Ns-Modbus:
Integration of Modbus with
ns-3 Network Simulator

by

Mohammad Reza Sahraei

B.Eng., Islamic Azad University, 1992

Research Project Submitted in Partial Fulfillment of the
Requirements for the Degree of
Master of Engineering

in the
School of Engineering Science
Faculty of Applied Sciences

SIMON FRASER UNIVERSITY

Summer 2013

Approval

Name:	**Mohammad Reza Sahraei**
Degree:	**Master of Engineering**
Title of Thesis:	***Ns-Modbus: Integration of Modbus with ns-3 Network Simulator***
Examining Committee:	**Chair:** Dr. Jie Liang Graduate Chair

Dr. Ljiljana Trajković
Senior Supervisor
Professor

Terrance John Hermary, P.Eng.
Supervisor
Hermary Opto Electronics Inc.
Coquitlam, British Columbia

Date Defended/Approved: May 8, 2013

Partial Copyright Licence

Abstract

Modbus, a de facto industry standard communication protocol, was first introduced by Modicon for serial communication networks. Modbus TCP emerged when Ethernet gained popularity in the industry. Modbus is simple, robust, and widely used in industrial applications. It has been simulated and emulated in a number of stand-alone applications. The goal of this project is to integrate Modbus TCP and UDP with the ns-3.13 network simulator.

Keywords: Communication technology; communication networks; protocols; industrial protocol; Modbus; ns-3.

To my wife Valla and my two daughters
Laleh and Isla, for their love and support.
Also, to my mom and dad,
for their invaluable guidance.

Acknowledgements

I would like to thank my senior supervisor Dr. Ljiljana Trajković for her valuable support and guidance. I would also like to thank my supervisor Mr. Terrance John Hermary for his invaluable supervision.

Table of Contents

List of Figures

List of Tables

List of Acronyms

ADU	Application Data Unit
ARP	Address Resolution Protocol
CIP	Common Industrial Protocol
EIA	Electronic Industries Association
HDLC	High-Level Data Link Control
HMI	Human Machine Interface
I/O	Input/Output
IETF	Internet Engineering Task Force
IP	Internet Protocol
ISO	International Organization for Standardization
MAC	Medium Access Control
MB	Modbus Protocol
MB+	Modbus Plus
MBAP	Modbus Application Protocol
MBP	Modbus Plus
ns	Network Simulator
ns-2	Network Simulator version 2
ns-3	Network Simulator version 3
ODVA	Open DeviceNet Vendors Association
PDU	Protocol Data Unit
PLC	Programmable Logic Controller
TCP	Transmission Control Protocol
TIA	Telecommunications Industry Association
UDP	User Datagram Protocol

Glossary

Common Industrial Protocol (CIP)	A comprehensive media-independent industrial protocol supported by hundreds of vendors around the World. It is used in CompoNet, ControlNet, DeviceNet, and EtherNet/IP.
CompoNet	A bit-level network for industrial automation that supports CIP.
ControlNet/fieldbus	A highly deterministic protocol due to a unique physical layer. It supports CIP.
DeviceNet	A network system for industrial automation developed by Allen-Bradley (now Rockwell Automation) that operates on Controller Area Network (CAN). This protocol supports CIP.
EtherNet/IP	Ethernet Industrial Protocol is a wrapper protocol to support CIP over Ethernet.
Open DeviceNet Vendors Association (ODVA)	A multi-vendor association that supports network technologies built on Common Industrial Protocol.

1. Introduction

Industrial plants contain and rely on a plethora of mechanical devices. These devices were originally designed to work together without any distant control capabilities. They were locally monitored and adjusted by operators. Gradually, electronic circuits and Programmable Logic Controllers (PLC) were introduced to enable operators to remotely monitor and control mechanical devices using point-to-point communication lines. Subsequently, communication protocols were defined. Each vendor developed its own requirements and, hence, a variety of standards and protocols have been introduced for industrial communications. Examples of such industrial protocols are Common Industrial Protocol (CIP), Ethernet Industrial Protocol (EtherNet/IP), DeviceNet, ControlNet, CompoNet, and Modbus. CIP was introduced by Rockwell Automation and is supported by the Open DeviceNet Vendors Association (ODVA). EtherNet/IP is an application layer protocol that relies on TCP/IP and is used to handle CIP on the Ethernet physical layer network infrastructure. Modbus protocol was introduced by Modicon (currently owned by Schneider Electric) in 1979. It was first used in serial communication networks. Modbus TCP emerged with introduction of Ethernet to industrial sites. Modbus is simple, robust, and widely used in the industry.

Modbus is an application layer protocol that may operate on EIA/TIA-232, EIA/TIA-485, or Ethernet standards. Modbus that uses serial communication networks is called Modbus serial. Modbus that uses Ethernet is presented as two flavours: Modbus TCP and Modbus UDP. Modbus has inherited handshake and timeouts from the serial communication protocols. Hence, it enables the implementation of the protocol on the UDP layer that advantageously does not have the overhead associated with the TCP layer. Modbus TCP and Modbus UDP utilize port 502 of the TCP/IP protocol.

Modbus is popular because it is simple and many users are familiar with the protocol. It is also an open standard supported by the Modbus-IDA association.

1.1. Project Scope

The scope of this project consists of three phases: understanding the Modbus protocol, gaining experience with the latest version of the ns-3 network simulator, and implementing and incorporating the Modbus TCP in ns-3. These three phases of the project were implemented sequentially. However, some reiteration and overlapping in the timelines of the project was inevitable. Before the implementation phase, it was decided to also incorporate Modbus UDP for three reasons:

- Modbus as an application layer protocol may work with both TCP and UDP, though TCP is more popular.
- To compare the overhead of TCP and UDP implementations.
- To get familiar with the network simulator because UDP implementation is simpler.

1.2. Organization of the Research Project

The organization of this research project is as follows: Modbus protocol, ns-3 network simulator, and work related to the Modbus implementation are presented in Chapter 2. The design and the implementation of ns-Modbus is described in Chapter 3. Validation and verification of the implementation are presented in Chapter 4. The scalability of ns-Modbus is addressed in Chapter 5. Possible future work is described in Chapter 6 followed by conclusion in Chapter 7. The generated message logs by the simulator scenarios are included in Appendix A. The simulation environment is described in Appendix D. The sample simulator scenarios written in C++ are presented in Appendix E. Ns-Modbus source code is presented in Appendix F. The techniques used in debugging ns-Modbus are given in Appendix G. The instructions to install ns-Modbus are shown in Appendix H. The instructions to run the provided scenarios are described in Appendix I. The Modbus exception codes are included in Appendix J. Finally, the “round-trip time” over the “total time” and the “round-trip time” over the “simulation time” ratios generated by each simulator scenario for the various number of nodes are presented in Appendix K.

2. Background Knowledge

In this Section, the Modbus protocol, the ns-3 network simulator, and work related to the Modbus implementation are introduced.

2.1. Modbus Protocol

Modbus is an application layer protocol that provides a client/server paradigm between the network devices [1]. The Modbus server waits for the arrival of the Modbus client request. After processing the request, it produces the proper response. If the request is invalid, the Modbus server generates an exception with an appropriate code. A simplified state diagram of the Modbus server is shown in Figure 1. Modbus client expects to receive a valid response to the request unless an exception or a timeout occurs:

- If no request is received by the server due to a communication failure or if the request has checksum error, then no response is generated and, subsequently, the client times out.
- If the received request is invalid, the server generates and responds with an exception with an appropriate code. The exception code indicates whether the request carried an illegal function (code 01), illegal data address (code 02), or illegal data value (code 03). An exception (code 05) is sent back as an acknowledgment when the request is accepted and relatively large period of time is required to process the request. Moreover, the exceptions show if there is a file consistency check failure (code 08) or a gateway issue encountered (0A and 0B). Additional details of the Modbus exceptions are listed in Appendix J.
- Otherwise, the Modbus server responds normally.

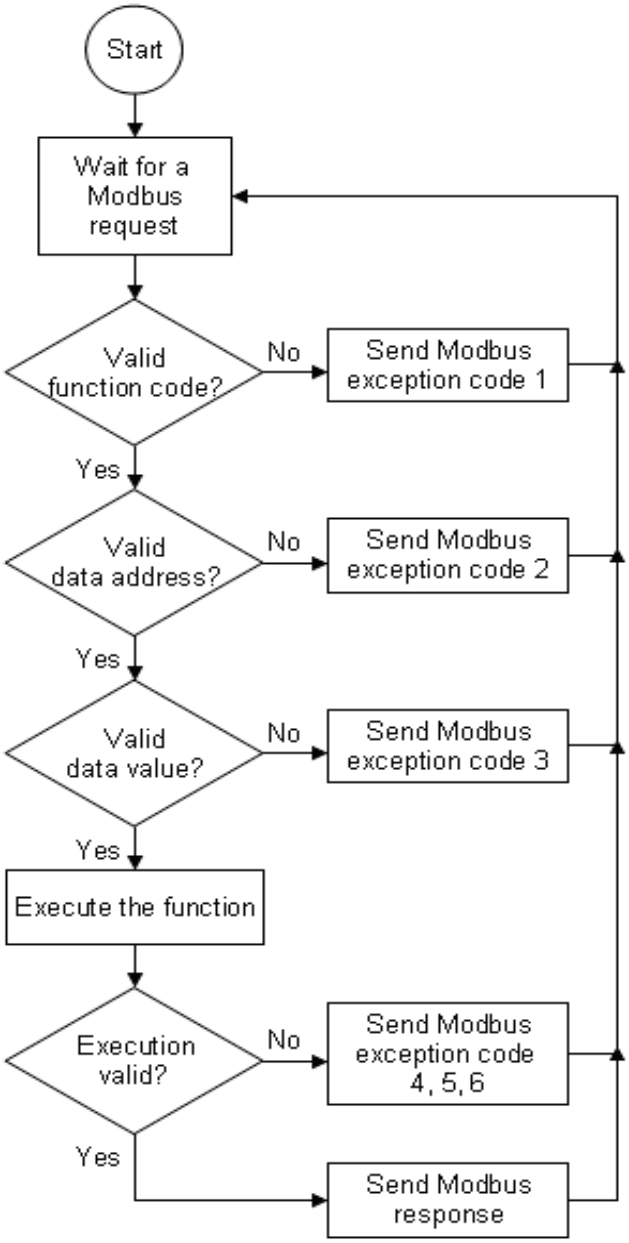

Figure 1. Modbus transaction state diagram [1].

Modbus protocol frames consist of a Protocol Data Unit (PDU) common to Modbus serial, Modbus TCP, and Modbus UDP. The PDU defines the data protocol independently from the underlying physical layer. Modbus requires additional overhead to handle the underlying network [2]. The combination of the PDU and the extra information forms Application Data Unit (ADU). The maximum size of the PDU is inherited from the serial communication. The maximum ADU size is 256 bytes for

RS485. The server address in this serial network protocol requires one byte and the checksum requires two bytes. Therefore, the maximum PDU size is 253 bytes. This restriction also limits the size of the PDU for Modbus TCP and Modbus UDP to 253 bytes. The maximum ADU size of Modbus TCP and Modbus UDP is not limited to 256 bytes because these protocols follow a different addressing and error checking scheme. The Modbus frame is shown in Figure 2.

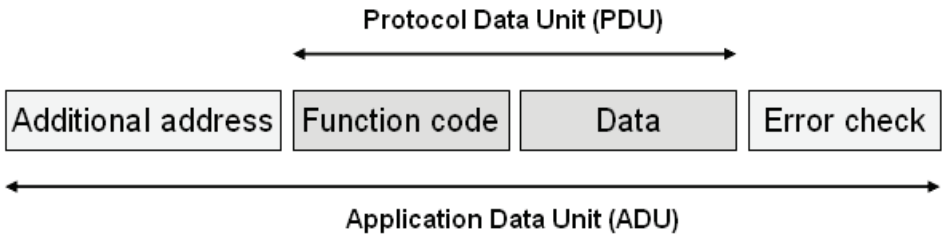

Figure 2. Modbus frame [5]. The ADU includes addressing, error checking, and the PDU. The maximum size of the PDU is 253 bytes.

2.1.1. Modbus PDUs

The Modbus protocol has three types of PDUs: Modbus request, Modbus response, and Modbus exception response.

- The Modbus request normally includes two sections: function code and request data. The function code section is one byte while the request data section is usually multiple bytes long. The values in the range from 1 to 127 are valid for the function codes while the values from 128 to 255 are reserved for the exception codes. The request data section depends on the function code and may contain other information.
- The Modbus response also consists of two sections: function code and response data. The Modbus request function code is echoed back in the response. The data section has variable length and depends on the function code. It may also contain other information.
- The Modbus exception response consists of exception function code and exception code. The exception function code is one byte long. It is an echo of the requested function code added to a constant offset with the value of 128 (0x80 hexadecimal) in order to map the function code to the reserved range for the exceptions. The exception code provides more information about the exception. For example, if the client sends a request with the function code set to 0x01, which checks whether a physical coil (or coils) is turned on or off, and if an error occurs then the Modbus server responds with the exception function code 0x81. The exception function code 0x81 is a summation of the requested

function code (0x01) and the exception offset (0x80). The exception code of the response may be 0x1, 0x2, 0x3, or 0x4 depending on the nature of the error. In this example, the exception code 0x2 is sent when the requested "start address" added to the "quantity of coils" is greater than 65,535. The exception code 0x3 is sent when the requested "quantity of coils" is not in the range of 1 to 2,000 (0x7D0 hexadecimal). The exception code 0x4 is sent when the Modbus server could not perform the process of reading the coils.

2.1.2. Modbus Data Model

Modbus recognizes input and output and also bit addressable and word addressable data items [1]. The Modbus data on the server are stored in series of tables in the memory that may be separated or overlapped. The four primary tables are listed in Table 1.

Table 1. Modbus Primary Tables.

Primary tables	Object type	Access type	Comments
Discrete input	bit	Read only	Provided by I/O system. Input only.
Coils	bit	Read/write	The one bit value may be read or changed.
Input registers	16-bit word	Read only	Provided by I/O system. Input only.
Holding registers	16-bit word	Read/write	The 16-bit value may be read or changed.

Modbus maps *Discrete Input*, *Coils*, *Input Registers*, and *Holding Registers* to assigned tables in the Modbus server memory. These tables may be mapped to separate blocks of memory or they may be overlapped. The allocation of the memory depends on the Modbus server and is vendor specific. Modbus allows 65,536 data items for each primary table. The read and write operations may be performed for multiple consecutive data items. The number of consecutive items to be processed is limited by the PDU and the function code. In practice, the data items are mapped to a physical address in the Modbus server. However, the addressing to the Modbus logical reference number is a zero-based 16-bit word addressing scheme.

2.1.3. Modbus Function Code

The Modbus function code is one byte long that indicates the action the Modbus server should take. Modbus uses values in a range from 1 to 127 for the function codes. Some

complex function codes use sub-codes to provide more options and functionalities. Modbus has three categories for the function codes: Public, User-defined, and Reserved.

- The public function codes are employed for general purpose usage and are well-documented. There are three ranges assigned to the public function codes. These ranges are 1-64, 73-99, and 111-127. Sections of these ranges are defined by the Modbus community (Modbus-IDA.org) and have available conformance tests. The rest of them are unallocated and set aside for future enhancement. The public functions codes are listed in Table 2.
- The user-defined function codes are vendor-specific and are not supported in the specification. There are two ranges allocated for this group of function codes: from 65 to 72 and from 100 to 110. The Modbus vendors may employ any valid user-defined function codes and assign them to their specific required functionalities, which subsequently needed to be published by the vendors. These functionalities are not unique because other vendors may assign different functionality to the same user-defined function code.
- The reserved function codes are used by some Modbus vendors for legacy applications and are not available to the public function codes. The function codes 9, 10, 13, 14, 41, 42, 90, 91, 125, 126, and 127 are reserved. Also, the sub-codes 19 and 21-65,535 for the function code 8 and the sub-codes 0-12 and 15-255 for the function code 43 are reserved.

2.2. Ns-3 Network Simulator

The ns network simulator is a discrete event simulator. It is popular in academia for its open source model, its extensibility, and plentiful online documentation. The ns simulator is often used in simulations of routing and multicast protocols and in simulation of ad-hoc networks. It supports popular network protocols for wired and wireless networks. It may be also used as a limited-functionality network emulator.

An early version of the ns simulator was developed in 1989 as a variant of the REAL network simulator. By 1995, ns had gained support from Defense Advanced Research Projects Agency (DARPA), the Virtual Inter Network Test-bed (VINT) project at Lawrence Berkeley National Laboratory (LBNL), Xerox Palo Alto Research Center (PARC), University of California, Berkeley (UCB), and University of Southern California Information Sciences Institute (USC/ISI).

The ns-2 version of ns has an established reputation in the research community and is used for educational purposes [3]. The ns-3 is the latest version with an entirely new design that is rapidly gaining popularity and is gradually replacing the well-liked ns-2. Ns-3 is a free software, licensed under GNU GPLv2 license [4], which allows copying, distribution, and modification of the application. Ns-3 is funded by institutes such as the University of Washington, Georgia Institute of Technology and the ICSI Center for Internet Research with collaborative support from the Planète research group at INRIA Sophia-Antipolis. Ns-3 is written in C++. Python may be used for developing, configuring, and testing various network simulation scenarios. The latest version of ns-3 (3.16) was released in December 2012. I implemented ns-Modbus as an extension of ns-3.13.

2.2.1. Ns-2 versus ns-3

In this Section, a brief comparison between ns-2 and ns-3 is presented. Ns-2 is written in C++ and OTcl (the object oriented Tcl). However, ns-3 is entirely implemented in C++. The scripts in ns-2 have to be implemented using OTcl while the simulation scripts in ns-3 may be written in C++ or Python. The result of ns-2 simulation may be viewed using the Network Animators (nam) while the result of ns-3 simulation may be visualized using NetAnim, which is still under development. Ns-3 is a newer version and has a more solid design and performance. However, not all models available in ns-2 have been implemented in ns-3. There are still activities and support for ns-2. Nevertheless, ns-3 compared to ns-2 gets considerably more support and development [5].

2.2.2. Ns-3 Ecosystem

Several useful tools play important roles in ns-3. These tools help manage the source code, compile the ns core and the scripts, debug, and view the outputs.

Mercurial

Mercurial is an open source, easy to learn, distributed, and robust source code control, available on several platforms [6]. Ns-3 uses Mercurial to manage its underlying code and documentation. Ns-3 code may be downloaded using the following Mercurial command:

hg clone http://code.nsnam.org/ns-3-dev

The details of how to install ns-3 are provided in Appendix H.

Waf

After downloading, the source code usually needs to be compiled into an executable format if such format is not already provided. Usually, the *make* tool is used under Linux operating system. Because of the project complexity, ns-3 uses Waf [7] for compilation and configuration [5]. Waf is a Python-based build system and relies on *wscript* files, which are integrated in the ns-3 class hierarchy. The classes to be included in the build are listed in the *wscript* files. Ns-3 uses a similar command to run a script:

*./waf --run "*scratch/my*First"*

The details of how to run the network scenarios are provided in Appendix I.

NetAnim

NetAnim is a graphical tool used to visualize the network activities during the simulation. Each simulation scenario needs to capture and save the activities in an XML file using *AnimationInterface anim("animation.xml")* and *anim.SetXMLOutput()* instructions and subsequently, the XML file may be loaded to NetAnim for animation.

The necessary instructions to utilize NetAnim are highlighted in Figure 3 and Figure 4. The instructions in this example are borrowed from *myfirstModbusUdp* network scenario.

```
18   * Author: Reza Sahraei <mrs16@sfu.ca>
19   */
20
21  #include "ns3/core-module.h"
22  #include "ns3/network-module.h"
23  #include "ns3/netanim-module.h"
24  #include "ns3/internet-module.h"
25  #include "ns3/point-to-point-module.h"
26  #include "ns3/applications-module.h"
27  #include "ns3/modbus-mgr.h"
28
29  // Network Topology
30  //
31  //     n0    n1
32  //     |     |
33  //     ======
34  //   LAN 10.1.1.0
```

Figure 3. The required NetAnim header file to be included in the network scenario.

```
117    clientApps.Start (Seconds (2.0));
118    clientApps.Stop (Seconds (10.0));
119
120    //configure tracing
121    AsciiTraceHelper ascii;
122    pointToPoint.EnableAsciiAll (ascii.CreateFileStream ("first-modbus-udp.tr"));
123    pointToPoint.EnablePcapAll ("first-modbus-udp");
124
125    // Create the animation object and configure for specified output
126    AnimationInterface anim (animFile);
127    anim.SetXMLOutput ();
128
129    Simulator::Run ();
130    Simulator::Destroy ();
131    return 0;
132  }
```

Figure 4. The instructions that generate NetAnim XML animation file.

A sample NetAnim screen shot showing the network traffic of *myfirstModbusUdp* scenario is illustrated in Figure 5.

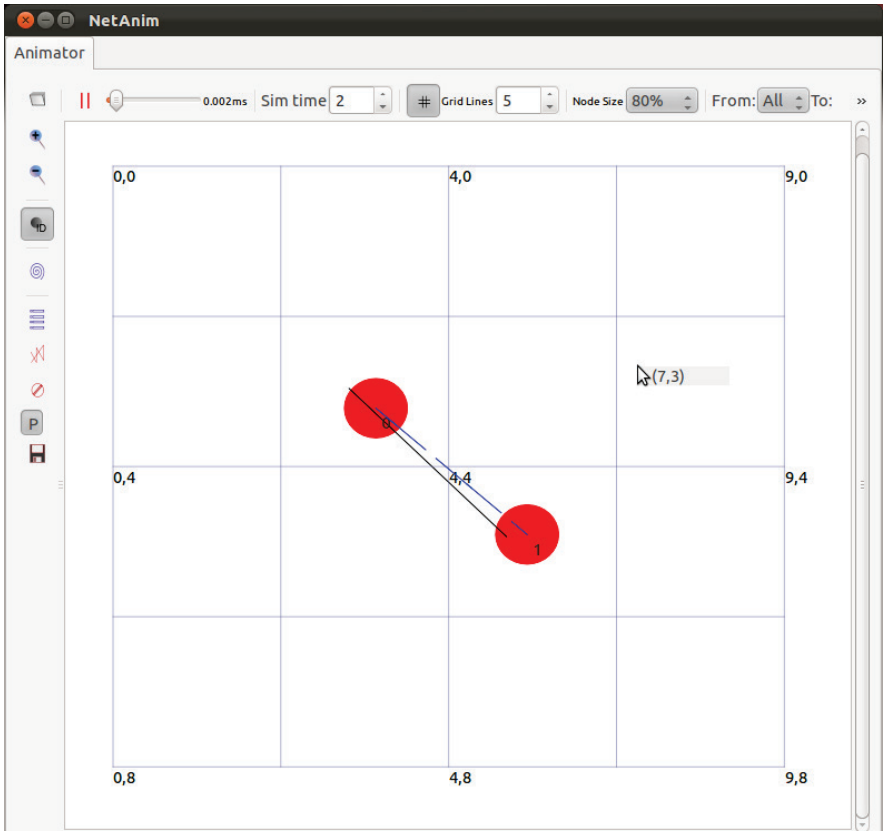

Figure 5. ***The NetAnim screen shot shows the two nodes on the network from the scenario myfirstModbusUdp. Node 0 is the Modbus server and node 1 is the Modbus client. The dotted line indicates the flow of network traffic between the two nodes.***

2.2.3. Ns-3 Reference Documents

Additional information about ns-3 is available from the documentation:

- Ns-3 Tutorial: Primary documentation [5].
- Ns-3 Manual [11].
- Ns-3 Model Library: ns-3 models and supporting software [12].
- Ns-3 Doxygen: The public APIs documentations [13].
- Coding Style: A set of coding standard for acceptable code layout [14].
- Ns-3 wiki [15].

3. Ns-Modbus Design and Implementation

The ns-Modbus classes are incorporated in the ns-3 class hierarchy. Ns-3 employs Waf, a software build tool written in the Python programming language to build and run the network simulation scenarios. The classes to be included in the build are listed in the *wscript* files. Ns-Modbus integrates classes under the ns-3 the sub-directories of scratch, helper, and model.

3.1. Simulation Scenario Classes

Various simulating scenarios may be implemented using C++ or Python. A user may employ the scratch sub-directory to execute simulation scenarios. The ns-Modbus implementation was verified using bus and star topologies. The bus topology is very common in the industry while the star topology is added for completeness. The ns-Modbus implementation is first verified using a simple scenario consisting of one client and one server based on a bus topology. The *myfirstModbusUdp* and *myfirstModbusTcp* network simulation scenarios are used to prove the concepts of Modbus UDP and Modbus TCP, respectively. The *busModbusUdp* and *busModbusTcp* scenarios improved the implementation of *myfirstModbusUdp* and *myfirstModbusTcp* by accepting variable number of the Modbus client nodes while there is still one Modbus server responding to the clients. The netmask of the bus topology is set to 255.255.255.0 and may accept up to 252 client nodes. The *starModbusUdp* and *starModbusTcp* scenarios implement star topology for Modbus UDP and Modbus TCP, respectively. These network simulation scenarios also accept variable number of the client nodes. These client nodes connect to a single Modbus server. The maximum number of clients for the tested star topology is 3,500 nodes. This number may potentially be higher.

All network simulation scenarios generate trace files, pcap files used in the Wireshark packet analyzer, and animation files used in the NetAnim packet animator [3]. These scenarios may generate detail log messages when the verbose flag is enabled.

Elements of the scenario setup in a sample file are shown in Figure 6, Figure 7, and Figure 8. Lines 50 to 54 in Figure 6 show the predefined Modbus requests that are used in various simulation scenarios.

```
49  // MODBUS requests
50  uint8_t ENCAP_INTERFACE_TRANS_CANOPEN_GENERAL_REF_REQ_RSP_PDU[] = {0x2B, 0x0D};
51  uint8_t ENCAP_INTERFACE_TRANS_READ_DEVICE_ID__BASIC_DEVICE_ID_OBJ_ID_00[] = {0x2
52  uint8_t ENCAP_INTERFACE_TRANS_READ_DEVICE_ID__REGULAR_DEVICE_ID_OBJ_ID_00[] = {0
53  uint8_t ENCAP_INTERFACE_TRANS_READ_DEVICE_ID__EXTENDED_DEVICE_ID_OBJ_ID_00[] = {
54  uint8_t ENCAP_INTERFACE_TRANS_READ_DEVICE_ID__SPECIFIC_DEVICE_ID_OBJ_ID_00[] = {
55
56
57  using namespace ns3;
58
59  NS_LOG_COMPONENT_DEFINE ("BusModbusTcp");
60
61  int
62  main (int argc, char *argv[])
```

Predefined Modbus Requests

Figure 6. A sample scenario with predefined Modbus requests.

Line 59 in Figure 6 uses *NS_LOG_COMPONENT_DEFINE* ns-3 macro to identify a name for the simulation scenario. Later, the name may be used by the network simulator to turn on or turn off the logging messages of the scenario.

Lines 98 to 100 in Figure 7 indicate the data rate and the delay of the channel used in the simulation scenario. Line 112 in Figure 7 sets the subnet address and the netmask used for the Modbus clients and the Modbus server in the scenario. Lines 122 and 123 set the time when the Modbus server application starts and stops running. Lines 94, 105, 110, 116 in Figure 7 call *NS_LOG_INFO* ns-3 macro to log messages showing each stage of the simulation.

```
94   NS_LOG_INFO ("Build bus topology.");
95   NodeContainer csmaNodes;
96   csmaNodes.Create (nCsma);
97
98   CsmaHelper csma;
99   csma.SetChannelAttribute ("DataRate", StringValue ("100Mbps"));
100  csma.SetChannelAttribute ("Delay", TimeValue (NanoSeconds (6560)));
101
102  NetDeviceContainer csmaDevices;
103  csmaDevices = csma.Install (csmaNodes);
104
105  NS_LOG_INFO ("Install internet stack on all nodes.");
106  InternetStackHelper internet;
107  internet.Install (csmaNodes);
108
109
110  NS_LOG_INFO ("Assign IP Addresses.");
111  Ipv4AddressHelper address;
112  address.SetBase ("10.1.1.0", "255.255.255.0");
113  Ipv4InterfaceContainer csmaInterfaces;
114  csmaInterfaces = address.Assign (csmaDevices);
115
116  NS_LOG_INFO ("Create applications.");
117  //
118  // Create a MODBUS TCP server on the star "hub"
119  //
120  ModbusTcpServerHelper cModbusTcpServerHelper;
121  ApplicationContainer serverApps = cModbusTcpServerHelper.Install (csmaNodes.Ge
122  serverApps.Start (Seconds (1.0));
123  serverApps.Stop (Seconds (10.0));
```

Server Node's Start/Stop Time

Figure 7. A sample scenario setup for channel data rate, channel delay, IP address, netmask, and the Modbus server start/stop time.

A sample scenario setting the Modbus requests, the request repetition, and the Modbus client start and stop time are shown in Figure 8. Each Modbus client starts two seconds after the beginning of the simulation and runs for eight seconds (lines 148 and 149). Each Modbus client sends the first request 2.01 seconds after the beginning of the simulation (lines 137 and 138) then 200 milliseconds later sends the second request (lines 140 and 141) and finally sends the third request 100 milliseconds afterwards (lines 143 and 144). This sequence repeats one more time, since the request repetition is set to 2 (line 146).

```
125   //
126   // Create MODBUS TCP client applications send UDP to the hub
127   //
128   for (uint32_t i = 1; i < nCsma; ++i)
129     {
130       ModbusTcpClientHelper cModbusTcpClientHelper (csmaInterfaces.GetAddress (0
131       cModbusTcpClientHelper.SetAttribute ("MaxPackets", UintegerValue (1));
132       cModbusTcpClientHelper.SetAttribute ("Interval", TimeValue (Seconds (1.0))
133
134       ApplicationContainer clientApps = cModbusTcpClientHelper.Install (csmaNode
135
136       // set MODBUS requests                    Setting a Request with a Start Delay
137       cModbusTcpClientHelper.SetRequest (clientApps.Get(0), Seconds (0.01),
138         ENCAP_INTERFACE_TRANS_CANOPEN_GENERAL_REF_REQ_RSP_PDU);
139
140       cModbusTcpClientHelper.SetRequest (clientApps.Get(0), Seconds (0.2),
141         ENCAP_INTERFACE_TRANS_READ_DEVICE_ID__BASIC_DEVICE_ID_OBJ_ID_00);
142
143       cModbusTcpClientHelper.SetRequest (clientApps.Get(0), Seconds (0.1),
144         ENCAP_INTERFACE_TRANS_READ_DEVICE_ID__SPECIFIC_DEVICE_ID_OBJ_ID_00);
145
146       cModbusTcpClientHelper.SetCount (clientApps.Get(0), 2);   Request Repetition
147
148       clientApps.Start (Seconds (2.0));
149       clientApps.Stop (Seconds (10.0));   Client Node's Start/Stop Time
150     }
```

Figure 8. ***A sample scenario setting the Modbus requests, the request repetition, and the Modbus client start/stop time.***

3.2. Topology Helper Classes

Common tasks in ns-3 such as assigning IP addresses, defining nodes, arranging connections between nodes and channels, and setting application running on each node define each simulation scenario [8]. I implemented helper classes to assist the user to quickly define Modbus UDP and Modbus TCP in a simulation scenario. The *Modbus-udp-helper* and *Modbus-tcp-helper* classes are added under the helper subdirectory in the ns-3 directory hierarchy.

3.3. Model Classes

The implementation of the Modbus protocol is incorporated into the ns-3 *model* subdirectory. The *Modbus-mgr* class is a core component that implements the Modbus application protocol specification. The *Modbus-pdu* class provides serialization and de-

serialization of the packets between the Modbus client and the Modbus server. The *Modbus-udp-client* and *Modbus-udp-server* classes implement Modbus UDP for the client and the server side, respectively. Whereas, the *Modbus-tcp-client* and *Modbus-tcp-server* classes implement Modbus TCP for the client and the server side, respectively.

3.4. Design Notes

In this Section, the design notes considered for the implementation of ns-Modbus are described.

3.4.1. Ns-Modbus Data Model

The ns-Modbus data model follows a separate addressing for all four primary tables of discrete input, coils, input registers, and holding registers (see Section 2.1.2 Modbus Data Model). In the code, there are four arrays of *m_inputsStatus*, *m_coilsStatus*, *m_inputRegisters*, and *m_holdingRegisters* to keep these primary tables. The *m_inputsStatus* and *m_coilsStatus* are byte arrays, although the requirement is bit-manipulation and there is opportunity to optimize the code. Both *m_inputRegisters* and *m_holdingRegisters* are 16-bit arrays. The size of all four arrays is 65,536. The implementation of the ns-Modbus data model in *modbus-mgr.h* and *modbus-mgr.cc* are illustrated in Figure 9 and Figure 10, respectively.

```
18   * Author: Reza Sahraei <mrs16@sfu.ca>
19   */
20  #ifndef MODBUS_MGR_H
21  #define MODBUS_MGR_H
22
23  #include <stdint.h>
24  #include <string>
25  #include "modbus-pdu.h"
26
27  const uint32_t MAX_MODBUS_PDU = 253; // Protocol Data Unit
28
29  const uint32_t MAX_COILS_NUM = 65536;
30  const uint32_t MAX_INPUTS_NUM = 65536;
31  const uint32_t MAX_HOLDING_REGISTERS = 65536;
32  const uint32_t MAX_INPUT_REGISTERS = 65536;
33  const uint32_t MAX_EVENTS = 64;
```

Figure 9. Ns-Modbus primary data model table sizes defined in modbus-mgr.h.

```
18   * Author: Reza Sahraei <mrs16@sfu.ca>
19   */
20
21  #include "modbus-mgr.h"
22
23  namespace ns3 {
24
25  ModbusMgr::ModbusMgr ()
26  {
27    m_rspPdu = new uint8_t [PDU_SIZE];
28    m_coilsStatus = new uint8_t [MAX_COILS_NUM];
29    m_inputsStatus = new uint8_t [MAX_INPUTS_NUM];
30    m_holdingRegisters = new uint16_t [MAX_HOLDING_REGISTERS];
31    m_inputRegisters = new uint16_t [MAX_INPUT_REGISTERS];
32    m_exceptionStatusRegister = 0;
```

Figure 10. Ns-Modbus primary data model arrays allocated in modbus-mgr.cc.

3.4.2. Ns-Modbus Function Codes

Ns-Modbus implemented the public assigned function codes, defined in Modbus application protocol specification V1.1b (see Section 2.1.3 Modbus Function Code). These function codes include bit and word data access, file record access, and diagnostics. A complete list of implemented function codes is given in Table 2.

Table 2. Definition of Public Function Codes.

Function Code	Sub Code	Description	Comments
02		Read Discrete Inputs	Physical Discrete Inputs (bit access)
01		Read Coils	Internal bits or Physical Coils (bit access)
05		Write Single Coil	Internal bits or Physical Coils (bit access)
15		Write Multiple Coils	Internal bits or Physical Coils (bit access)
04		Read Input Register	Physical Input Registers (16-bit access)
03		Read Holding Registers	Internal Registers or Physical Output Registers (16-bit access)
06		Write Signle Register	Internal Registers or Physical Output Registers (16-bit access)
16		Write Multiple Registers	Internal Registers or Physical Output Registers (16-bit access)
23		Read/Write Multiple Registers	Internal Registers or Physical Output Registers (16-bit access)
22		Mask Write Register	Internal Registers or Physical Output Registers (16-bit access)
24		Read FIFO queue	Internal Registers or Physical Output Registers (16-bit access)
20		Read File Record	File Record access
21		Write File Record	File Record access
07		Read Exception Status	Diagnostics
08	0-18, 20	Diagnostic	Diagnostics
11		Get Com Event Counter	Diagnostics
12		Get Com Event Log	Diagnostics
17		Report Slave ID	Diagnostics
43	14	Read Device Identification	Diagnostics
43	13, 14	Encapsulated Interface Transport	Other
43	13	CANopen General Reference	Other

Note: Additional details are given in the Public Function Code descriptions in Modbus application protocol specification V1.1b [1].

3.4.3. Ns-Modbus Incorporation into the ns Distribution

Waf uses *wscript* files under the ns directory structure to configure and build the network simulation application. One specific *wscript* needs to be modified in order to incorporate the ns-Modbus code under the ns directory structure. The location of the *wscript* is under the relative path of */ns-3-dev/src/applications*. This *wscript* file specifies the header files and the C++ files of both *model* and *helper* subdirectories to be included in the build. The *wscript* file provides two sections of *headers.source* and *module.source* to incorporate the header files and the C++ files respectively. Ns-Modbus has six header files and six C++ files under *model* subdirectory. Furthermore, ns-Modbus has two header files and two C++ files under *helper* subdirectory to be included in the build. I included *modbus-udp-client.h, modbus-udp-server.h, modbus-tcp-client.h, modbus-tcp-server.h, modbus-pdu.h, modbus-mgr.h, modbus-udp-helper.h,* and *modbus-tcp-helper.h* under the *headers.source* section. Also, I included *modbus-udp-client.cc, modbus-udp-server.cc, modbus-tcp-client.cc, modbus-tcp-server.cc, modbus-pdu.cc, modbus-mgr.cc, modbus-udp-helper.cc,* and *modbus-tcp-helper.cc* under the *module.source* section. There is no Linux patch provided to automate the insertion of the instructions to incorporate files in the *wscript.* It is because of the slight variation of the *wscript* file contents for different ns-3 distributions. The added instructions to the *wscript* are shown in Figure 11 and Figure 12.

```
## -*- Mode: python; py-indent-offset: 4; indent-tabs-mode: nil; coding: utf-8; -*
def build(bld):
    module = bld.create_ns3_module('applications', ['internet', 'config-store', 't
    module.source = [
        'model/bulk-send-application.cc',
        'model/onoff-application.cc',
        'model/packet-sink.cc',
        'model/ping6.cc',
        'model/radvd.cc',
        'model/radvd-interface.cc',
        'model/radvd-prefix.cc',
        'model/udp-client.cc',
        'model/udp-server.cc',
        'model/seq-ts-header.cc',
        'model/udp-trace-client.cc',
        'model/packet-loss-counter.cc',
        'model/udp-echo-client.cc',
        'model/udp-echo-server.cc',
        'model/v4ping.cc',
        'model/modbus-udp-client.cc',
        'model/modbus-udp-server.cc',
        'model/modbus-tcp-client.cc',
        'model/modbus-tcp-server.cc',
        'model/modbus-pdu.cc',
        'model/modbus-mgr.cc',
        'model/bgp-router.cc',
        'model/bgp-pdu.cc',
        'helper/bulk-send-helper.cc',
        'helper/on-off-helper.cc',
        'helper/packet-sink-helper.cc',
        'helper/ping6-helper.cc',
        'helper/udp-client-server-helper.cc',
        'helper/udp-echo-helper.cc',
        'helper/v4ping-helper.cc',
        'helper/modbus-udp-helper.cc',
        'helper/modbus-tcp-helper.cc',
        'helper/bgp-router-helper.cc',
```

Figure 11. The instructions to incorporate the ns-Modbus C++ files into the …/ns-3-dev/src/applications/wscript.

```
48      headers.source = [
49          'model/bulk-send-application.h',
50          'model/onoff-application.h',
51          'model/packet-sink.h',
52          'model/ping6.h',
53          'model/radvd.h',
54          'model/radvd-interface.h',
55          'model/radvd-prefix.h',
56          'model/udp-client.h',
57          'model/udp-server.h',
58          'model/seq-ts-header.h',
59          'model/udp-trace-client.h',
60          'model/packet-loss-counter.h',
61          'model/udp-echo-client.h',
62          'model/udp-echo-server.h',
63          'model/v4ping.h',
64          'model/modbus-udp-client.h',
65          'model/modbus-udp-server.h',
66          'model/modbus-tcp-client.h',
67          'model/modbus-tcp-server.h',
68          'model/modbus-pdu.h',
69          'model/modbus-mgr.h',
70          'model/bgp-router.h',
71          'model/bgp-pdu.h',
72          'helper/bulk-send-helper.h',
73          'helper/on-off-helper.h',
74          'helper/packet-sink-helper.h',
75          'helper/ping6-helper.h',
76          'helper/udp-client-server-helper.h',
77          'helper/udp-echo-helper.h',
78          'helper/v4ping-helper.h',
79          'helper/modbus-udp-helper.h',
80          'helper/modbus-tcp-helper.h',
81          'helper/bgp-router-helper.h',
82          ]
83
84      bld.ns3_python_bindings()
```

Figure 12. The instructions to incorporate the ns-Modbus header files into the …/ns-3-dev/src/applications/wscript.

3.4.4. Ns-Modbus Files

The ns-Modbus files and directories are archived into a .tgz tar ball. A complete list of the files and the relative path is shown in Table 3. The *wscript* is the only file in the ns-3 distribution that was modified to specify the Modbus files for Waf to include in the build process (Section 3.4.3 ns-Modbus Incorporation into ns Distribution). The contents of the ns-Modbus files are given in Appendix E and Appendix F.

Table 3. The ns-Modbus File Names and Relative Path.

File Name	Relative Path	Action
myFirstModbusUdp.cc	~/ns-3-allinone/ns-3-dev/scratch/	Added
myFirstModbusTcp.cc	~/ns-3-allinone/ns-3-dev/scratch/	Added
busModbusUdp.cc	~/ns-3-allinone/ns-3-dev/scratch/	Added
busModbusTcp.cc	~/ns-3-allinone/ns-3-dev/scratch/	Added
busModbusTcpScenario1.cc	~/ns-3-allinone/ns-3-dev/scratch/	Added
busModbusTcpScenario2.cc	~/ns-3-allinone/ns-3-dev/scratch/	Added
busModbusTcpScenario3.cc	~/ns-3-allinone/ns-3-dev/scratch/	Added
starModbusUdp.cc	~/ns-3-allinone/ns-3-dev/scratch/	Added
starModbusTcp.cc	~/ns-3-allinone/ns-3-dev/scratch/	Added
wscript	~/ns-3-allinone/ns-3-dev/src/applications/	Modified
modbus-udp-helper.h	~/ns-3-allinone/ns-3-dev/src/applications/helper/	Added
modbus-udp-helper.cc	~/ns-3-allinone/ns-3-dev/src/applications/helper/	Added
modbus-tcp-helper.h	~/ns-3-allinone/ns-3-dev/src/applications/helper/	Added
modbus-tcp-helper.cc	~/ns-3-allinone/ns-3-dev/src/applications/helper/	Added
modbus-pdu.h	~/ns-3-allinone/ns-3-dev/src/applications/model/	Added
modbus-pdu.cc	~/ns-3-allinone/ns-3-dev/src/applications/model/	Added
modbus-mgr.h	~/ns-3-allinone/ns-3-dev/src/applications/model/	Added
modbus-mgr.cc	~/ns-3-allinone/ns-3-dev/src/applications/model/	Added
modbus-udp-client.h	~/ns-3-allinone/ns-3-dev/src/applications/model/	Added
modbus-udp-client.cc	~/ns-3-allinone/ns-3-dev/src/applications/model/	Added
modbus-udp-server.h	~/ns-3-allinone/ns-3-dev/src/applications/model/	Added
modbus-udp-server.cc	~/ns-3-allinone/ns-3-dev/src/applications/model/	Added
modbus-tcp-client.h	~/ns-3-allinone/ns-3-dev/src/applications/model/	Added
modbus-tcp-client.cc	~/ns-3-allinone/ns-3-dev/src/applications/model/	Added
modbus-tcp-server.h	~/ns-3-allinone/ns-3-dev/src/applications/model/	Added
modbus-tcp-server.cc	~/ns-3-allinone/ns-3-dev/src/applications/model/	Added

3.4.5. Ns-Modbus Netmask

The netmask in the bus topology scenarios is set to 255.255.255.0. Therefore the maximum number of permissible Carrier Sense Multiple Access (CSMA) nodes is limited to 253 (the values 0 and 255 are not allowed). The maximum number of nodes limits the scalability test discussed in Chapter 5. The netmask may be modified to allow a higher number of nodes (i.e. 255.255.0.0). However, this has not been tested.

