Using Mini Protocol Stacks to Guide Research

Mr. Anand Richard, Indiana State University

I am a recent entrant into the world of academia. I am currently an Assistant Professor of Computer Science at Saint Joseph's College, Rensselaer, Indiana. Prior to this I was in industry for 23 years where my world was Embedded Systems Software applications and networking protocols. My toolbox is filled with C/RISC Assembly/C++/C# and a heady mix of processor expertise like ARM, StrongARM, PPC 603, 603e, 604, 860, 8260 for most of which I once wrote board support packages for RTOSes or device drivers. My career pinnacle so far was working with Dr.Mick Seaman of IEEE 802.1D working group and Dr.Simon Knee to implement rapid (802.1w) and multiple spanning tree (802.1s) protocols for Intel Netstructure Gigabit switches. I am currently in the closing phases of a PhD in Technology Management at Indiana State University where my dissertation is on refinements to the DNP3 (Distributed Network Protocol) using Split protocol techniques. I have written a bare bones DNP3 stack in C# to serve as the test bed for my experiments.

Dr. Patrick Appiah-Kubi

Using ‘mini’ network protocol stacks to facilitate and guide research

Protocol research can be a demanding task due to the steep learning curve associated with the subject. We can characterize protocol research as a quest to improve a protocol in some way. A good example is an attempt to solve specific problems with security or throughput for example. A protocol is implemented predominantly in software although it is common to sequester portions of it in hardware for more speed. A pre-requisite for research into this area is turgid understanding of the protocol. Network simulators like NS2 (Network Simulator 2) and NS3 can be used to study and implement protocols. There are also special purpose simulators that are protocol specific like the DEVS-Suite (Discrete Event Discrete Time Simulator) which is aimed at the OSPF (Open Shortest Path First) protocol (Zengin & Sarjoughian, 2010). Layered protocols are easier to understand and modify in this regard. Yang et al suggest a layered task based method using the TCP/IP layers as an aid to teaching network protocols (Yang et al., 2010). We suggest that implementing a simplified ‘mini’ protocol stack can greatly aid understanding and serve as a test bed for research. In this technique one selects portions of the protocol in each layer and implements them rather than the whole. In this paper we follow this idea and implement a mini DNP3 protocol stack purely with a view to applying a new split design to it and study the effects. A further way to simplify debugging protocol implementation is to select a ‘tunneling’ version of it like MODBUS over TCP/IP or DNP3 over TCP/IP. Since the protocols under study (MODBUS. DNP3) are being carried over TCP packets in these examples; it becomes easier to debug them. Errors in packet formation etc. will not hamper transportation from one device to another.

Figure 1 DNP3 Protocol Layers (IEEE 1815)

DNP3 is predominantly used in the electric utility industry. As shown in Figure 1 it is a 3 layer protocol (User Layer not counted). It carries information about power systems and transmission parameters like voltage levels, power levels etc. Since inception in the 1990s and standardization by the IEEE in 2009 (“IEEE SA - 1815-2012 - IEEE Standard for Electric Power Systems Communications-Distributed Network Protocol (DNP3),” 2016), there have been

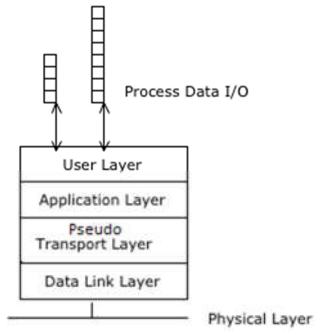

Figure 1

various exploits against commercial DNP3 stacks (East, Butts, Papa, & Shenoi, 2009) .

Table 1 Mini DNP3 stack features

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

Table 1 above shows the features implemented in the mini protocol stack (Richard, Anand, 2016). Only the items in bold were included. The columns serve as a summary of the protocol layers. Application, Pseudo-Transport and Data Link are the 3 main layers in DNP3. We chose to implement the 'Read' and 'Write' functions in the Application layer for example. It becomes immediately clear from this table that the source code for the stack is greatly simplified due to the 'mini' approach. This helps students to understand the protocol and also to make changes from a research perspective. In the case of this paper specifically, the implemented features listed in bold enabled the testing of the Class 0 poll that a Master DNP3 device would send to an Outstation device in order to retrieve all the data configured therein as Class 0. Class 0 data in a DNP3 device is merely 'static' data which cannot be configured for event messages. For example a current or voltage level could be configured as Class 0 data.

