Writing Mini-protocol stacks as an aid to teaching networking protocols

Anand Richard, Saint Joseph's College, Rensselaer, IN

Introduction

Current research literature abounds with many innovations in approaches to teaching networking. Simulators like NS2 (Network Simulator version 2) and NS3 are widely used in academia to teach network protocols. Zenghin and Saroughian present an OSPF (Open Shortest Path First) simulator called the DEVS-Suite (Discrete Event Discrete Time Simulator) that helps students study and understand the OSPF protocol (Zengin & Sarjoughian, 2010). Yang, Yang, Gao, Shen, Zhu and Tan contend "Network protocols are mass, stuffy, abstract and difficult to understand for students to learn" and suggest a layered task based method based on the layered TCP/IP protocol itself as a solution to teach networking (Yang et al., 2010). Feitelson looks at the pros and cons of using two different textbooks by different authors that can be used to teach networking courses (Feitelson, 2007). These text books emphasize building small networks that students can use as labs to gain a better understanding. Feitelson concludes that while these approaches equip the students to become better network administrators they do not do much to make them network researchers. One is inclined to agree with Feitelson because knowing how to use a protocol is only a first step to understanding it. The 'learn by implementation' paradigm is also recommended by Uldag and McBride in implementing Bluetooth stacks as a method to teach networking (Uludag & McBride, 2010).

We can summarize the approaches in the research literature as a 4 step continuum:

1. Learn what the protocol does.
2. Use it in laboratory settings.
3. Analyze it using standard tools like simulators and protocol analyzers.
4. Study the specification with a view to implement it.

We posit the 4th step as the most important, particularly from the perspective of equipping the student to undertake research in the protocol.

Understanding a network protocol's inner workings demands a close scrutiny of its standard or specification. For example, to understand the Spanning Tree algorithm one must read and understand the appropriate sections in the IEEE 802.1D standard. In teaching these protocols while the main concepts can be 'Power Pointed', such an approach imparts only a superficial understanding. We suggest protocol implementation as a sine-qua-non for deeper understanding. This can be daunting due to the size of the undertaking. However we show here a methodology that can reduce the complexity and magnitude of the task significantly.

Leveraging the layered model

In any commercial software product there is core functionality with a mandatory set of features built in a layered model. The minimum product definition applies to each layer and conveniently enables us to leave out features from each layer and still end up with a working whole protocol stack. This rule can be used in minimizing protocol specifications resulting in a ‘feature set of interest’. The implementation of such a design will conform to only those parts of the specification that have been implemented. Such a protocol stack can even be shown to interwork with other fully implemented commercial stacks if we restrict the interworking to the feature sets implemented.

Suitability of tunneling protocols for mini stack implementations

Tunneling protocols can be defined as those that depend on a traditional TCP/IP stack to get from source to destination. DNP3 (Distributed Network Protocol) over TCP/IP and Modbus over TCP/IP are good examples. Since all the hard work of communicating the bits from source to destination is done by the tunnel protocol (TCP/IP), the tunneling protocol (DNP3) implementation need not fully implement layers that ensure guaranteed and error free delivery of packets. We will show this in our implementation of DNP3 over TCP/IP.

Overview of DNP3

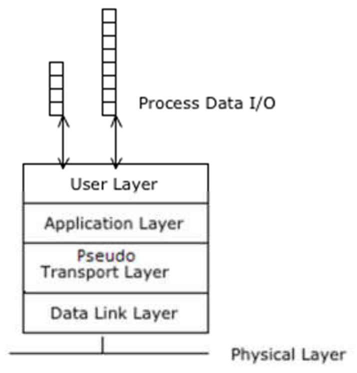

Fig 1 DNP3 Protocol Stack

Fig 1 shows the DNP3 protocol stack (“DNP3 Primer,” n.d.).

DNP3 is a request-response protocol used primarily in the electric utility industry. Messages are exchanged between Master devices that are clients and Outstation devices that are servers. Outstations are connected to process machinery, control equipment etc. They are in direct contact and control of process parameters. Master devices are often upstream in the monitoring and control area and regularly poll Outstations in the field for data and status information. As Fig 1 above shows, DNP3 has a 4 layer structure. We will briefly describe these layers.

User Layer

The User layer is not actually a part of the stack but is shown for completeness. This layer would be the HMI or SCADA application on a Master that is issuing control commands or polling for data. These are encapsulated into Application Layer packets.

