executable form for comparison.

6. *Design the learning environment to support and challenge the learner's thinking*. The NW projects become progressively more difficult and build on what was learned in previous projects.
7. *Encourage testing ideas against alternative views and alternative contexts*. Multiple protocols exist at each layer for comparison. Also the student can create a completely new protocol at any layer within the framework provided.
8. *Provide opportunity for and support reflection on both the content learned and the learning process*. A capstone project assembles all of the student's projects into a working whole. Descriptions of the supporting software mechanisms and the motivation for the method used are imbedded in the supporting materials.

CONCLUSION

Clearly, the project environment provided by NW might have been developed under the constructivist philosophy as advocated by Gruender, Lebow, and Savery and Duffy. In point of fact, however, NW was developed without recourse to any particular formal educational philosophy. We conclude that NW provides a practical demonstration that the constructivist philosophy describes a natural approach to teaching protocols for computer networking. The reasons NW works well to develop student understanding of network protocols are, without exception, directly related to educational principles derived from the constructivist philosophy.

It is reasonable to question whether this result is a general property of computer science or at least of computer networking. Some other factor, for example, the background of NW's developer, might account for the close of alignment of NW with constructivist principles. Answering this question will require detailed analysis of the educational methods associated with a significant number of tools used to teach computer science, in the manner presented above for NW. As a first step in this direction, it is intriguing to note that learning tools in widely differing areas of computer science have properties similar to NW in that they support a learning process where the student constructs a model of reality by programming within the learning environment. Examples are the Micro language and compiler presented by Fisher and LeBlanc (1991) and the *x*-kernel framework for network programming presented by Peterson and Davie (2000). These examples

support the conjecture that application of constructivist principles may in fact be a natural path for effective education in computer science. We believe this thesis deserves further investigation.

REFERENCES

Gruender, C.D. (1996). Constructivism and learning: A philosophical appraisal. *Educational Technology*, *36*, 21–29.

Fischer, C.N., & LeBlanc, R.J. (1991). *Crafting a compiler with C*. Menlo park, CA: Benjamin/Cummins.

Lebow, D. (1993). Constructivist values for instructional systems design: Five principles toward a new mindset. *Educational Technology Research and Development*, *41*, 4–16.

Peterson, L.L., & Davie, B.S. (2000). *Computer networks: A systems approach* (2nd ed.). Los Altos, CA: Morgan Kaufman.

Pullen, J.M. (2000a). The Network Workbench: Network simulation software for academic investigation of internet concepts. *Computer Networks*, *32*, 365–378.

Pullen, J.M. (2000b). *Understanding Internet protocols through hands-on programming*. New York: Wiley.

Savery, J.R., & Duffy, T.M. (1995). Problem based learning: An instructional model and its constructivist framework. *Educational Technology*, *35*, 31–38.