4. Verification and Validation

Both the bus and the star topologies were tested to validate the implementation. The Modbus server in each scenario runs from the 1st second and stops running at the 10th second of the simulation. The clients start running at 2 seconds and stop at 10 seconds of the simulation. Each client sends to the server a Modbus "Encap Interface Trans Canopen General Ref" request at 2.1 seconds. Subsequently, the server responds to the request. I measured the "*total execution*" time, "*Modbus simulation*" time, "*node and link creation*" time, and "*round-trip Modbus request/response*" time for various numbers of the Modbus clients. The "*round-trip Modbus request/response*" time is the average value of all the measured round-trip times between a Modbus client and the Modbus server. The verbose log messages, the generation of trace files, the generation of pcap files, and the generation of animation files were disabled during the software profiling.

In this Section, I described the test cases that were implemented in three scenarios to verify and validate ns-Modbus. The first test case verifies the process of sending two Modbus requests by the client and receiving the proper Modbus responses. The second test case writes new values to the primary table of the Modbus server then reads them back and verifies the values. The third test case sends requests with illegal function code, invalid data address, and incorrect data value and verifies the proper exceptions are sent back by the Modbus server.

4.1. Sending Request and Receiving Response Test Case

I used *busModbusTcpScenario1* to send the "*encapsulated interface transport*" and the "*read coils*" requests and receive the associated responses. I collected the verbose message logs, which were generated in both the Modbus client and the Modbus server sides. Subsequently, I used Wireshark to analyze the generated network traffic.

The contents of the two requests used by this scenario are shown in Figure 13. The first Modbus request of this test case is *ENCAP_INTERFACE_TRANS_CANOPEN_GENERAL_REF_REQ_RSP_PDU* (line 51). The function code of the request (first byte) is 0x2B (43 decimal). This function code requires a sub code, which is stored in the second byte. The sub code of the request is 0x0D (13 decimal). The combination of the function code and the sub code signifies "*encapsulated interface transport*" (details are shown in Table 2). The second Modbus request is *READ_COILS_REQ* (line 50). The function code of the request is 0x01 that indicates "*read coils*". The next two bytes of the request indicate the starting address with the value of 0x0013 (19 decimal). The last two bytes of the request denote the quantity of coils with the value of 0x0013 (19 decimal) as well.

```
49  // MODBUS requests
50  uint8_t READ_COILS_REQ[] = {0x01, 0x00, 0x13, 0x00, 0x13};
51  uint8_t ENCAP_INTERFACE_TRANS_CANOPEN_GENERAL_REF_REQ_RSP_PDU[] = {0x2B, 0x0D};
52  uint8_t ENCAP_INTERFACE_TRANS_READ_DEVICE_ID__BASIC_DEVICE_ID_OBJ_ID_00[] = {0x2
53  uint8_t ENCAP_INTERFACE_TRANS_READ_DEVICE_ID__REGULAR_DEVICE_ID_OBJ_ID_00[] = {0
54  uint8_t ENCAP_INTERFACE_TRANS_READ_DEVICE_ID__EXTENDED_DEVICE_ID_OBJ_ID_00[] = {
55  uint8_t ENCAP_INTERFACE_TRANS_READ_DEVICE_ID__SPECIFIC_DEVICE_ID_OBJ_ID_00[] = {
56
57
58  using namespace ns3;
```

Figure 13. Requested function codes of the sending request and receiving response test case.

The instructions setting the two Modbus requests in the scenario are shown in Figure 14 (lines 138 to 142). The start delay for each request is set to one second, which makes the Modbus client to send the first request (*ENCAP_INTERFACE_TRANS_CANOPEN_GENERAL_REF_REQ_RSP_PDU*) one second after it starts running. Then it waits for another second to send the *READ_COILS_REQ* request. The Modbus client starts running after 2 seconds have elapsed from the beginning of the simulation (line 146). Therefore, the first request is sent after the 3 seconds then the next request is sent after the 4 seconds after the start of the simulation. I examine the timing in the log messages and also show the network activities using the Wireshark in the next Section.

```
121   ModbusTcpServerHelper cModbusTcpServerHelper;
122   ApplicationContainer serverApps = cModbusTcpServerHelper.Install (csmaNodes.Ge
123   serverApps.Start (Seconds (1.0));
124   serverApps.Stop (Seconds (10.0));
125
126   //
127   // Create MODBUS TCP client applications send UDP to the hub
128   //
129   for (uint32_t i = 1; i < nCsma; ++i)
130     {
131       ModbusTcpClientHelper cModbusTcpClientHelper (csmaInterfaces.GetAddress (0
132       cModbusTcpClientHelper.SetAttribute ("MaxPackets", UintegerValue (1));
133       cModbusTcpClientHelper.SetAttribute ("Interval", TimeValue (Seconds (1.0))
134
135       ApplicationContainer clientApps = cModbusTcpClientHelper.Install (csmaNode
136
137       // set MODBUS requests                                   one second delay each
138       cModbusTcpClientHelper.SetRequest (clientApps.Get(0), Seconds (1),
139         ENCAP_INTERFACE_TRANS_CANOPEN_GENERAL_REF_REQ_RSP_PDU);
140
141       cModbusTcpClientHelper.SetRequest (clientApps.Get(0), Seconds (1),
142         READ_COILS_REQ);
143
144       cModbusTcpClientHelper.SetCount (clientApps.Get(0), 1);
145
146       clientApps.Start (Seconds (2.0));   starts at the 2nd second
147       clientApps.Stop (Seconds (10.0));
148     }
```

Figure 14. Scheduled requests of the sending request and receiving response test case.

4.1.1. Sending Request and Receiving Response Test Case Results Verification

The following two Linux commands are used to run the scenario:

$export NS_LOG=BusModbusTcpScenario1=level_all

$./waf --run "scratch/busModbusTcpScenario1 --verbose=1"

The message logs of the scenario are shown in Figure 15 and Figure 16. Lines 6 to 13 in Figure 15 show the preparation of the scenario. The simulation starts running at line 14. Lines 15 and 16 show that at the third second the client sends a Modbus request to the server with IP address 10.1.1.1. This time agrees with the client start time added to the first request delay. Lines 18, 19, and 20 illustrate parsing the request by the Modbus server, which correctly identified the request. Line 22 indicates that the response is sent back. Lines 23 to 35 confirm the receipt of the response by the Modbus client from the

server with IP address 10.1.1.1. The values received indicate the echo of the function code and sub function code (MEI Type) followed by the values of the coils (lines 26 and 27). The response content agrees with the Modbus application protocol specification [1] encapsulated interface transport (Section 6.19).

```
1  reza@Dena:~/temp/ns-3-allinone/ns-3-dev$ export NS_LOG=BusModbusTcpScenario1=level
2  reza@Dena:~/temp/ns-3-allinone/ns-3-dev$ ./waf --run "scratch/busModbusTcpScenaric
3  Waf: Entering directory `/home/reza/temp/ns-3-allinone/ns-3-dev/build'
4  Waf: Leaving directory `/home/reza/temp/ns-3-allinone/ns-3-dev/build'
5  'build' finished successfully (1.694s)
6  Build bus topology.
7  Install internet stack on all nodes.
8  Assign IP Addresses.
9  Create applications.
10 ModbusTcpClient::SetRequest ()
11 ModbusTcpClient::SetRequest ()
12 ModbusTcpClient::SetCount ()
13 Enable pcap tracing.
14 Run Simulation.
15 ModbusTcpClient::Send ()
16 TraceDelay TX: 253 bytes to 10.1.1.1 Uid: 14 Time: 3
17 Parsing MODBUS TCP function code
18 Encapsulated Interface Transport
19 meiType: 0xd
20 CANopen General Reference Request and Response PDU
21 TraceDelay: RX 253 bytes from 10.1.1.2 Sequence Number: 0 Uid: 14 TXtime: +3000000
22 Sending MODBUS TCP response packet
23 ModbusTcpClient::HandleRead ()
24 Received 253 bytes from 10.1.1.1
25 Success - function code: 0x2b
26 pdu[0]: 0x2b
27 pdu[1]: 0xd
28 pdu[2]: 0x2b
29 pdu[3]: 0xe
30 pdu[4]: 0x1
31 pdu[5]: 0x0
32 pdu[6]: 0x2b
33 pdu[7]: 0xe
34 pdu[8]: 0x2
35 pdu[9]: 0x0
```

Client sends the request at the third second

Server parses the request

Client receives the response

Figure 15. Verbose message logs of the sending request and receiving response test case.

Lines 37 and 38 in Figure 16 denote sending of the second request at the forth second. This timing also agrees with the summation of the client start up, the first request delay, and the second request delay. The request is identified correctly by the Modbus server shown at lines 40, 41, and 42. The Modbus response is generated and sent back at line 44. Lines 45 to 57 confirm the receipt of the response from the server with IP address

10.1.1.1. The first byte in the response (line 48) is an echo of the request. According to the Modbus specification the second byte (line 49) indicates the “byte count”, which is the function of “quantity of coils” in the request (19 decimal). The “byte count” is calculated by dividing the “quantity of coils” by 8 and if the remainder is not zero then increment the “byte count” by one. Line 49 in Figure 16 proves that the “byte count” is calculated correctly. The additional three bytes are the value of the requested coils (lines 50 to 52). The rest of the bytes are invalid (lines 53 to 57), since the length of the packet is fixed to a constant length in the simulation. Lines 59 and 60 show the two TCP closures.

```
37  ModbusTcpClient::Send ()
38  TraceDelay TX: 253 bytes to 10.1.1.1 Uid: 22 Time: 4
39  Parsing MODBUS TCP function code
40  Read Coils
41  startAdr: 19
42  quantityOfCoils: 19
43  TraceDelay: RX 253 bytes from 10.1.1.2 Sequence Number: 1 Uid: 22 TXtime: +4000000
44  Sending MODBUS TCP response packet
45  ModbusTcpClient::HandleRead ()
46  Received 253 bytes from 10.1.1.1
47  Success - function code: 0x1
48  pdu[0]: 0x1
49  pdu[1]: 0x3
50  pdu[2]: 0xba
51  pdu[3]: 0x1
52  pdu[4]: 0x28
53  pdu[5]: 0x34
54  pdu[6]: 0x2
55  pdu[7]: 0x1
56  pdu[8]: 0x1
57  pdu[9]: 0x0
58
59  ModbusTcpServer, peerClose
60  ModbusTcpServer, peerClose
61  Done.
62  Total (ms): 111.01
63  Modbus Simulation time (ms): 97.7521
64  Node and Link creation (ms): 13.258
65  reza@Dena:~/temp/ns-3-allinone/ns-3-dev$
```

Client sends the request at the fourth second

Server parses the request

Client receives the response

Figure 16. Verbose message logs of the sending request and receiving response test case (continuation).

The network activities between the Modbus client and the Modbus server nodes are shown in Figure 17. The first packet is an Address Resolution Protocol (ARP) request issued by the client and the server replies with an ARP response in the second packet.

Subsequently, the client sends a SYN packet to open a TCP connection to the server port 502. The packet number 4 is an ARP request issued by the server and the client answers with an ARP response. This is the current ns implementation behaviour however, ns should keep a record of the client Medium Access Control (MAC) address in the server MAC table when it receives the client ARP request (packet number 1). Afterwards, the server sends a SYN/ACK packet to the client (packet number 6). The client completes the TCP three-way handshake in packet 7 by sending an acknowledgment.

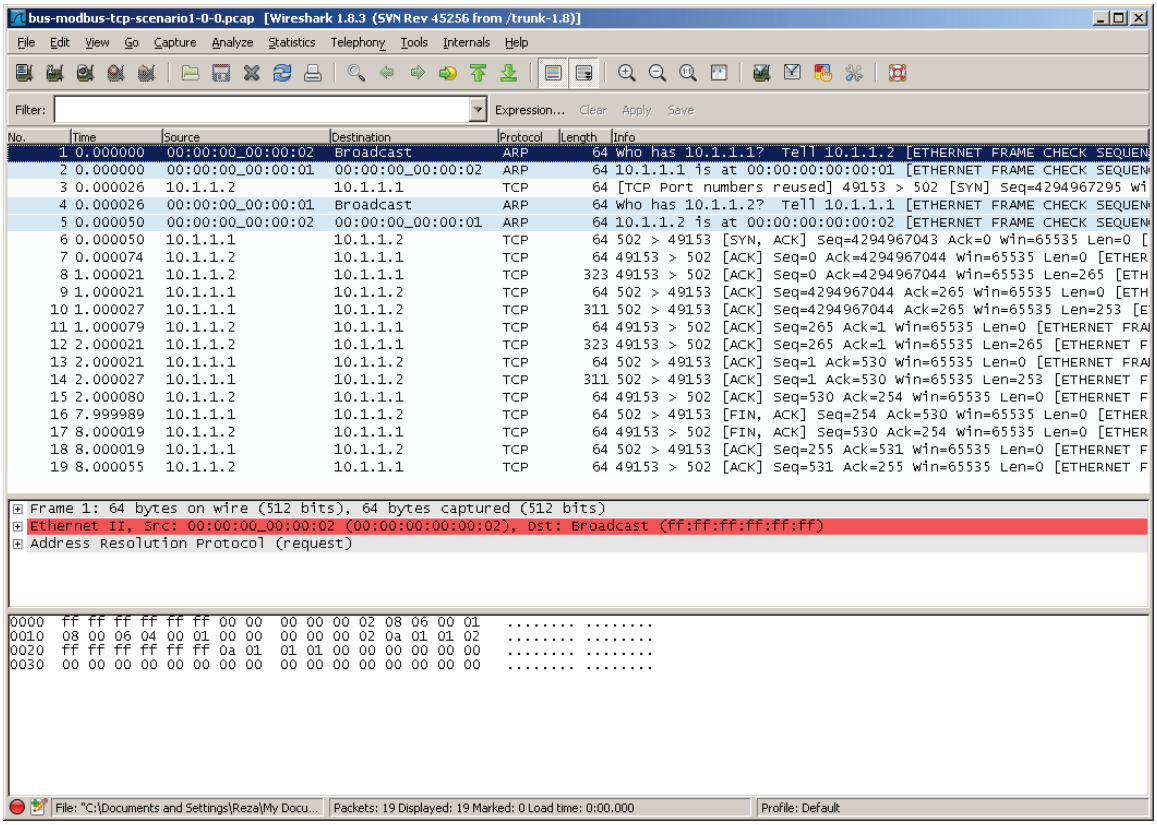

Figure 17. Wireshark capture of the sending request and receiving response test case.

Thus far, all the activities occur at the beginning of the capture, when the client application starts the simulation at the 2nd second (Figure 14 line 146). Packets number 8 to 11 take place one second after the start of the capture, which is 3 seconds after the beginning of the simulation. The client sends a Modbus request in the packet number 8, which is acknowledged by the server in the packet number 9. The server sends the

associated Modbus response in the packet number 10, which is followed by the client acknowledgment in the packet number 11. The packets number 12 to 15 occur two seconds after the start of the capture. The sending and receiving activities of the second request are similar to the first one. In the packets number 16 to 19, both the Modbus client and the Modbus server close the TCP connection. The termination of the TCP connection takes place 8 seconds after the start of the capture, which is 10 seconds after the beginning of the simulation (Figure 14 line 147).

The analysis of the timing and the content of the packets show that ns-Modbus generates the network traffic timely and correctly according to the defined scenario for this test case.

4.2. Writing to and Reading from Primary Table Test Case

I used *busModbusTcpScenario2* to send the "*write multiple coils*" and "*read coils*" Modbus requests and receive the associated responses. I showed the verbose message logs that were generated by the Modbus client and the Modbus server side to verify that the coils value modification has been done successfully.

The content of the two requests used in this scenario is shown in Figure 18. The first Modbus request of this test case is *WRITE_MULTIPLE_COILS_REQ* (line 50). The function code of the request is 0x0F (15 decimal) that indicates "write multiple coils" (Table 2, definition of public function codes). The second and the third bytes of the request (set to 0) signify the "starting address" of the coils. The next two bytes of the request represent the "number of coils" with the value of 0x0010 (16 decimal). The byte number six of the request indicates the length of the coil values to be set in bytes, which is set to 0x02. The next two bytes are the values to be set in the sixteen coils, since each coil is assigned to a bit to turn on or off. In this test case, 0xA5 and 0xF0 (10100101 and 11110000 binary) are chosen for the coil values. These values are commonly used for testing memories in order to toggle each bit or nibble independently. The second Modbus request is *READ_COILS_REQ* (line 52). The function code of the request is 0x01 that "read coils". The next two bytes of the request indicate the "starting address" with the value of 0. The last two bytes of the request denote the "quantity of

coils" with the value of 0x0010 (16 decimal). The "starting address" and the "number of coils" are identical to the "write multiple coils" request.

```
49  // MODBUS requests
50  uint8_t WRITE_MULTIPLE_COILS_REQ[] =
51     {0x0F, 0x00, 0x00, 0x00, 0x10, 0x02, 0xA5, 0xF0};
52  uint8_t READ_COILS_REQ[] =
53     {0x01, 0x00, 0x00, 0x00, 0x10};
54
55  using namespace ns3;
56
57  NS_LOG_COMPONENT_DEFINE ("BusModbusTcpScenario2");
```

Write Coils and
Read Coils Requests

Figure 18. Requested function codes of the writing to and reading from primary table test case.

The instructions setting the two Modbus requests in the scenario are shown in Figure 19 (lines 135 to 139). The start delay for each request is set to one second, which makes the Modbus client to send the first request (*WRITE_MULTIPLE_COILS_REQ)* then it waits for another second to send the next request (*READ_COILS_REQ)*. The first request is sent after the 3rd second then the next request is sent 4 seconds after the start of the simulation. I examine the timing and the coil values in the log messages.

```
134       // set MODBUS requests
135       cModbusTcpClientHelper.SetRequest (clientApps.Get(0), Seconds (1),
136         WRITE_MULTIPLE_COILS_REQ);
137
138       cModbusTcpClientHelper.SetRequest (clientApps.Get(0), Seconds (1),
139         READ_COILS_REQ);
140
141       cModbusTcpClientHelper.SetCount (clientApps.Get(0), 1);
142
143       clientApps.Start (Seconds (2.0));
144       clientApps.Stop (Seconds (10.0));
145   }
```

Figure 19. Scheduled requests of the writing to and reading from primary table test case.

4.2.1. Writing to and Reading from Primary Table Results Verification

The following two Linux commands are used to run the scenario:

$export NS_LOG=BusModbusTcpScenario2=level_all
$./waf --run "scratch/busModbusTcpScenario2 --verbose=1"

The message logs of the scenario are shown in Figure 20 and Figure 21. The lines 5 to 12 in Figure 20 show the preparation of the scenario to run. The simulation starts running at line 13. The client sends the *WRITE_MULTIPLE_COILS_REQ* request to the server with IP address 10.1.1.1 at lines 14 and 15. The timing of sending the request agrees with the Modbus client start time added to the first request delay. The server parses the request and identifies it as “write multiple coils” (line 17) with the “starting address” set to zero (line 18), the “quantity of outputs” set to 16 (line 19), and the “byte count” set to 2 (line20). Line 22 indicates that a response is sent back. The lines 23 to 35 confirm the receipt of the response by the Modbus client from the server with the IP address 10.1.1.1. The values received indicate the echo of the “function code”, “starting address”, and “quantity of outputs” (lines 26 to 30). The response content agrees with the Modbus application protocol specification [1] “write multiple coils” (Section 6.11). The lines 31 to 35 in Figure 20 are not part of the standard response and are ignored by the Modbus client. Fixed number bytes from the response get printed out regardless of the response type.

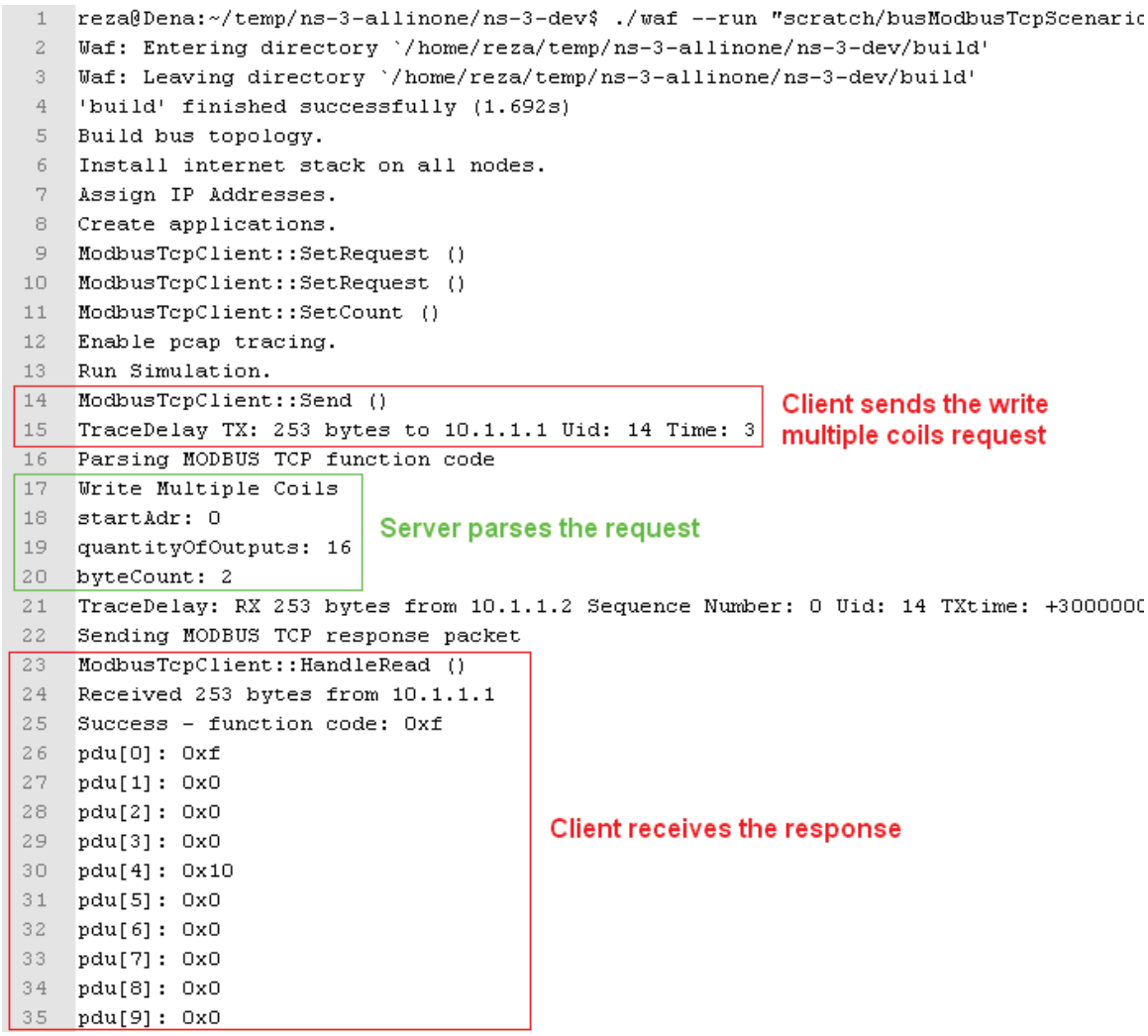

Figure 20. Verbose message logs of the writing to and reading from primary table test case.

Lines 37 and 38 in Figure 21 denote sending of the *READ_COILS_REQ* request at the 4th second. This timing also agrees with the summation of the client start up, the first request delay, and the second request delay. The request is identified correctly by the Modbus server shown at lines 40, 41, and 42. The Modbus response is generated and sent back at line 44. Lines 45 to 57 confirm the receipt of the response from the server with IP address 10.1.1.1. The first byte in the response (line 48) is an echo of the request. The second byte in line 49 is the “byte count”, which is the function of “quantity of coils” in the request (the “byte count” equals the “quantity of coils” divided by 8, if the remainder is different than 0 then increment the result by one). The additional two bytes (lines 50 and 51) are the current values of the requested coils (0xA5 and 0x10), which

are identical to the values that are set by the *WRITE_MULTIPLE_COILS_REQ*, confirming a successful operation of setting the coils. Lines 59 and 60 show the two TCP closures.

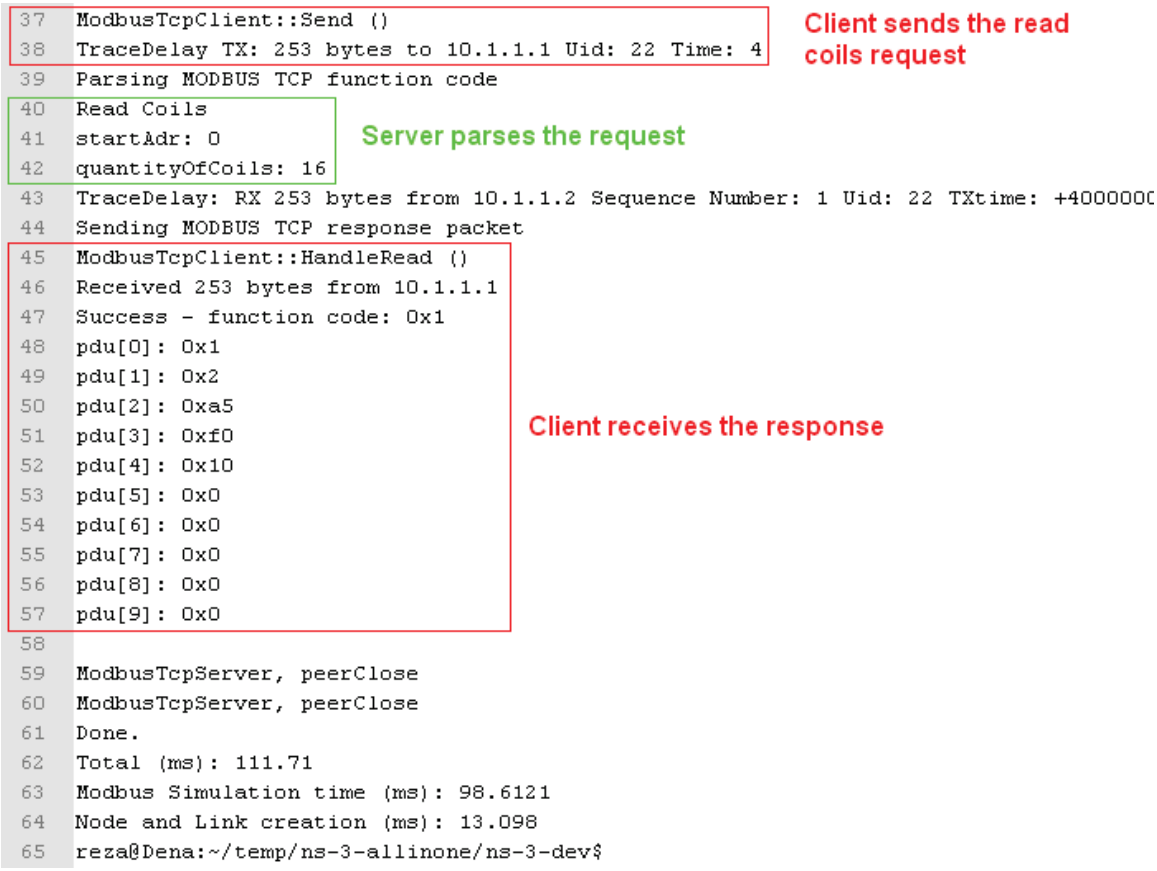

Figure 21. Verbose message logs of the writing to and reading from primary table test case (continuation).

4.3. Sending Modbus Request with Illegal Data Test Case

I used *busModbusTcpScenario3* to generate three Modbus requests with illegal data. The first request has an undefined “function code”. The second request has an invalid “data address”. The third request has an invalid data value. These requests with illegal data cause exception codes 1, 2, and 3 respectively. I showed the verbose message logs, which were generated in both the Modbus client and the Modbus server.

The content of the three Modbus requests used in this test case is shown in Figure 22. The first Modbus request (line 52) has an illegal function code 0. The second Modbus request (lines 55 and 56) is a *WRITE_MULTIPLE_COILS_REQ* with an invalid data address because the summation of "starting address" of *0xFFFF* (65,535 decimal) and the "quantity of outputs" of *0x10* (16 decimal) exceeds the total allowable and allocated area for the coils values. Finally, the last Modbus request (lines 59 and 60) has an illegal "quantity of outputs" with the value of *0xFFFF* (65,535 decimal) that exceeds the specification maximum allowed number, which is 2000 decimal (Modbus application protocol specification Section 6.11).

```
49  // Invalid MODBUS requests
50  //
51  // Function code is not supported: exception code = 1
52  uint8_t WRONG_FUNCTION_CODE_REQ[] = {0x00};
53  //
54  // Illegal data address: exception code = 2
55  uint8_t WRITE_MULTIPLE_COILS_REQ_WRONG_START_ADR_PLUS_QUANTITY[] =
56     {0x0F, 0xFF, 0xFF, 0x00, 0x10, 0x02, 0xA5, 0xF0};
57  //
58  // Illegal data value: exception code = 3
59  uint8_t WRITE_MULTIPLE_COILS_REQ_WRONG_QUANTITY_OF_OUTPUT[] =
60     {0x0F, 0x00, 0x00, 0xFF, 0xFF, 0x02, 0xA5, 0xF0};
61
62  using namespace ns3;
63
64  NS_LOG_COMPONENT_DEFINE ("BusModbusTcpScenario3");
```

Figure 22. Requested function codes of the sending Modbus request with illegal data test case.

The instructions setting the three Modbus requests in the scenario are shown in Figure 23 (lines 142 to 149). The first request has wrong function code (*0x00)* at line 142. The second request is a "write multiple coils" with illegal "data address" at line 145. Finally, the last request is a "write multiple coils" with illegal "data value" at line 148. Each request has a starting delay of 1 second.

The SetCount function at line 151 indicates how many times the sequence of sending the requests occurs. The sequence is set to one, therefore, the three Modbus requests are only sent once. The function clientApps.Start (line 153) instructs the Modbus client to start after 2 seconds have elapsed since the beginning of the simulation. Therefore, the first request (*WRONG_FUNCTION_CODE_REQ)* is sent after 3 seconds have

elapsed from the beginning of the simulation. Subsequently, the second request is sent after another second. The last request is forwarded to the Modbus server 5 seconds after the start of the simulation.

The function serverApps.Start (line 127) instructs the Modbus server to start one second after the beginning of the simulation. Therefore, the Modbus server is ready to receive the Modbus clients' requests.

```
125   ModbusTcpServerHelper cModbusTcpServerHelper;
126   ApplicationContainer serverApps = cModbusTcpServerHelper.Install (csmaNodes.Ge
127   serverApps.Start (Seconds (1.0));
128   serverApps.Stop (Seconds (10.0));
129
130   //
131   // Create MODBUS TCP client applications send UDP to the hub
132   //
133   for (uint32_t i = 1; i < nCsma; ++i)
134     {
135       ModbusTcpClientHelper cModbusTcpClientHelper (csmaInterfaces.GetAddress (0
136       cModbusTcpClientHelper.SetAttribute ("MaxPackets", UintegerValue (1));
137       cModbusTcpClientHelper.SetAttribute ("Interval", TimeValue (Seconds (1.0))
138
139       ApplicationContainer clientApps = cModbusTcpClientHelper.Install (csmaNode
140
141       // set MODBUS requests
142       cModbusTcpClientHelper.SetRequest (clientApps.Get(0), Seconds (1),
143         WRONG_FUNCTION_CODE_REQ);
144
145       cModbusTcpClientHelper.SetRequest (clientApps.Get(0), Seconds (1),
146         WRITE_MULTIPLE_COILS_REQ_WRONG_START_ADR_PLUS_QUANTITY);
147
148       cModbusTcpClientHelper.SetRequest (clientApps.Get(0), Seconds (1),
149         WRITE_MULTIPLE_COILS_REQ_WRONG_QUANTITY_OF_OUTPUT);
150
151       cModbusTcpClientHelper.SetCount (clientApps.Get(0), 1);
152
153       clientApps.Start (Seconds (2.0));
154       clientApps.Stop (Seconds (10.0));
155     }
```

Figure 23. Scheduled requests of the sending Modbus request with illegal data test case.

4.3.1. Sending Modbus Request with Illegal Data Results Verification

The following Linux command lines are used to run the scenario:

$export NS_LOG=BusModbusTcpScenario3=level_all

$./waf --run "scratch/busModbusTcpScenario3 --verbose=1"

The message logs of the scenario are shown in Figures 24, 25, and 26. Lines 6 to 14 in Figure 24 are the preparation of the scenario to run. The simulation starts running at line 15. Lines 16 and 17 show the transmission of the first request with the wrong “function code” when the simulation time reaches the 3 seconds mark.

```
1  reza@Dena:~/temp/ns-3-allinone/ns-3-dev$ export NS_LOG=BusModbusTcpScenario3=level
2  reza@Dena:~/temp/ns-3-allinone/ns-3-dev$ ./waf --run "scratch/busModbusTcpScenaric
3  Waf: Entering directory `/home/reza/temp/ns-3-allinone/ns-3-dev/build'
4  Waf: Leaving directory `/home/reza/temp/ns-3-allinone/ns-3-dev/build'
5  'build' finished successfully (1.761s)
6  Build bus topology.
7  Install internet stack on all nodes.
8  Assign IP Addresses.
9  Create applications.
10 ModbusTcpClient::SetRequest ()
11 ModbusTcpClient::SetRequest ()
12 ModbusTcpClient::SetRequest ()
13 ModbusTcpClient::SetCount ()
14 Enable pcap tracing.
15 Run Simulation.
16 ModbusTcpClient::Send ()                                          Client sends wrong
17 TraceDelay TX: 253 bytes to 10.1.1.1 Uid: 14 Time: 3               function code request
18 Parsing MODBUS TCP function code
19 error: undefined function code                                    Server parses the request
20 TraceDelay: RX 253 bytes from 10.1.1.2 Sequence Number: 0 Uid: 14 TXtime: +3000000
21 Sending MODBUS TCP response packet
22 ModbusTcpClient::HandleRead ()
23 Received 253 bytes from 10.1.1.1
24 MODBUS response error - function code: 0x0 exception code: 0x1     Client receives
25 Invalid funcion or sub-function code                               the response
26 pdu[0]: 0x80
27 pdu[1]: 0x1
28 pdu[2]: 0x0
29 pdu[3]: 0x0
30 pdu[4]: 0x0
```

Figure 24. Verbose message logs of the sending Modbus request with illegal data test case.

The log message at line 19 indicates that the Modbus server received the request and identified it as an invalid function code. The log message at line 21 shows that the Modbus server generated the proper exception message back to the Modbus client.

Lines 22 to 30 confirm the receipt of the response by the Modbus client from the Modbus server with the IP address 10.1.1.1. The values received indicate the echo of the "function code" (*0x00)* added to the constant *0x80* (128 decimal), which correctly mapped to the allocated exception code area. The second byte of the Modbus exception response signifies the "exception code", which gives additional information about the type of the received exception (details are given in Appendix J). The "exception code" at line 27 is *0x01,* which means an invalid "function code" detected in the request.

The log message at lines 32 and 33 in Figure 25 shows that the second request with an illegal data address is sent when the simulation time reached to the 4 seconds mark. The Modbus server received the second request and identified it correctly as the "write multiple coils" request with the "start address" set to 65,535, the "quantity of outputs" set to 16, and the "byte count" set to 2 logged at the lines 35 to 38.

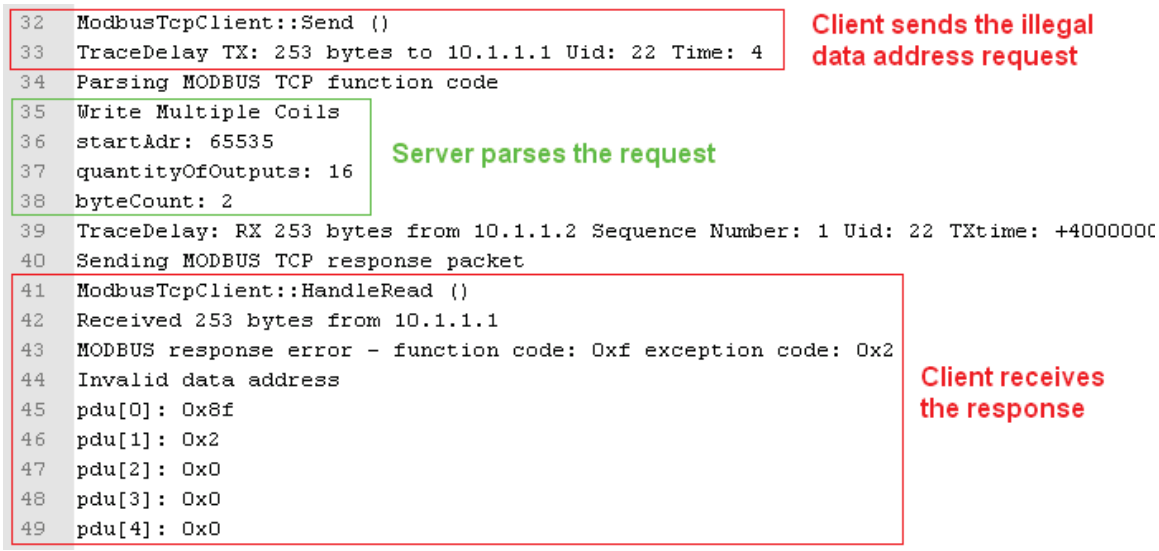

Figure 25. Verbose message logs of the sending Modbus request with illegal data test case (continuation).

The Modbus server examines if the summation of the "start address" and the "quantity of outputs" exceeds the coils memory maximum allowable and allocated area. The

maximum allowable number is 65,535, which is violated in this request. The Modbus server responds back to the Modbus client, which is logged at line 40. The client receives the response and identifies it as an exception (*0x8F* = *0x0F* + *0x80*). The second byte of the response is the "exception code" with the value of *0x02* (line 46), which signifies an invalid data address exception.

The log messages at the lines 51 and 52 in Figure 26 show that the third request is sent to the Modbus server when the simulation time reached the 5 seconds mark.

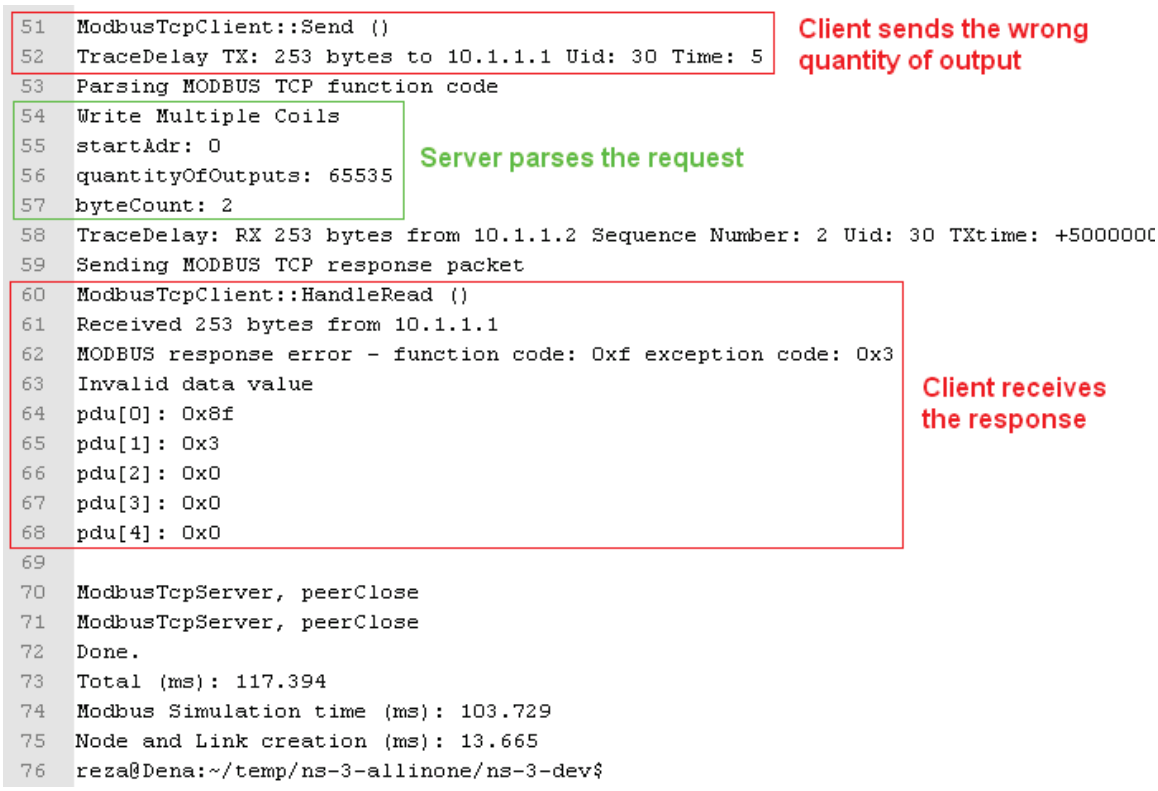

Figure 26. Verbose message logs of the sending Modbus request with illegal data test case (continuation 2).

The Modbus server received the request and identified it correctly as the "write multiple coils" request with the "start address" set to 0, the "quantity of outputs" set to 65,535, and the "byte count" set to 2 logged at the lines 54 to 57. The "quantity of outputs" exceeds the maximum allowable by the specification for this parameter and besides it is too large for all of the coils values to fit in a single response. Therefore, the Modbus server generates an exception instead of a standard response and sends it back to the

Modbus client logged at the line 59. The client receives the response and identifies it as an exception. The exception code of the response correctly indicates that the “exception type” is an “invalid data value”. Lines 70 and 71 show the TCP connection closure of the client and the server nodes.