Application Layer

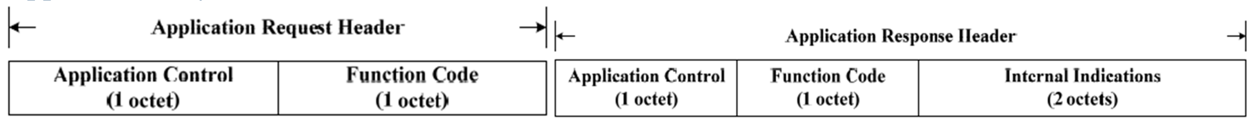
Application Request Header
Application Control (1 octet)
Function Code (1 octet)
Application Response Header
Application Control (1 octet)
Function Code (1 octet)
Internal Indications (2 octets)

Fig 2 Application Layer Header

Fig 2 above shows the Application layer packet(“IEEE SA - 1815-2012 - IEEE Standard for Electric Power Systems Communications-Distributed Network Protocol (DNP3),” n.d., p. 21) . There is a request and a response type. The function code value tells the DNP3 device what task is being requested. The Application layer takes the data from the user layer and splits it into manageable chunks with a maximum of 2048 bytes and adds a 2 or 4 byte header. This unit is called an APDU (Application Protocol Data Unit). The APDU is passed to the Pseudo Transport layer.

Pseudo Transport Layer

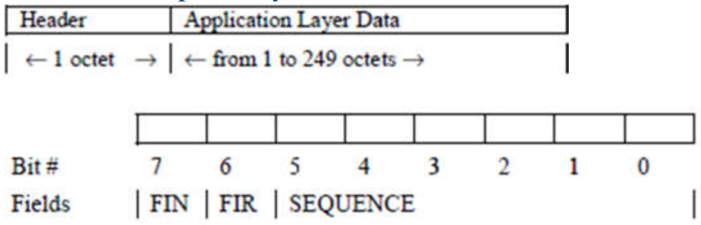

Fig 3 Transport Layer Header

Fig 3 above shows the transport layer packet and the header fields (“IEEE SA - 1815-2012 - IEEE Standard for Electric Power Systems Communications-Distributed Network Protocol (DNP3),” n.d., p. 267,68). The transport layer’s main function is to break the application layer packet into 249 data bytes + a one byte header. This unit is called a ‘Frame’. It is also termed a TPDU (Transport Protocol Data Unit) that is handed to the data link layer. The FIR (First) bit set indicates this is the first packet in a sequence and the FIN (Final) bit set indicates this is the final packet in the sequence. Sequence numbers are 4-bits wide and indicate the order of the frames.

Data Link Layer

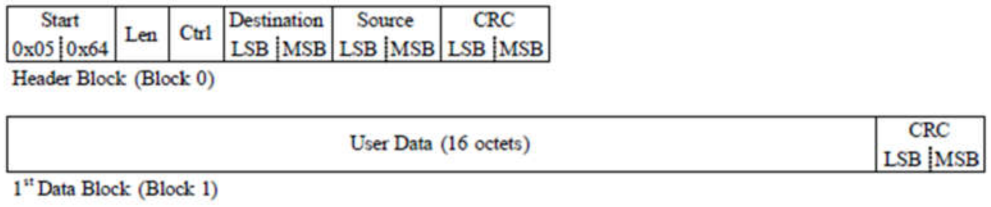

Fig 4 Data Link Layer Header

Fig 4 shows the data link layer packet header and payload (“IEEE SA - 1815-2012 - IEEE Standard for Electric Power Systems Communications-Distributed Network Protocol (DNP3),” n.d., p. 276). The data link layer is responsible for error correction and adds a 10-byte header to the TPDU which is the overhead for the correction tasks. A 16-bit CRC is used and the frame becomes a 292-byte sized unit. Multiple 16-octet frames can be added to the data link header but each carries its own 16-bit CRC as shown.

Physical Layer

The physical layer converts the TPDU into an asynchronous bit stream over a physical medium such as RS-232C, RS-485 or Ethernet.

Physical Layer Details

The physical layer protocol uses 8-bits, one start bit, one stop bit and one parity bit. The voltage levels are RS-232C and conform to RS-232C control signaling. The CCIT V.24 protocol is used for DTE/DCE communications.

Physical Layer Tasks

The physical layer must provide for the following:

1. Connection establishment
2. Disconnection
3. Transmit
4. Receive
5. Status

Methodology

Table 1 below shows the most important features of the DNP3 stack. The items in bold italics show a possible selection of what features in each layer could be implemented.

Table 1 Methodology Summary

Application Request	Application Response	Pseudo-Transport	Data Link
Read	***Response***	***Encapsulation into transport frames***	***Encapsulation of transport packet into data link frames***
Write	***Unsolicited Response***	Segmentation	***De-capsulation from data link to transport frames***
Select	Authentication Response	***De-capsulation into application frames***	***Error detection via checksums***
Operate			***Source and Destination Addressing***
Direct Operate			Send/Receive Confirm
Direct Operate –no response			Lost/Repeat packet detect

Freeze			Flow Control

Application Layer
Mandatory features to be implemented in the application layer could be to write packet building routines conforming to the packet header and payload structure. Optional items would be the number and kind of function codes we choose to implement/support. For example we could choose to implement only the first three codes of Confirm, Read and Write and the Response codes to build and handle response packets.