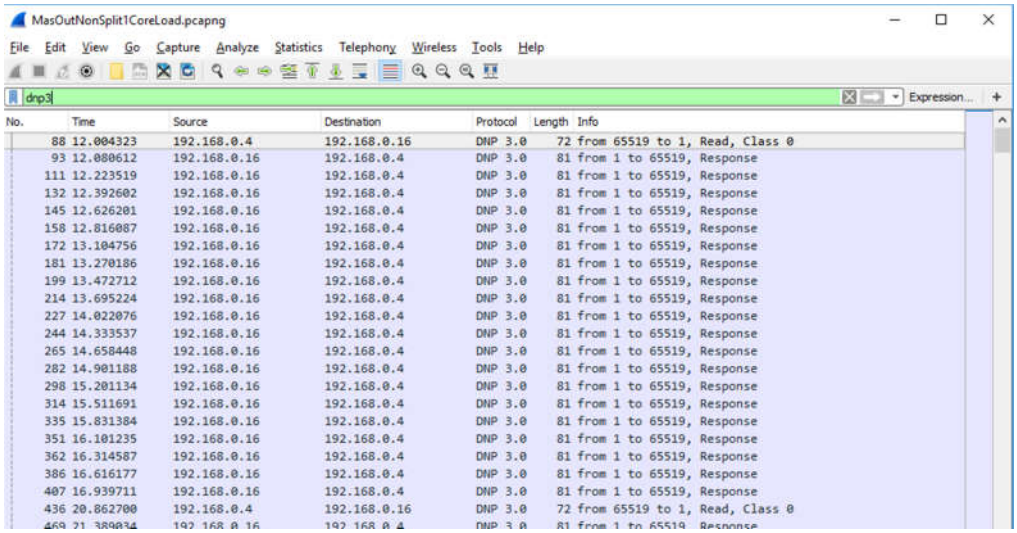

Figure 2 Wire Shark Decode of Class 0 poll and response

Figure 2 shows the Wire Shark decode of a Class 0 poll issued from the Master DNP3 device and the response from the Outstation. Both were designed using the mini DNP3 protocol stack.

A TCP connection can be divided into connection tasks (Connection) and data transfer (Data Transfer) tasks as shown in Figure 3.

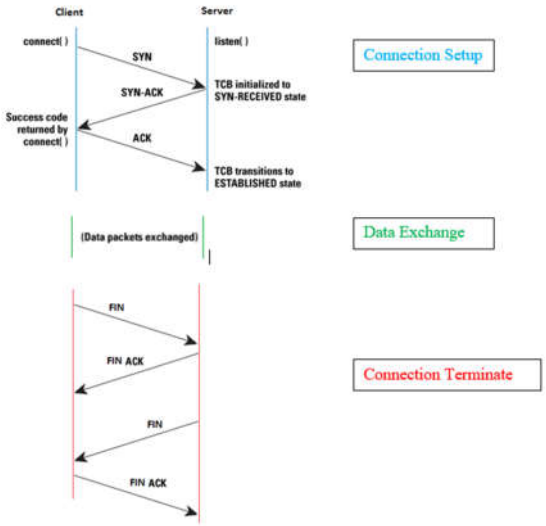

Figure 3 Traditional TCP Design

Connection tasks deal with establishing and tearing down a connection. Data Transfer tasks handle data transmission from the Server to the Client. Since the tasks associated with these categories have clear boundaries we can situated them in multiple machines. Figure 4 shows such a design.