5. Model Performance

Performance measurement is an important quantitative metric to evaluate the quality of the application. Some other measurements are designed to cover the areas such as robustness, scalability, run-time speed, and the overhead. In this section, I made an effort to gauge some of these subjects. The simulation experiments were performed on a Pentium® Dual-Core CPU T4200 @ 2.00 GHz x 2 host with 2.9 Gigabytes memory and Ubuntu 11.10 Linux operating system. The details of the simulation environment are given in Appendix D.

5.1. Application Profiling

The runtime speed of the code is an important factor that contributes to the quality of the application. Four sections of the code are profiled:

- Node and Link creation
- Modbus Simulation time
- Round-trip Modbus request/response
- Total time

The “Node and Link creation” is the initialization time that includes arguments parsing, building topology, setting channel speed and delay attributes, setting the topology of the network, installing internet stack on the nodes, assigning IP addresses, creating application on the nodes, enabling Pcap tracing, and creating animation object. The start point where the time is measured for the profiling is shown in Figure 27 at line 69. The end point of the profiling is at line 166 in Figure 28.

The “Modbus Simulation” is the time elapsed for the simulator to run and free the objects and the memory allocated by ns-Modbus. This time includes the execution of *Simulator::run()* and *Simulator::Destroy()* instructions (lines 169 and 170 in Figure 28).

```
60  NS_LOG_COMPONENT_DEFINE ("BusModbusUdp");
61
62  int
63  main (int argc, char *argv[])
64  {
65    struct timeval detail_time1, detail_time2, detail_time3;
66
67    // to get current time
68    gettimeofday(&detail_time1, NULL);
69    double t1 = detail_time1.tv_sec + (detail_time1.tv_usec/1000000.0);
70                                                  Node and Link Creation - Start Point
71    bool verbose = false;
72    std::string animFile = "bus-modbus-udp-animfile.xml";
73
74    //
75    // Default number of nodes.  Overridable by command line argument.
76    //
77    uint32_t nCsma = 4;
78
79    // under ns-3-dev type ./waf --run 'scratch/myfirst --PrintHelp'
80    // to see a list of parameters can be passed to the parser
81    CommandLine cmd;
82    cmd.AddValue ("nCsma", "Number of CSMA nodes/devices", nCsma);
83    cmd.AddValue ("verbose", "Tell MODBUS applications to log if true", verbose);
84    cmd.AddValue ("animFile",  "File Name for Animation Output", animFile);
85    cmd.Parse (argc, argv);
```

Figure 27. The start location of ns-Modbus application profiling for "node and link creation". The current time is queried and stored in seconds in variable t1 (line 69).

The "Round-trip Modbus request/response" is an average time elapsed from the instant when a request is sent from a Modbus client to the Modbus server until the related response is received by the Modbus client. Therefore, the "Round-trip Modbus request/response time" is part of the "Modbus Simulation time".

The "Total time" is the summation of "Modbus Simulation time" and "Node and Link creation time". Line 176 in Figure 28 shows the instruction that logs the "Total time".

```
153   NS_LOG_INFO ("Enable pcap tracing.");
154   //configure tracing
155   AsciiTraceHelper ascii;
156   csma.EnableAsciiAll (ascii.CreateFileStream ("bus-modbus-udp.tr"));
157   csma.EnablePcapAll ("bus-modbus-udp");
158
159   // Create the animation object and configure for specified output
160   AnimationInterface anim (animFile);
161   anim.SetXMLOutput ();
162   //anim.StartAnimation ();
163
164   // to get current time                    Node and Link Creation - End Point
165   gettimeofday(&detail_time2, NULL);
166   double t2 = detail_time2.tv_sec + (detail_time2.tv_usec/1000000.0);
167
168   NS_LOG_INFO ("Run Simulation.");
169   Simulator::Run ();                       Modbus Simulation Time
170   Simulator::Destroy ();
171   NS_LOG_INFO ("Done.");
172
173   // print total time elapsed in microsecond
174   gettimeofday(&detail_time3, NULL);
175   double t3 = detail_time3.tv_sec + (detail_time3.tv_usec/1000000.0);
176   NS_LOG_INFO ("Total (ms): " << (t3 - t1)*1000);
177   NS_LOG_INFO ("Modbus Simulation time (ms): " << (t3 - t2)*1000);
178   NS_LOG_INFO ("Node and Link creation (ms): " << (t2 - t1)*1000);
179
180   return 0;
181 }
```

Figure 28. ***The end location of ns-Modbus profiling for "node and link creation". The current time in seconds is stored in t2 (line 166). Line 175 is another location in the code that holds the current time in variable t3. The time difference between t2 and t1 indicates "node and link creation time" (line 176). The time difference between t3 and t2 indicates "Modbus simulation time" (line 177). The time difference between t3 and t1 indicates the "total time" (line 176).***

The application profiling of ns-Modbus UDP for bus topology is shown in Table 4. The profiling is done for various number of nodes, which are recorded in the first column of the Table. The Table starts with two nodes (a single Modbus client and a single Modbus server) and ends with 253 nodes (252 Modbus clients and a single Modbus server). The data rate of the channel is set to 100 Mbps and the delay is set to 6,560 nanoseconds.

Table 4. Bus Modbus UDP Application Profiling (seconds).

Number of Nodes	Total	Modbus Simulation time	Node and Link creation	Round-trip Modbus request/response
2	0.01323	0.00564	0.0076	0.00193
5	0.02028	0.01028	0.00999	0.0052
10	0.03634	0.02178	0.01456	0.01371
25	0.10465	0.07512	0.02953	0.05854
50	0.28223	0.22999	0.05224	0.19583
75	0.58004	0.50477	0.07528	0.43801
100	0.93474	0.84049	0.09425	0.7419
125	1.4787	1.35743	0.12127	1.05154
150	2.10735	1.97132	0.13602	1.46554
175	2.79368	2.63152	0.16216	1.92743
200	3.74294	3.56397	0.17898	2.61863
225	4.77056	4.56402	0.20654	3.22501
253	6.71951	6.47547	0.24404	4.3678

Notes:

- ‘./waf --run "scratch/busModbusUdp --verbose=0 --nCsma=xxx" > deimi.txt’ is the Linux command line used for the profiling.
- One message (ENCAP_INTERFACE_TRANS_CANOPEN_GENERAL_REF_REQ_RSP_PDU) at 2.1 second of startup (0.1 second of client startup) is requested from the Modbus server.
- “Round-trip Modbus request/response” are the average of all individual Modbus clients recorded in deimi.txt file.
- The nCsma, which is passed by the command line to ns-Modbus, includes one Modbus server and one or more Modbus clients.
- The profiling is done by Pentium® Dual-Core CPU T4200 @ 2.00 GHz x 2 host with 2.9 GBytes of memory and the Ubuntu 11.10 Linux operating system.

The application profiling of ns-Modbus TCP for bus topology is shown in Table 5. The Table starts with two nodes (a single Modbus client and a single Modbus server) and ends with 253 nodes (252 clients and a single Modbus server). The data rate of the channel is set to 100 Mbps and the delay is set to 6,560 nanoseconds.

Table 5. Bus Modbus TCP Application Profiling (seconds).

Number of Nodes	Total	Modbus Simulation time	Node and Link creation	Round-trip Modbus request/response
2	0.01671	0.00948	0.00722	0.00137
5	0.03509	0.02519	0.0099	0.00446
10	0.06852	0.05399	0.01453	0.00946
25	0.21209	0.18313	0.02896	0.02928
50	0.59978	0.54791	0.05187	0.09128
75	41.9799	41.9072	0.07248	0.18714
100	77.265	77.1723	0.09266	0.32336
125	126.265	126.151	0.11446	0.55602
150	173.478	173.344	0.134	1.00262
175	200.436	200.278	0.15878	1.69987
200	242.603	242.425	0.17789	2.32805
225	274.381	274.185	0.19566	2.96981
253	303.449	303.222	0.22604	4.72135

Note: ‘./waf --run "scratch/busModbusTcp --verbose=0 --nCsma=xxx" > deimi.txt’ is the Linux command line used for the profiling.

The application profiling of ns-Modbus UDP for star (point-to-point) topology is shown in Table 6. The Table starts with two nodes (a single Modbus client and a single Modbus server) and ends with 3,500 nodes (3,499 clients and a single Modbus server). The data rate of the channel is set to 5 Mbps and the delay is set to 2 milliseconds.

Table 6. Star (P2P) Modbus UDP Application Profiling (seconds).

Number of Nodes	Total	Modbus Simulation time	Node and Link creation	Round-trip Modbus request/response
2	0.01449	0.00519	0.0093	0.0015
5	0.01942	0.00653	0.01289	0.00227
10	0.02919	0.01007	0.01911	0.00404
25	0.05476	0.01871	0.03604	0.00934
50	0.10461	0.0361	0.0685	0.02073
75	0.15982	0.05425	0.10557	0.03257
100	0.2126	0.06927	0.14333	0.0432
125	0.28678	0.09322	0.19356	0.06093
150	0.36441	0.11839	0.24602	0.07875
175	0.45875	0.14217	0.31658	0.09951
200	0.54642	0.16671	0.37971	0.11762
225	0.68401	0.1931	0.4909	0.13835
253	0.82045	0.22592	0.59453	0.16442
500	3.2979	0.62436	2.67354	0.50269
1000	20.0584	1.9777	18.0807	1.73973
1500	61.4051	4.21852	57.1866	3.86056
2000	140.099	7.45611	132.643	6.98735
2500	270.994	11.4441	259.545	10.83923
3000	471.37	16.6923	454.678	15.95855
3500	739.443	22.8586	716.584	22.02983

Notes:

- 'export NS_LOG=StarModbusUdp=level_all' is the Linux command line used to enable the related logging messages.
- './waf --run "scratch/starModbusUdp --verbose=0 --nSpokes=xxx" > deimi.txt' is the Linux command line used for the profiling.

The application profiling of ns-Modbus TCP for star (point-to-point) topology is shown in Table 7.

Table 7. Star (P2P) Modbus TCP Application Profiling (seconds).

Number of Nodes	Total	Modbus Simulation time	Node and Link creation	Round-trip Modbus request/response
2	0.01989	0.01055	0.00935	0.00194
5	0.03156	0.01888	0.01269	0.004
10	0.05102	0.03267	0.01835	0.00753
25	0.10892	0.07149	0.03743	0.02052
50	0.23233	0.15785	0.07449	0.04064
75	0.32948	0.22589	0.10359	0.0593
100	0.48953	0.34431	0.14522	0.09508
125	0.63751	0.44607	0.19144	0.12289
150	0.82712	0.57691	0.25021	0.16084
175	1.04446	0.7251	0.31936	0.20072
200	1.27412	0.87943	0.39469	0.24471
225	1.49689	1.01418	0.48272	0.28635
253	1.79164	1.193	0.59864	0.33567
500	6.42257	3.69915	2.72342	1.05179
1000	30.5966	12.9376	17.6591	3.69869
1500	88.2595	29.189	59.0705	8.31091
2000	184.873	50.5859	134.287	14.34634
2500	338.636	78.9367	259.7	22.28083
3000	563.448	116.733	446.714	33.08998
3500	863.055	157.677	705.378	44.25535

Notes: ‘

- export NS_LOG=StarModbusTcp=level_all’ is the Linux command line used to enable the related logging messages.
- ‘./waf --run "scratch/starModbusTcp --verbose=0 --nSpokes=xxx" > deimi.txt’ is the Linux command line used for the profiling.

The Table starts with two nodes (a single Modbus client and a single Modbus server) and ends with 3,500 nodes (3,499 clients and a single Modbus server). The data rate of the channel is set to 5 Mbps and the delay is set to 2 milliseconds.

5.1.1. Model Scalability

Addressing the scalability of the simulated network is very important especially when the model becomes complex. I validated the implementation on a network with maximum 3,500 nodes. The validation scenarios for ns-Modbus permit 252 client nodes for bus topology and up to 3,500 client nodes for star topology.

Scalability: Number of Modbus Clients in a Bus Topology

I considered “Modbus simulation time”, “node and link creation time”, “round-trip Modbus request/response time”, and “total execution time” of a bus topology model for both UDP and TCP connections. The execution time for various numbers of Modbus UDP clients is shown in Figure 29. The ratio of the averaged “round-trip request/response time” over the “total time” is 0.6–0.7 (60%–70%).

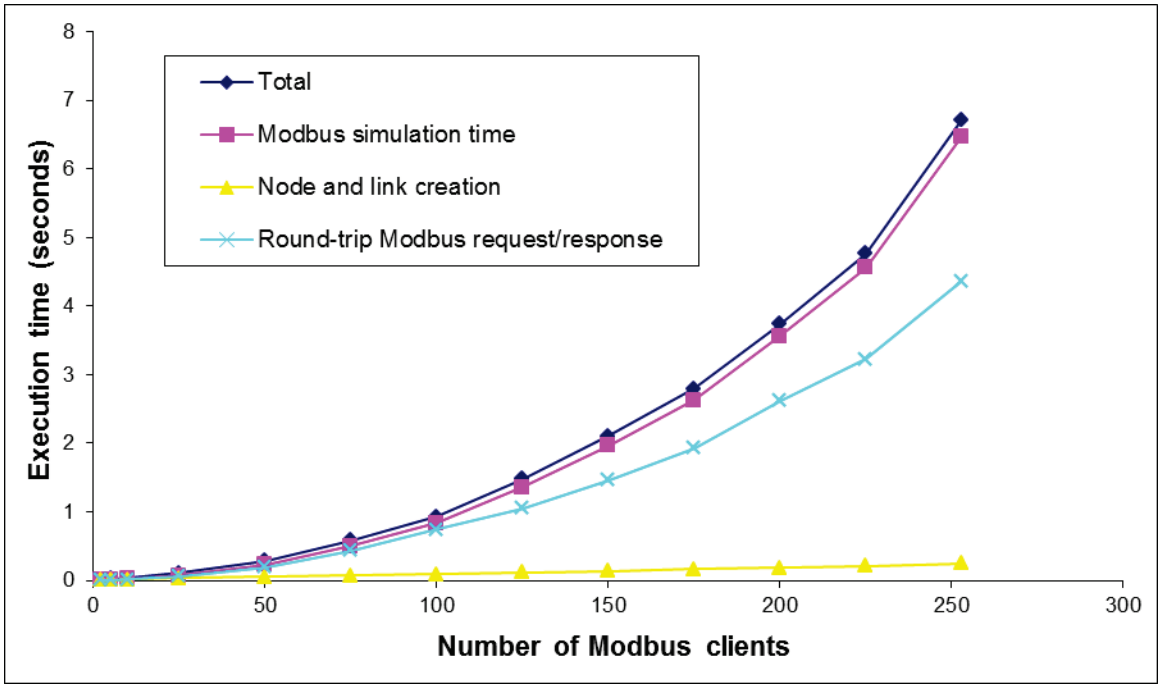

Figure 29. Modbus UDP bus topology: execution times.

The execution times versus the number of Modbus TCP clients are shown in Figure 30. The ratio of the averaged “round-trip request/response time” over the “total time” is approximately 12.8% for up to 50 nodes. The ratio is approximately 0.8% for the larger number of nodes. As shown in Figure 30, the slopes of the “Modbus simulation time” and also the “total time” are approximately 1.5 (for number of nodes larger than 50). However, the slope of the “node and link creation time” remains constant (approximately 0.0009) regardless of the number of nodes (2 through 253).

The execution time of Modbus TCP clients for “node and link creation” and “round-trip request/response” are shown in Figure 31.

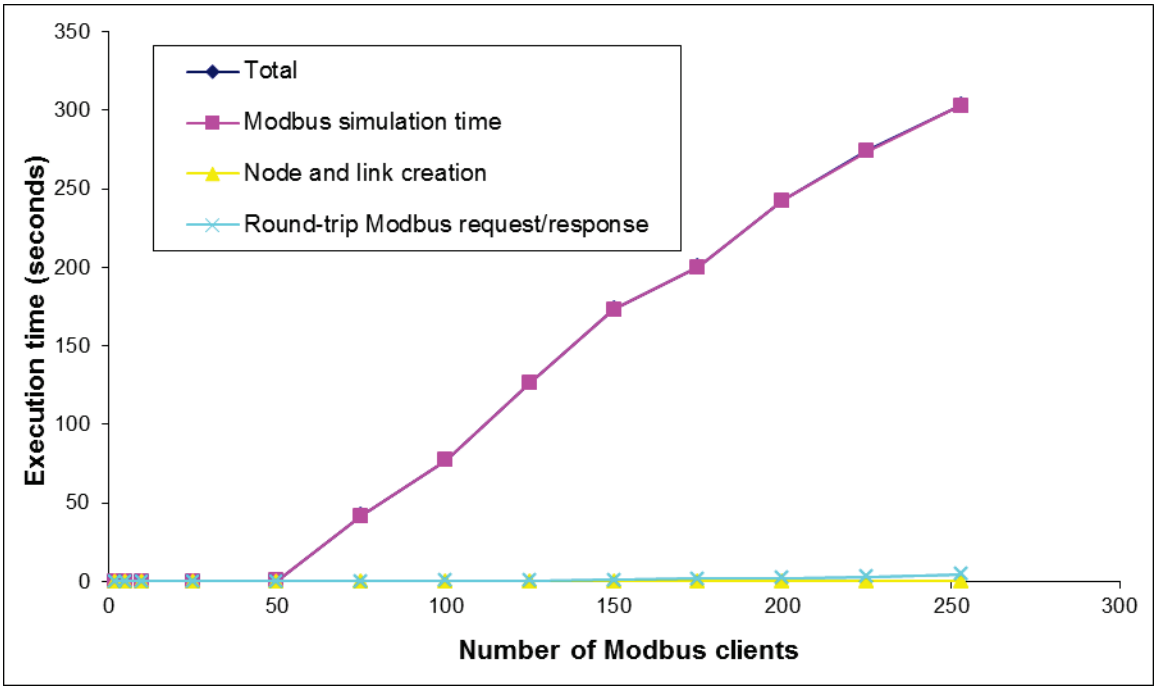

Figure 30. Modbus TCP bus topology: execution times.

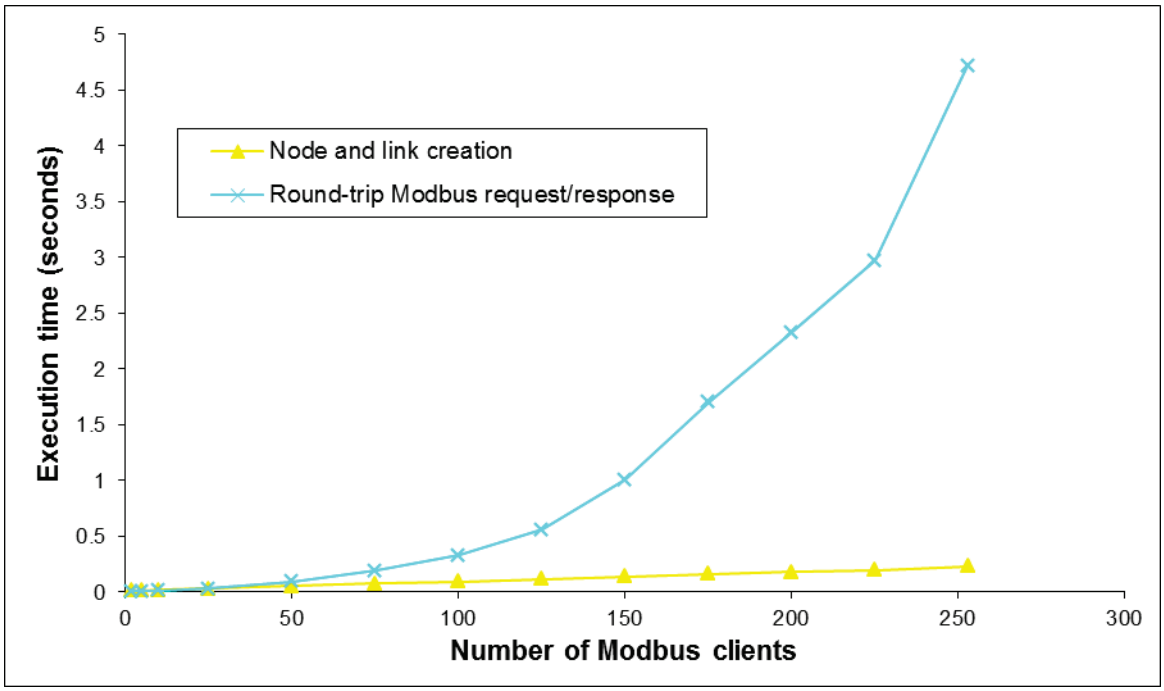

Figure 31. The execution times of "node and link creation" and "round-trip request/response" for Modbus TCP bus topology.

Scalability: Number of Modbus Clients in a Star (point-to-point) Topology

I considered "Modbus simulation time", "node and link creation time", "round-trip Modbus request/response time", and "total execution time" for a point-to-point (P2P) star topology model for both UDP and TCP connections. The execution times as a function of the number of Modbus UDP clients are shown in Figure 32. The ratio of averaged "round-trip request/response time" over the "total time" is approximately 14%.

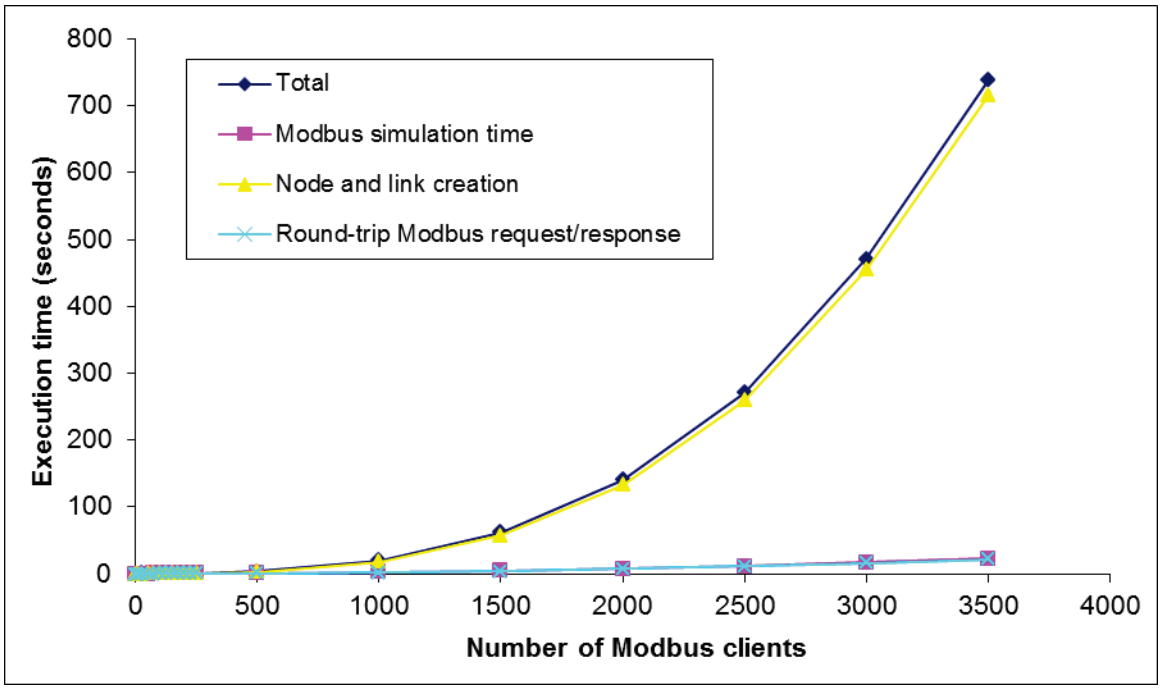

Figure 32. Modbus UDP star topology: execution times.

The execution times versus the number of Modbus TCP clients with star topology are shown in Figure 33. The ratio of averaged “round-trip request/response time” over the “total time” is around 14.5%.

As shown in Figure 32 and Figure 33, the “node and link creation time” for Modbus star topology is considerably higher compared to “node and link creation time” for Modbus bus topology, which is mentioned in Figure 29 and Figure 30.

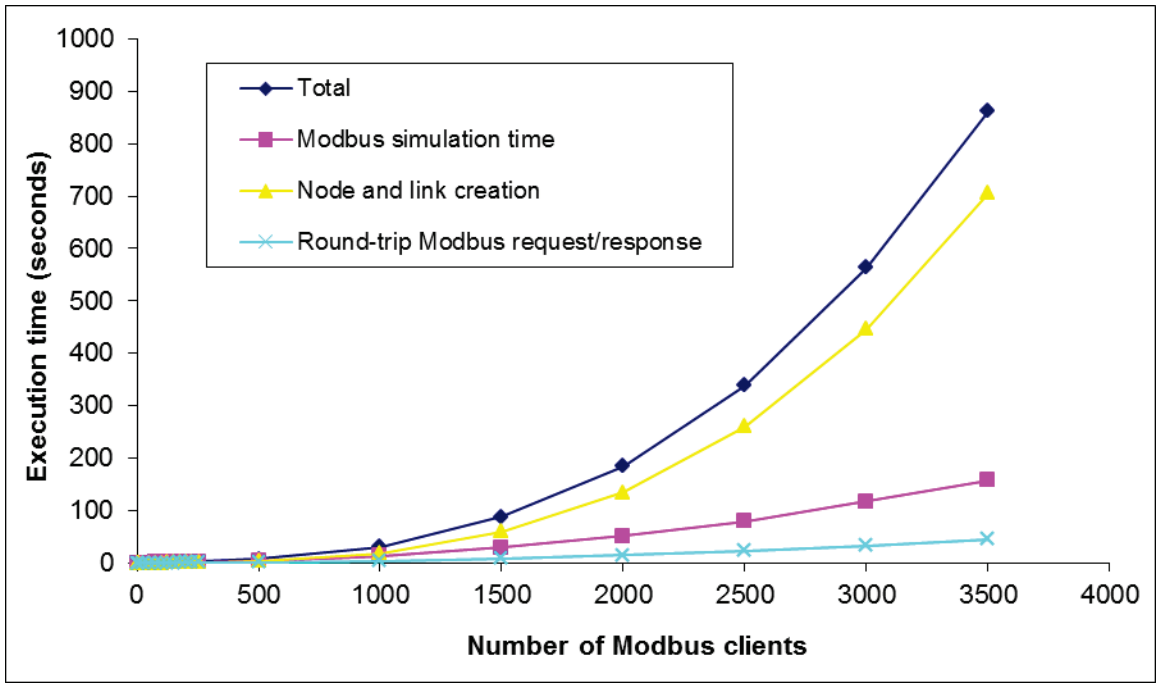

Figure 33. Modbus TCP star topology: execution times.

5.1.2. Runtime Overhead

I illustrated the overhead of ns-Modbus application by showing the ratio of “round-trip Modbus request/response” over “Modbus simulation time” and also the ratio of “round-trip Modbus request/response” over “total time”.

The “round-trip Modbus request/response time” over “Modbus simulation time” and “round-trip Modbus request/response time” over “total time” for Modbus UDP bus topology are shown in Figure 34. The ratios settle down when the number of nodes is higher than 125, which are around 75% and 65% for “round-trip” over “simulation time” and “round-trip” over “total time” respectively.

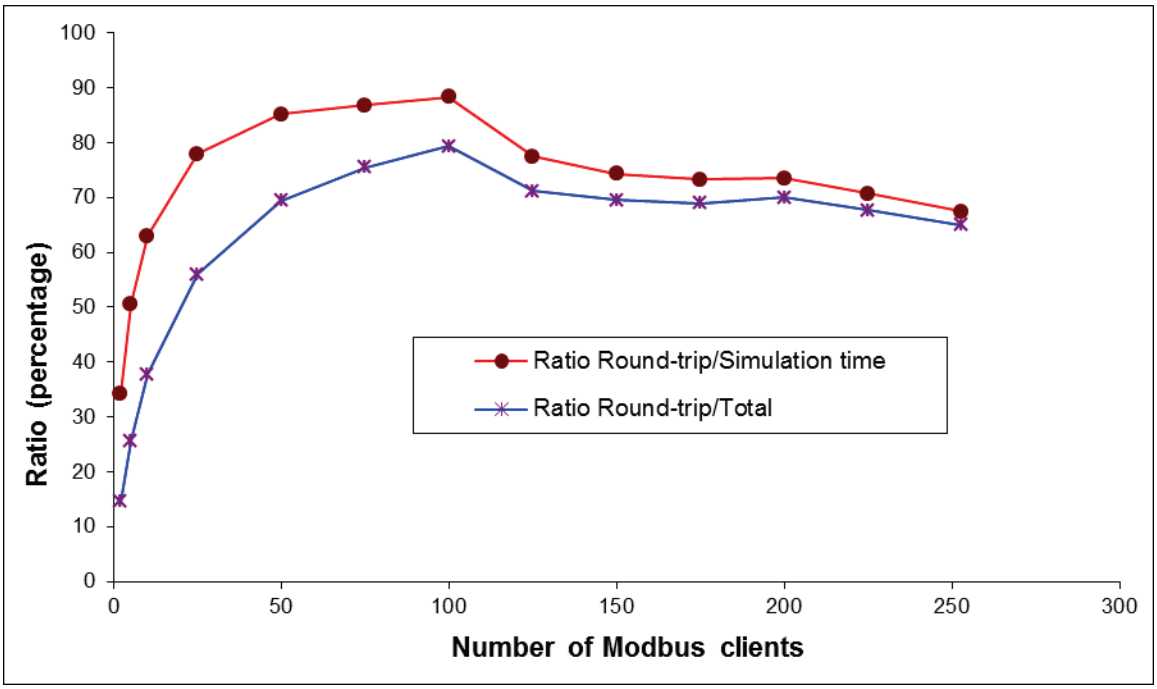

Figure 34. The ratios of the "round-trip Modbus request/response time" over the "Modbus simulation time" and "round-trip Modbus request/response time" over the "total time" for Modbus UDP bus topology.

The "round-trip Modbus request/response time" over the "Modbus simulation time" and the "round-trip Modbus request/response time" over the "total time" for Modbus TCP bus topology are shown in Figure 35. When the number of nodes is higher than 50, the "Modbus simulation time" rapidly increases with the average slope of 1.5 (56 degrees), which causes the "total time" to increase at the same rate. This phenomenon explains the large drop in both ratios when the number of nodes is higher than 50.

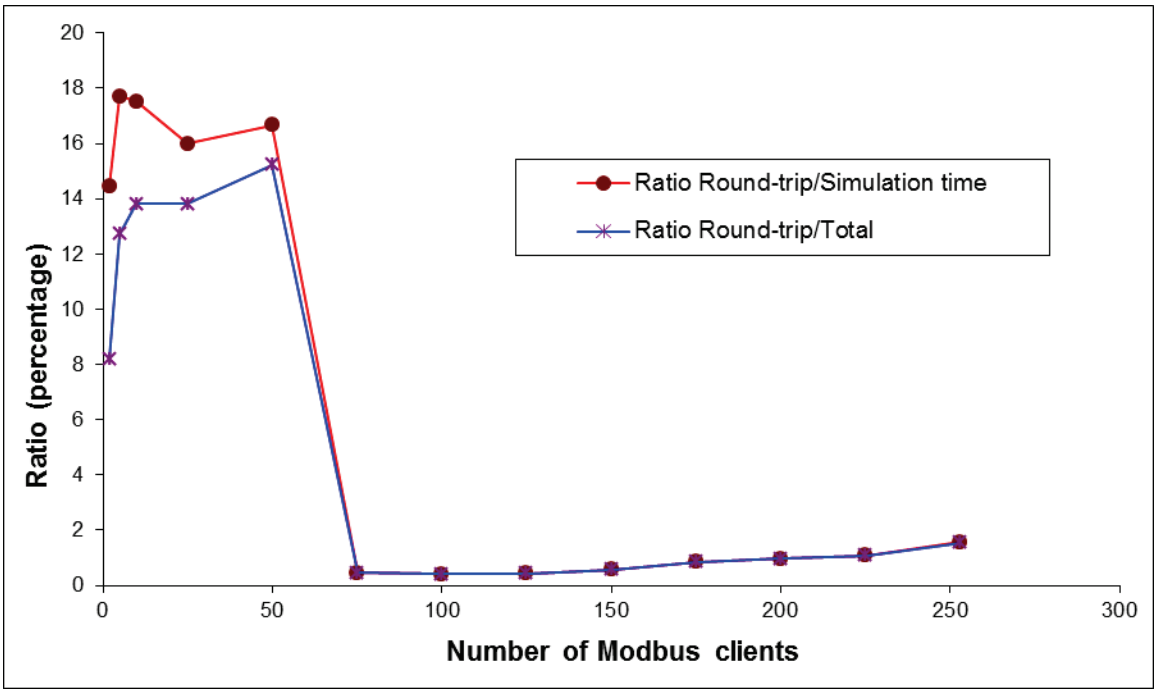

Figure 35. The ratios of the "round-trip Modbus request/response time" over the "Modbus simulation time" and "round-trip Modbus request/response time" over the "total time" for Modbus TCP bus topology.

The "round-trip Modbus request/response time" over the "Modbus simulation time" and the "round-trip Modbus request/response time" over the "total time" for Modbus UDP star topology are shown in Figure 36. The ratio of the "round-trip Modbus request/response time" over the "Modbus simulation time" reaches to 96% for the higher number of nodes. The "node and link creation time" for this model is much higher than bus topology and increases further for a larger number of nodes (see Sub-Section 5.1.1). Therefore, the ratio of the "round-trip Modbus request/response time" over the "total time" gradually decreases as the number of nodes increases and declines to 3%.

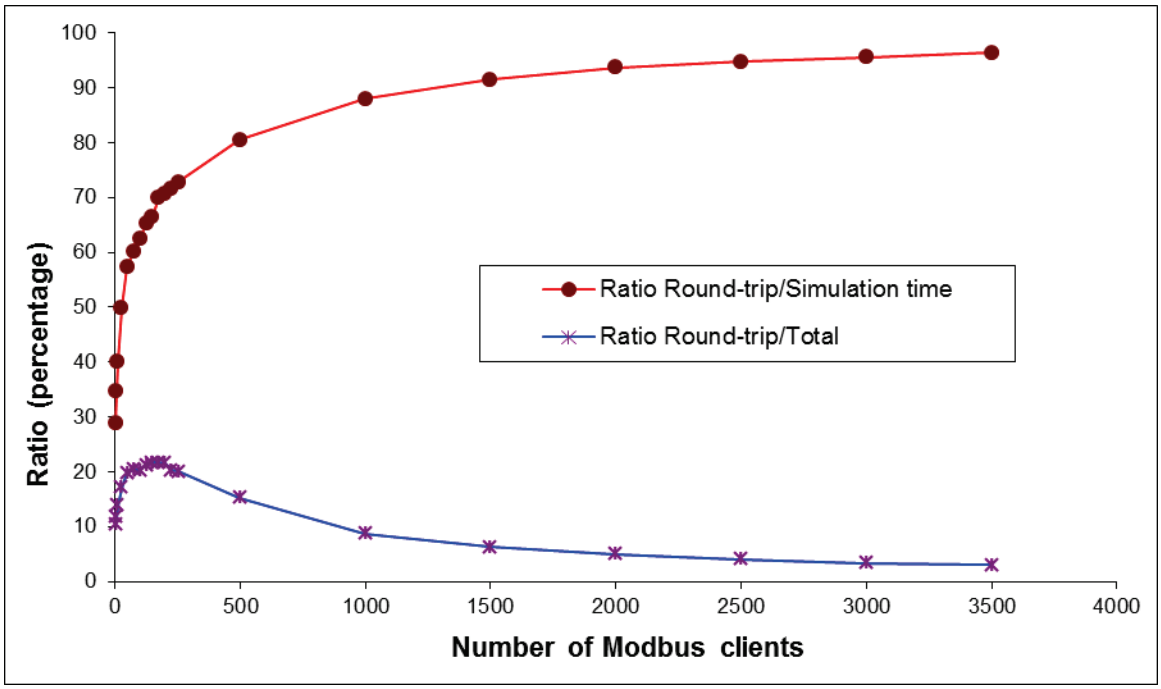

Figure 36. The ratios of the "round-trip Modbus request/response time" over the "Modbus simulation time" and "round-trip Modbus request/response time" over the "total time" for Modbus UDP star topology.

The "round-trip Modbus request/response time" over the "Modbus simulation time" and the "round-trip Modbus request/response time" over the "total time" for Modbus TCP star topology are shown in Figure 37. The trends of the ratios are similar to Modbus UDP star topology. The ratio of the "round-trip Modbus request/response time" over the "Modbus simulation time" reaches to 28% for the higher number of nodes. However, the ratio is lower (compared to Modbus UDP star topology ratio, which is 96%) and the simulation settles to this value more quickly when the number of nodes is approximately 25. The ratio of the "round-trip Modbus request/response time" over the "total time" comparably decreases to approximately 5.

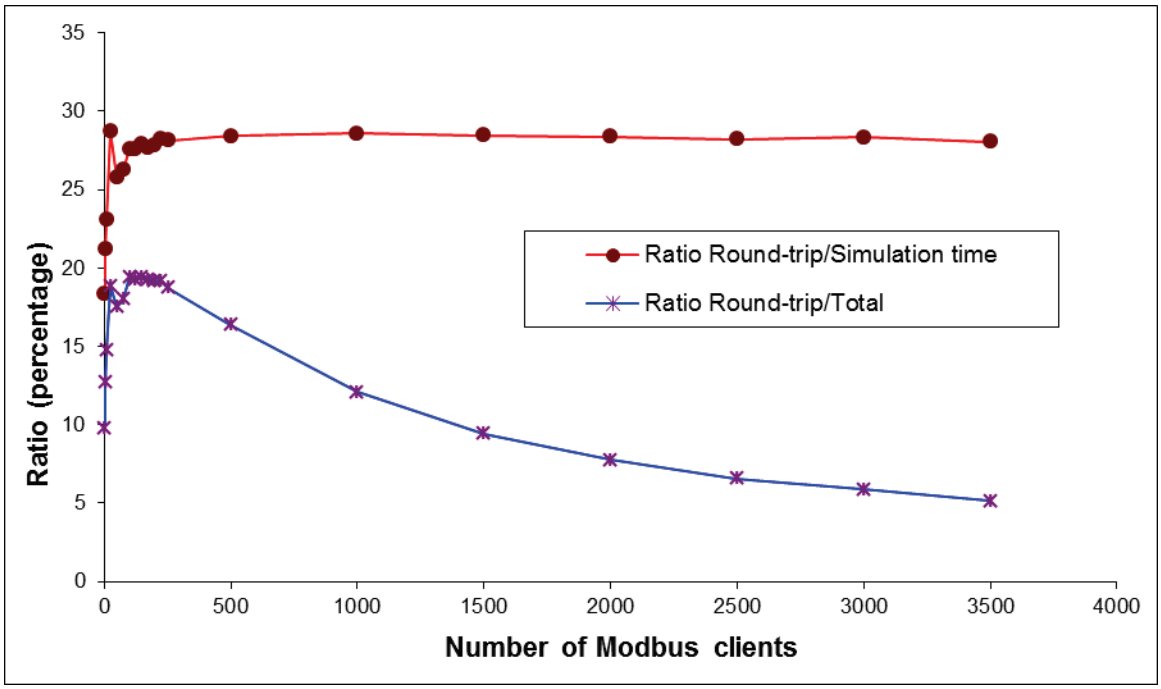

Figure 37. The ratios of the "round-trip Modbus request/response time" over the "Modbus simulation time" and "round-trip Modbus request/response time" over the "total time" for Modbus TCP star topology.

5.2. Simulation Robustness

No application crashes have been observed. The maximum number of nodes tested for star topology is 3,500. Higher number of nodes is also achievable. Since the execution time for the large number of nodes is relatively high (12 min for UDP and 14 min for TCP star topologies), only the maximum of 3,500 nodes was successfully simulated.

6. Future Work

In this project, I implemented Modbus in ns-3 distribution for UDP and TCP. I also analyzed the performance of the application for the provided script scenarios for bus and star topologies. Furthermore, I reviewed the overhead, scalability, and robustness of ns-Modbus. These are the opportunities for improvement:

I followed closely the ns-3 recommendation for coding style including function names, variable names, and formatting that make integration of ns-Modbus in the ns-3 distribution much easier. There is an opportunity to include ns-Modbus in the ns-3 distribution.

There is an option to upgrade ns-Modbus to the latest ns-3 build (ns-3.16) released in December 2012. The effort to upgrade to the latest release should be minimal, since I have already executed the upgrade during the implementation.

There is an opportunity to add an emulating feature to the simulation that may send or receive genuine traffic over the network. In this case, the system operates in hybrid mode.

There is a possibility to send and receive requests and responses in variable length. This is a relatively minimal effort that has already been implemented in another ns-3 project called ns3-BGP.

The simulation generates trace files with extension tr, which is in ASCII format. However, apart from the information about the packet header, the content of the Modbus request and response is shown in an unreadable ASCII representation of the binary data. It might be useful to convert the content to a human readable format.

The client Modbus timeout is not provisioned and implemented in the code. Hence, it might be beneficial to incorporate the concept of a timeout, especially when the traffic

congestion and packet loss is an issue and the client needs to be timed out. This might be even more useful when the emulating feature is added.

The bit manipulation memory blocks, which are represented in the variables *m_inputsStatus* and *m_coilsStatus,* are not optimized for memory footprint and they allocate byte arrays and assign a byte for each bit. This improvement is specifically useful when the Modbus code is incorporated in the firmware of a device with limited memory size.

Lastly, it might be prudent to employ Modbus conformance test specifications for verification of the application.

7. Conclusion

In this project, I described the architecture and implementation of a Modbus model for ns-3 network simulator. The project scope included understanding the Modbus protocol, learning the ns-3 network simulator, and implementing Modbus TCP and UDP in ns-3. The ns-Modbus enables the evaluation and simulation of Modbus protocol. The developed code may be used in industrial device implementations, which require Modbus UDP or Modbus TCP functionalities. In the future, emulation of the protocol may be added and ns-Modbus may be subjected to the Modbus conformance test [9]. The ns-Modbus implementation was evaluated through validation and verification tests. The implementation was tested for bus and star network topologies. The scalability tests show that ns-Modbus is able to simulate senarios with a large number of Modbus clients. The comparison of Modbus UDP and Modbus TCP for bus topology shows that Modbus UDP has much better performance for the number of nodes higher than 50.

References

[1] Modbus-IDA. (2006, December) Modbus application protocol specification V1.1b. [Online]. Available: http://www.modbus.com/docs/Modbus_Application_Protocol_V1_1b.pdf.

[2] D. Peng, H. Zhang, L. Yang, and H. Li, “Design and realization of Modbus protocol based on embedded Linux system,” in *Proc*. *ICESS* ’08, July 2008, pp. 275–280.

[3] ns-3. (2012, November) Network simulator release 3.15. [Online]. Available: http://www.nsnam.org.