Transport Layer
The main function of the transport layer is to break up APDUs into smaller chunks of 250 bytes to facilitate error-free transport over noisy links in factory environments. Restricting the APDU size to be < 250 bytes would obviate the need for implementing the segmentation feature and tracking of sequence numbers. The work in the transport layer is then reduced to merely adding the TPDU header.

Data Link Layer
The data link layer performs encapsulation of TPDU frames at the source and de-capsulation at the destination. It provides checksums that ensure error free delivery. It also provides a 2-byte source and destination address for each DNP3 device. These features are redundant due to the TCP/IP tunnel via Ethernet. However a third party protocol analyzer would not recognize the packet as a DNP3 packet if these features are left out. This makes them mandatory. Send/Receive confirmation, Lost/Duplicate packet detection and flow control are truly optional and can either be totally left out or implemented at a later stage when all the other parts are found to be working fine.

Choice of Programming Language
In the matter of choosing a language, speed and simplicity restrict our choices to high level languages like C#, Ruby and Python. The main virtue of these languages is, they come with built in libraries to handle the TCP/IP work.

Verification with Third Party Tools
We recommend that Third Party tools like online DNP3 decoders and/or Wireshark can serve as a simple protocol integrity check.

Results
In our mini protocol stack for DNP3 we implemented the following:

a. Binary Outputs
b. 32-bit Counter
c. Class 0 poll

Binary Outputs
We set up 3 binary outputs. This was a Group 10 Variation 1 implementation of binary output packed format. We conducted a write operation followed by a read operation to determine if the values written were correct. All the exchanges were monitored with Wireshark and analyzed for

correctness. The packets were tested with the online opendnp3 protocol analyzer to determine if they were correct.

32-bit Counter

This was a Group 20 Variation 1 implementation of 32-bit binary counter with flag. The counters were configured in the Outstation device and initialized with values and read from the master device. The packets were tested with the online opendnp3 protocol analyzer for correctness.

Class 0 Poll

All static and non-event type data is categorized as Class 0. A Class 0 poll will return the data in this class. In our case it returned data for the 32-bit counter since that was the only one configured in the Outstation.

Conclusion

Based on our work, a curriculum for 3rd and 4th year students majoring in computer networking can be designed for study of a protocol that could be split into a 2 semester effort. The planning and analyzing can be done in one semester followed in the next semester where the students implement the protocol. Our C# DNP3 stack is available for examination and use with permission at https://github.com/kiranand/GitCode.

References

1. DNP3 Primer. (n.d.). Retrieved September 5, 2016, from http://www.dnp.org/AboutUs/DNP3%20Primer%20Rev%20A.pdf
2. Feitelson, D. (2007). Teaching TCP/IP Hands-On. *IEEE Distributed Systems Online*, *8*(11), 5–5. https://doi.org/10.1109/MDSO.2007.66
3. IEEE SA - 1815-2012 - IEEE Standard for Electric Power Systems Communications-Distributed Network Protocol (DNP3). (n.d.). Retrieved from https://standards.ieee.org/findstds/standard/1815-2012.html
4. Uludag, S., & McBride, B. (2010). Work in progress #x2014; Teaching networking concepts through Bluetooth software implementation. In *2010 IEEE Frontiers in Education Conference (FIE)* (p. F4F–1–F4F–2). https://doi.org/10.1109/FIE.2010.5673191
5. Yang, W., Yang, G., Gao, T., Shen, X., Zhu, Z., & Tan, Z. (2010). Research application of task-driven method in teaching of network protocols. In *2010 2nd International Conference on Education Technology and Computer* (Vol. 1, pp. V1-408-V1-412). https://doi.org/10.1109/ICETC.2010.5529219
6. Zengin, A., & Sarjoughian, H. (2010). DEVS-Suite simulator: A tool teaching network protocols. In *Simulation Conference (WSC), Proceedings of the 2010 Winter* (pp. 2947–2957). https://doi.org/10.1109/WSC.2010.5678989

Biographical Information

Anand Richard is a 23 year industry veteran in the field of embedded systems software design and networking. In 2000 he assisted in the writing of the IEEE 802.1S and 802.1W spanning tree protocols while heading the Layer-2 networking group at Intel. He is currently finishing up a Ph.D. in Technology Management at Indiana State University. He is employed full time at Saint Joseph's College, Rensselaer, Indiana as Assistant Professor of Computer Science. His research interests include networking protocols and effective methods to teach computer languages.