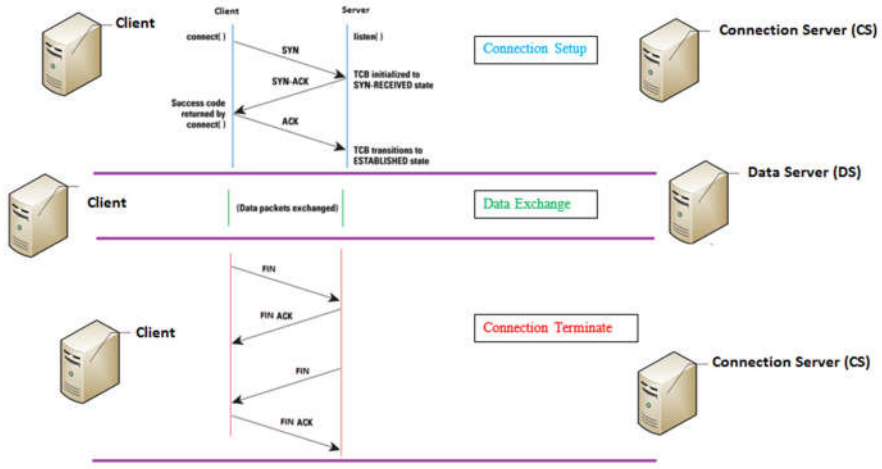

Figure 4 Splitting TCP across machines

In such a design the machine that does the connection work is termed the 'Connection Server' (CS) and the machine handling all data transfer, the 'Data Server' (DS). A machine running the split design TCP/IP stack can request one or more available machines on the network to become its Data Server. On acceptance, the CS handles all the connection work of incoming TCP connections and offloads all the data transfer work of a connection to the DS with information on where to send the data. When the Data Servers send out requested data, they use the IP address of the CS in the source IP field of their IP header. This obscures the fact to the client that a different machine is sending the requested data. This distribution of work from a single machine to two machines immediately yields, redundancy, scaling in data reception time and security benefits. Rawal et al (Rawal, Karne, & Wijesinha, 2012) discuss this approach where they report that adding multiple Data Servers to a single Connection Server shows a significant reduction in client request processing time with the number of Data Servers added. In this paper we discuss the application of a split design to the Distributed Network Protocol 3 (DNP3) over TCP/IP.

We examine if applying the split design to our mini DNP3 over TCP/IP protocol stack could address security in a general way through giving an Outstation under attack a possible mitigation strategy. We posit that DNP3 administrators could identify vulnerable Outstations and implement a split configuration using multiple Outstations. This will allow the victim of an attack to offload its processing to other Outstations by making them its Data Servers. This strategy can also be used to dynamically scale the number of Outstations to distribute Outstation processing load. In this case we presume that the Outstations will share a common DNP3 database.

One of the objectives of this paper was to prove that a 'home grown' mini protocol stack is a sine-qua-non to conducting research into its fundamental design. Making changes in a full protocol stack would be highly complicated, demanding and require a large team in which everyone is a DNP3 subject matter expert. The team that worked on this paper consisted of only two members. The lead author was part of the team that worked on the original split protocol design of TCP (Rawal et al). The co-author undertook the work of reading and comprehending the IEEE 1815 DNP3 specification and implementing the mini protocol stack itself. The split

design was devised by the lead author and implemented by the co-author. The development work of writing and testing the code took a semester to accomplish. It must be stressed however that the Instructor should be a DNP3 subject matter expert to some degree. This is because understanding the IEEE 1815 specification requires a significant investment of time but once accomplished, the knowledge can be used by the Instructor to shorten the learning curve for students considerably. Students stand to benefit enormously by having the source code and documentation for such a protocol stack. It serves a dual purpose of being an instructional aid as well as a research test bed on which future investigations can be carried out.

In the 'mini DNP3 over TCP/IP protocol stack'(Richard, Anand, 2016) we applied the split at the TCP level. The DNP3 protocol per se was not changed. The stack was written in C# on Windows following the IEEE 1815 standard. The split design was implemented using the SharpPcap library ("Tamir Gal | SharpPcap," n.d.). SharpPcap enables users to build and transmit TCP packets where packet headers can be modified. Rawal et al changed the source code of a Linux TCP/IP stack to incorporate the split design. Here we used an extra library to build, send and intercept packets using SharpPcap with an existing traditional Windows TCP/IP connection already running to initiate the first connection between the client and server. This approach while not ideal provides a faster way to implement the split design and also to test it. We built a Master and Outstation device. The Outstation was configured with 20 Class 0 Data Counters. We then measured the time taken for the Master to complete a Class 0 Data Poll from the Outstation. The poll from the Master to the Outstation retrieves all Class 0 Data from the Outstation. We next enabled the split configuration in the Outstation to use 1, 2, 3 and 4 Data Servers and measured the time again.

We conducted 40 trials for each configuration of Non-Split, 1DS, 2DS, 3DS and 4DS. All packet data was captured using Wireshark. The mean for each sample was then taken to build the final bar graph in Figure 5.