[4] GNU Operating System. (2012, November) GNU General Public License. [Online]. Available: http://www.gnu.org/copyleft/gpl.html.

[5] ns-3. (2012, November) ns-3 Tutorial, Release ns-3.15 [Online]. Available: http://www.nsnam.org/docs/release/3.15/tutorial/ns-3-tutorial.pdf.

[6] Mercurial. (2012, November) mercurial [Online]. Available: http://www.mercurial.selenic.com.

[7] waf. (2012, November) waf [Online]. Available: http://code.google.com/p/waf.

[8] ns-3. (2012, April) ns-3 Tutorial [Online]. Available: http://www.nsnam.org/docs/release/3.12/tutorial/singlehtml/index.html.

[9] Modbus Organization (2009, December) Conformance test specification for Modbus TCP Version 3.0. [Online]. Available: http://www.modbus.org/docs/MBConformanceTestSpec_v3.0.pdf.

[10] ns-3. (2012, November) Download [Online]. Available: http://www.nsnam.org/ns-3-dev/download.

[11] ns-3 Network Simulator. (2012, December) ns-3 Manual, Release ns-3.15 [Online]. Available: http://www.nsnam.org/docs/release/3.15/manual/ns-3-manual.pdf.

[12] ns-3 Network Simulator. (2012, December) ns-3 Model Library, Release ns-3.15 [Online]. Available: http://www.nsnam.org/docs/release/3.15/models/ns-3-model-library.pdf.

[13] ns-3 A Discrete-Event Network Simulator. (2012, December) ns-3 API, Release ns-3.15 [Online]. Available: http://www.nsnam.org/docs/release/3.15/doxygen/index.html.

[14] ns-3. (2012, December) Coding Style [Online]. Available:
http://www.nsnam.org/developers/contributing-code/coding-style.

[15] ns-3. (2012, December) wiki [Online]. Available:
http://www.nsnam.org/wiki/index.php/Main_Page.

Appendices

Appendix A.

Traces – Message Logs

Ns-Modbus generates output on the screen. The output screen messages log the trace of code during executions. These messages can be shown in more detail by sending verbose argument to the program (see appendix H, How to Use) .

myFirstModbusUdp

```
reza@Dena:~/temp/ns-3-allinone/ns-3-dev$
export NS_LOG=FirstModbusUdpScriptExample=level_all
reza@Dena:~/temp/ns-3-allinone/ns-3-dev$ ./waf --run "scratch/myfirstModbusUdp"

Waf: Entering directory `/home/reza/temp/ns-3-allinone/ns-3-dev/build'
Waf: Leaving directory `/home/reza/temp/ns-3-allinone/ns-3-dev/build'
'build' finished successfully (1.695s)
Round-trip Modbus request/response (ms): 1.47796
Read Device Identification
readDeviceIdCode 1
objectId 0
Round-trip Modbus request/response (ms): 1.11079
Read Device Identification
readDeviceIdCode 4
objectId 0
Round-trip Modbus request/response (ms): 1.05381
Round-trip Modbus request/response (ms): 0.848055
Read Device Identification
readDeviceIdCode 1
objectId 0
Round-trip Modbus request/response (ms): 0.989199
Read Device Identification
readDeviceIdCode 4
objectId 0
Round-trip Modbus request/response (ms): 1.03807
Round-trip Modbus request/response (ms): 0.869989
Read Device Identification
readDeviceIdCode 1
objectId 0
Round-trip Modbus request/response (ms): 1.06788
Read Device Identification
readDeviceIdCode 4
objectId 0
Round-trip Modbus request/response (ms): 1.21999
```

Round-trip Modbus request/response (ms): 0.880003
Read Device Identification
readDeviceIdCode 1
objectId 0
Round-trip Modbus request/response (ms): 1.05405
Read Device Identification
readDeviceIdCode 4
objectId 0
Round-trip Modbus request/response (ms): 0.9799
Round-trip Modbus request/response (ms): 0.840902
Read Device Identification
readDeviceIdCode 1
objectId 0
Round-trip Modbus request/response (ms): 1.07694
Read Device Identification
readDeviceIdCode 4
objectId 0
Round-trip Modbus request/response (ms): 1.00493
Round-trip Modbus request/response (ms): 0.932932
Read Device Identification
readDeviceIdCode 1
objectId 0
Round-trip Modbus request/response (ms): 1.46389
Read Device Identification
readDeviceIdCode 4
objectId 0
Round-trip Modbus request/response (ms): 1.36685
Round-trip Modbus request/response (ms): 1.26219
Read Device Identification
readDeviceIdCode 1
objectId 0
Round-trip Modbus request/response (ms): 1.16801
Read Device Identification
readDeviceIdCode 4
objectId 0
Round-trip Modbus request/response (ms): 1.03283
Round-trip Modbus request/response (ms): 0.874043
Read Device Identification
readDeviceIdCode 1
objectId 0
Round-trip Modbus request/response (ms): 1.05405
Read Device Identification
readDeviceIdCode 4

objectId 0
Round-trip Modbus request/response (ms): 1.55091
Round-trip Modbus request/response (ms): 1.04904
Read Device Identification
readDeviceIdCode 1
objectId 0
Round-trip Modbus request/response (ms): 1.06692
Read Device Identification
readDeviceIdCode 4
objectId 0
Round-trip Modbus request/response (ms): 1.16587
Round-trip Modbus request/response (ms): 0.920773
Read Device Identification
readDeviceIdCode 1
objectId 0
Round-trip Modbus request/response (ms): 1.06788
Read Device Identification
readDeviceIdCode 4
objectId 0
Round-trip Modbus request/response (ms): 1.03807
Round-trip Modbus request/response (ms): 0.916958
Read Device Identification
readDeviceIdCode 1
objectId 0
Round-trip Modbus request/response (ms): 1.14608
Read Device Identification
readDeviceIdCode 4
objectId 0
Round-trip Modbus request/response (ms): 1.06621
Round-trip Modbus request/response (ms): 0.859976
Read Device Identification
readDeviceIdCode 1
objectId 0
Round-trip Modbus request/response (ms): 0.995874
Read Device Identification
readDeviceIdCode 4
objectId 0
Round-trip Modbus request/response (ms): 0.974894
Round-trip Modbus request/response (ms): 0.931978
Read Device Identification
readDeviceIdCode 1
objectId 0
Round-trip Modbus request/response (ms): 1.21117

Read Device Identification
readDeviceIdCode 4
objectId 0
Round-trip Modbus request/response (ms): 1.35589
Round-trip Modbus request/response (ms): 0.899076
Read Device Identification
readDeviceIdCode 1
objectId 0
Round-trip Modbus request/response (ms): 1.04499
Read Device Identification
readDeviceIdCode 4
objectId 0
Round-trip Modbus request/response (ms): 1.06692
Round-trip Modbus request/response (ms): 0.864983
Read Device Identification
readDeviceIdCode 1
objectId 0
Round-trip Modbus request/response (ms): 0.994921
Read Device Identification
readDeviceIdCode 4
objectId 0
Round-trip Modbus request/response (ms): 1.08099
Round-trip Modbus request/response (ms): 1.11103
Read Device Identification
readDeviceIdCode 1
objectId 0
Round-trip Modbus request/response (ms): 1.49512
Read Device Identification
readDeviceIdCode 4
objectId 0
Round-trip Modbus request/response (ms): 1.08695
Round-trip Modbus request/response (ms): 0.979185
Read Device Identification
readDeviceIdCode 1
objectId 0
Round-trip Modbus request/response (ms): 1.01519
Read Device Identification
readDeviceIdCode 4
objectId 0
Round-trip Modbus request/response (ms): 1.09601
Round-trip Modbus request/response (ms): 0.910997
Read Device Identification
readDeviceIdCode 1

objectId 0
Round-trip Modbus request/response (ms): 1.07789
Read Device Identification
readDeviceIdCode 4
objectId 0
Round-trip Modbus request/response (ms): 1.10984
Round-trip Modbus request/response (ms): 0.880003
Read Device Identification
readDeviceIdCode 1
objectId 0
Round-trip Modbus request/response (ms): 1.17302
Read Device Identification
readDeviceIdCode 4
objectId 0
Round-trip Modbus request/response (ms): 1.32513
Round-trip Modbus request/response (ms): 0.907898
Read Device Identification
readDeviceIdCode 1
objectId 0
Round-trip Modbus request/response (ms): 1.0221
reza@Dena:~/temp/ns-3-allinone/ns-3-dev$

myFirstModbusTcp

reza@Dena:~/temp/ns-3-allinone/ns-3-dev$
export NS_LOG=FirstModbusTcpScriptExample=level_all
reza@Dena:~/temp/ns-3-allinone/ns-3-dev$ ./waf --run "scratch/myfirstModbusTcp"
Waf: Entering directory `/home/reza/temp/ns-3-allinone/ns-3-dev/build'
Waf: Leaving directory `/home/reza/temp/ns-3-allinone/ns-3-dev/build'
'build' finished successfully (1.689s)
Read Device Identification
readDeviceIdCode 1
objectId 0
Read Device Identification
readDeviceIdCode 4
objectId 0
Read Device Identification
readDeviceIdCode 1
objectId 0
Read Device Identification
readDeviceIdCode 4
objectId 0
Read Device Identification
readDeviceIdCode 1

objectId 0
Read Device Identification
readDeviceIdCode 4
objectId 0
Read Device Identification
readDeviceIdCode 1
objectId 0
Read Device Identification
readDeviceIdCode 4
objectId 0
Read Device Identification
readDeviceIdCode 1
objectId 0
Read Device Identification
readDeviceIdCode 4
objectId 0
Read Device Identification
readDeviceIdCode 1
objectId 0
Read Device Identification
readDeviceIdCode 4
objectId 0
Read Device Identification
readDeviceIdCode 1
objectId 0
Read Device Identification
readDeviceIdCode 4
objectId 0
Read Device Identification
readDeviceIdCode 1
objectId 0
Read Device Identification
readDeviceIdCode 4
objectId 0
Read Device Identification
readDeviceIdCode 1
objectId 0
Read Device Identification
readDeviceIdCode 4
objectId 0
Read Device Identification
readDeviceIdCode 1
objectId 0

Read Device Identification
readDeviceIdCode 4
objectId 0
Read Device Identification
readDeviceIdCode 1
objectId 0
Read Device Identification
readDeviceIdCode 4
objectId 0
Read Device Identification
readDeviceIdCode 1
objectId 0
Read Device Identification
readDeviceIdCode 4
objectId 0
Read Device Identification
readDeviceIdCode 1
objectId 0
Read Device Identification
readDeviceIdCode 4
objectId 0
Read Device Identification
readDeviceIdCode 1
objectId 0
Read Device Identification
readDeviceIdCode 4
objectId 0
Read Device Identification
readDeviceIdCode 1
objectId 0
Read Device Identification
readDeviceIdCode 4
objectId 0
Read Device Identification
readDeviceIdCode 1
objectId 0
Read Device Identification
readDeviceIdCode 4
objectId 0
Read Device Identification
readDeviceIdCode 1
objectId 0
Read Device Identification

readDeviceIdCode 4
objectId 0
Read Device Identification
readDeviceIdCode 1
objectId 0
Read Device Identification
readDeviceIdCode 4
objectId 0
Read Device Identification
readDeviceIdCode 1
objectId 0
Read Device Identification
readDeviceIdCode 4
objectId 0
Read Device Identification
readDeviceIdCode 1
objectId 0
reza@Dena:~/temp/ns-3-allinone/ns-3-dev$

busModbusUdp

reza@Dena:~/temp/ns-3-allinone/ns-3-dev$ export NS_LOG=BusModbusUdp=level_all
reza@Dena:~/temp/ns-3-allinone/ns-3-dev$ ./waf --run "scratch/busModbusUdp"
Waf: Entering directory `/home/reza/temp/ns-3-allinone/ns-3-dev/build'
Waf: Leaving directory `/home/reza/temp/ns-3-allinone/ns-3-dev/build'
'build' finished successfully (1.722s)
Build bus topology.
Install internet stack on all nodes.
Assign IP Addresses.
Create applications.
Enable pcap tracing.
Run Simulation.
Round-trip Modbus request/response (ms): 12.2209
Round-trip Modbus request/response (ms): 15.214
Round-trip Modbus request/response (ms): 19.686
Read Device Identification
readDeviceIdCode 1
objectId 0
Round-trip Modbus request/response (ms): 2.86698
Read Device Identification
readDeviceIdCode 1
objectId 0
Round-trip Modbus request/response (ms): 3.92389
Read Device Identification

readDeviceIdCode 1
objectId 0
Round-trip Modbus request/response (ms): 6.30808
Read Device Identification
readDeviceIdCode 4
objectId 0
Round-trip Modbus request/response (ms): 2.71988
Read Device Identification
readDeviceIdCode 4
objectId 0
Round-trip Modbus request/response (ms): 3.65305
Read Device Identification
readDeviceIdCode 4
objectId 0
Round-trip Modbus request/response (ms): 5.92184
Done.
Total (ms): 63.6158
Modbus Simulation time (ms): 45.8539
Node and Link creation (ms): 17.7619
reza@Dena:~/temp/ns-3-allinone/ns-3-dev$

busModbusUdp (verbose)

reza@Dena:~/temp/ns-3-allinone/ns-3-dev$ export NS_LOG=BusModbusUdp=level_all
reza@Dena:~/temp/ns-3-allinone/ns-3-dev$ ./waf --run "scratch/busModbusUdp --verbose=1"
Waf: Entering directory `/home/reza/temp/ns-3-allinone/ns-3-dev/build'
Waf: Leaving directory `/home/reza/temp/ns-3-allinone/ns-3-dev/build'
'build' finished successfully (1.713s)
Build bus topology.
Install internet stack on all nodes.
Assign IP Addresses.
Create applications.
ModbusUdpClient::SetRequest ()
ModbusUdpClient::SetRequest ()
ModbusUdpClient::SetRequest ()
ModbusUdpClient::SetCount ()
ModbusUdpClient::SetRequest ()
ModbusUdpClient::SetRequest ()
ModbusUdpClient::SetRequest ()
ModbusUdpClient::SetCount ()
ModbusUdpClient::SetRequest ()
ModbusUdpClient::SetRequest ()
ModbusUdpClient::SetRequest ()
ModbusUdpClient::SetCount ()

Enable pcap tracing.

Run Simulation.

ModbusUdpClient::Send ()

TraceDelay TX: 253 bytes to 10.1.1.1 Uid: 0 Time: 2.1

ModbusUdpClient::Send ()

TraceDelay TX: 253 bytes to 10.1.1.1 Uid: 3 Time: 2.1

ModbusUdpClient::Send ()

TraceDelay TX: 253 bytes to 10.1.1.1 Uid: 6 Time: 2.1

Parsing MODBUS UDP function code

TraceDelay: RX 253 bytes from 10.1.1.2 Sequence Number: 0 Uid: 0 TXtime: +2100000000.0ns RXtime: +2100056797.0ns Delay: +56797.0ns

Sending MODBUS UDP response packet

Parsing MODBUS UDP function code

TraceDelay: RX 253 bytes from 10.1.1.4 Sequence Number: 0 Uid: 6 TXtime: +2100000000.0ns RXtime: +2100125797.0ns Delay: +125797.0ns

Sending MODBUS UDP response packet

ModbusUdpClient::HandleRead ()

Round-trip Modbus request/response (ms): 13.3791

Received 253 bytes from 10.1.1.1

Success - function code: 0x2b

pdu[0]: 0x2b

pdu[1]: 0xd

pdu[2]: 0x2b

pdu[3]: 0xe

pdu[4]: 0x1

pdu[5]: 0x0

pdu[6]: 0x2b

pdu[7]: 0xe

pdu[8]: 0x2

pdu[9]: 0x0

ModbusUdpClient::HandleRead ()

Round-trip Modbus request/response (ms): 17.0269

Received 253 bytes from 10.1.1.1

Success - function code: 0x2b

pdu[0]: 0x2b

pdu[1]: 0xd

pdu[2]: 0x2b

pdu[3]: 0xe

pdu[4]: 0x1

pdu[5]: 0x0

pdu[6]: 0x2b

pdu[7]: 0xe

pdu[8]: 0x2
pdu[9]: 0x0

Parsing MODBUS UDP function code
TraceDelay: RX 253 bytes from 10.1.1.3 Sequence Number: 0 Uid: 3 TXtime: +2100000000.0ns RXtime: +2100374797.0ns Delay: +374797.0ns
Sending MODBUS UDP response packet
ModbusUdpClient::HandleRead ()
Round-trip Modbus request/response (ms): 22.2449
Received 253 bytes from 10.1.1.1
Success - function code: 0x2b
pdu[0]: 0x2b
pdu[1]: 0xd
pdu[2]: 0x2b
pdu[3]: 0xe
pdu[4]: 0x1
pdu[5]: 0x0
pdu[6]: 0x2b
pdu[7]: 0xe
pdu[8]: 0x2
pdu[9]: 0x0

ModbusUdpClient::Send ()
TraceDelay TX: 253 bytes to 10.1.1.1 Uid: 27 Time: 2.3
ModbusUdpClient::Send ()
TraceDelay TX: 253 bytes to 10.1.1.1 Uid: 28 Time: 2.3
ModbusUdpClient::Send ()
TraceDelay TX: 253 bytes to 10.1.1.1 Uid: 29 Time: 2.3
Parsing MODBUS UDP function code
Read Device Identification
readDeviceIdCode 1
objectId 0
TraceDelay: RX 253 bytes from 10.1.1.2 Sequence Number: 1 Uid: 27 TXtime: +2300000000.0ns RXtime: +2300031439.0ns Delay: +31439.0ns
Sending MODBUS UDP response packet
ModbusUdpClient::HandleRead ()
Round-trip Modbus request/response (ms): 3.79491
Received 253 bytes from 10.1.1.1
Success - function code: 0x2b
pdu[0]: 0x2b
pdu[1]: 0xe
pdu[2]: 0x1
pdu[3]: 0x81

pdu[4]: 0x0
pdu[5]: 0x0
pdu[6]: 0x3
pdu[7]: 0x0
pdu[8]: 0x37
pdu[9]: 0x53

Parsing MODBUS UDP function code
Read Device Identification
readDeviceIdCode 1
objectId 0
TraceDelay: RX 253 bytes from 10.1.1.4 Sequence Number: 1 Uid: 29 TXtime: +2300000000.0ns RXtime: +2300110439.0ns Delay: +110439.0ns
Sending MODBUS UDP response packet
ModbusUdpClient::HandleRead ()
Round-trip Modbus request/response (ms): 5.14793
Received 253 bytes from 10.1.1.1
Success - function code: 0x2b
pdu[0]: 0x2b
pdu[1]: 0xe
pdu[2]: 0x1
pdu[3]: 0x81
pdu[4]: 0x0
pdu[5]: 0x0
pdu[6]: 0x3
pdu[7]: 0x0
pdu[8]: 0x37
pdu[9]: 0x53

Parsing MODBUS UDP function code
Read Device Identification
readDeviceIdCode 1
objectId 0
TraceDelay: RX 253 bytes from 10.1.1.3 Sequence Number: 1 Uid: 28 TXtime: +2300000000.0ns RXtime: +2300642439.0ns Delay: +642439.0ns
Sending MODBUS UDP response packet
ModbusUdpClient::HandleRead ()
Round-trip Modbus request/response (ms): 8.36611
Received 253 bytes from 10.1.1.1
Success - function code: 0x2b
pdu[0]: 0x2b
pdu[1]: 0xe
pdu[2]: 0x1

pdu[3]: 0x81
pdu[4]: 0x0
pdu[5]: 0x0
pdu[6]: 0x3
pdu[7]: 0x0
pdu[8]: 0x37
pdu[9]: 0x53

ModbusUdpClient::Send ()
TraceDelay TX: 253 bytes to 10.1.1.1 Uid: 30 Time: 2.4
ModbusUdpClient::Send ()
TraceDelay TX: 253 bytes to 10.1.1.1 Uid: 31 Time: 2.4
ModbusUdpClient::Send ()
TraceDelay TX: 253 bytes to 10.1.1.1 Uid: 32 Time: 2.4
Parsing MODBUS UDP function code
Read Device Identification
readDeviceIdCode 4
objectId 0
TraceDelay: RX 253 bytes from 10.1.1.2 Sequence Number: 2 Uid: 30 TXtime: +2400000000.0ns RXtime: +2400031439.0ns Delay: +31439.0ns
Sending MODBUS UDP response packet
ModbusUdpClient::HandleRead ()
Round-trip Modbus request/response (ms): 3.4008
Received 253 bytes from 10.1.1.1
Success - function code: 0x2b
pdu[0]: 0x2b
pdu[1]: 0xe
pdu[2]: 0x4
pdu[3]: 0x81
pdu[4]: 0x0
pdu[5]: 0x0
pdu[6]: 0x1
pdu[7]: 0x0
pdu[8]: 0x37
pdu[9]: 0x53

Parsing MODBUS UDP function code
Read Device Identification
readDeviceIdCode 4
objectId 0
TraceDelay: RX 253 bytes from 10.1.1.4 Sequence Number: 2 Uid: 32 TXtime: +2400000000.0ns RXtime: +2400155439.0ns Delay: +155439.0ns
Sending MODBUS UDP response packet

ModbusUdpClient::HandleRead ()
Round-trip Modbus request/response (ms): 5.10097
Received 253 bytes from 10.1.1.1
Success - function code: 0x2b
pdu[0]: 0x2b
pdu[1]: 0xe
pdu[2]: 0x4
pdu[3]: 0x81
pdu[4]: 0x0
pdu[5]: 0x0
pdu[6]: 0x1
pdu[7]: 0x0
pdu[8]: 0x37
pdu[9]: 0x53

Parsing MODBUS UDP function code
Read Device Identification
readDeviceIdCode 4
objectId 0
TraceDelay: RX 253 bytes from 10.1.1.3 Sequence Number: 2 Uid: 31 TXtime: +2400000000.0ns RXtime: +2400540439.0ns Delay: +540439.0ns
Sending MODBUS UDP response packet
ModbusUdpClient::HandleRead ()
Round-trip Modbus request/response (ms): 8.52704
Received 253 bytes from 10.1.1.1
Success - function code: 0x2b
pdu[0]: 0x2b
pdu[1]: 0xe
pdu[2]: 0x4
pdu[3]: 0x81
pdu[4]: 0x0
pdu[5]: 0x0
pdu[6]: 0x1
pdu[7]: 0x0
pdu[8]: 0x37
pdu[9]: 0x53

Done.
Total (ms): 72.6211
Modbus Simulation time (ms): 54.646
Node and Link creation (ms): 17.9751
reza@Dena:~/temp/ns-3-allinone/ns-3-dev$

busModbusTcp

```
reza@Dena:~/temp/ns-3-allinone/ns-3-dev$ export NS_LOG=BusModbusTcp=level_all
reza@Dena:~/temp/ns-3-allinone/ns-3-dev$ ./waf --run "scratch/busModbusTcp"
Waf: Entering directory `/home/reza/temp/ns-3-allinone/ns-3-dev/build'
[ 635/1362] cxx: scratch/busModbusTcp.cc -> build/scratch/busModbusTcp.cc.6.o
[1348/1362] cxxprogram: build/scratch/busModbusTcp.cc.6.o -> build/scratch/busModbusTcp
Waf: Leaving directory `/home/reza/temp/ns-3-allinone/ns-3-dev/build'
'build' finished successfully (4.482s)
Build bus topology.
Install internet stack on all nodes.
Assign IP Addresses.
Create applications.
Enable pcap tracing.
Run Simulation.
Read Device Identification
readDeviceIdCode 1
objectId 0
Read Device Identification
readDeviceIdCode 1
objectId 0
Read Device Identification
readDeviceIdCode 1
objectId 0
Read Device Identification
readDeviceIdCode 4
objectId 0
Read Device Identification
readDeviceIdCode 4
objectId 0
Read Device Identification
readDeviceIdCode 4
objectId 0
Done.
Total (ms): 188.925
Modbus Simulation time (ms): 171.287
Node and Link creation (ms): 17.6382
reza@Dena:~/temp/ns-3-allinone/ns-3-dev$
```

busModbusTcp (verbose)

```
reza@Dena:~/temp/ns-3-allinone/ns-3-dev$ export NS_LOG=BusModbusTcp=level_all
reza@Dena:~/temp/ns-3-allinone/ns-3-dev$ ./waf --run "scratch/busModbusTcp --verbose=1"
Waf: Entering directory `/home/reza/temp/ns-3-allinone/ns-3-dev/build'
```

```
Waf: Leaving directory `/home/reza/temp/ns-3-allinone/ns-3-dev/build'
'build' finished successfully (1.690s)
Build bus topology.
Install internet stack on all nodes.
Assign IP Addresses.
Create applications.
ModbusTcpClient::SetRequest ()
ModbusTcpClient::SetRequest ()
ModbusTcpClient::SetRequest ()
ModbusTcpClient::SetCount ()
ModbusTcpClient::SetRequest ()
ModbusTcpClient::SetRequest ()
ModbusTcpClient::SetRequest ()
ModbusTcpClient::SetCount ()
ModbusTcpClient::SetRequest ()
ModbusTcpClient::SetRequest ()
ModbusTcpClient::SetRequest ()
ModbusTcpClient::SetCount ()
Enable pcap tracing.
Run Simulation.
ModbusTcpClient::Send ()
TraceDelay TX: 253 bytes to 10.1.1.1 Uid: 42 Time: 2.01
ModbusTcpClient::Send ()
TraceDelay TX: 253 bytes to 10.1.1.1 Uid: 43 Time: 2.01
ModbusTcpClient::Send ()
TraceDelay TX: 253 bytes to 10.1.1.1 Uid: 44 Time: 2.01
Parsing MODBUS TCP function code
TraceDelay: RX 253 bytes from 10.1.1.2 Sequence Number: 0 Uid: 42 TXtime: +2010000000.0ns
RXtime: +2010032399.0ns Delay: +32399.0ns
Sending MODBUS TCP response packet
ModbusTcpClient::HandleRead ()
Received 253 bytes from 10.1.1.1
Success - function code: 0x2b
pdu[0]: 0x2b
pdu[1]: 0xd
pdu[2]: 0x2b
pdu[3]: 0xe
pdu[4]: 0x1
pdu[5]: 0x0
pdu[6]: 0x2b
pdu[7]: 0xe
pdu[8]: 0x2
pdu[9]: 0x0
```

Parsing MODBUS TCP function code

TraceDelay: RX 253 bytes from 10.1.1.4 Sequence Number: 0 Uid: 44 TXtime: +2010000000.0ns RXtime: +2010130399.0ns Delay: +130399.0ns

Sending MODBUS TCP response packet

ModbusTcpClient::HandleRead ()

Received 253 bytes from 10.1.1.1

Success - function code: 0x2b

pdu[0]: 0x2b

pdu[1]: 0xd

pdu[2]: 0x2b

pdu[3]: 0xe

pdu[4]: 0x1

pdu[5]: 0x0

pdu[6]: 0x2b

pdu[7]: 0xe

pdu[8]: 0x2

pdu[9]: 0x0

Parsing MODBUS TCP function code

TraceDelay: RX 253 bytes from 10.1.1.3 Sequence Number: 0 Uid: 43 TXtime: +2010000000.0ns RXtime: +2010255399.0ns Delay: +255399.0ns

Sending MODBUS TCP response packet

ModbusTcpClient::HandleRead ()

Received 253 bytes from 10.1.1.1

Success - function code: 0x2b

pdu[0]: 0x2b

pdu[1]: 0xd

pdu[2]: 0x2b

pdu[3]: 0xe

pdu[4]: 0x1

pdu[5]: 0x0

pdu[6]: 0x2b

pdu[7]: 0xe

pdu[8]: 0x2

pdu[9]: 0x0

ModbusTcpClient::Send ()

TraceDelay TX: 253 bytes to 10.1.1.1 Uid: 66 Time: 2.21

ModbusTcpClient::Send ()

TraceDelay TX: 253 bytes to 10.1.1.1 Uid: 67 Time: 2.21

ModbusTcpClient::Send ()

TraceDelay TX: 253 bytes to 10.1.1.1 Uid: 68 Time: 2.21

Parsing MODBUS TCP function code
Read Device Identification
readDeviceIdCode 1
objectId 0
TraceDelay: RX 253 bytes from 10.1.1.2 Sequence Number: 1 Uid: 66 TXtime: +2210000000.0ns RXtime: +2210032399.0ns Delay: +32399.0ns
Sending MODBUS TCP response packet
ModbusTcpClient::HandleRead ()
Received 253 bytes from 10.1.1.1
Success - function code: 0x2b
pdu[0]: 0x2b
pdu[1]: 0xe
pdu[2]: 0x1
pdu[3]: 0x81
pdu[4]: 0x0
pdu[5]: 0x0
pdu[6]: 0x3
pdu[7]: 0x0
pdu[8]: 0x37
pdu[9]: 0x53

Parsing MODBUS TCP function code
Read Device Identification
readDeviceIdCode 1
objectId 0
TraceDelay: RX 253 bytes from 10.1.1.4 Sequence Number: 1 Uid: 68 TXtime: +2210000000.0ns RXtime: +2210124399.0ns Delay: +124399.0ns
Sending MODBUS TCP response packet
ModbusTcpClient::HandleRead ()
Received 253 bytes from 10.1.1.1
Success - function code: 0x2b
pdu[0]: 0x2b
pdu[1]: 0xe
pdu[2]: 0x1
pdu[3]: 0x81
pdu[4]: 0x0
pdu[5]: 0x0
pdu[6]: 0x3
pdu[7]: 0x0
pdu[8]: 0x37
pdu[9]: 0x53

Parsing MODBUS TCP function code

Read Device Identification
readDeviceIdCode 1
objectId 0
TraceDelay: RX 253 bytes from 10.1.1.3 Sequence Number: 1 Uid: 67 TXtime: +2210000000.0ns RXtime: +2210398399.0ns Delay: +398399.0ns
Sending MODBUS TCP response packet
ModbusTcpClient::HandleRead ()
Received 253 bytes from 10.1.1.1
Success - function code: 0x2b
pdu[0]: 0x2b
pdu[1]: 0xe
pdu[2]: 0x1
pdu[3]: 0x81
pdu[4]: 0x0
pdu[5]: 0x0
pdu[6]: 0x3
pdu[7]: 0x0
pdu[8]: 0x37
pdu[9]: 0x53

ModbusTcpClient::Send ()
TraceDelay TX: 253 bytes to 10.1.1.1 Uid: 90 Time: 2.31
ModbusTcpClient::Send ()
TraceDelay TX: 253 bytes to 10.1.1.1 Uid: 91 Time: 2.31
ModbusTcpClient::Send ()
TraceDelay TX: 253 bytes to 10.1.1.1 Uid: 92 Time: 2.31
Parsing MODBUS TCP function code
Read Device Identification
readDeviceIdCode 4
objectId 0
TraceDelay: RX 253 bytes from 10.1.1.2 Sequence Number: 2 Uid: 90 TXtime: +2310000000.0ns RXtime: +2310032399.0ns Delay: +32399.0ns
Sending MODBUS TCP response packet
Parsing MODBUS TCP function code
Read Device Identification
readDeviceIdCode 4
objectId 0
TraceDelay: RX 253 bytes from 10.1.1.3 Sequence Number: 2 Uid: 91 TXtime: +2310000000.0ns RXtime: +2310087399.0ns Delay: +87399.0ns
Sending MODBUS TCP response packet
ModbusTcpClient::HandleRead ()
Received 253 bytes from 10.1.1.1
Success - function code: 0x2b
pdu[0]: 0x2b

pdu[1]: 0xe
pdu[2]: 0x4
pdu[3]: 0x81
pdu[4]: 0x0
pdu[5]: 0x0
pdu[6]: 0x1
pdu[7]: 0x0
pdu[8]: 0x37
pdu[9]: 0x53

ModbusTcpClient::HandleRead ()
Received 253 bytes from 10.1.1.1
Success - function code: 0x2b
pdu[0]: 0x2b
pdu[1]: 0xe
pdu[2]: 0x4
pdu[3]: 0x81
pdu[4]: 0x0
pdu[5]: 0x0
pdu[6]: 0x1
pdu[7]: 0x0
pdu[8]: 0x37
pdu[9]: 0x53

Parsing MODBUS TCP function code
Read Device Identification
readDeviceIdCode 4
objectId 0
TraceDelay: RX 253 bytes from 10.1.1.4 Sequence Number: 2 Uid: 92 TXtime: +2310000000.0ns RXtime: +2310259399.0ns Delay: +259399.0ns
Sending MODBUS TCP response packet
ModbusTcpClient::HandleRead ()
Received 253 bytes from 10.1.1.1
Success - function code: 0x2b
pdu[0]: 0x2b
pdu[1]: 0xe
pdu[2]: 0x4
pdu[3]: 0x81
pdu[4]: 0x0
pdu[5]: 0x0
pdu[6]: 0x1
pdu[7]: 0x0
pdu[8]: 0x37

pdu[9]: 0x53

ModbusTcpServer, peerClose
ModbusTcpServer, peerClose
ModbusTcpServer, peerClose
ModbusTcpServer, peerClose
Done.
Total (ms): 198.328
Modbus Simulation time (ms): 180.393
Node and Link creation (ms): 17.935
reza@Dena:~/temp/ns-3-allinone/ns-3-dev$

busModbusTcpScenario1

reza@Dena:~/temp/ns-3-allinone/ns-3-dev$ export NS_LOG=BusModbusTcpScenario1=level_all
reza@Dena:~/temp/ns-3-allinone/ns-3-dev$ ./waf --run "scratch/busModbusTcpScenario1"
Waf: Entering directory `/home/reza/temp/ns-3-allinone/ns-3-dev/build'
Waf: Leaving directory `/home/reza/temp/ns-3-allinone/ns-3-dev/build'
'build' finished successfully (1.709s)
Build bus topology.
Install internet stack on all nodes.
Assign IP Addresses.
Create applications.
Enable pcap tracing.
Run Simulation.
Encapsulated Interface Transport
meiType: 0xd
CANopen General Reference Request and Response PDU
Read Coils
startAdr: 19
quantityOfCoils: 19
Done.
Total (ms): 110.989
Modbus Simulation time (ms): 97.6889
Node and Link creation (ms): 13.3002
reza@Dena:~/temp/ns-3-allinone/ns-3-dev$

busModbusTcpScenario1 (verbose)

reza@Dena:~/temp/ns-3-allinone/ns-3-dev$ export NS_LOG=BusModbusTcpScenario1=level_all
reza@Dena:~/temp/ns-3-allinone/ns-3-dev$ ./waf --run "scratch/busModbusTcpScenario1 --verbose=1"
Waf: Entering directory `/home/reza/temp/ns-3-allinone/ns-3-dev/build'
Waf: Leaving directory `/home/reza/temp/ns-3-allinone/ns-3-dev/build'
'build' finished successfully (1.694s)

Build bus topology.
Install internet stack on all nodes.
Assign IP Addresses.
Create applications.
ModbusTcpClient::SetRequest ()
ModbusTcpClient::SetRequest ()
ModbusTcpClient::SetCount ()
Enable pcap tracing.
Run Simulation.
ModbusTcpClient::Send ()
TraceDelay TX: 253 bytes to 10.1.1.1 Uid: 14 Time: 3
Parsing MODBUS TCP function code
Encapsulated Interface Transport
meiType: 0xd
CANopen General Reference Request and Response PDU
TraceDelay: RX 253 bytes from 10.1.1.2 Sequence Number: 0 Uid: 14 TXtime: +3000000000.0ns RXtime: +3000032399.0ns Delay: +32399.0ns
Sending MODBUS TCP response packet
ModbusTcpClient::HandleRead ()
Received 253 bytes from 10.1.1.1
Success - function code: 0x2b
pdu[0]: 0x2b
pdu[1]: 0xd
pdu[2]: 0x2b
pdu[3]: 0xe
pdu[4]: 0x1
pdu[5]: 0x0
pdu[6]: 0x2b
pdu[7]: 0xe
pdu[8]: 0x2
pdu[9]: 0x0

ModbusTcpClient::Send ()
TraceDelay TX: 253 bytes to 10.1.1.1 Uid: 22 Time: 4
Parsing MODBUS TCP function code
Read Coils
startAdr: 19
quantityOfCoils: 19
TraceDelay: RX 253 bytes from 10.1.1.2 Sequence Number: 1 Uid: 22 TXtime: +4000000000.0ns RXtime: +4000032399.0ns Delay: +32399.0ns
Sending MODBUS TCP response packet
ModbusTcpClient::HandleRead ()
Received 253 bytes from 10.1.1.1

Success - function code: 0x1
pdu[0]: 0x1
pdu[1]: 0x3
pdu[2]: 0xba
pdu[3]: 0x1
pdu[4]: 0x28
pdu[5]: 0x34
pdu[6]: 0x2
pdu[7]: 0x1
pdu[8]: 0x1
pdu[9]: 0x0

ModbusTcpServer, peerClose
ModbusTcpServer, peerClose
Done.
Total (ms): 111.01
Modbus Simulation time (ms): 97.7521
Node and Link creation (ms): 13.258
reza@Dena:~/temp/ns-3-allinone/ns-3-dev$

starModbusUdp

reza@Dena:~/temp/ns-3-allinone/ns-3-dev$ export NS_LOG=StarModbusUdp=level_all
reza@Dena:~/temp/ns-3-allinone/ns-3-dev$ ./waf --run "scratch/starModbusUdp"
Waf: Entering directory `/home/reza/temp/ns-3-allinone/ns-3-dev/build'
[1250/1362] cxx: scratch/starModbusUdp.cc -> build/scratch/starModbusUdp.cc.7.o
[1351/1362] cxxprogram: build/scratch/starModbusUdp.cc.7.o -> build/scratch/starModbusUdp
Waf: Leaving directory `/home/reza/temp/ns-3-allinone/ns-3-dev/build'
'build' finished successfully (4.756s)
Build star topology.
Install internet stack on all nodes.
Assign IP Addresses.
Create applications.
Enable static global routing.
Enable pcap tracing.
Run Simulation.
Round-trip Modbus request/response (ms): 11.071
Round-trip Modbus request/response (ms): 10.7539
Round-trip Modbus request/response (ms): 10.396
Round-trip Modbus request/response (ms): 10.036
Round-trip Modbus request/response (ms): 9.64308
Round-trip Modbus request/response (ms): 9.269
Round-trip Modbus request/response (ms): 8.89897
Round-trip Modbus request/response (ms): 8.17585

Read Device Identification
readDeviceIdCode 1
objectId 0
Read Device Identification
readDeviceIdCode 1
objectId 0
Read Device Identification
readDeviceIdCode 1
objectId 0
Read Device Identification
readDeviceIdCode 1
objectId 0
Read Device Identification
readDeviceIdCode 1
objectId 0
Read Device Identification
readDeviceIdCode 1
objectId 0
Read Device Identification
readDeviceIdCode 1
objectId 0
Read Device Identification
readDeviceIdCode 1
objectId 0
Round-trip Modbus request/response (ms): 11.8508
Round-trip Modbus request/response (ms): 11.538
Round-trip Modbus request/response (ms): 11.1959
Round-trip Modbus request/response (ms): 10.818
Round-trip Modbus request/response (ms): 10.4351
Round-trip Modbus request/response (ms): 10.041
Round-trip Modbus request/response (ms): 9.67097
Round-trip Modbus request/response (ms): 9.23204
Read Device Identification
readDeviceIdCode 4
objectId 0
Read Device Identification
readDeviceIdCode 4
objectId 0
Read Device Identification
readDeviceIdCode 4
objectId 0
Read Device Identification
readDeviceIdCode 4

objectId 0
Read Device Identification
readDeviceIdCode 4
objectId 0
Read Device Identification
readDeviceIdCode 4
objectId 0
Read Device Identification
readDeviceIdCode 4
objectId 0
Read Device Identification
readDeviceIdCode 4
objectId 0
Round-trip Modbus request/response (ms): 11.8389
Round-trip Modbus request/response (ms): 11.497
Round-trip Modbus request/response (ms): 11.143
Round-trip Modbus request/response (ms): 10.793
Round-trip Modbus request/response (ms): 10.442
Round-trip Modbus request/response (ms): 10.093
Round-trip Modbus request/response (ms): 9.75418
Round-trip Modbus request/response (ms): 9.43112
Done.
Total (ms): 90.523
Modbus Simulation time (ms): 48.275
Node and Link creation (ms): 42.248
reza@Dena:~/temp/ns-3-allinone/ns-3-dev$

starModbusUdp (verbose)

reza@Dena:~/temp/ns-3-allinone/ns-3-dev$ export NS_LOG=StarModbusUdp=level_all
reza@Dena:~/temp/ns-3-allinone/ns-3-dev$ ./waf --run "scratch/starModbusUdp"
Waf: Entering directory `/home/reza/temp/ns-3-allinone/ns-3-dev/build'
reza@Dena:~/temp/ns-3-allinone/ns-3-dev$ ./waf --run "scratch/starModbusUdp --verbose=1"
Waf: Entering directory `/home/reza/temp/ns-3-allinone/ns-3-dev/build'
Waf: Leaving directory `/home/reza/temp/ns-3-allinone/ns-3-dev/build'
'build' finished successfully (1.696s)
Build star topology.
Install internet stack on all nodes.
Assign IP Addresses.
Create applications.
ModbusUdpClient::SetRequest ()
ModbusUdpClient::SetRequest ()
ModbusUdpClient::SetRequest ()
ModbusUdpClient::SetCount ()

```
ModbusUdpClient::SetRequest ()
ModbusUdpClient::SetRequest ()
ModbusUdpClient::SetRequest ()
ModbusUdpClient::SetCount ()
ModbusUdpClient::SetRequest ()
ModbusUdpClient::SetRequest ()
ModbusUdpClient::SetRequest ()
ModbusUdpClient::SetCount ()
ModbusUdpClient::SetRequest ()
ModbusUdpClient::SetRequest ()
ModbusUdpClient::SetRequest ()
ModbusUdpClient::SetCount ()
ModbusUdpClient::SetRequest ()
ModbusUdpClient::SetRequest ()
ModbusUdpClient::SetRequest ()
ModbusUdpClient::SetCount ()
ModbusUdpClient::SetRequest ()
ModbusUdpClient::SetRequest ()
ModbusUdpClient::SetRequest ()
ModbusUdpClient::SetCount ()
ModbusUdpClient::SetRequest ()
ModbusUdpClient::SetRequest ()
ModbusUdpClient::SetRequest ()
ModbusUdpClient::SetCount ()
ModbusUdpClient::SetRequest ()
ModbusUdpClient::SetRequest ()
ModbusUdpClient::SetRequest ()
ModbusUdpClient::SetCount ()
Enable static global routing.
Enable pcap tracing.
Run Simulation.
ModbusUdpClient::Send ()
TraceDelay TX: 253 bytes to 10.1.1.1 Uid: 0 Time: 2.1
ModbusUdpClient::Send ()
TraceDelay TX: 253 bytes to 10.1.2.1 Uid: 1 Time: 2.1
ModbusUdpClient::Send ()
TraceDelay TX: 253 bytes to 10.1.3.1 Uid: 2 Time: 2.1
ModbusUdpClient::Send ()
TraceDelay TX: 253 bytes to 10.1.4.1 Uid: 3 Time: 2.1
ModbusUdpClient::Send ()
TraceDelay TX: 253 bytes to 10.1.5.1 Uid: 4 Time: 2.1
ModbusUdpClient::Send ()
TraceDelay TX: 253 bytes to 10.1.6.1 Uid: 5 Time: 2.1
```

ModbusUdpClient::Send ()

TraceDelay TX: 253 bytes to 10.1.7.1 Uid: 6 Time: 2.1

ModbusUdpClient::Send ()

TraceDelay TX: 253 bytes to 10.1.8.1 Uid: 7 Time: 2.1

Parsing MODBUS UDP function code

TraceDelay: RX 253 bytes from 10.1.1.2 Sequence Number: 0 Uid: 0 TXtime: +2100000000.0ns RXtime: +2102471999.0ns Delay: +2471999.0ns

Sending MODBUS UDP response packet

Parsing MODBUS UDP function code

TraceDelay: RX 253 bytes from 10.1.2.2 Sequence Number: 0 Uid: 1 TXtime: +2100000000.0ns RXtime: +2102471999.0ns Delay: +2471999.0ns

Sending MODBUS UDP response packet

Parsing MODBUS UDP function code

TraceDelay: RX 253 bytes from 10.1.3.2 Sequence Number: 0 Uid: 2 TXtime: +2100000000.0ns RXtime: +2102471999.0ns Delay: +2471999.0ns

Sending MODBUS UDP response packet

Parsing MODBUS UDP function code

TraceDelay: RX 253 bytes from 10.1.4.2 Sequence Number: 0 Uid: 3 TXtime: +2100000000.0ns RXtime: +2102471999.0ns Delay: +2471999.0ns

Sending MODBUS UDP response packet

Parsing MODBUS UDP function code

TraceDelay: RX 253 bytes from 10.1.5.2 Sequence Number: 0 Uid: 4 TXtime: +2100000000.0ns RXtime: +2102471999.0ns Delay: +2471999.0ns

Sending MODBUS UDP response packet

Parsing MODBUS UDP function code

TraceDelay: RX 253 bytes from 10.1.6.2 Sequence Number: 0 Uid: 5 TXtime: +2100000000.0ns RXtime: +2102471999.0ns Delay: +2471999.0ns

Sending MODBUS UDP response packet

Parsing MODBUS UDP function code

TraceDelay: RX 253 bytes from 10.1.7.2 Sequence Number: 0 Uid: 6 TXtime: +2100000000.0ns RXtime: +2102471999.0ns Delay: +2471999.0ns

Sending MODBUS UDP response packet

Parsing MODBUS UDP function code

TraceDelay: RX 253 bytes from 10.1.8.2 Sequence Number: 0 Uid: 7 TXtime: +2100000000.0ns RXtime: +2102471999.0ns Delay: +2471999.0ns

Sending MODBUS UDP response packet

ModbusUdpClient::HandleRead ()

Round-trip Modbus request/response (ms): 15.27

Received 253 bytes from 10.1.1.1

Success - function code: 0x2b

pdu[0]: 0x2b

pdu[1]: 0xd

pdu[2]: 0x2b

pdu[3]: 0xe

pdu[4]: 0x1

pdu[5]: 0x0
pdu[6]: 0x2b
pdu[7]: 0xe
pdu[8]: 0x2
pdu[9]: 0x0

ModbusUdpClient::HandleRead ()
Round-trip Modbus request/response (ms): 15.157
Received 253 bytes from 10.1.2.1
Success - function code: 0x2b
pdu[0]: 0x2b
pdu[1]: 0xd
pdu[2]: 0x2b
pdu[3]: 0xe
pdu[4]: 0x1
pdu[5]: 0x0
pdu[6]: 0x2b
pdu[7]: 0xe
pdu[8]: 0x2
pdu[9]: 0x0

ModbusUdpClient::HandleRead ()
Round-trip Modbus request/response (ms): 15.027
Received 253 bytes from 10.1.3.1
Success - function code: 0x2b
pdu[0]: 0x2b
pdu[1]: 0xd
pdu[2]: 0x2b
pdu[3]: 0xe
pdu[4]: 0x1
pdu[5]: 0x0
pdu[6]: 0x2b
pdu[7]: 0xe
pdu[8]: 0x2
pdu[9]: 0x0

ModbusUdpClient::HandleRead ()
Round-trip Modbus request/response (ms): 14.8749
Received 253 bytes from 10.1.4.1
Success - function code: 0x2b
pdu[0]: 0x2b
pdu[1]: 0xd
pdu[2]: 0x2b

pdu[3]: 0xe
pdu[4]: 0x1
pdu[5]: 0x0
pdu[6]: 0x2b
pdu[7]: 0xe
pdu[8]: 0x2
pdu[9]: 0x0

ModbusUdpClient::HandleRead ()
Round-trip Modbus request/response (ms): 14.74
Received 253 bytes from 10.1.5.1
Success - function code: 0x2b
pdu[0]: 0x2b
pdu[1]: 0xd
pdu[2]: 0x2b
pdu[3]: 0xe
pdu[4]: 0x1
pdu[5]: 0x0
pdu[6]: 0x2b
pdu[7]: 0xe
pdu[8]: 0x2
pdu[9]: 0x0

ModbusUdpClient::HandleRead ()
Round-trip Modbus request/response (ms): 14.5791
Received 253 bytes from 10.1.6.1
Success - function code: 0x2b
pdu[0]: 0x2b
pdu[1]: 0xd
pdu[2]: 0x2b
pdu[3]: 0xe
pdu[4]: 0x1
pdu[5]: 0x0
pdu[6]: 0x2b
pdu[7]: 0xe
pdu[8]: 0x2
pdu[9]: 0x0

ModbusUdpClient::HandleRead ()
Round-trip Modbus request/response (ms): 14.3659
Received 253 bytes from 10.1.7.1
Success - function code: 0x2b
pdu[0]: 0x2b

pdu[1]: 0xd
pdu[2]: 0x2b
pdu[3]: 0xe
pdu[4]: 0x1
pdu[5]: 0x0
pdu[6]: 0x2b
pdu[7]: 0xe
pdu[8]: 0x2
pdu[9]: 0x0

ModbusUdpClient::HandleRead ()
Round-trip Modbus request/response (ms): 14.236
Received 253 bytes from 10.1.8.1
Success - function code: 0x2b
pdu[0]: 0x2b
pdu[1]: 0xd
pdu[2]: 0x2b
pdu[3]: 0xe
pdu[4]: 0x1
pdu[5]: 0x0
pdu[6]: 0x2b
pdu[7]: 0xe
pdu[8]: 0x2
pdu[9]: 0x0

ModbusUdpClient::Send ()
TraceDelay TX: 253 bytes to 10.1.1.1 Uid: 8 Time: 2.3
ModbusUdpClient::Send ()
TraceDelay TX: 253 bytes to 10.1.2.1 Uid: 9 Time: 2.3
ModbusUdpClient::Send ()
TraceDelay TX: 253 bytes to 10.1.3.1 Uid: 10 Time: 2.3
ModbusUdpClient::Send ()
TraceDelay TX: 253 bytes to 10.1.4.1 Uid: 11 Time: 2.3
ModbusUdpClient::Send ()
TraceDelay TX: 253 bytes to 10.1.5.1 Uid: 12 Time: 2.3
ModbusUdpClient::Send ()
TraceDelay TX: 253 bytes to 10.1.6.1 Uid: 13 Time: 2.3
ModbusUdpClient::Send ()
TraceDelay TX: 253 bytes to 10.1.7.1 Uid: 14 Time: 2.3
ModbusUdpClient::Send ()
TraceDelay TX: 253 bytes to 10.1.8.1 Uid: 15 Time: 2.3
Parsing MODBUS UDP function code
Read Device Identification

readDeviceIdCode 1

objectId 0

TraceDelay: RX 253 bytes from 10.1.1.2 Sequence Number: 1 Uid: 8 TXtime: +2300000000.0ns RXtime: +2302471999.0ns Delay: +2471999.0ns

Sending MODBUS UDP response packet

Parsing MODBUS UDP function code

Read Device Identification

readDeviceIdCode 1

objectId 0

TraceDelay: RX 253 bytes from 10.1.2.2 Sequence Number: 1 Uid: 9 TXtime: +2300000000.0ns RXtime: +2302471999.0ns Delay: +2471999.0ns

Sending MODBUS UDP response packet

Parsing MODBUS UDP function code

Read Device Identification

readDeviceIdCode 1

objectId 0

TraceDelay: RX 253 bytes from 10.1.3.2 Sequence Number: 1 Uid: 10 TXtime: +2300000000.0ns RXtime: +2302471999.0ns Delay: +2471999.0ns

Sending MODBUS UDP response packet

Parsing MODBUS UDP function code

Read Device Identification

readDeviceIdCode 1

objectId 0

TraceDelay: RX 253 bytes from 10.1.4.2 Sequence Number: 1 Uid: 11 TXtime: +2300000000.0ns RXtime: +2302471999.0ns Delay: +2471999.0ns

Sending MODBUS UDP response packet

Parsing MODBUS UDP function code

Read Device Identification

readDeviceIdCode 1

objectId 0

TraceDelay: RX 253 bytes from 10.1.5.2 Sequence Number: 1 Uid: 12 TXtime: +2300000000.0ns RXtime: +2302471999.0ns Delay: +2471999.0ns

Sending MODBUS UDP response packet

Parsing MODBUS UDP function code

Read Device Identification

readDeviceIdCode 1

objectId 0

TraceDelay: RX 253 bytes from 10.1.6.2 Sequence Number: 1 Uid: 13 TXtime: +2300000000.0ns RXtime: +2302471999.0ns Delay: +2471999.0ns

Sending MODBUS UDP response packet

Parsing MODBUS UDP function code

Read Device Identification

readDeviceIdCode 1

objectId 0

TraceDelay: RX 253 bytes from 10.1.7.2 Sequence Number: 1 Uid: 14 TXtime: +2300000000.0ns RXtime: +2302471999.0ns Delay: +2471999.0ns

Sending MODBUS UDP response packet

Parsing MODBUS UDP function code

Read Device Identification

readDeviceIdCode 1

objectId 0

TraceDelay: RX 253 bytes from 10.1.8.2 Sequence Number: 1 Uid: 15 TXtime: +2300000000.0ns RXtime: +2302471999.0ns Delay: +2471999.0ns

Sending MODBUS UDP response packet

ModbusUdpClient::HandleRead ()

Round-trip Modbus request/response (ms): 15.245

Received 253 bytes from 10.1.1.1

Success - function code: 0x2b

pdu[0]: 0x2b

pdu[1]: 0xe

pdu[2]: 0x1

pdu[3]: 0x81

pdu[4]: 0x0

pdu[5]: 0x0

pdu[6]: 0x3

pdu[7]: 0x0

pdu[8]: 0x37

pdu[9]: 0x53

ModbusUdpClient::HandleRead ()

Round-trip Modbus request/response (ms): 15.0688

Received 253 bytes from 10.1.2.1

Success - function code: 0x2b

pdu[0]: 0x2b

pdu[1]: 0xe

pdu[2]: 0x1

pdu[3]: 0x81

pdu[4]: 0x0

pdu[5]: 0x0

pdu[6]: 0x3

pdu[7]: 0x0

pdu[8]: 0x37

pdu[9]: 0x53

ModbusUdpClient::HandleRead ()

Round-trip Modbus request/response (ms): 14.8389

Received 253 bytes from 10.1.3.1

Success - function code: 0x2b
pdu[0]: 0x2b
pdu[1]: 0xe
pdu[2]: 0x1
pdu[3]: 0x81
pdu[4]: 0x0
pdu[5]: 0x0
pdu[6]: 0x3
pdu[7]: 0x0
pdu[8]: 0x37
pdu[9]: 0x53

ModbusUdpClient::HandleRead ()
Round-trip Modbus request/response (ms): 14.6661
Received 253 bytes from 10.1.4.1
Success - function code: 0x2b
pdu[0]: 0x2b
pdu[1]: 0xe
pdu[2]: 0x1
pdu[3]: 0x81
pdu[4]: 0x0
pdu[5]: 0x0
pdu[6]: 0x3
pdu[7]: 0x0
pdu[8]: 0x37
pdu[9]: 0x53

ModbusUdpClient::HandleRead ()
Round-trip Modbus request/response (ms): 14.503
Received 253 bytes from 10.1.5.1
Success - function code: 0x2b
pdu[0]: 0x2b
pdu[1]: 0xe
pdu[2]: 0x1
pdu[3]: 0x81
pdu[4]: 0x0
pdu[5]: 0x0
pdu[6]: 0x3
pdu[7]: 0x0
pdu[8]: 0x37
pdu[9]: 0x53

ModbusUdpClient::HandleRead ()

Round-trip Modbus request/response (ms): 14.406
Received 253 bytes from 10.1.6.1
Success - function code: 0x2b
pdu[0]: 0x2b
pdu[1]: 0xe
pdu[2]: 0x1
pdu[3]: 0x81
pdu[4]: 0x0
pdu[5]: 0x0
pdu[6]: 0x3
pdu[7]: 0x0
pdu[8]: 0x37
pdu[9]: 0x53

ModbusUdpClient::HandleRead ()
Round-trip Modbus request/response (ms): 14.194
Received 253 bytes from 10.1.7.1
Success - function code: 0x2b
pdu[0]: 0x2b
pdu[1]: 0xe
pdu[2]: 0x1
pdu[3]: 0x81
pdu[4]: 0x0
pdu[5]: 0x0
pdu[6]: 0x3
pdu[7]: 0x0
pdu[8]: 0x37
pdu[9]: 0x53

ModbusUdpClient::HandleRead ()
Round-trip Modbus request/response (ms): 13.829
Received 253 bytes from 10.1.8.1
Success - function code: 0x2b
pdu[0]: 0x2b
pdu[1]: 0xe
pdu[2]: 0x1
pdu[3]: 0x81
pdu[4]: 0x0
pdu[5]: 0x0
pdu[6]: 0x3
pdu[7]: 0x0
pdu[8]: 0x37
pdu[9]: 0x53

ModbusUdpClient::Send ()
TraceDelay TX: 253 bytes to 10.1.1.1 Uid: 16 Time: 2.4
ModbusUdpClient::Send ()
TraceDelay TX: 253 bytes to 10.1.2.1 Uid: 17 Time: 2.4
ModbusUdpClient::Send ()
TraceDelay TX: 253 bytes to 10.1.3.1 Uid: 18 Time: 2.4
ModbusUdpClient::Send ()
TraceDelay TX: 253 bytes to 10.1.4.1 Uid: 19 Time: 2.4
ModbusUdpClient::Send ()
TraceDelay TX: 253 bytes to 10.1.5.1 Uid: 20 Time: 2.4
ModbusUdpClient::Send ()
TraceDelay TX: 253 bytes to 10.1.6.1 Uid: 21 Time: 2.4
ModbusUdpClient::Send ()
TraceDelay TX: 253 bytes to 10.1.7.1 Uid: 22 Time: 2.4
ModbusUdpClient::Send ()
TraceDelay TX: 253 bytes to 10.1.8.1 Uid: 23 Time: 2.4
Parsing MODBUS UDP function code
Read Device Identification
readDeviceIdCode 4
objectId 0
TraceDelay: RX 253 bytes from 10.1.1.2 Sequence Number: 2 Uid: 16 TXtime: +2400000000.0ns RXtime: +2402471999.0ns Delay: +2471999.0ns
Sending MODBUS UDP response packet
Parsing MODBUS UDP function code
Read Device Identification
readDeviceIdCode 4
objectId 0
TraceDelay: RX 253 bytes from 10.1.2.2 Sequence Number: 2 Uid: 17 TXtime: +2400000000.0ns RXtime: +2402471999.0ns Delay: +2471999.0ns
Sending MODBUS UDP response packet
Parsing MODBUS UDP function code
Read Device Identification
readDeviceIdCode 4
objectId 0
TraceDelay: RX 253 bytes from 10.1.3.2 Sequence Number: 2 Uid: 18 TXtime: +2400000000.0ns RXtime: +2402471999.0ns Delay: +2471999.0ns
Sending MODBUS UDP response packet
Parsing MODBUS UDP function code
Read Device Identification
readDeviceIdCode 4
objectId 0
TraceDelay: RX 253 bytes from 10.1.4.2 Sequence Number: 2 Uid: 19 TXtime: +2400000000.0ns RXtime: +2402471999.0ns Delay: +2471999.0ns

Sending MODBUS UDP response packet

Parsing MODBUS UDP function code

Read Device Identification

readDeviceIdCode 4

objectId 0

TraceDelay: RX 253 bytes from 10.1.5.2 Sequence Number: 2 Uid: 20 TXtime: +2400000000.0ns RXtime: +2402471999.0ns Delay: +2471999.0ns

Sending MODBUS UDP response packet

Parsing MODBUS UDP function code

Read Device Identification

readDeviceIdCode 4

objectId 0

TraceDelay: RX 253 bytes from 10.1.6.2 Sequence Number: 2 Uid: 21 TXtime: +2400000000.0ns RXtime: +2402471999.0ns Delay: +2471999.0ns

Sending MODBUS UDP response packet

Parsing MODBUS UDP function code

Read Device Identification

readDeviceIdCode 4

objectId 0

TraceDelay: RX 253 bytes from 10.1.7.2 Sequence Number: 2 Uid: 22 TXtime: +2400000000.0ns RXtime: +2402471999.0ns Delay: +2471999.0ns

Sending MODBUS UDP response packet

Parsing MODBUS UDP function code

Read Device Identification

readDeviceIdCode 4

objectId 0

TraceDelay: RX 253 bytes from 10.1.8.2 Sequence Number: 2 Uid: 23 TXtime: +2400000000.0ns RXtime: +2402471999.0ns Delay: +2471999.0ns

Sending MODBUS UDP response packet

ModbusUdpClient::HandleRead ()

Round-trip Modbus request/response (ms): 14.9488

Received 253 bytes from 10.1.1.1

Success - function code: 0x2b

pdu[0]: 0x2b

pdu[1]: 0xe

pdu[2]: 0x4

pdu[3]: 0x81

pdu[4]: 0x0

pdu[5]: 0x0

pdu[6]: 0x1

pdu[7]: 0x0

pdu[8]: 0x37

pdu[9]: 0x53

ModbusUdpClient::HandleRead ()
Round-trip Modbus request/response (ms): 14.879
Received 253 bytes from 10.1.2.1
Success - function code: 0x2b
pdu[0]: 0x2b
pdu[1]: 0xe
pdu[2]: 0x4
pdu[3]: 0x81
pdu[4]: 0x0
pdu[5]: 0x0
pdu[6]: 0x1
pdu[7]: 0x0
pdu[8]: 0x37
pdu[9]: 0x53

ModbusUdpClient::HandleRead ()
Round-trip Modbus request/response (ms): 14.818
Received 253 bytes from 10.1.3.1
Success - function code: 0x2b
pdu[0]: 0x2b
pdu[1]: 0xe
pdu[2]: 0x4
pdu[3]: 0x81
pdu[4]: 0x0
pdu[5]: 0x0
pdu[6]: 0x1
pdu[7]: 0x0
pdu[8]: 0x37
pdu[9]: 0x53

ModbusUdpClient::HandleRead ()
Round-trip Modbus request/response (ms): 14.715
Received 253 bytes from 10.1.4.1
Success - function code: 0x2b
pdu[0]: 0x2b
pdu[1]: 0xe
pdu[2]: 0x4
pdu[3]: 0x81
pdu[4]: 0x0
pdu[5]: 0x0
pdu[6]: 0x1
pdu[7]: 0x0
pdu[8]: 0x37

pdu[9]: 0x53

ModbusUdpClient::HandleRead ()
Round-trip Modbus request/response (ms): 14.59
Received 253 bytes from 10.1.5.1
Success - function code: 0x2b
pdu[0]: 0x2b
pdu[1]: 0xe
pdu[2]: 0x4
pdu[3]: 0x81
pdu[4]: 0x0
pdu[5]: 0x0
pdu[6]: 0x1
pdu[7]: 0x0
pdu[8]: 0x37
pdu[9]: 0x53

ModbusUdpClient::HandleRead ()
Round-trip Modbus request/response (ms): 14.5121
Received 253 bytes from 10.1.6.1
Success - function code: 0x2b
pdu[0]: 0x2b
pdu[1]: 0xe
pdu[2]: 0x4
pdu[3]: 0x81
pdu[4]: 0x0
pdu[5]: 0x0
pdu[6]: 0x1
pdu[7]: 0x0
pdu[8]: 0x37
pdu[9]: 0x53

ModbusUdpClient::HandleRead ()
Round-trip Modbus request/response (ms): 14.4658
Received 253 bytes from 10.1.7.1
Success - function code: 0x2b
pdu[0]: 0x2b
pdu[1]: 0xe
pdu[2]: 0x4
pdu[3]: 0x81
pdu[4]: 0x0
pdu[5]: 0x0
pdu[6]: 0x1

pdu[7]: 0x0
pdu[8]: 0x37
pdu[9]: 0x53

ModbusUdpClient::HandleRead ()
Round-trip Modbus request/response (ms): 14.4382
Received 253 bytes from 10.1.8.1
Success - function code: 0x2b
pdu[0]: 0x2b
pdu[1]: 0xe
pdu[2]: 0x4
pdu[3]: 0x81
pdu[4]: 0x0
pdu[5]: 0x0
pdu[6]: 0x1
pdu[7]: 0x0
pdu[8]: 0x37
pdu[9]: 0x53

Done.
Total (ms): 111.567
Modbus Simulation time (ms): 68.424
Node and Link creation (ms): 43.143
reza@Dena:~/temp/ns-3-allinone/ns-3-dev$

starModbusTcp

reza@Dena:~/temp/ns-3-allinone/ns-3-dev$ export NS_LOG=StarModbusTcp=level_all
reza@Dena:~/temp/ns-3-allinone/ns-3-dev$ ./waf --run "scratch/starModbusTcp"
Waf: Entering directory `/home/reza/temp/ns-3-allinone/ns-3-dev/build'
[1240/1362] cxx: scratch/starModbusTcp.cc -> build/scratch/starModbusTcp.cc.1.o
[1356/1362] cxxprogram: build/scratch/starModbusTcp.cc.1.o -> build/scratch/starModbusTcp
Waf: Leaving directory `/home/reza/temp/ns-3-allinone/ns-3-dev/build'
'build' finished successfully (4.737s)
Build star topology.
Install internet stack on all nodes.
Assign IP Addresses.
Create applications.
Enable static global routing.
Enable pcap tracing.
Run Simulation.
Read Device Identification
readDeviceIdCode 1
objectId 0

Read Device Identification
readDeviceIdCode 1
objectId 0
Read Device Identification
readDeviceIdCode 1
objectId 0
Read Device Identification
readDeviceIdCode 1
objectId 0
Read Device Identification
readDeviceIdCode 1
objectId 0
Read Device Identification
readDeviceIdCode 1
objectId 0
Read Device Identification
readDeviceIdCode 1
objectId 0
Read Device Identification
readDeviceIdCode 1
objectId 0
Read Device Identification
readDeviceIdCode 4
objectId 0
Read Device Identification
readDeviceIdCode 4
objectId 0
Read Device Identification
readDeviceIdCode 4
objectId 0
Read Device Identification
readDeviceIdCode 4
objectId 0
Read Device Identification
readDeviceIdCode 4
objectId 0
Read Device Identification
readDeviceIdCode 4
objectId 0
Read Device Identification
readDeviceIdCode 4
objectId 0
Read Device Identification

```
readDeviceIdCode 4
objectId 0
Done.
Total (ms): 269.734
Modbus Simulation time (ms): 227.468
Node and Link creation (ms): 42.2659
reza@Dena:~/temp/ns-3-allinone/ns-3-dev$
```

starModbusTcp (verbose)

```
reza@Dena:~/temp/ns-3-allinone/ns-3-dev$ export NS_LOG=StarModbusTcp=level_all
reza@Dena:~/temp/ns-3-allinone/ns-3-dev$ ./waf --run "scratch/starModbusTcp --verbose=1"
Waf: Entering directory `/home/reza/temp/ns-3-allinone/ns-3-dev/build'
Waf: Leaving directory `/home/reza/temp/ns-3-allinone/ns-3-dev/build'
'build' finished successfully (1.690s)
Build star topology.
Install internet stack on all nodes.
Assign IP Addresses.
Create applications.
ModbusTcpClient::SetRequest ()
ModbusTcpClient::SetRequest ()
ModbusTcpClient::SetRequest ()
ModbusTcpClient::SetCount ()
ModbusTcpClient::SetRequest ()
ModbusTcpClient::SetRequest ()
ModbusTcpClient::SetRequest ()
ModbusTcpClient::SetCount ()
ModbusTcpClient::SetRequest ()
ModbusTcpClient::SetRequest ()
ModbusTcpClient::SetRequest ()
ModbusTcpClient::SetCount ()
ModbusTcpClient::SetRequest ()
ModbusTcpClient::SetRequest ()
ModbusTcpClient::SetRequest ()
ModbusTcpClient::SetCount ()
ModbusTcpClient::SetRequest ()
ModbusTcpClient::SetRequest ()
ModbusTcpClient::SetRequest ()
ModbusTcpClient::SetCount ()
ModbusTcpClient::SetRequest ()
ModbusTcpClient::SetRequest ()
ModbusTcpClient::SetRequest ()
ModbusTcpClient::SetCount ()
ModbusTcpClient::SetRequest ()
```

ModbusTcpClient::SetRequest ()
ModbusTcpClient::SetRequest ()
ModbusTcpClient::SetCount ()
ModbusTcpClient::SetRequest ()
ModbusTcpClient::SetRequest ()
ModbusTcpClient::SetRequest ()
ModbusTcpClient::SetCount ()
Enable static global routing.
Enable pcap tracing.
Run Simulation.
ModbusTcpClient::Send ()
TraceDelay TX: 253 bytes to 10.1.1.1 Uid: 24 Time: 2.1
ModbusTcpClient::Send ()
TraceDelay TX: 253 bytes to 10.1.2.1 Uid: 25 Time: 2.1
ModbusTcpClient::Send ()
TraceDelay TX: 253 bytes to 10.1.3.1 Uid: 26 Time: 2.1
ModbusTcpClient::Send ()
TraceDelay TX: 253 bytes to 10.1.4.1 Uid: 27 Time: 2.1
ModbusTcpClient::Send ()
TraceDelay TX: 253 bytes to 10.1.5.1 Uid: 28 Time: 2.1
ModbusTcpClient::Send ()
TraceDelay TX: 253 bytes to 10.1.6.1 Uid: 29 Time: 2.1
ModbusTcpClient::Send ()
TraceDelay TX: 253 bytes to 10.1.7.1 Uid: 30 Time: 2.1
ModbusTcpClient::Send ()
TraceDelay TX: 253 bytes to 10.1.8.1 Uid: 31 Time: 2.1
Parsing MODBUS TCP function code
TraceDelay: RX 253 bytes from 10.1.1.2 Sequence Number: 0 Uid: 24 TXtime: +2100000000.0ns RXtime: +2102491200.0ns Delay: +2491200.0ns
Sending MODBUS TCP response packet
Parsing MODBUS TCP function code
TraceDelay: RX 253 bytes from 10.1.2.2 Sequence Number: 0 Uid: 25 TXtime: +2100000000.0ns RXtime: +2102491200.0ns Delay: +2491200.0ns
Sending MODBUS TCP response packet
Parsing MODBUS TCP function code
TraceDelay: RX 253 bytes from 10.1.3.2 Sequence Number: 0 Uid: 26 TXtime: +2100000000.0ns RXtime: +2102491200.0ns Delay: +2491200.0ns
Sending MODBUS TCP response packet
Parsing MODBUS TCP function code
TraceDelay: RX 253 bytes from 10.1.4.2 Sequence Number: 0 Uid: 27 TXtime: +2100000000.0ns RXtime: +2102491200.0ns Delay: +2491200.0ns
Sending MODBUS TCP response packet
Parsing MODBUS TCP function code

TraceDelay: RX 253 bytes from 10.1.5.2 Sequence Number: 0 Uid: 28 TXtime: +2100000000.0ns RXtime: +2102491200.0ns Delay: +2491200.0ns

Sending MODBUS TCP response packet

Parsing MODBUS TCP function code

TraceDelay: RX 253 bytes from 10.1.6.2 Sequence Number: 0 Uid: 29 TXtime: +2100000000.0ns RXtime: +2102491200.0ns Delay: +2491200.0ns

Sending MODBUS TCP response packet

Parsing MODBUS TCP function code

TraceDelay: RX 253 bytes from 10.1.7.2 Sequence Number: 0 Uid: 30 TXtime: +2100000000.0ns RXtime: +2102491200.0ns Delay: +2491200.0ns

Sending MODBUS TCP response packet

Parsing MODBUS TCP function code

TraceDelay: RX 253 bytes from 10.1.8.2 Sequence Number: 0 Uid: 31 TXtime: +2100000000.0ns RXtime: +2102491200.0ns Delay: +2491200.0ns

Sending MODBUS TCP response packet

ModbusTcpClient::HandleRead ()

Received 253 bytes from 10.1.1.1

Success - function code: 0x2b

pdu[0]: 0x2b

pdu[1]: 0xd

pdu[2]: 0x2b

pdu[3]: 0xe

pdu[4]: 0x1

pdu[5]: 0x0

pdu[6]: 0x2b

pdu[7]: 0xe

pdu[8]: 0x2

pdu[9]: 0x0

ModbusTcpClient::HandleRead ()

Received 253 bytes from 10.1.2.1

Success - function code: 0x2b

pdu[0]: 0x2b

pdu[1]: 0xd

pdu[2]: 0x2b

pdu[3]: 0xe

pdu[4]: 0x1

pdu[5]: 0x0

pdu[6]: 0x2b

pdu[7]: 0xe

pdu[8]: 0x2

pdu[9]: 0x0

ModbusTcpClient::HandleRead ()

Received 253 bytes from 10.1.3.1
Success - function code: 0x2b
pdu[0]: 0x2b
pdu[1]: 0xd
pdu[2]: 0x2b
pdu[3]: 0xe
pdu[4]: 0x1
pdu[5]: 0x0
pdu[6]: 0x2b
pdu[7]: 0xe
pdu[8]: 0x2
pdu[9]: 0x0

ModbusTcpClient::HandleRead ()
Received 253 bytes from 10.1.4.1
Success - function code: 0x2b
pdu[0]: 0x2b
pdu[1]: 0xd
pdu[2]: 0x2b
pdu[3]: 0xe
pdu[4]: 0x1
pdu[5]: 0x0
pdu[6]: 0x2b
pdu[7]: 0xe
pdu[8]: 0x2
pdu[9]: 0x0

ModbusTcpClient::HandleRead ()
Received 253 bytes from 10.1.5.1
Success - function code: 0x2b
pdu[0]: 0x2b
pdu[1]: 0xd
pdu[2]: 0x2b
pdu[3]: 0xe
pdu[4]: 0x1
pdu[5]: 0x0
pdu[6]: 0x2b
pdu[7]: 0xe
pdu[8]: 0x2
pdu[9]: 0x0

ModbusTcpClient::HandleRead ()
Received 253 bytes from 10.1.6.1

Success - function code: 0x2b
pdu[0]: 0x2b
pdu[1]: 0xd
pdu[2]: 0x2b
pdu[3]: 0xe
pdu[4]: 0x1
pdu[5]: 0x0
pdu[6]: 0x2b
pdu[7]: 0xe
pdu[8]: 0x2
pdu[9]: 0x0

ModbusTcpClient::HandleRead ()
Received 253 bytes from 10.1.7.1
Success - function code: 0x2b
pdu[0]: 0x2b
pdu[1]: 0xd
pdu[2]: 0x2b
pdu[3]: 0xe
pdu[4]: 0x1
pdu[5]: 0x0
pdu[6]: 0x2b
pdu[7]: 0xe
pdu[8]: 0x2
pdu[9]: 0x0

ModbusTcpClient::HandleRead ()
Received 253 bytes from 10.1.8.1
Success - function code: 0x2b
pdu[0]: 0x2b
pdu[1]: 0xd
pdu[2]: 0x2b
pdu[3]: 0xe
pdu[4]: 0x1
pdu[5]: 0x0
pdu[6]: 0x2b
pdu[7]: 0xe
pdu[8]: 0x2
pdu[9]: 0x0

ModbusTcpClient::Send ()
TraceDelay TX: 253 bytes to 10.1.1.1 Uid: 72 Time: 2.3
ModbusTcpClient::Send ()

TraceDelay TX: 253 bytes to 10.1.2.1 Uid: 73 Time: 2.3
ModbusTcpClient::Send ()
TraceDelay TX: 253 bytes to 10.1.3.1 Uid: 74 Time: 2.3
ModbusTcpClient::Send ()
TraceDelay TX: 253 bytes to 10.1.4.1 Uid: 75 Time: 2.3
ModbusTcpClient::Send ()
TraceDelay TX: 253 bytes to 10.1.5.1 Uid: 76 Time: 2.3
ModbusTcpClient::Send ()
TraceDelay TX: 253 bytes to 10.1.6.1 Uid: 77 Time: 2.3
ModbusTcpClient::Send ()
TraceDelay TX: 253 bytes to 10.1.7.1 Uid: 78 Time: 2.3
ModbusTcpClient::Send ()
TraceDelay TX: 253 bytes to 10.1.8.1 Uid: 79 Time: 2.3
Parsing MODBUS TCP function code
Read Device Identification
readDeviceIdCode 1
objectId 0
TraceDelay: RX 253 bytes from 10.1.1.2 Sequence Number: 1 Uid: 72 TXtime: +2300000000.0ns RXtime: +2302491200.0ns Delay: +2491200.0ns
Sending MODBUS TCP response packet
Parsing MODBUS TCP function code
Read Device Identification
readDeviceIdCode 1
objectId 0
TraceDelay: RX 253 bytes from 10.1.2.2 Sequence Number: 1 Uid: 73 TXtime: +2300000000.0ns RXtime: +2302491200.0ns Delay: +2491200.0ns
Sending MODBUS TCP response packet
Parsing MODBUS TCP function code
Read Device Identification
readDeviceIdCode 1
objectId 0
TraceDelay: RX 253 bytes from 10.1.3.2 Sequence Number: 1 Uid: 74 TXtime: +2300000000.0ns RXtime: +2302491200.0ns Delay: +2491200.0ns
Sending MODBUS TCP response packet
Parsing MODBUS TCP function code
Read Device Identification
readDeviceIdCode 1
objectId 0
TraceDelay: RX 253 bytes from 10.1.4.2 Sequence Number: 1 Uid: 75 TXtime: +2300000000.0ns RXtime: +2302491200.0ns Delay: +2491200.0ns
Sending MODBUS TCP response packet
Parsing MODBUS TCP function code
Read Device Identification
readDeviceIdCode 1

objectId 0

TraceDelay: RX 253 bytes from 10.1.5.2 Sequence Number: 1 Uid: 76 TXtime: +2300000000.0ns RXtime: +2302491200.0ns Delay: +2491200.0ns

Sending MODBUS TCP response packet

Parsing MODBUS TCP function code

Read Device Identification

readDeviceIdCode 1

objectId 0

TraceDelay: RX 253 bytes from 10.1.6.2 Sequence Number: 1 Uid: 77 TXtime: +2300000000.0ns RXtime: +2302491200.0ns Delay: +2491200.0ns

Sending MODBUS TCP response packet

Parsing MODBUS TCP function code

Read Device Identification

readDeviceIdCode 1

objectId 0

TraceDelay: RX 253 bytes from 10.1.7.2 Sequence Number: 1 Uid: 78 TXtime: +2300000000.0ns RXtime: +2302491200.0ns Delay: +2491200.0ns

Sending MODBUS TCP response packet

Parsing MODBUS TCP function code

Read Device Identification

readDeviceIdCode 1

objectId 0

TraceDelay: RX 253 bytes from 10.1.8.2 Sequence Number: 1 Uid: 79 TXtime: +2300000000.0ns RXtime: +2302491200.0ns Delay: +2491200.0ns

Sending MODBUS TCP response packet

ModbusTcpClient::HandleRead ()

Received 253 bytes from 10.1.1.1

Success - function code: 0x2b

pdu[0]: 0x2b

pdu[1]: 0xe

pdu[2]: 0x1

pdu[3]: 0x81

pdu[4]: 0x0

pdu[5]: 0x0

pdu[6]: 0x3

pdu[7]: 0x0

pdu[8]: 0x37

pdu[9]: 0x53

ModbusTcpClient::HandleRead ()

Received 253 bytes from 10.1.2.1

Success - function code: 0x2b

pdu[0]: 0x2b

pdu[1]: 0xe

pdu[2]: 0x1
pdu[3]: 0x81
pdu[4]: 0x0
pdu[5]: 0x0
pdu[6]: 0x3
pdu[7]: 0x0
pdu[8]: 0x37
pdu[9]: 0x53

ModbusTcpClient::HandleRead ()
Received 253 bytes from 10.1.3.1
Success - function code: 0x2b
pdu[0]: 0x2b
pdu[1]: 0xe
pdu[2]: 0x1
pdu[3]: 0x81
pdu[4]: 0x0
pdu[5]: 0x0
pdu[6]: 0x3
pdu[7]: 0x0
pdu[8]: 0x37
pdu[9]: 0x53

ModbusTcpClient::HandleRead ()
Received 253 bytes from 10.1.4.1
Success - function code: 0x2b
pdu[0]: 0x2b
pdu[1]: 0xe
pdu[2]: 0x1
pdu[3]: 0x81
pdu[4]: 0x0
pdu[5]: 0x0
pdu[6]: 0x3
pdu[7]: 0x0
pdu[8]: 0x37
pdu[9]: 0x53

ModbusTcpClient::HandleRead ()
Received 253 bytes from 10.1.5.1
Success - function code: 0x2b
pdu[0]: 0x2b
pdu[1]: 0xe
pdu[2]: 0x1

pdu[3]: 0x81
pdu[4]: 0x0
pdu[5]: 0x0
pdu[6]: 0x3
pdu[7]: 0x0
pdu[8]: 0x37
pdu[9]: 0x53

ModbusTcpClient::HandleRead ()
Received 253 bytes from 10.1.6.1
Success - function code: 0x2b
pdu[0]: 0x2b
pdu[1]: 0xe
pdu[2]: 0x1
pdu[3]: 0x81
pdu[4]: 0x0
pdu[5]: 0x0
pdu[6]: 0x3
pdu[7]: 0x0
pdu[8]: 0x37
pdu[9]: 0x53

ModbusTcpClient::HandleRead ()
Received 253 bytes from 10.1.7.1
Success - function code: 0x2b
pdu[0]: 0x2b
pdu[1]: 0xe
pdu[2]: 0x1
pdu[3]: 0x81
pdu[4]: 0x0
pdu[5]: 0x0
pdu[6]: 0x3
pdu[7]: 0x0
pdu[8]: 0x37
pdu[9]: 0x53

ModbusTcpClient::HandleRead ()
Received 253 bytes from 10.1.8.1
Success - function code: 0x2b
pdu[0]: 0x2b
pdu[1]: 0xe
pdu[2]: 0x1
pdu[3]: 0x81

pdu[4]: 0x0
pdu[5]: 0x0
pdu[6]: 0x3
pdu[7]: 0x0
pdu[8]: 0x37
pdu[9]: 0x53

ModbusTcpClient::Send ()
TraceDelay TX: 253 bytes to 10.1.1.1 Uid: 120 Time: 2.4
ModbusTcpClient::Send ()
TraceDelay TX: 253 bytes to 10.1.2.1 Uid: 121 Time: 2.4
ModbusTcpClient::Send ()
TraceDelay TX: 253 bytes to 10.1.3.1 Uid: 122 Time: 2.4
ModbusTcpClient::Send ()
TraceDelay TX: 253 bytes to 10.1.4.1 Uid: 123 Time: 2.4
ModbusTcpClient::Send ()
TraceDelay TX: 253 bytes to 10.1.5.1 Uid: 124 Time: 2.4
ModbusTcpClient::Send ()
TraceDelay TX: 253 bytes to 10.1.6.1 Uid: 125 Time: 2.4
ModbusTcpClient::Send ()
TraceDelay TX: 253 bytes to 10.1.7.1 Uid: 126 Time: 2.4
ModbusTcpClient::Send ()
TraceDelay TX: 253 bytes to 10.1.8.1 Uid: 127 Time: 2.4
Parsing MODBUS TCP function code
Read Device Identification
readDeviceIdCode 4
objectId 0
TraceDelay: RX 253 bytes from 10.1.1.2 Sequence Number: 2 Uid: 120 TXtime: +2400000000.0ns RXtime: +2402491200.0ns Delay: +2491200.0ns
Sending MODBUS TCP response packet
Parsing MODBUS TCP function code
Read Device Identification
readDeviceIdCode 4
objectId 0
TraceDelay: RX 253 bytes from 10.1.2.2 Sequence Number: 2 Uid: 121 TXtime: +2400000000.0ns RXtime: +2402491200.0ns Delay: +2491200.0ns
Sending MODBUS TCP response packet
Parsing MODBUS TCP function code
Read Device Identification
readDeviceIdCode 4
objectId 0
TraceDelay: RX 253 bytes from 10.1.3.2 Sequence Number: 2 Uid: 122 TXtime: +2400000000.0ns RXtime: +2402491200.0ns Delay: +2491200.0ns
Sending MODBUS TCP response packet

Parsing MODBUS TCP function code

Read Device Identification

readDeviceIdCode 4

objectId 0

TraceDelay: RX 253 bytes from 10.1.4.2 Sequence Number: 2 Uid: 123 TXtime: +2400000000.0ns RXtime: +2402491200.0ns Delay: +2491200.0ns

Sending MODBUS TCP response packet

Parsing MODBUS TCP function code

Read Device Identification

readDeviceIdCode 4

objectId 0

TraceDelay: RX 253 bytes from 10.1.5.2 Sequence Number: 2 Uid: 124 TXtime: +2400000000.0ns RXtime: +2402491200.0ns Delay: +2491200.0ns

Sending MODBUS TCP response packet

Parsing MODBUS TCP function code

Read Device Identification

readDeviceIdCode 4

objectId 0

TraceDelay: RX 253 bytes from 10.1.6.2 Sequence Number: 2 Uid: 125 TXtime: +2400000000.0ns RXtime: +2402491200.0ns Delay: +2491200.0ns

Sending MODBUS TCP response packet

Parsing MODBUS TCP function code

Read Device Identification

readDeviceIdCode 4

objectId 0

TraceDelay: RX 253 bytes from 10.1.7.2 Sequence Number: 2 Uid: 126 TXtime: +2400000000.0ns RXtime: +2402491200.0ns Delay: +2491200.0ns

Sending MODBUS TCP response packet

Parsing MODBUS TCP function code

Read Device Identification

readDeviceIdCode 4

objectId 0

TraceDelay: RX 253 bytes from 10.1.8.2 Sequence Number: 2 Uid: 127 TXtime: +2400000000.0ns RXtime: +2402491200.0ns Delay: +2491200.0ns

Sending MODBUS TCP response packet

ModbusTcpClient::HandleRead ()

Received 253 bytes from 10.1.1.1

Success - function code: 0x2b

pdu[0]: 0x2b

pdu[1]: 0xe

pdu[2]: 0x4

pdu[3]: 0x81

pdu[4]: 0x0

pdu[5]: 0x0

pdu[6]: 0x1
pdu[7]: 0x0
pdu[8]: 0x37
pdu[9]: 0x53

ModbusTcpClient::HandleRead ()
Received 253 bytes from 10.1.2.1
Success - function code: 0x2b
pdu[0]: 0x2b
pdu[1]: 0xe
pdu[2]: 0x4
pdu[3]: 0x81
pdu[4]: 0x0
pdu[5]: 0x0
pdu[6]: 0x1
pdu[7]: 0x0
pdu[8]: 0x37
pdu[9]: 0x53

ModbusTcpClient::HandleRead ()
Received 253 bytes from 10.1.3.1
Success - function code: 0x2b
pdu[0]: 0x2b
pdu[1]: 0xe
pdu[2]: 0x4
pdu[3]: 0x81
pdu[4]: 0x0
pdu[5]: 0x0
pdu[6]: 0x1
pdu[7]: 0x0
pdu[8]: 0x37
pdu[9]: 0x53

ModbusTcpClient::HandleRead ()
Received 253 bytes from 10.1.4.1
Success - function code: 0x2b
pdu[0]: 0x2b
pdu[1]: 0xe
pdu[2]: 0x4
pdu[3]: 0x81
pdu[4]: 0x0
pdu[5]: 0x0
pdu[6]: 0x1

pdu[7]: 0x0
pdu[8]: 0x37
pdu[9]: 0x53

ModbusTcpClient::HandleRead ()
Received 253 bytes from 10.1.5.1
Success - function code: 0x2b
pdu[0]: 0x2b
pdu[1]: 0xe
pdu[2]: 0x4
pdu[3]: 0x81
pdu[4]: 0x0
pdu[5]: 0x0
pdu[6]: 0x1
pdu[7]: 0x0
pdu[8]: 0x37
pdu[9]: 0x53

ModbusTcpClient::HandleRead ()
Received 253 bytes from 10.1.6.1
Success - function code: 0x2b
pdu[0]: 0x2b
pdu[1]: 0xe
pdu[2]: 0x4
pdu[3]: 0x81
pdu[4]: 0x0
pdu[5]: 0x0
pdu[6]: 0x1
pdu[7]: 0x0
pdu[8]: 0x37
pdu[9]: 0x53

ModbusTcpClient::HandleRead ()
Received 253 bytes from 10.1.7.1
Success - function code: 0x2b
pdu[0]: 0x2b
pdu[1]: 0xe
pdu[2]: 0x4
pdu[3]: 0x81
pdu[4]: 0x0
pdu[5]: 0x0
pdu[6]: 0x1
pdu[7]: 0x0

```
pdu[8]: 0x37
pdu[9]: 0x53

ModbusTcpClient::HandleRead ()
Received 253 bytes from 10.1.8.1
Success - function code: 0x2b
pdu[0]: 0x2b
pdu[1]: 0xe
pdu[2]: 0x4
pdu[3]: 0x81
pdu[4]: 0x0
pdu[5]: 0x0
pdu[6]: 0x1
pdu[7]: 0x0
pdu[8]: 0x37
pdu[9]: 0x53

ModbusTcpServer, peerClose
ModbusTcpServer, peerClose
ModbusTcpServer, peerClose
ModbusTcpServer, peerClose
ModbusTcpServer, peerClose
ModbusTcpServer, peerClose
ModbusTcpServer, peerClose
ModbusTcpServer, peerClose
ModbusTcpServer, peerClose
Done.
Total (ms): 295.794
Modbus Simulation time (ms): 253.589
Node and Link creation (ms): 42.2049
reza@Dena:~/temp/ns-3-allinone/ns-3-dev$
```

Appendix B.