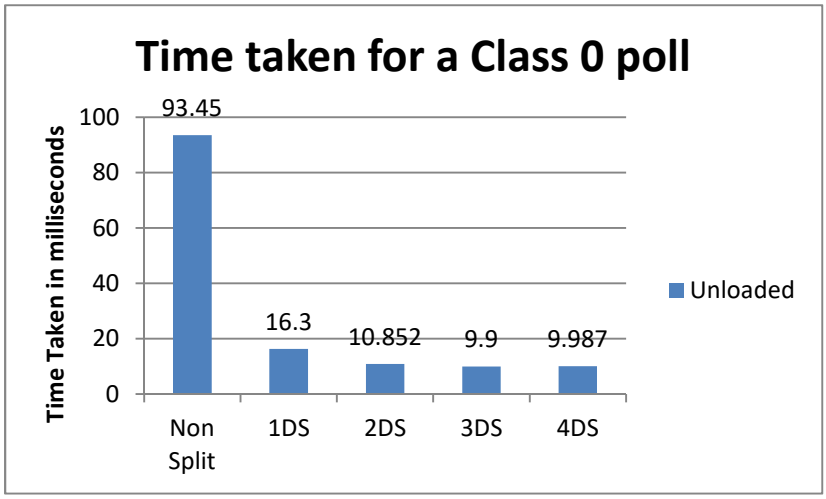

Figure 5 Class 0 poll in non split mode

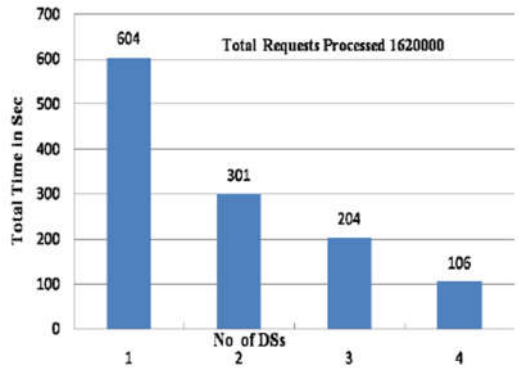

Figure 6 Results from Rawal et al ("IEEE Xplore Abstract Record," n.d., p. 4)

Our results in Figure-5 show an 82.5% decrease in time taken for the Class 0 Poll from a non-split approach to a single DS. There is a further 34% decrease with 2 DSes. Diminishing returns are noticed with a 3rd DS and 4th DS. Adding a 4th DS actually slightly increases the response time. In this regard we were unable to duplicate the results of Rawal et al shown in Figure 6.

We suggest this is due to the small data size of Class 0 20 counters used in the study. Dividing 20 counters across 2 machines gives us faster response times because each DS handles the transmission of 10 DNP3 counter messages which is a significant load taken away from the CS. When we move to 3 and 4 DSes the counter data handled per DS reduces to 7 and 5. The difference in work handled by each DS becomes marginal and hence we see marginal improvement on adding more DSes.

Using 4 DSes in fact very slightly increases the response time as can be seen in Figure 3 from 9 ms to 9.87 ms approximately. It is possible this increase is our application response time to handle the CS redirect of the client request to the 4th DS which gets added on to the peak performance time of 9 ms.

Based on our data we conclude the following:

1. A mini DNP3 protocol stack can be built with a reduced feature set. The packets generated by this stack can be read and deciphered by Wireshark proving independent 3rd party confirmation of our design.
2. A split design can be applied to the DNP3 stack indirectly using the SharpPcap library and shown to work. This is facilitated by the reduced complexity of the 'mini' stack.
3. The split design tests reveal a significant reduction in the time taken for a Class 0 poll.
4. Based on '3' above we can state that in the event of a DNP3 Outstation coming under attack, it could mitigate the attack via use of the split design to offload processing work to a Data Server DNP3 device.

We suggest further research in the following areas:

1. Examination of the tests here under a larger data size, i.e. 100 counters instead of 20 to determine if benefits continue as suggested by Rawal et al across 4 DSes.

2. Investigate if increasing the packet size without crossing the threshold of DNP3 message fragmentation has an impact on the results of adding more DSes.
3. Our mini protocol stack source is available on Git Hub https://github.com/kiranand/DNP3 and available on request.

Bibliography

East, S., Butts, J., Papa, M., & Shenoi, S. (2009). A Taxonomy of Attacks on the DNP3 Protocol. In C. Palmer & S. Shenoi (Eds.), *Critical Infrastructure Protection III* (Vol. 311, pp. 67–81). Berlin, Heidelberg: Springer Berlin Heidelberg. Retrieved from http://link.springer.com/10.1007/978-3-642-04798-5_5

IEEE SA - 1815-2012 - IEEE Standard for Electric Power Systems Communications-Distributed Network Protocol (DNP3). (2016, September 25). Retrieved September 25, 2016, from https://standards.ieee.org/findstds/standard/1815-2012.html

IEEE Xplore Abstract Record. (n.d.). Retrieved from http://ieeexplore.ieee.org/document/6249320/

Rawal, B. S., Karne, R. K., & Wijesinha, A. L. (2012). Split protocol client/server architecture. In *2012 IEEE Symposium on Computers and Communications (ISCC)* (pp. 000348–000353). https://doi.org/10.1109/ISCC.2012.6249320

Richard, Anand. (2016, March 4). *Writing Mini Protocol Stacks as an aid to teaching Networking protocols*. American Society of Engineering Educators Zone 2 Conference, San Juan, Puerto Rico.

Tamir Gal | SharpPcap. (n.d.). Retrieved January 27, 2017, from http://www.tamirgal.com/blog/page/sharppcap.aspx

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

Zengin, A., & Sarjoughian, H. (2010). DEVS-Suite simulator: A tool teaching network protocols. In *Proceedings of the 2010 Winter Simulation Conference* (pp. 2947–2957). https://doi.org/10.1109/WSC.2010.5678989