Traces – Wireshark Screen Shots

In this Appendix, the screen shots of Wireshark pcap captured files generated by ns-Modbus are included. The screen shots are categorized based on each scenario in separate sections. The Wireshark capture shows the traffic each node transmits or receives.

The number of pcap files depends of the number of nodes in the scenario, which running. The pcap files contain the traces of packets between the nodes and can be viewed using Wireshark or tcpdump tools.

myFirstModbusUdp scenario

This scenario includes two nodes connected over a LAN with network address of 10.1.1.0. The connection between the two nodes is point-to-point with a data rate of 5 Mbps and a delay of 2 ms. Node0 is a Modbus UDP client and node1 is a Modbus UDP server with 10.1.1.1 and 10.1.1.2 ipaddresses respectively. The whole simulation length is 10 seconds, but the server node application starts after one second and the client node application starts after two seconds. The client node sends three different Modbus requests, which are Encapsulated Interface Transport of Canopen General, Read Device ID Basic, and Read Device ID Specific, after a 100 ms, 200 ms, and 100 ms delay respectively. The sequence of sending the above requests set to repeat one hundred times, but the end of simulation arrives earlier in this case. Please see appendix E for more details.

The Figure 38 shows the beginning of the capture for node0, whereas, Figure 39 shows the end of the capture for node1.

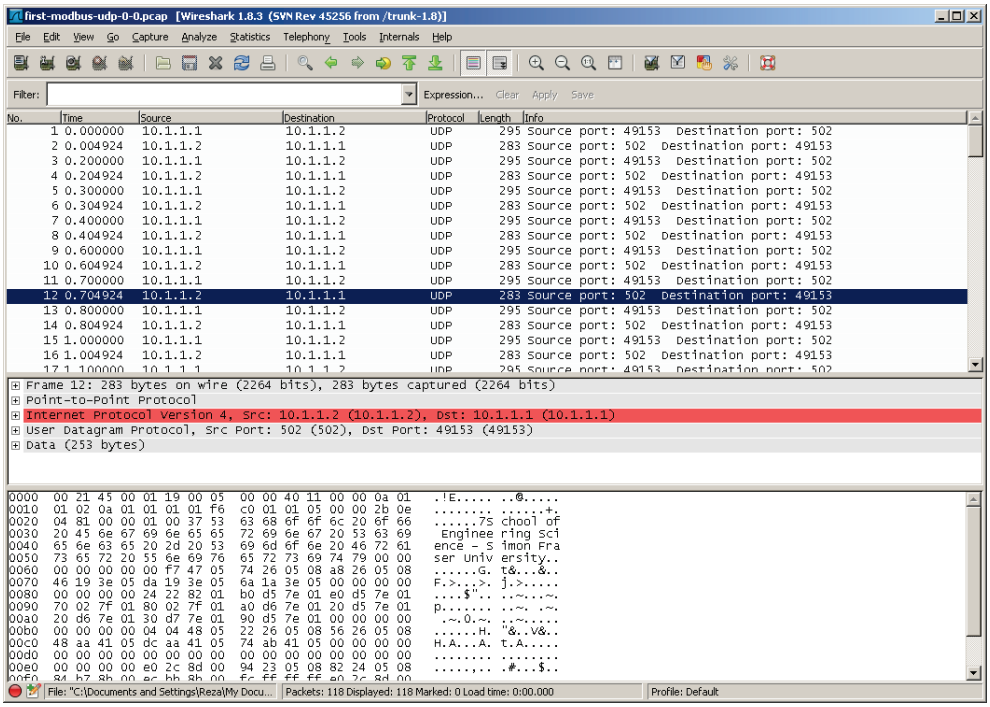

Figure B1. myFirstModbusUdp scenario Wireshark capture node0 (client)

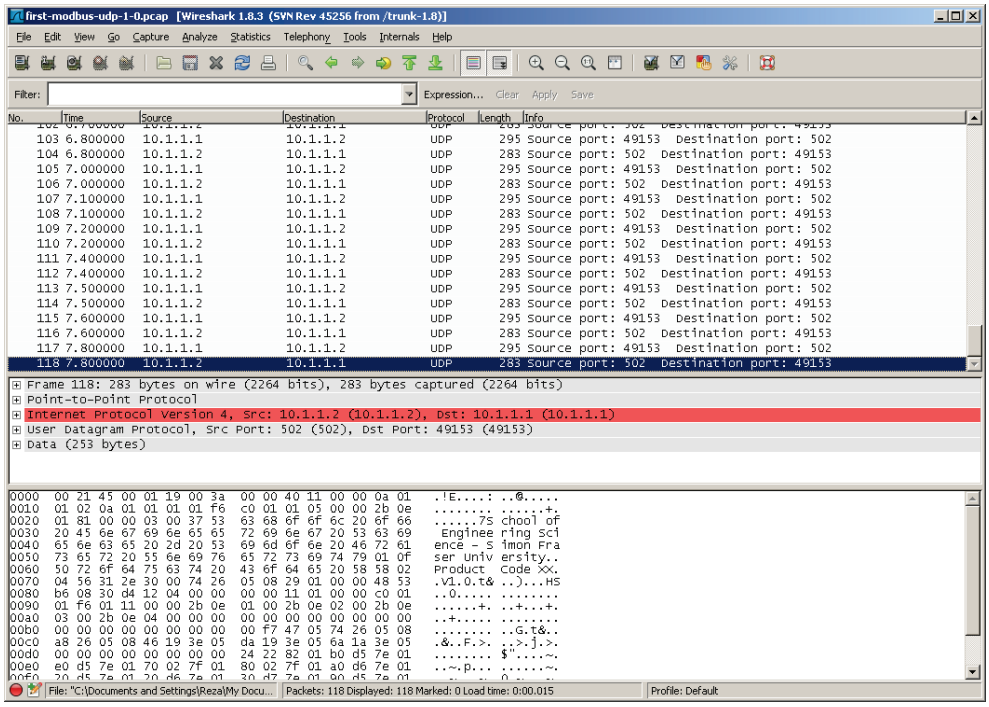

Figure B2. myFirstModbusUdp scenario Wireshark capture node1 (server)

myFirstModbusTcp scenario

This scenario includes two nodes connected over a LAN with network address of 10.1.1.0. The connection between the two nodes is point-to-point with a data rate of 5 Mbps and a delay of 2 ms. Node0 is a Modbus TCP client and node1 is a Modbus TCP server with 10.1.1.1 and 10.1.1.2 ipaddresses respectively. The whole simulation length is 10 seconds, but the server node application starts after one second and the client node application starts after two seconds. The client node sends three different Modbus requests, which are Encapsulated Interface Transport of Canopen General, Read Device ID Basic, and Read Device ID Specific, after a 100 ms, 200 ms, and 100 ms delay respectively. The sequence of sending the above requests set to repeat one hundred times, but the end of simulation arrives earlier in this case. Please see appendix E for more details.

The Figure 40 shows the beginning of the capture for node0, whereas, Figure 41 shows the end of the capture for node1.

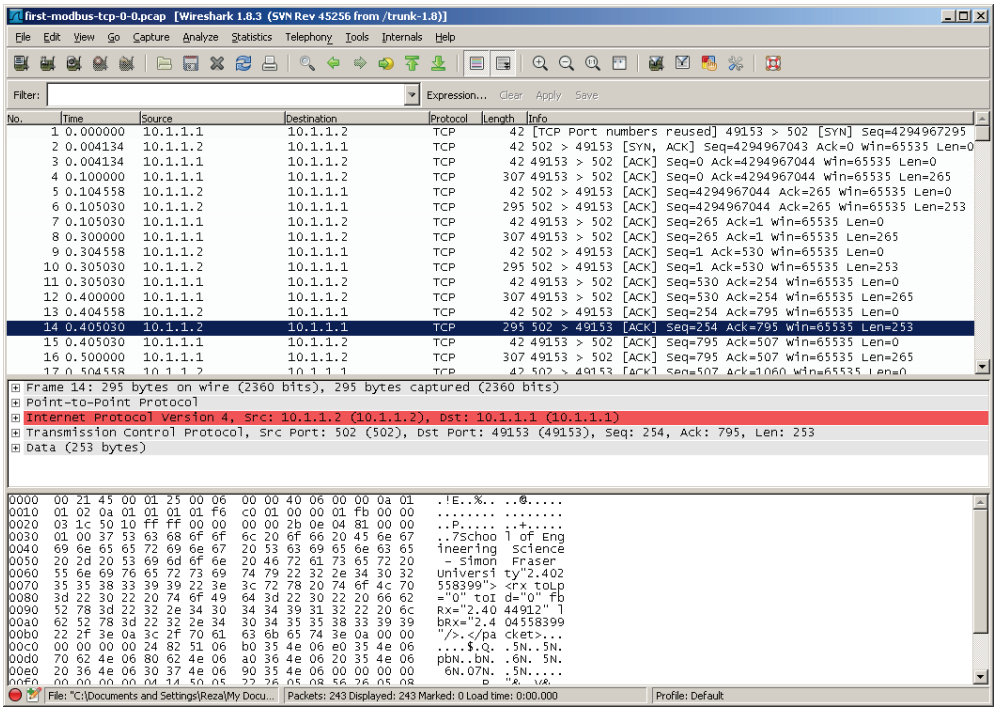

Figure B3. myFirstModbusTcp scenario Wireshark capture node0 (client)

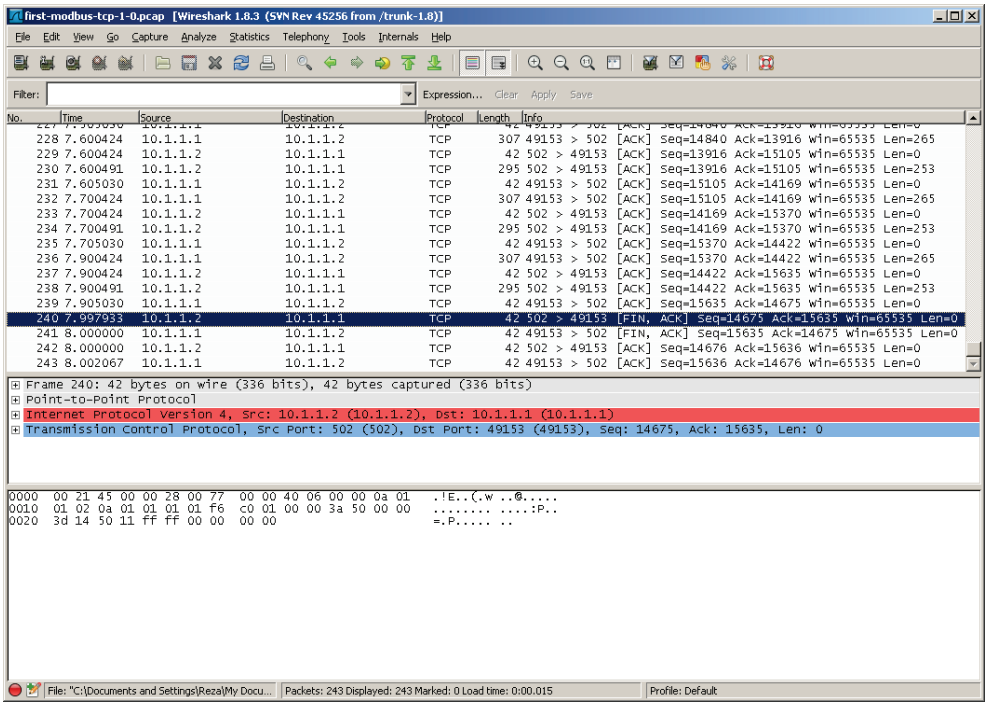

Figure B4. myFirstModbusTcp scenario Wireshark capture node1 (server)

busModbusUdp scenario

This scenario includes four nodes connected over a LAN with network address of 10.1.1.0. The connection between the four nodes is CSMA with a data rate of 100 Mbps and a delay of 6560 nanoseconds. Node0 is a Modbus UDP server, whereas, node1, 2, and 3 are Modbus UDP clients with 10.1.1.1, 10.1.1.2, 10.1.1.3, and 10.1.1.4 ipaddresses respectively. The whole simulation length is 10 seconds, but the server node application starts after one second and the client nodes application start after two seconds. The client nodes send three different Modbus requests, which are Encapsulated Interface Transport of Canopen General, Read Device ID Basic, and Read Device ID Specific, after a 100 ms, 200 ms, and 100 ms delay respectively. Please see appendix E for more details.

The Figure 42 shows the beginning of the capture for node0; which is the server, whereas, Figure 43, 44, and 45 show the clients' captures.

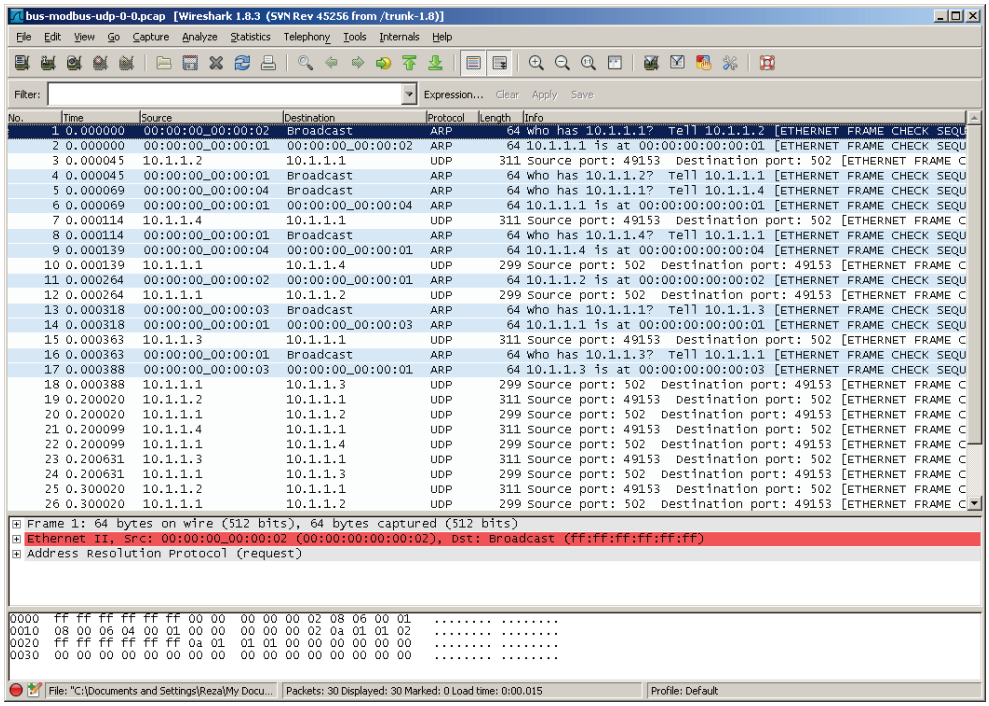

Figure B5. busModbusUdp scenario Wireshark capture node0 (server)

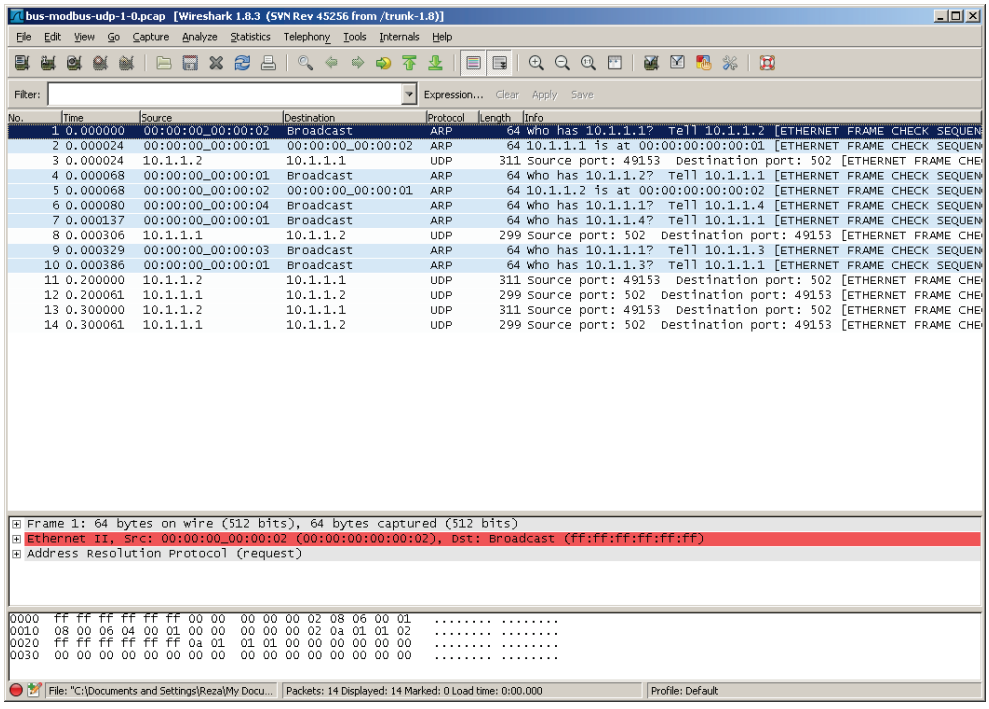

Figure B6. busModbusUdp scenario Wireshark capture node1 (client)

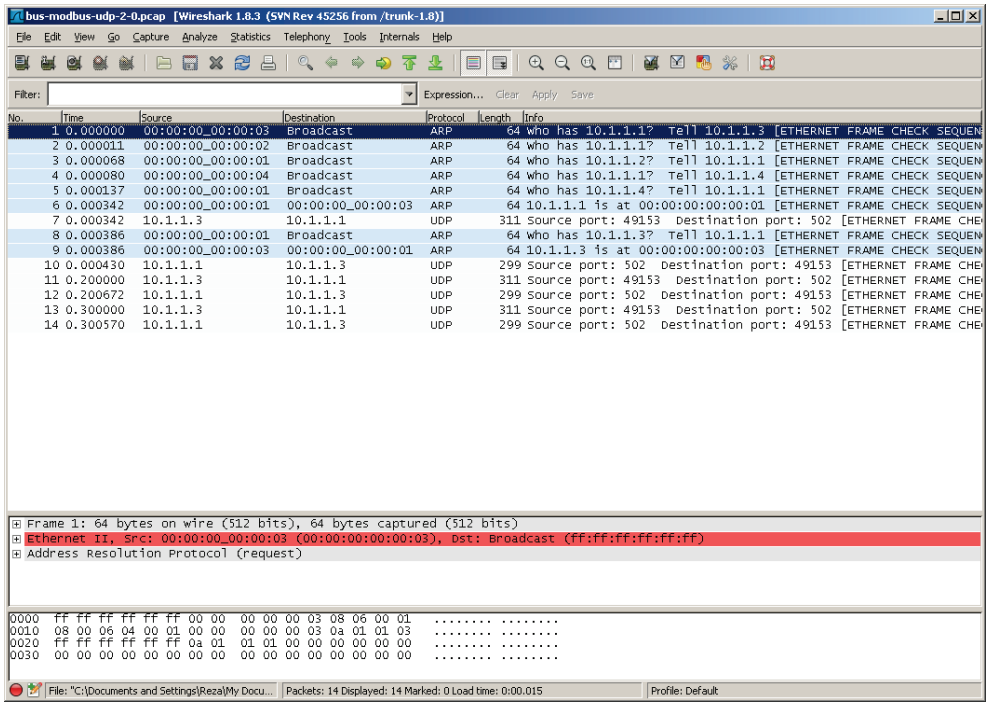

Figure B7. busModbusUdp scenario Wireshark capture node2 (client)

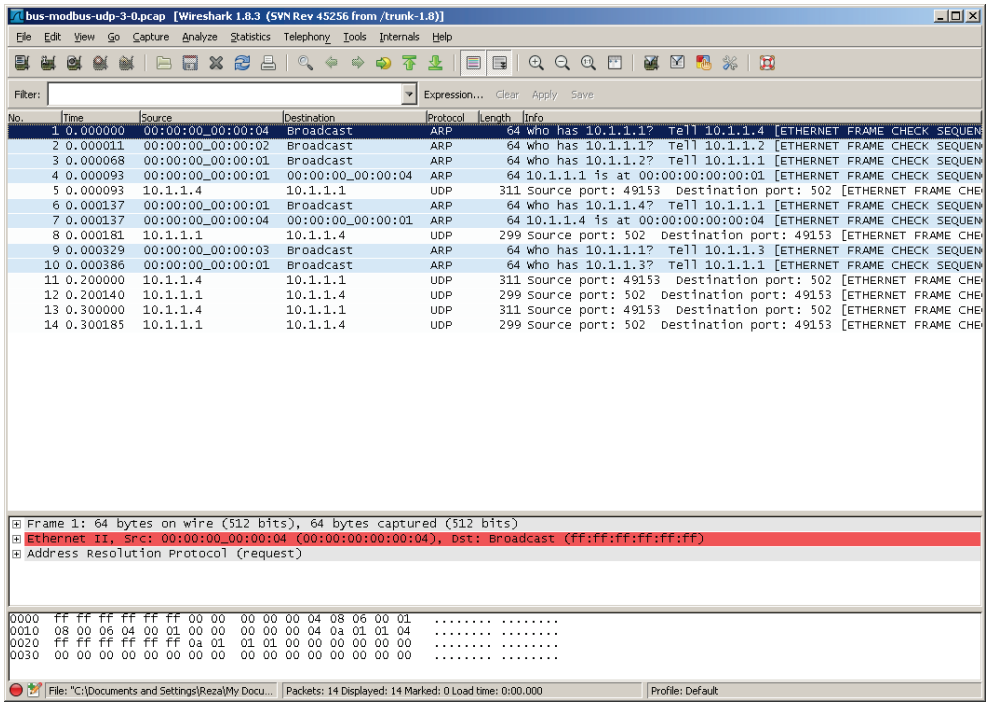

Figure B8. busModbusUdp scenario Wireshark capture node3 (client)

busModbusTcp scenario

This scenario includes four nodes connected over a LAN with network address of 10.1.1.0. The connection between the four nodes is CSMA with a data rate of 100 Mbps and a delay of 6560 nanoseconds. Node0 is a Modbus TCP server, whereas, node1, 2, and 3 are Modbus TCP clients with 10.1.1.1, 10.1.1.2, 10.1.1.3, and 10.1.1.4 ipaddresses respectively. The whole simulation length is 10 seconds, but the server node application starts after one second and the client nodes application start after two seconds. The client nodes send three different Modbus requests, which are Encapsulated Interface Transport of Canopen General, Read Device ID Basic, and Read Device ID Specific, after a 100 ms, 200 ms, and 100 ms delay respectively. Please see appendix E for more details.

The Figure 46 shows the beginning and the Figure 47 shows the end of the capture for node0; which is the server, whereas, Figure 48, 49, and 50 show the clients' captures.

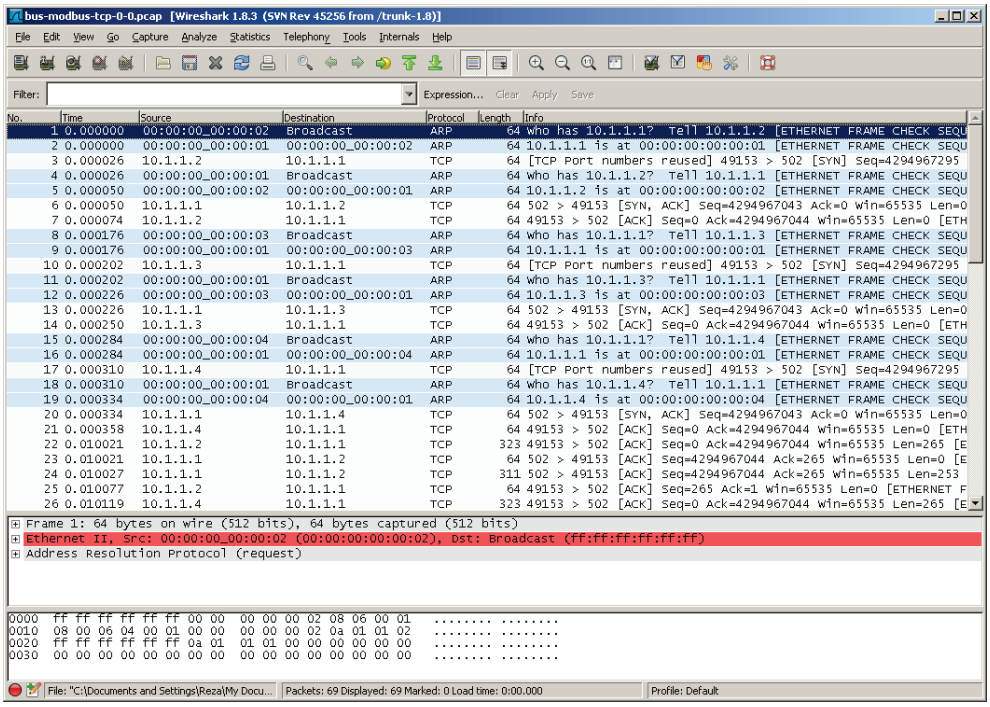

Figure B9. busModbusTcp scenario Wireshark capture node0 (server)

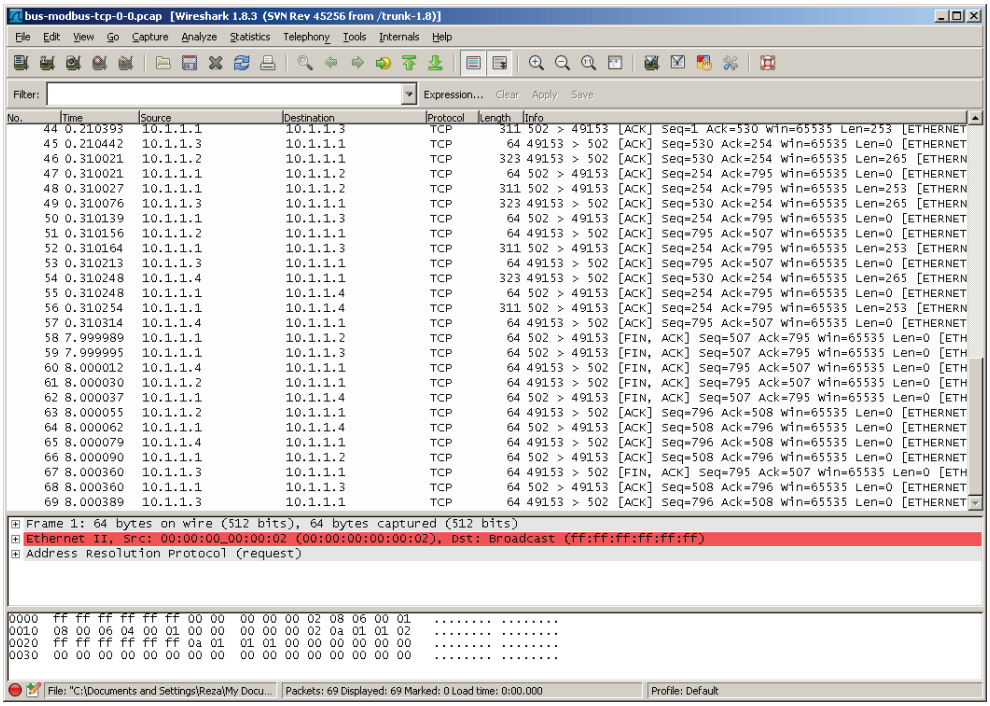

Figure B10. busModbusTcp scenario Wireshark capture node0 (server) continuation.

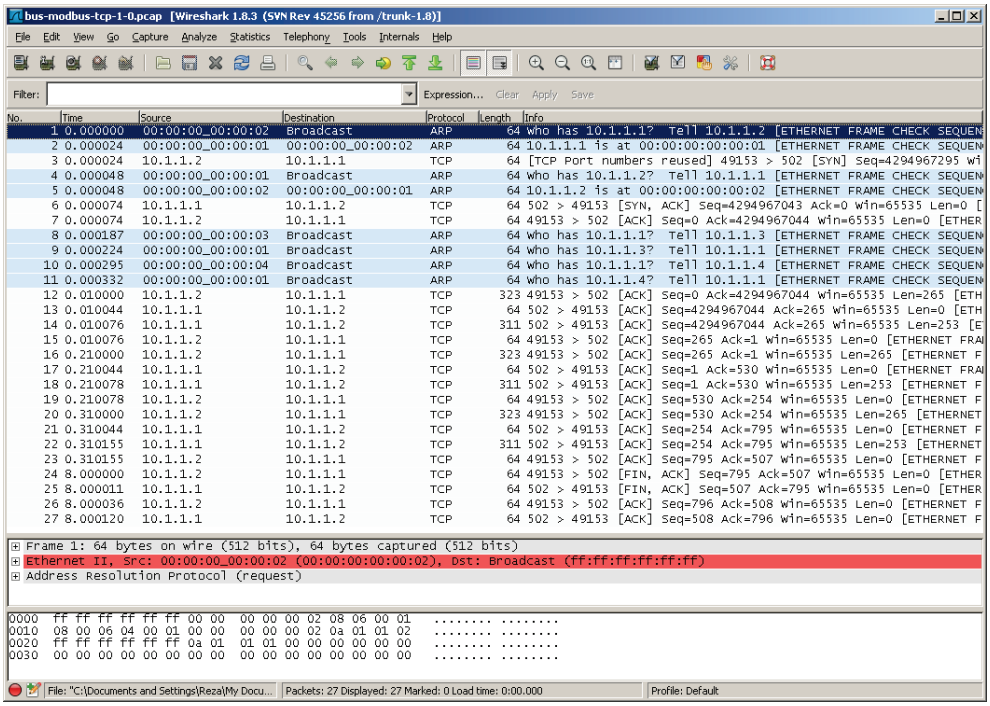

Figure B11. busModbusTcp scenario Wireshark capture node1 (client)

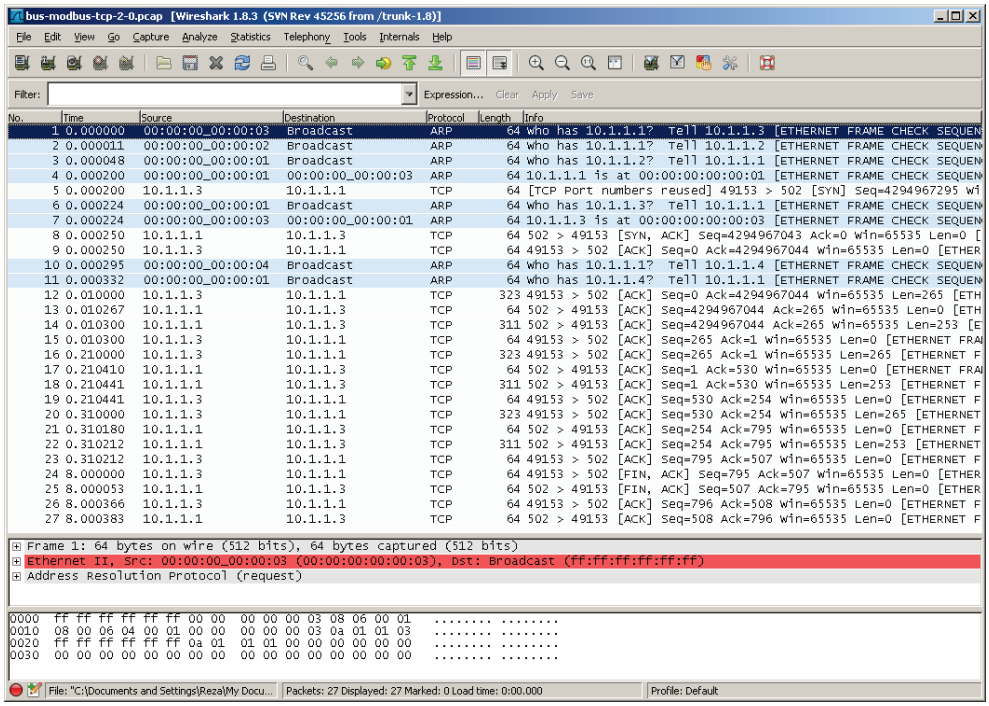

Figure B12. busModbusTcp scenario Wireshark capture node2 (client)

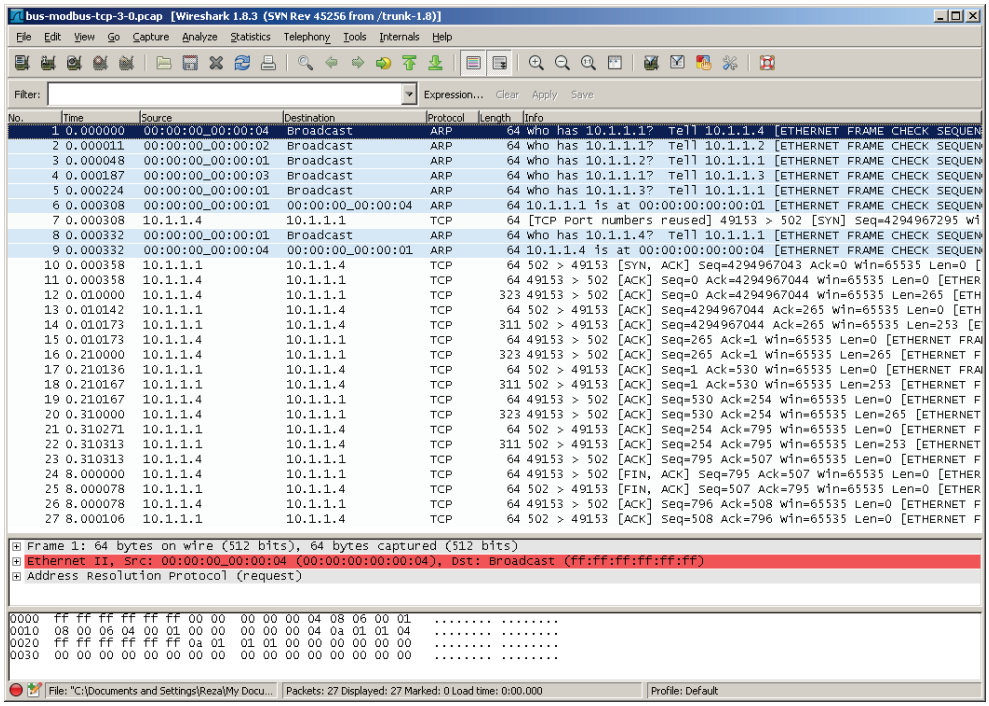

Figure B13. busModbusTcp scenario Wireshark capture node3 (client)

busModbusTcpScenario1 scenario

This scenario includes two nodes connected over a LAN with network address of 10.1.1.0. The connection between the four nodes is CSMA with a data rate of 100 Mbps and a delay of 6560 nanoseconds. Node0 is a Modbus TCP server, whereas, node1 is Modbus TCP client with 10.1.1.1, and 10.1.1.2 ipaddresses respectively. The whole simulation length is 10 seconds, but the server node application starts after one second and the client node application starts after two seconds. The client node sends two different Modbus requests, which are Encapsulated Interface Transport of Canopen General, and Read Coils, after a 1 second, and 2 seconds delay respectively. Please see appendix E for more details.

The Figure 51 shows the capture for node0; which is the server, whereas, Figure 52 shows the clients' capture.

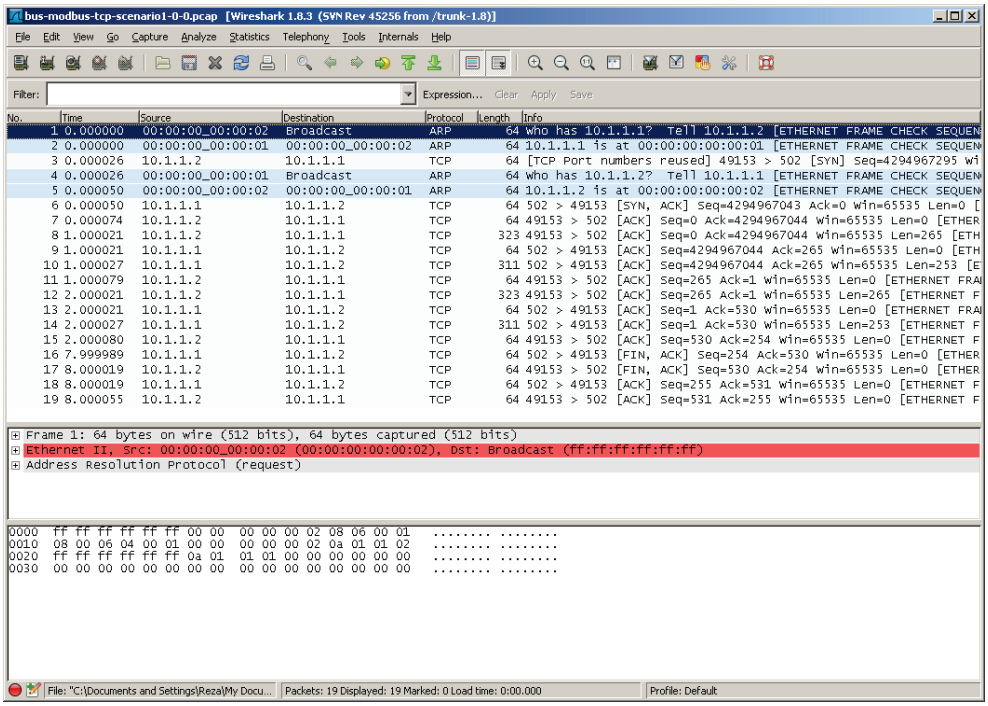

Figure B14. busModbusTcpScenario1 scenario Wireshark capture node0 (server)

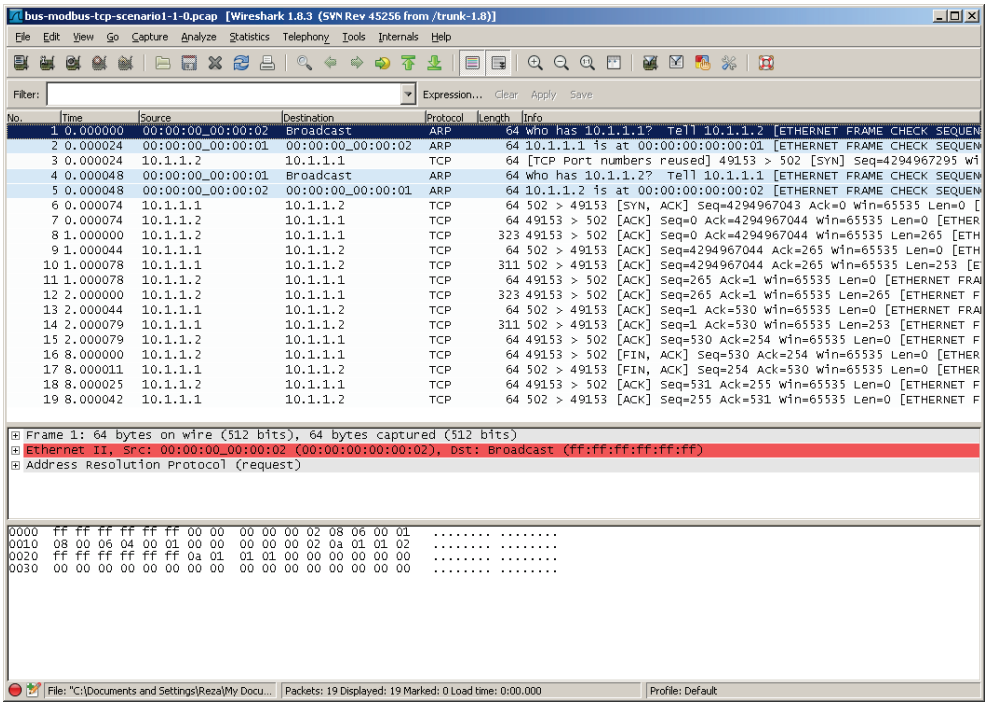

Figure B15. busModbusTcpScenario1 scenario Wireshark capture node1 (client)

starModbusUdp scenario

This scenario includes nine nodes connected over a LAN with network address of 10.1.1.0. The connection between the four nodes is point to point with a data rate of 5 Mbps and a delay of 2 ms. Node0 is a Modbus UDP server, whereas, node1 through 8 are Modbus UDP clients with 10.1.1.1, and 10.1.1.2 through 10.1.1.9 ipaddresses respectively. The whole simulation length is 10 seconds, but the server node application starts after one second and the client nodes application start after two seconds. The client nodes send three different Modbus requests, which are Encapsulated Interface Transport of Canopen General, Read Device ID Basic, and Read Device ID Specific, after a 100 ms, 200 ms, and 100 ms delay respectively. Please see appendix E for more details.

The Figure 53 through Figure 60 show the capture for node0; which is the server, whereas, Figure 61 through Figure A12 show the clients' captures.

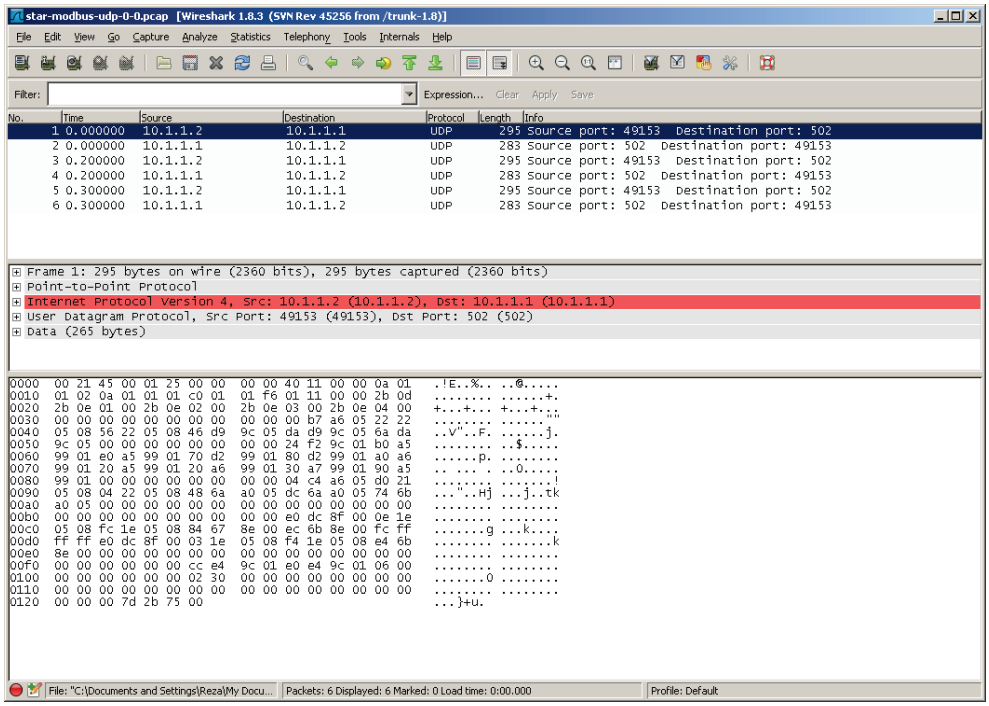

Figure B16. starModbusUdp scenario Wireshark capture node0 (server) to node1

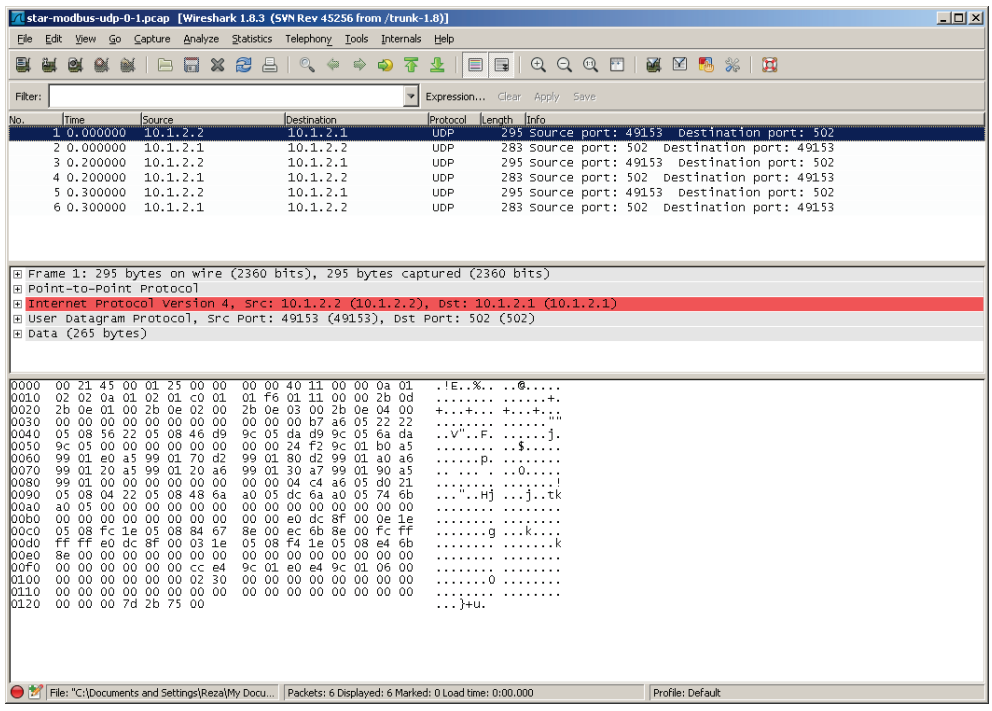

Figure B17. starModbusUdp scenario Wireshark capture node0 (server) to node2

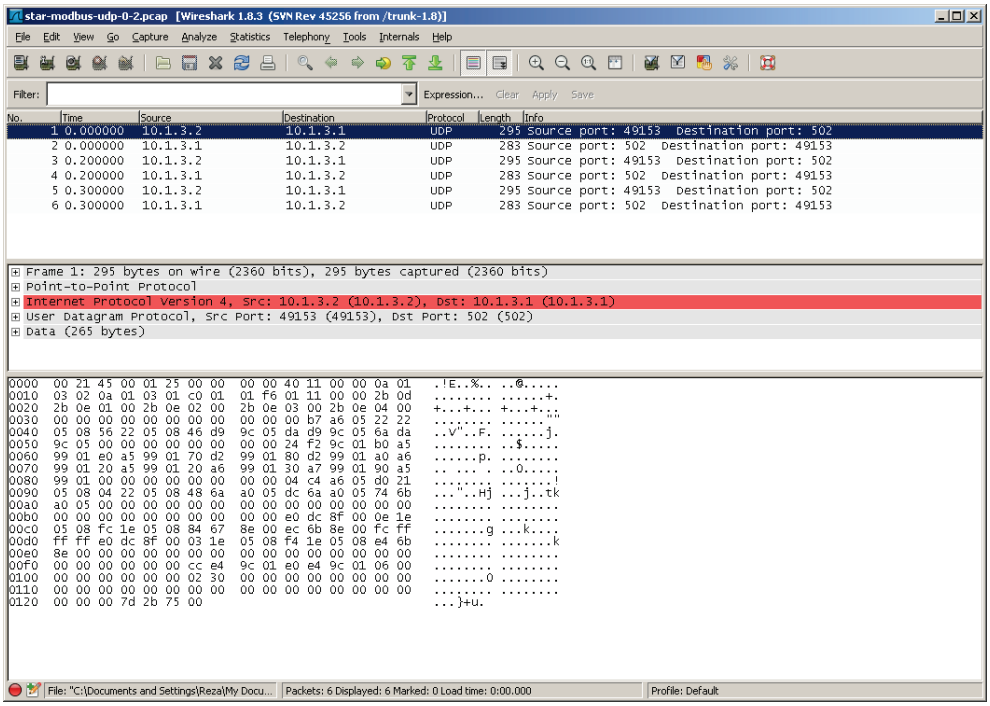

Figure B18. starModbusUdp scenario Wireshark capture node0 (server) to node2

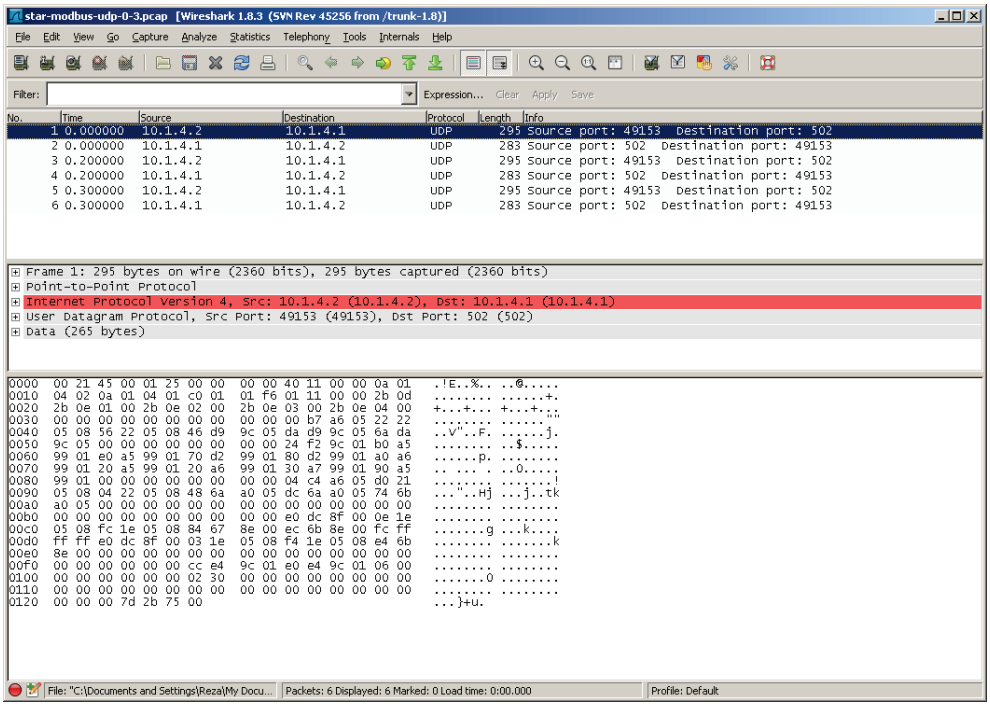

Figure B19. starModbusUdp scenario Wireshark capture node0 (server) to node3

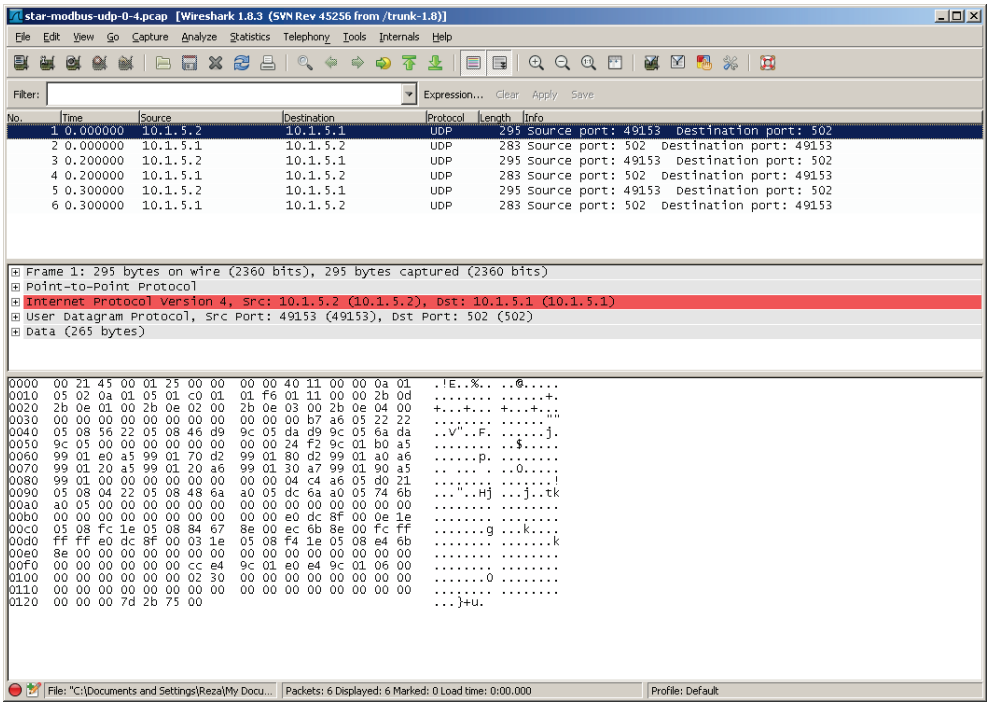

Figure B20. starModbusUdp scenario Wireshark capture node0 (server) to node4

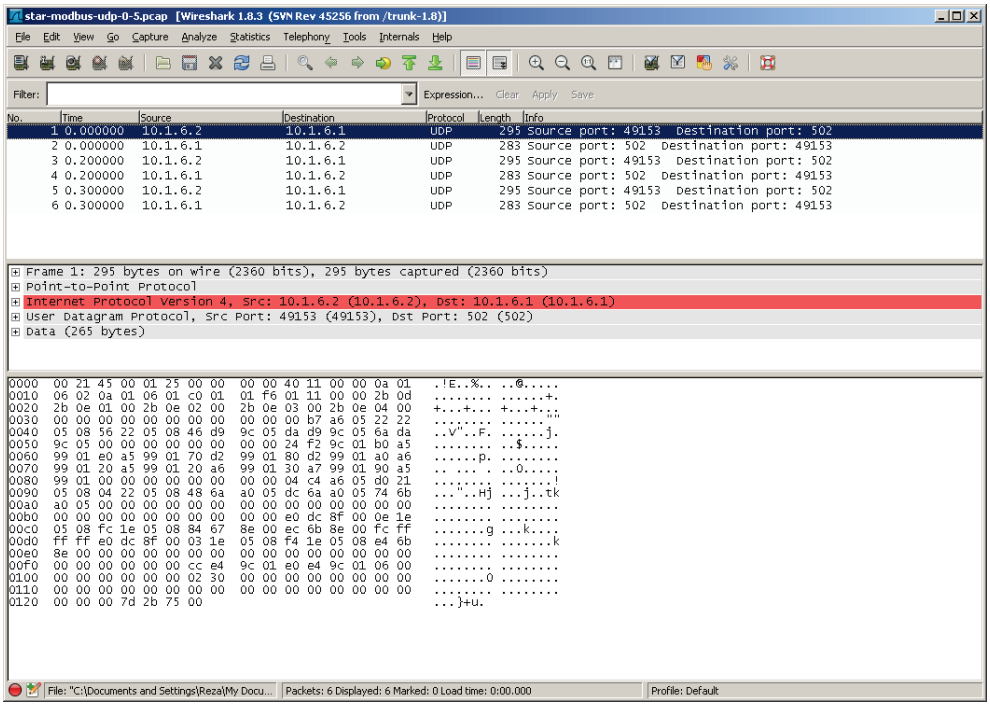

Figure B21. starModbusUdp scenario Wireshark capture node0 (server) to node5

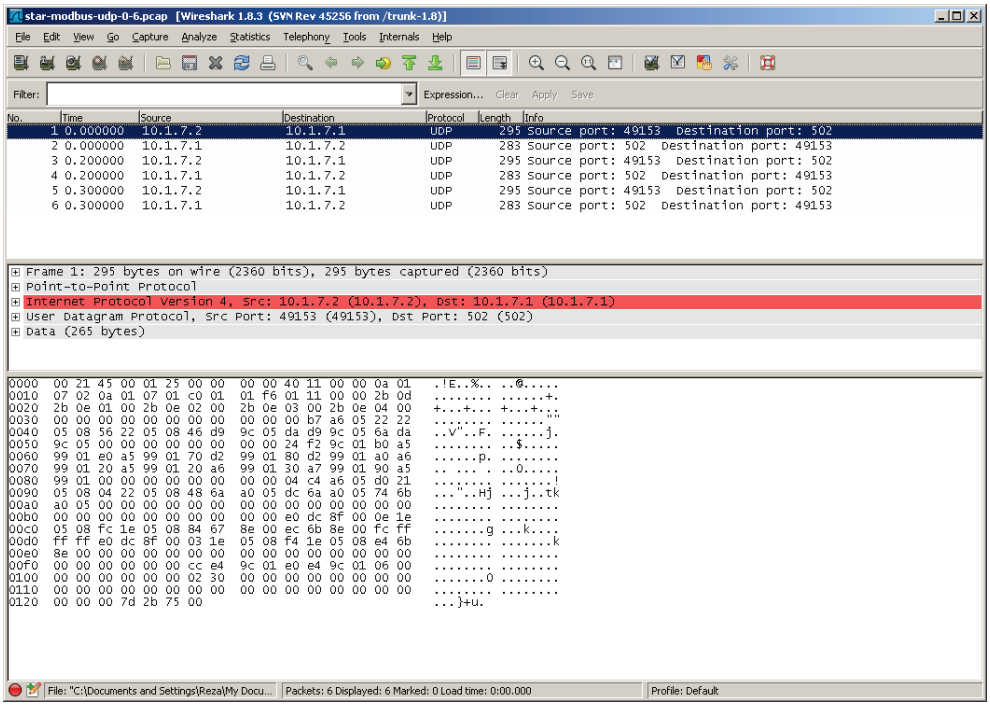

Figure B22. starModbusUdp scenario Wireshark capture node0 (server) to node6

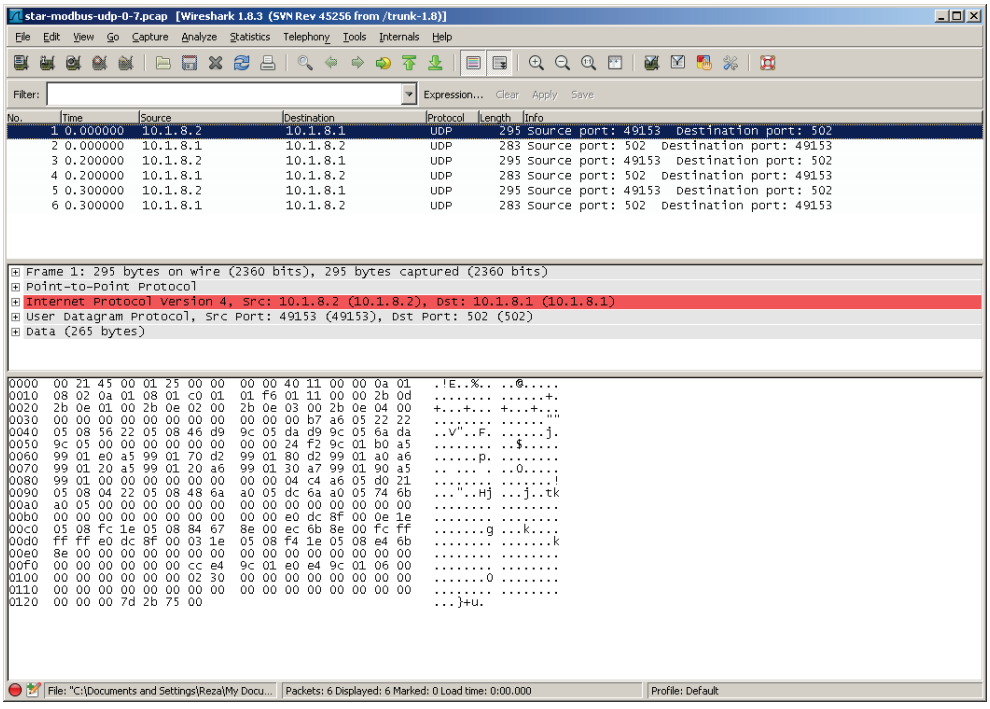

Figure B23. starModbusUdp scenario Wireshark capture node0 (server) to node7

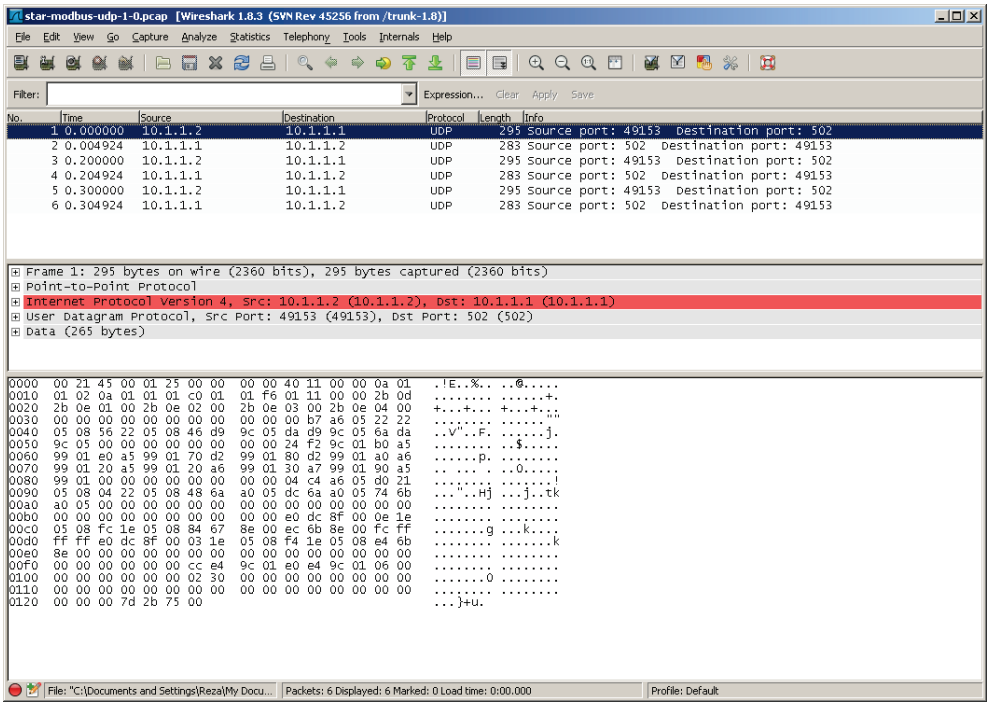

Figure B24. starModbusUdp scenario Wireshark capture node1 (client)

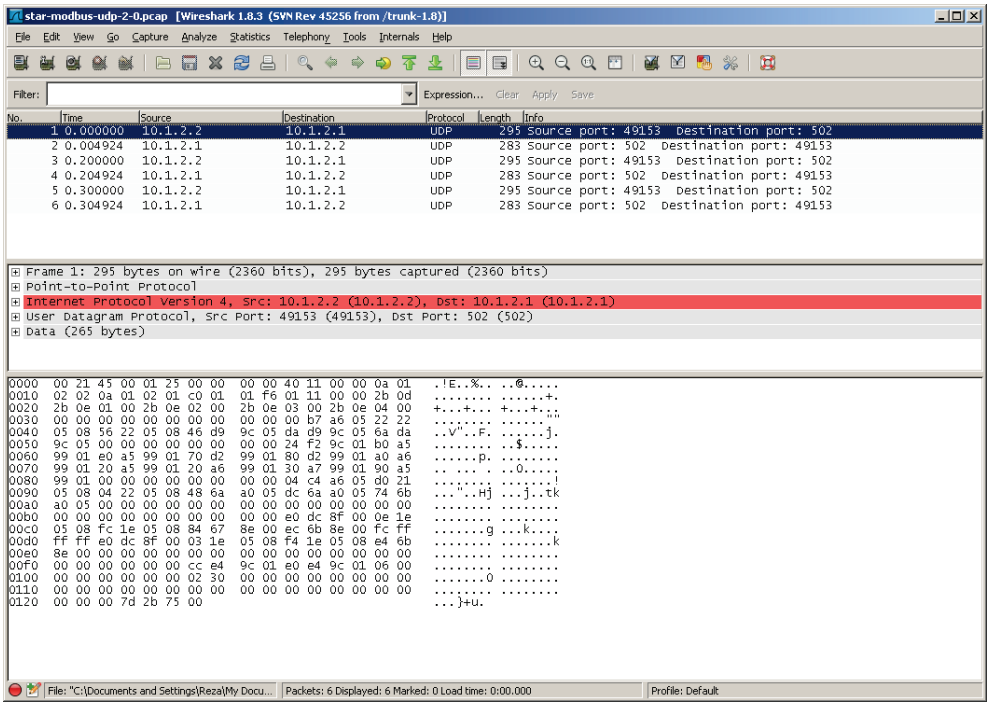

Figure B25. starModbusUdp scenario Wireshark capture node2 (client)

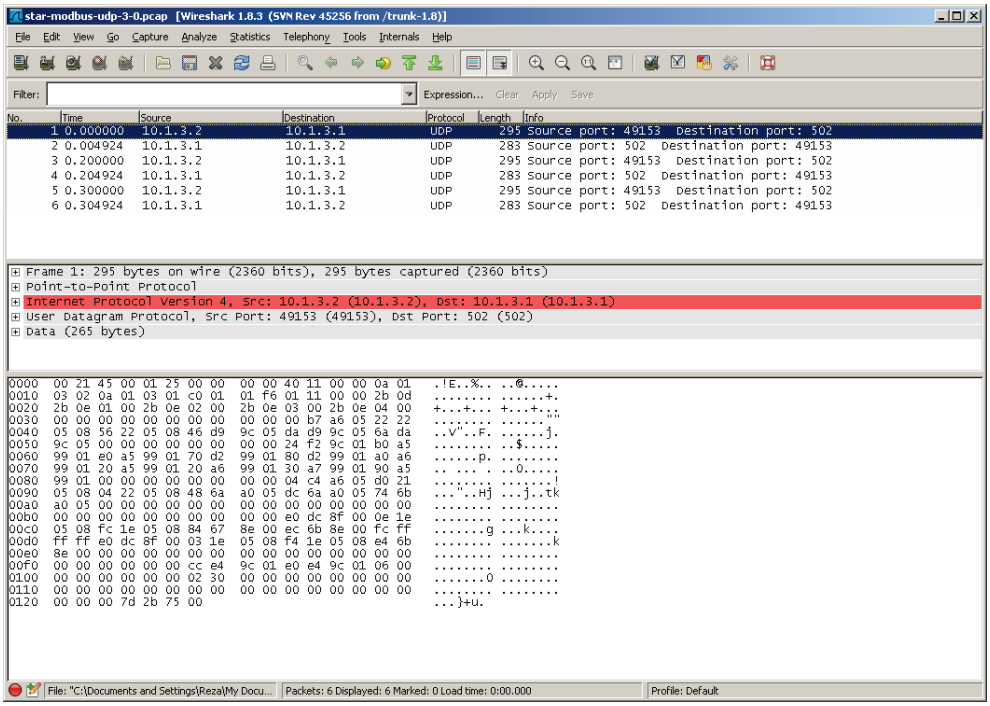

Figure B26. starModbusUdp scenario Wireshark capture node3 (client)

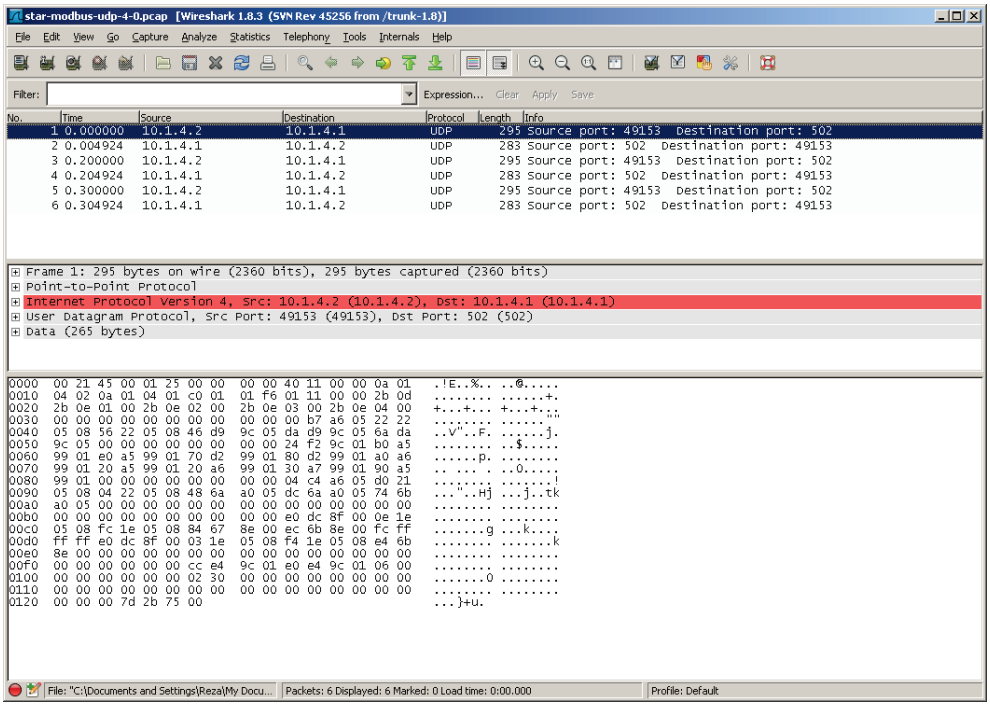

Figure B27. starModbusUdp scenario Wireshark capture node4 (client)

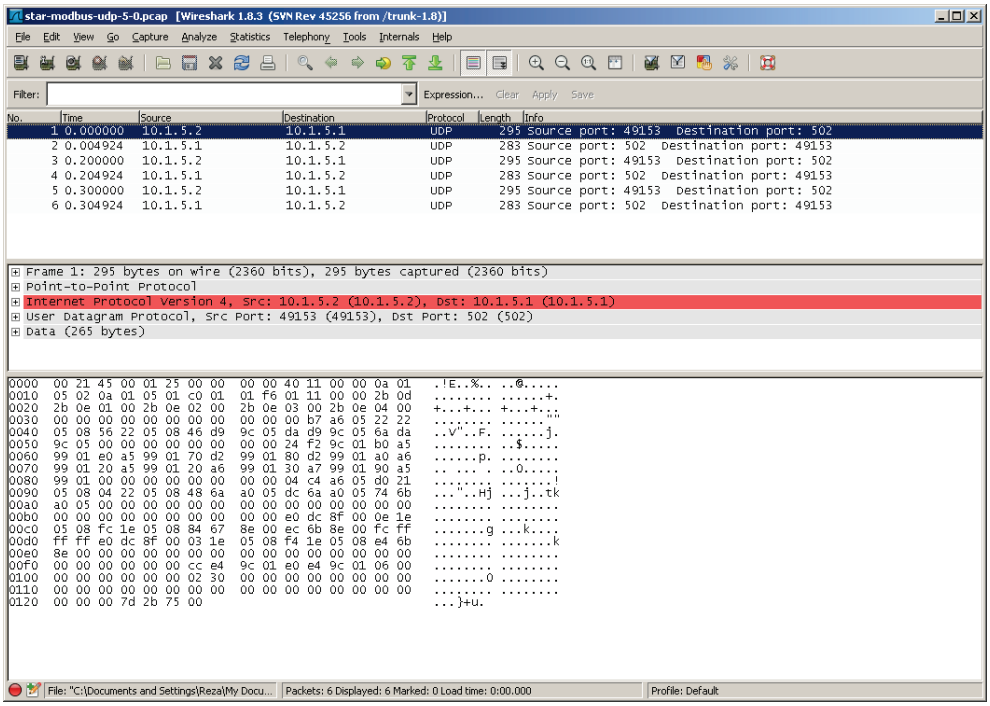

Figure B28. starModbusUdp scenario Wireshark capture node5 (client)

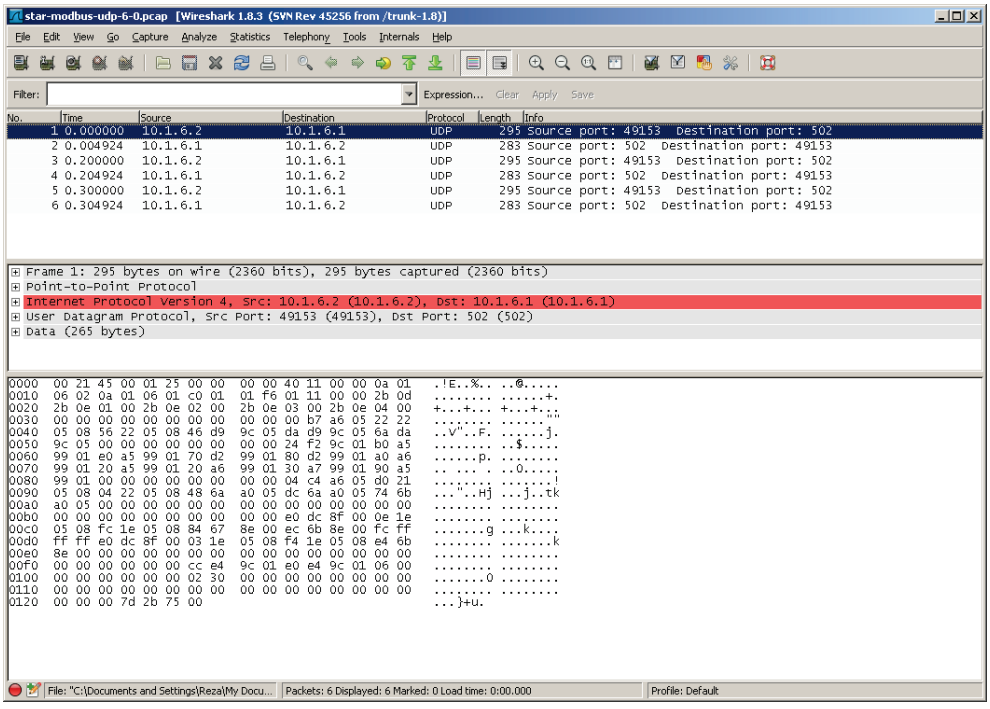

Figure B29. starModbusUdp scenario Wireshark capture node6 (client)

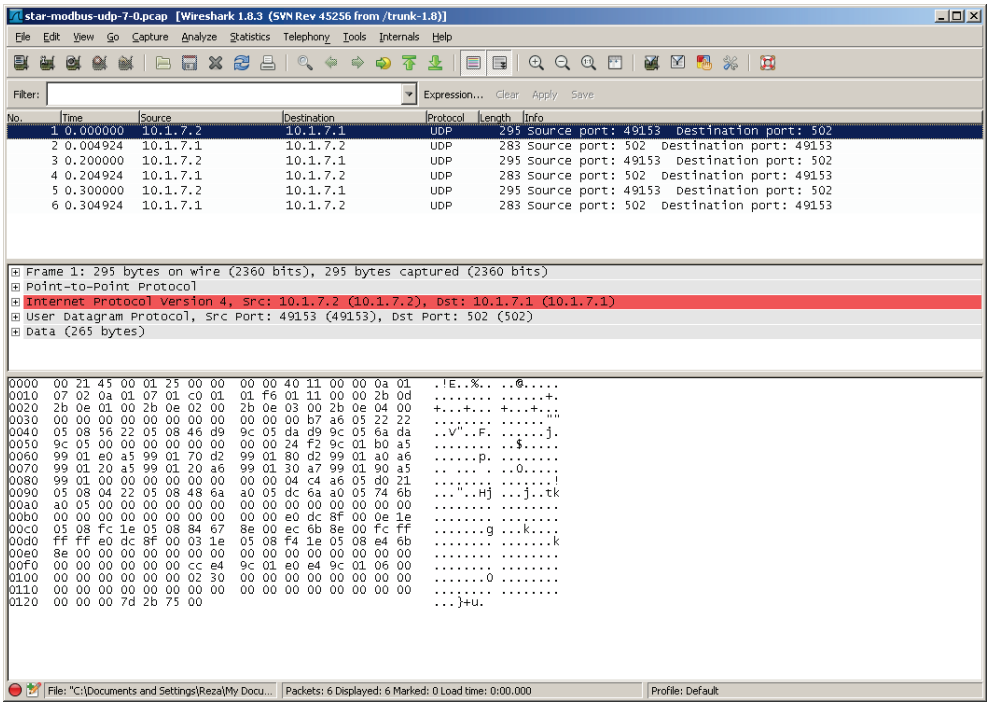

Figure B30. starModbusUdp scenario Wireshark capture node7 (client)

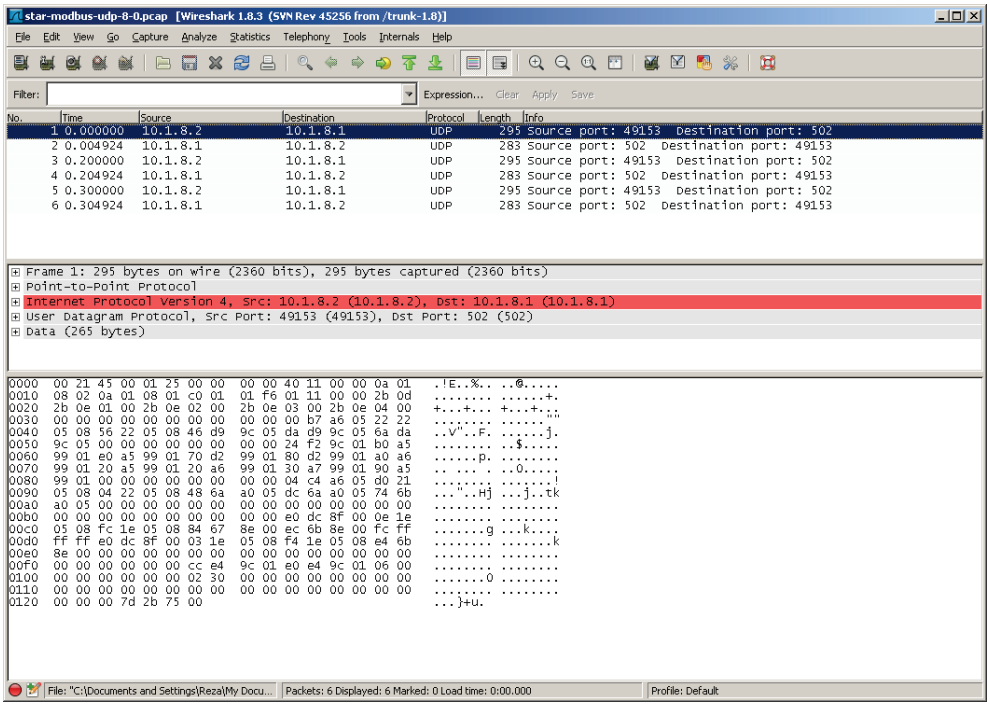

Figure B31. starModbusUdp scenario Wireshark capture node8 (client)

starModbusTcp scenario

This scenario includes nine nodes connected over a LAN with network address of 10.1.1.0. The connection between the four nodes is point to point with a data rate of 5 Mbps and a delay of 2 ms. Node0 is a Modbus TCP server, whereas, node1 through 8 are Modbus TCP clients with 10.1.1.1, and 10.1.1.2 through 10.1.1.9 ipaddresses respectively. The whole simulation length is 10 seconds, but the server node application starts after one second and the client nodes application start after two seconds. The client nodes send three different Modbus requests, which are Encapsulated Interface Transport of Canopen General, Read Device ID Basic, and Read Device ID Specific, after a 100 ms, 200 ms, and 100 ms delay respectively. Please see appendix E for more details.

The Figure 69 through Figure 77 show the capture for node0; which is the server, whereas, Figure 78 through Figure 85 show the clients' captures.

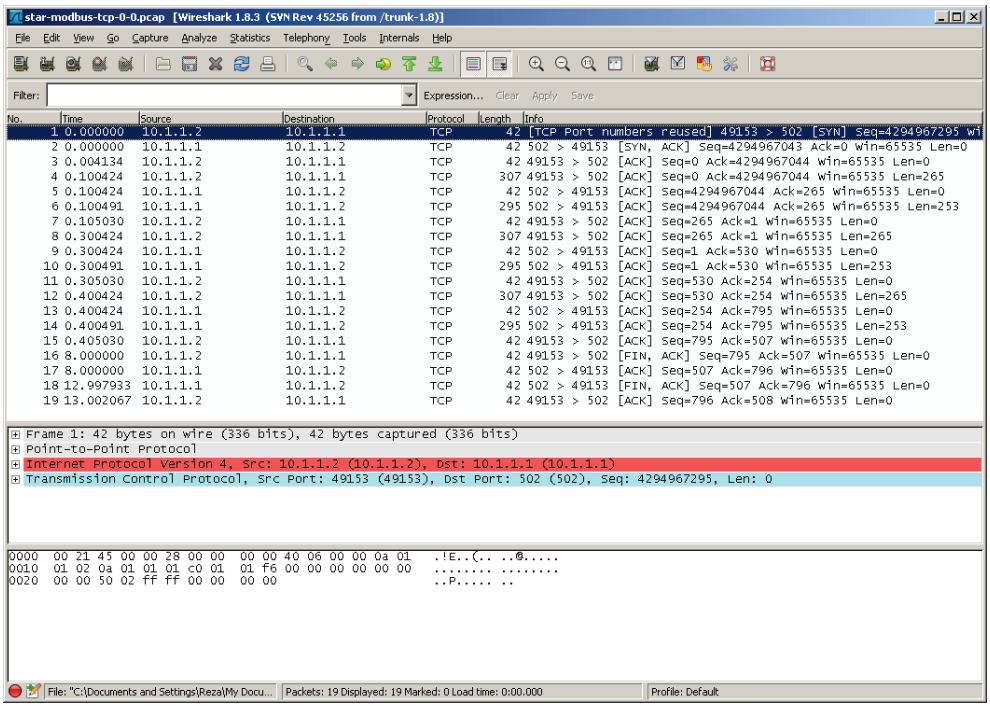

Figure B32. starModbusTcp scenario Wireshark capture node0 (server) to node1

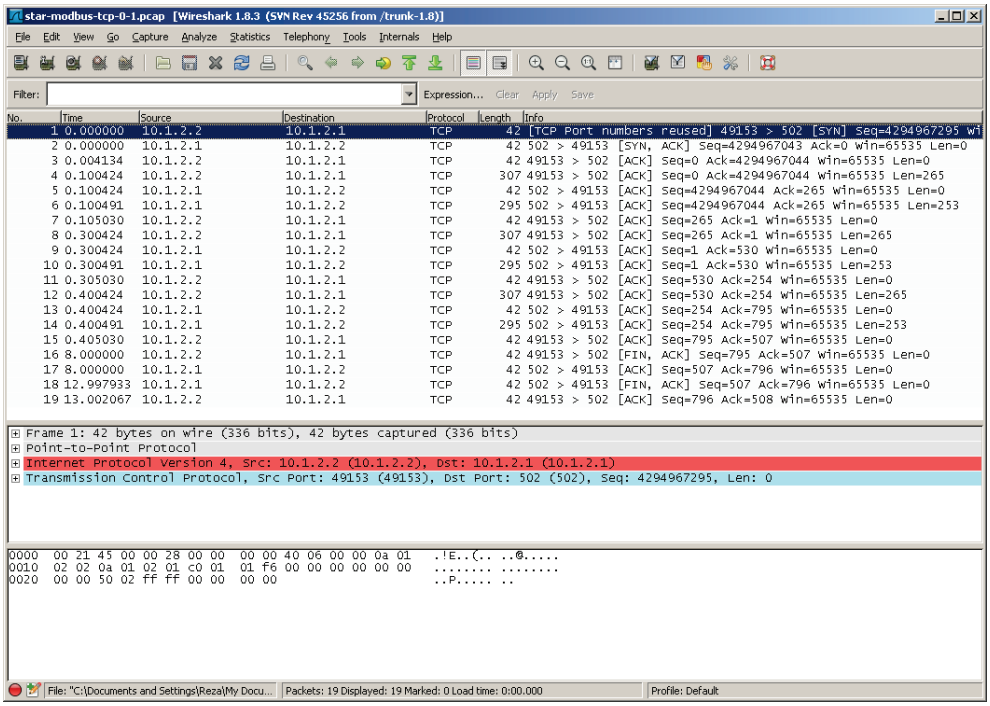

Figure B33. starModbusTcp scenario Wireshark capture node0 (server) to node2

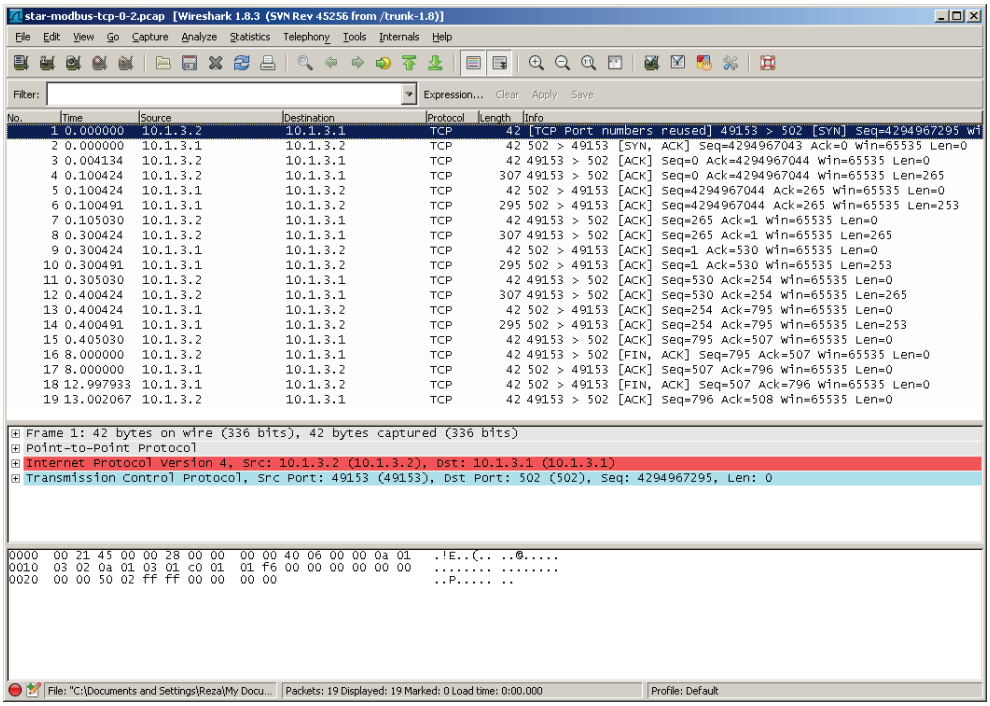

Figure B34. starModbusTcp scenario Wireshark capture node0 (server) to node3

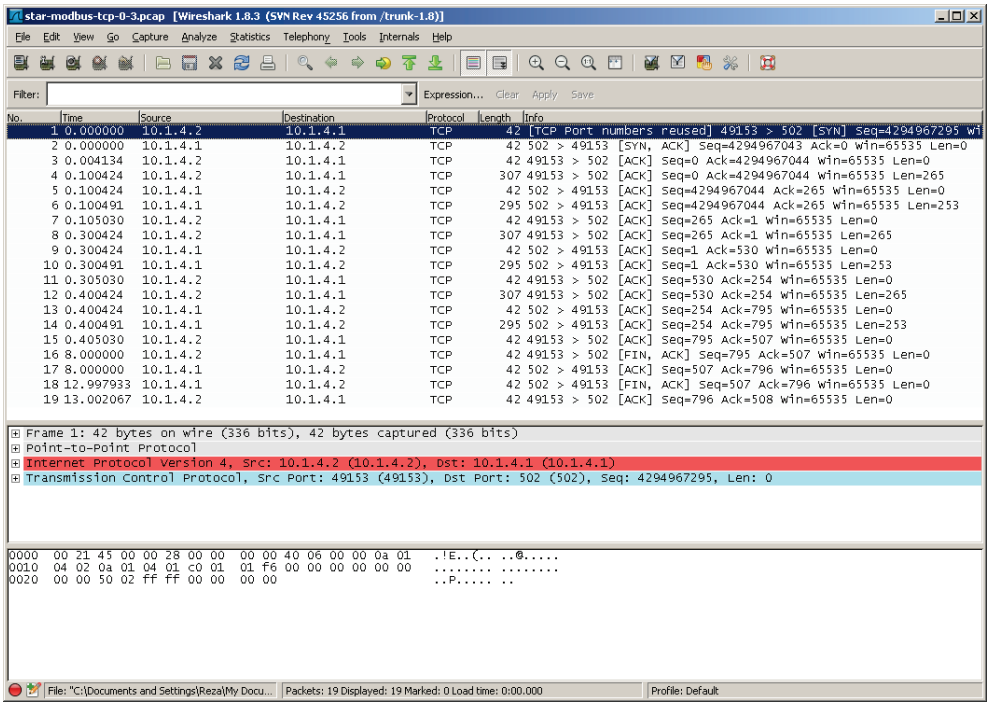

Figure B35. starModbusTcp scenario Wireshark capture node0 (server) to node4

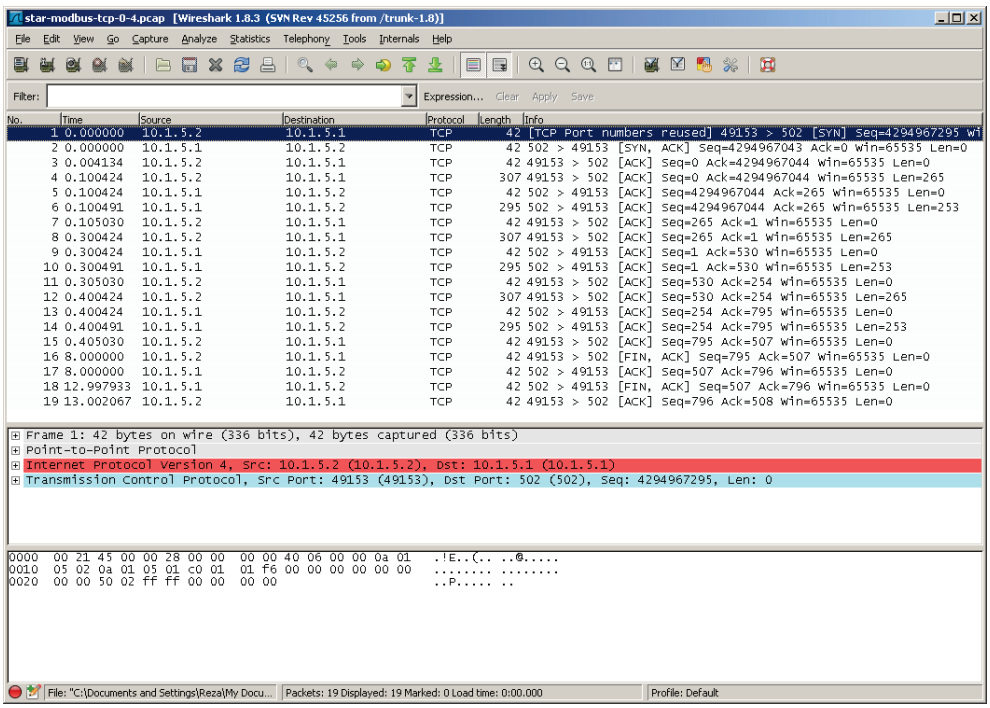

Figure B36. starModbusTcp scenario Wireshark capture node0 (server) to node5

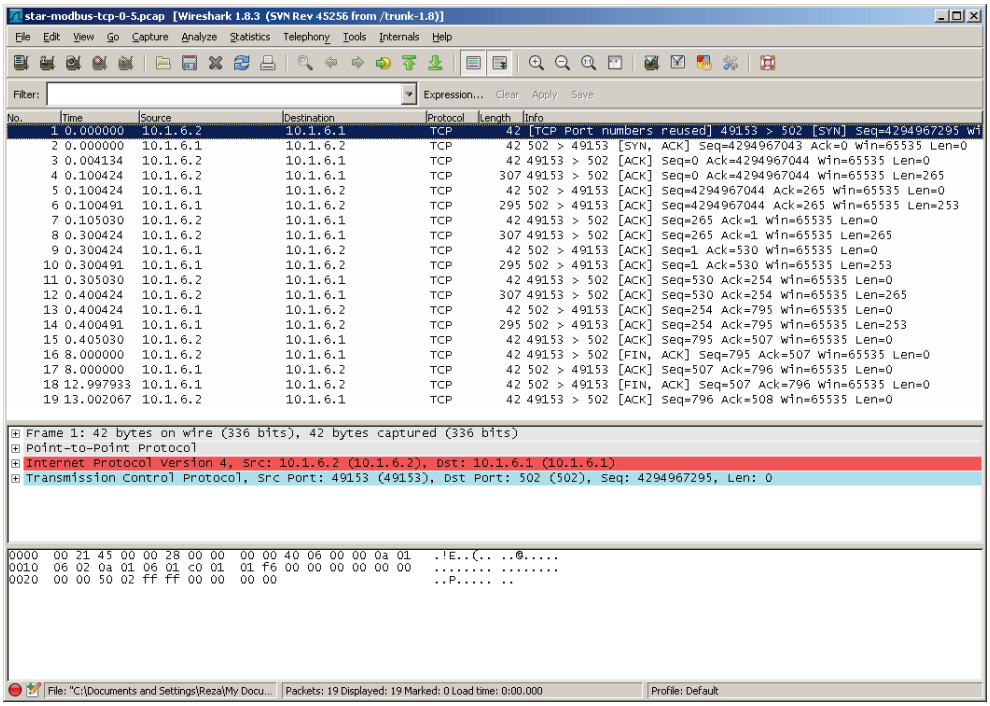

Figure B37. starModbusTcp scenario Wireshark capture node0 (server) to node6

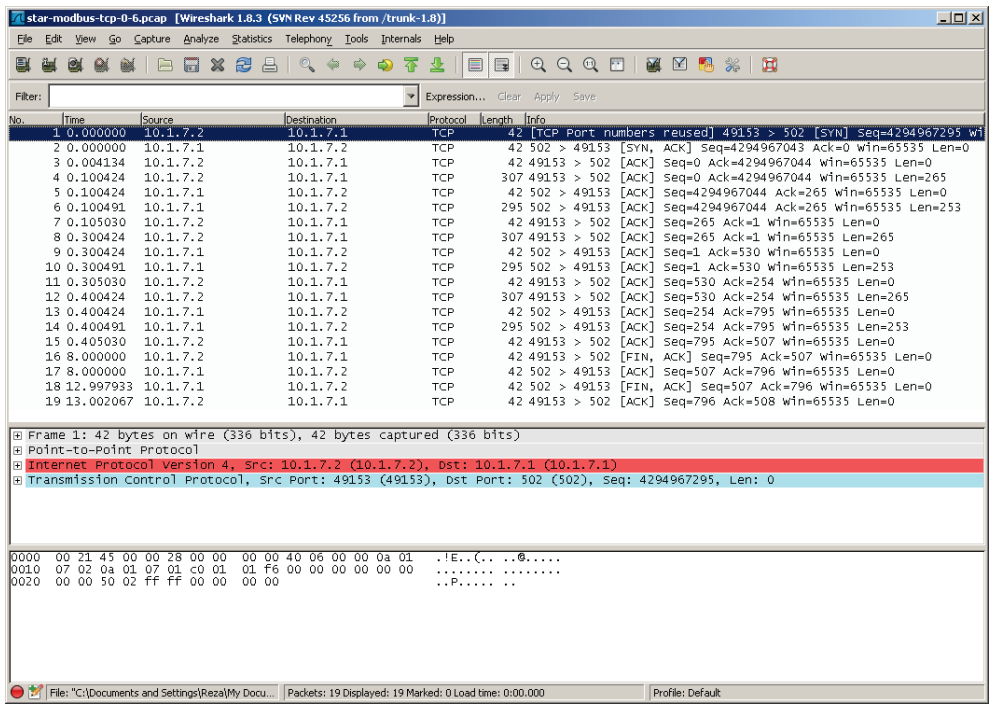

Figure B38. starModbusTcp scenario Wireshark capture node0 (server) to node7

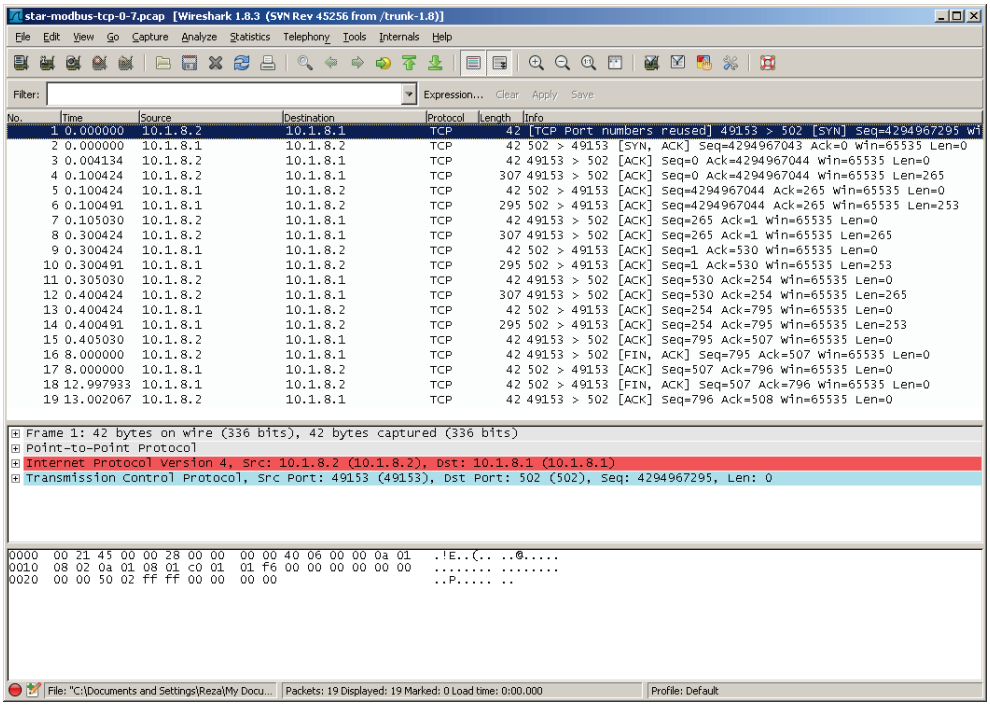

Figure B39. starModbusTcp scenario Wireshark capture node0 (server) to node8

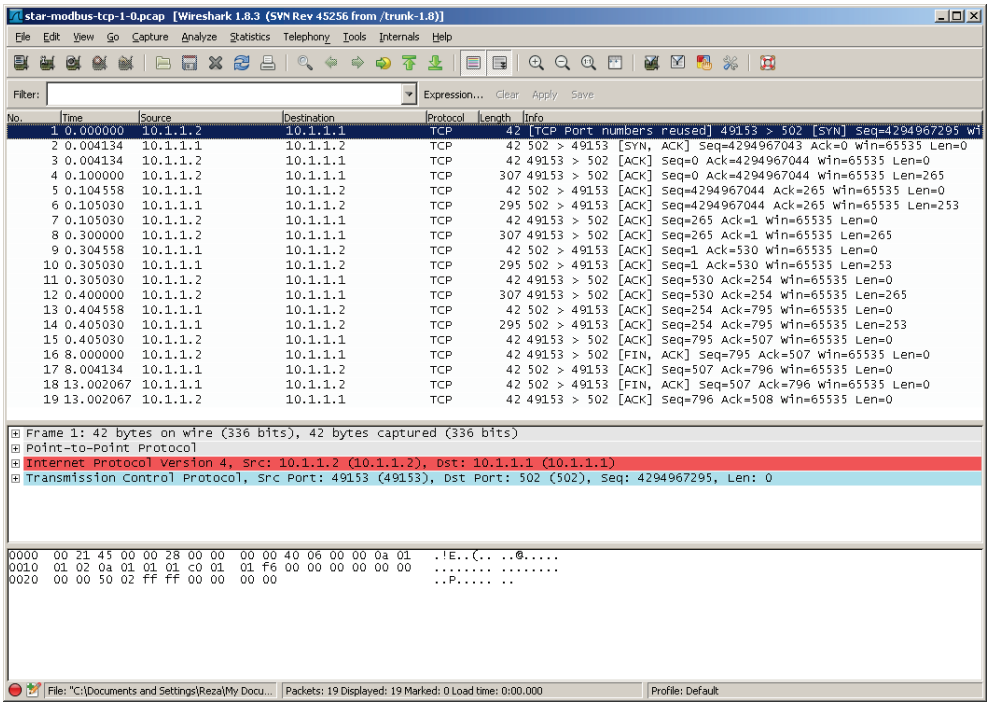

Figure B40. starModbusTcp scenario Wireshark capture node1 (client)

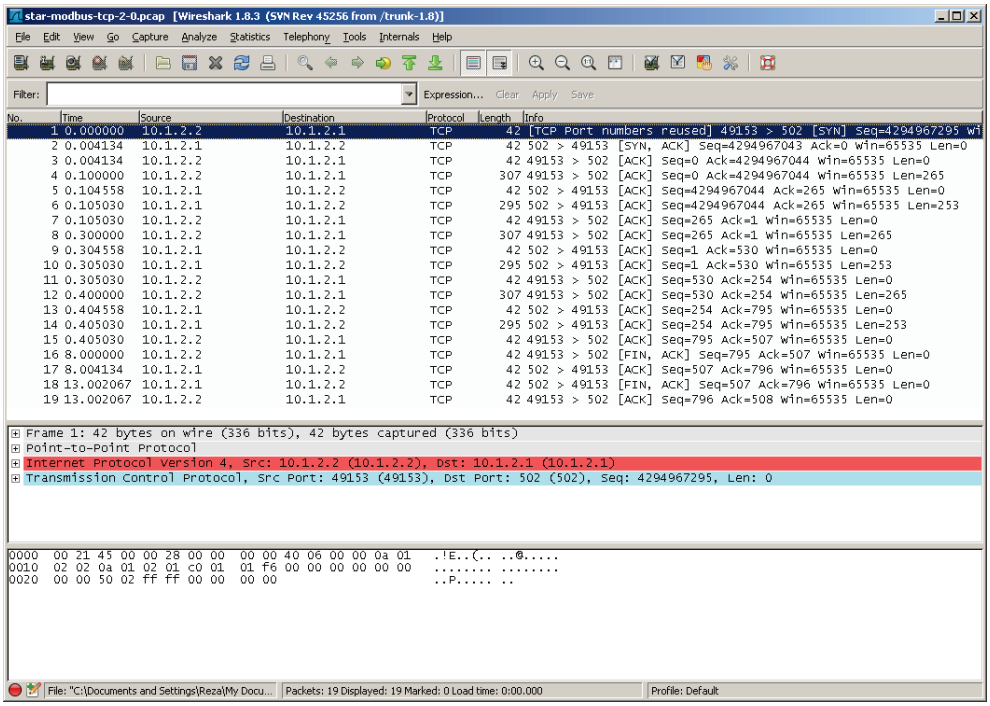

Figure B41. starModbusTcp scenario Wireshark capture node2 (client)

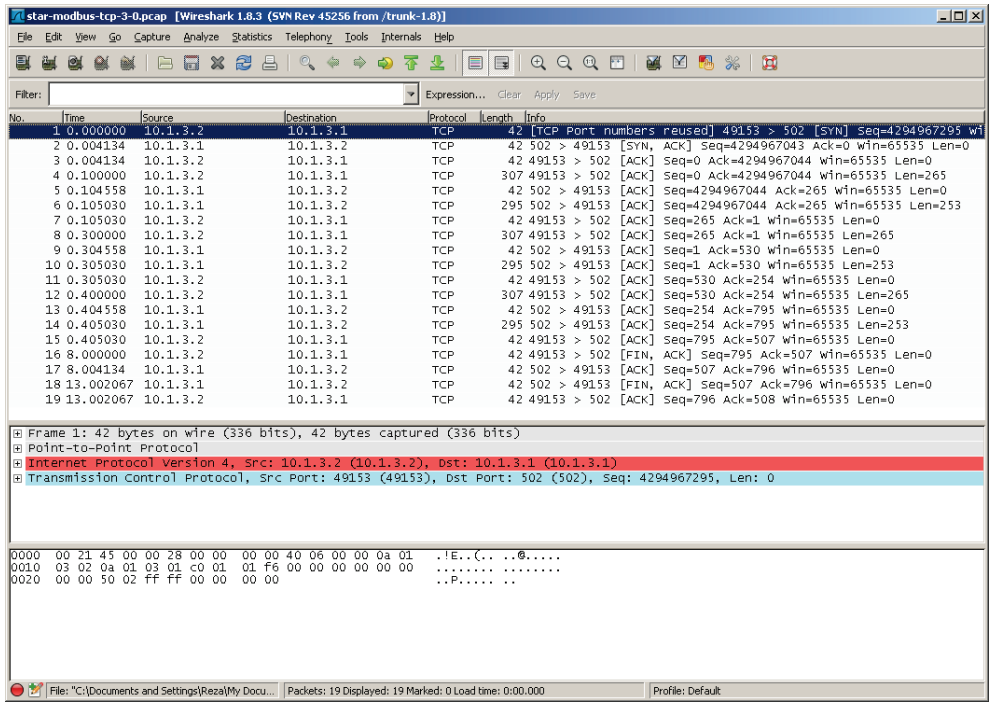

Figure B42. starModbusTcp scenario Wireshark capture node3 (client)

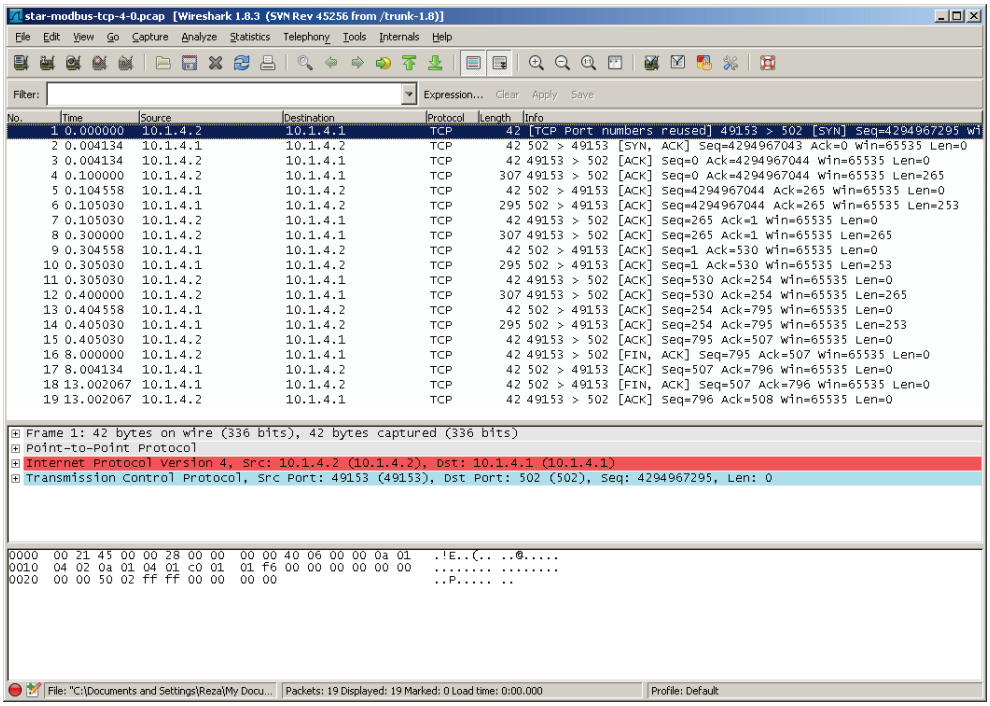

Figure B43. starModbusTcp scenario Wireshark capture node4 (client)

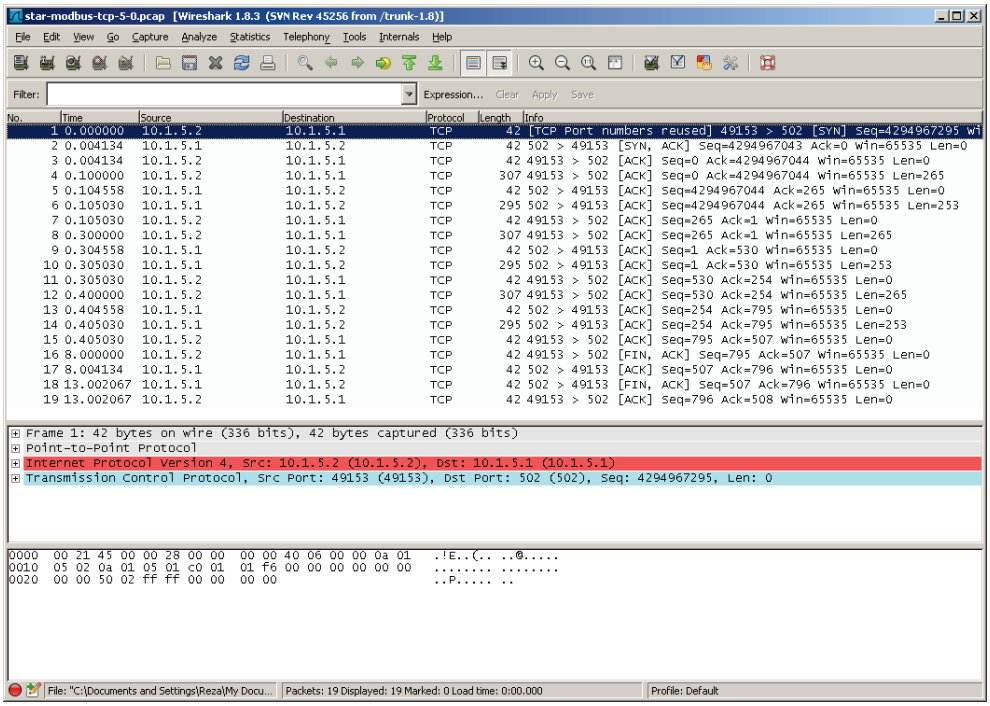

Figure B44. starModbusTcp scenario Wireshark capture node5 (client)

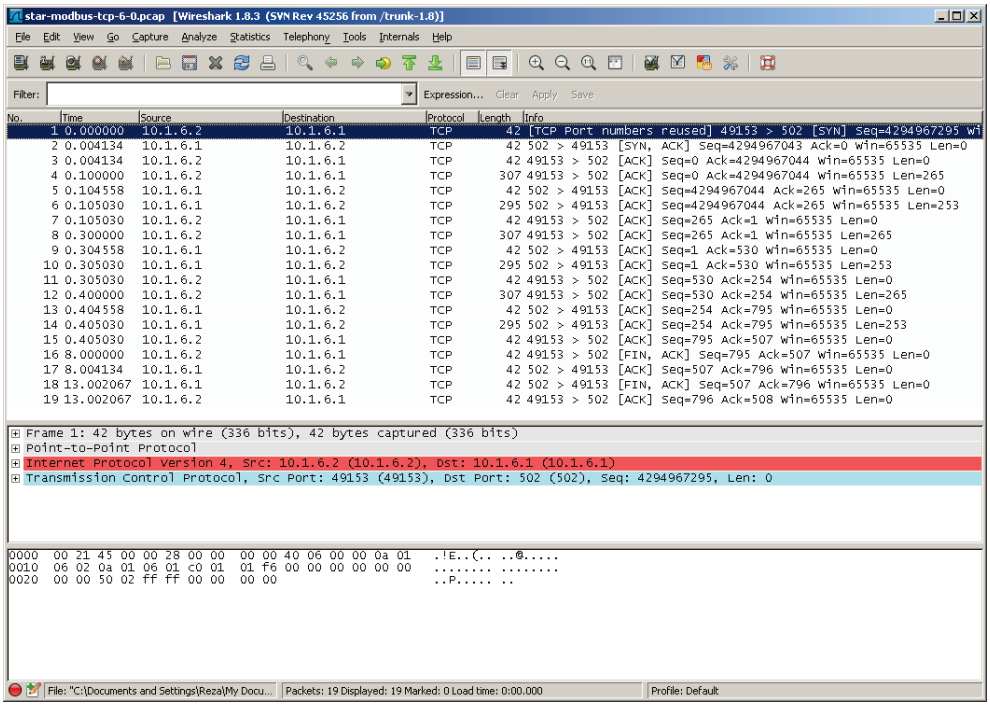

Figure B45. starModbusTcp scenario Wireshark capture node6 (client)

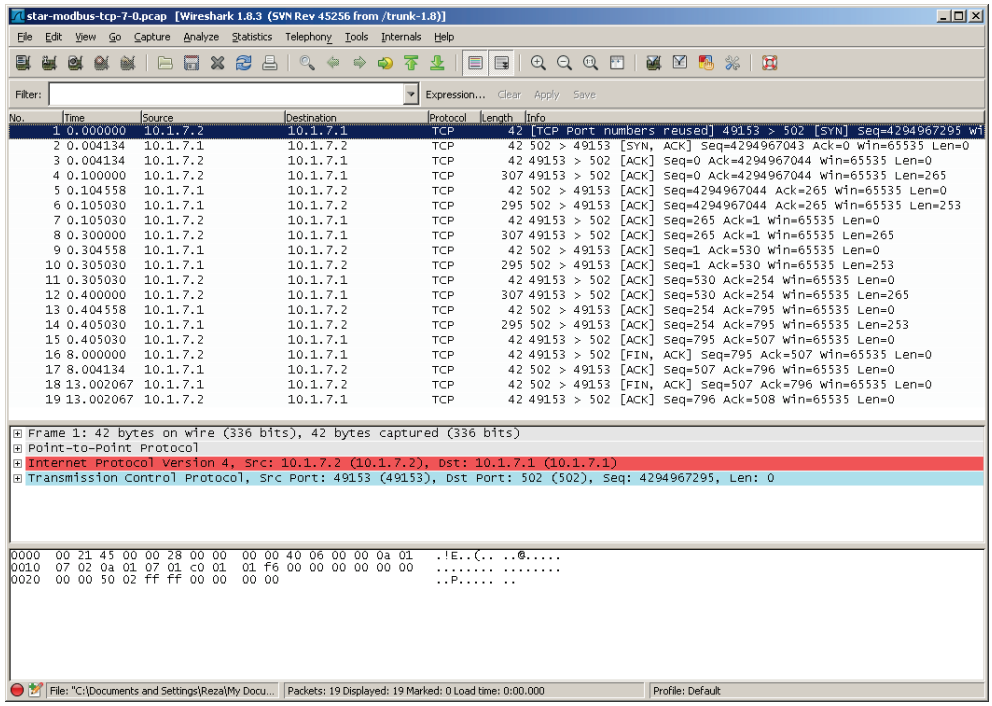

Figure B46. starModbusTcp scenario Wireshark capture node7 (client)

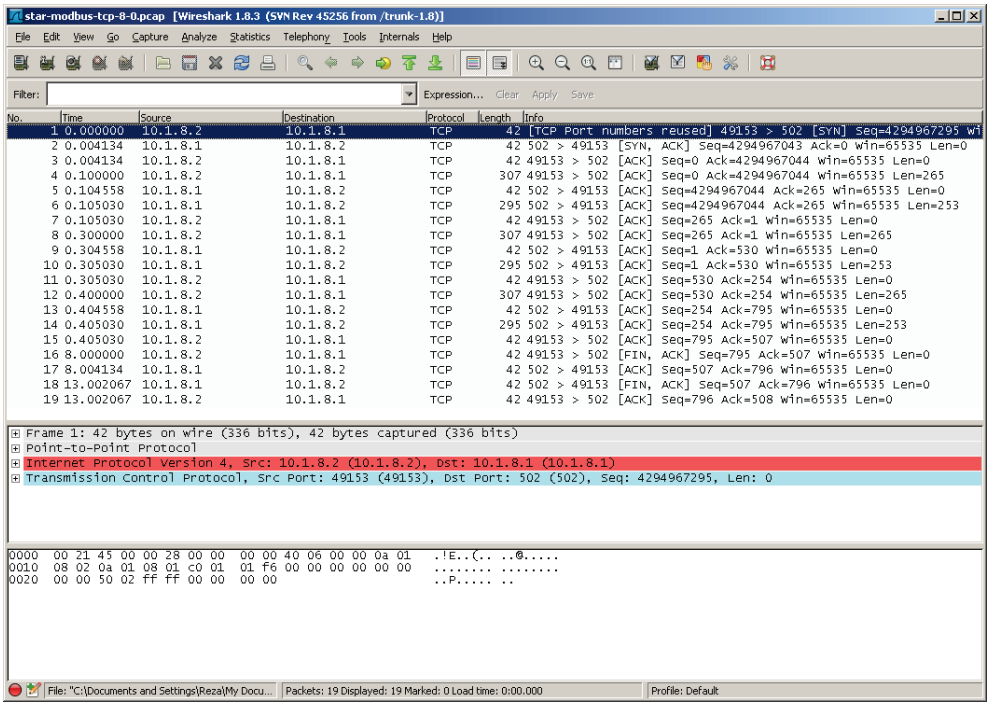

Figure B47. starModbusTcp scenario Wireshark capture node8 (client)

Appendix C.

Traces – NetAnim Screen Shots

Ns-Modbus can generate trace files with extension tr, and xml. There are only one tr and xml file are generated for each scenario. The xml files can be used as an input to NetAnim tool to animate the simulation. In this Appendix, I included the screen shots of NetAnim tool, which are generated from xml files produced by ns-Modbus.

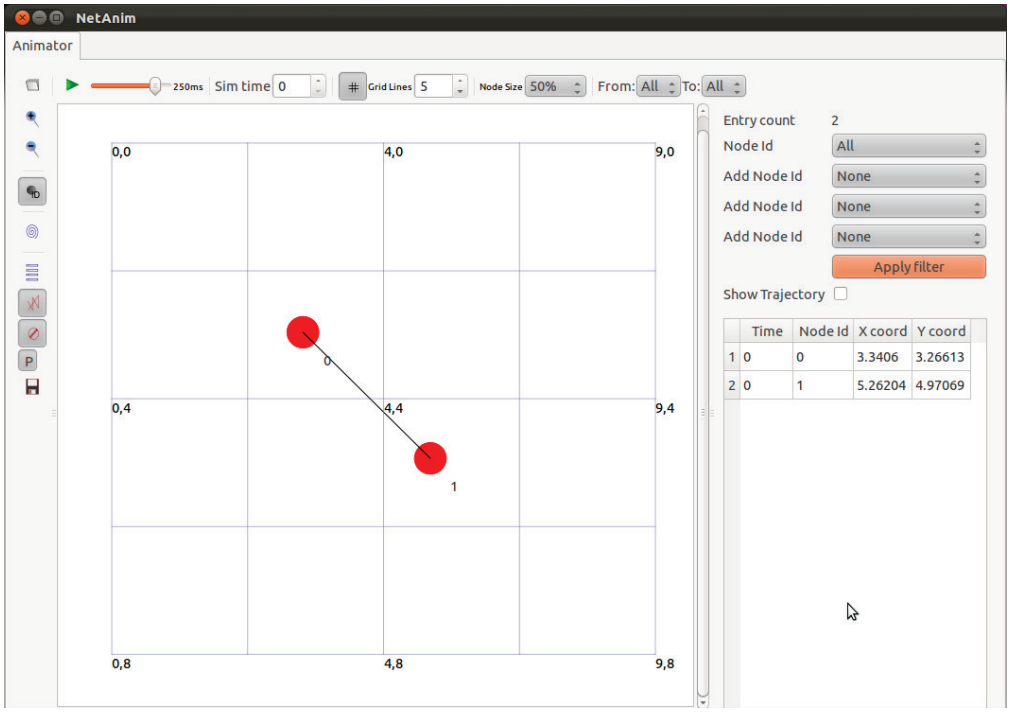

Figure C1. NetAnim, myFirstModbusUdp scenario

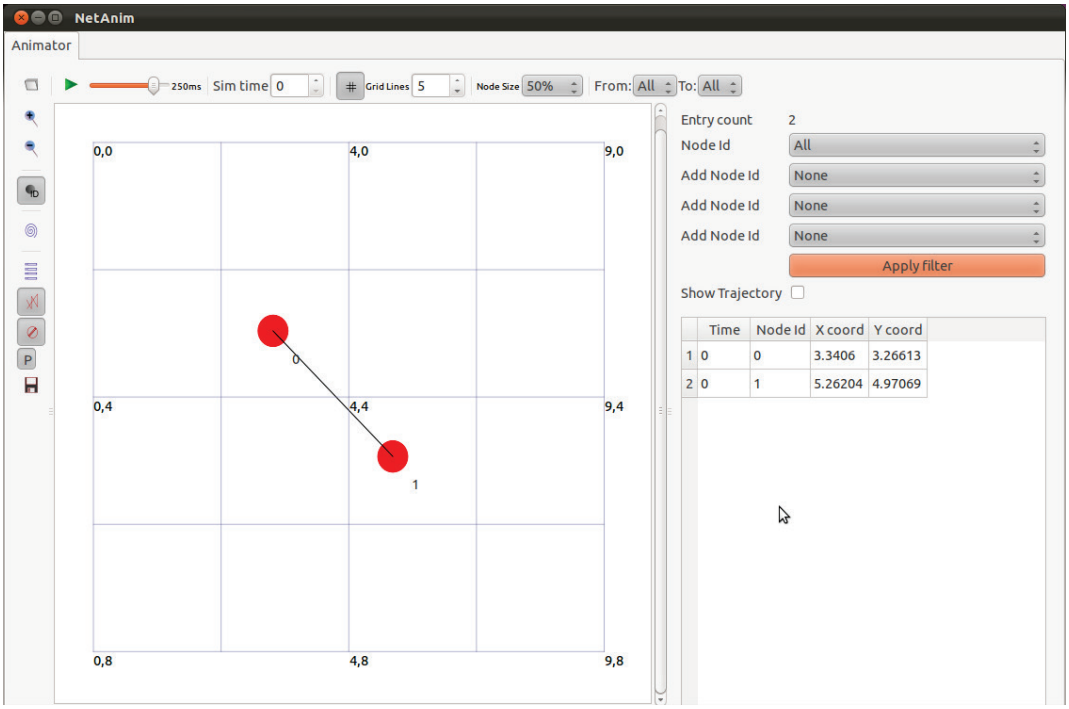

Figure C2. NetAnim, myFirstModbusTcp scenario

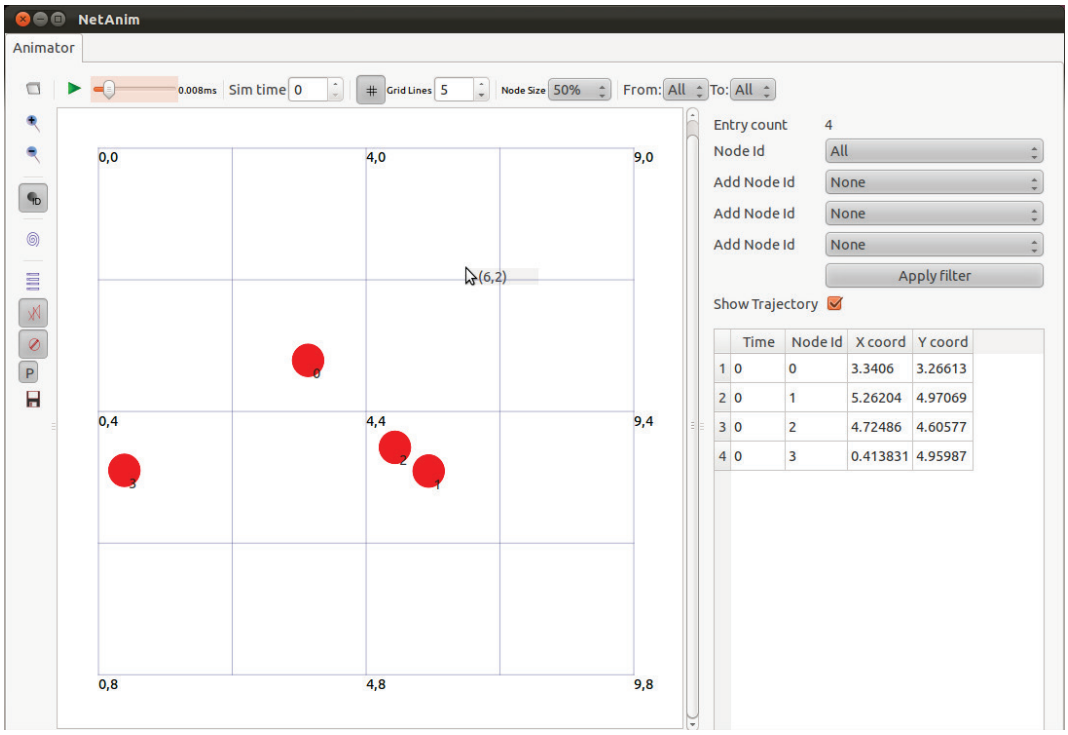

Figure C3. NetAnim, busModbusUdp scenario

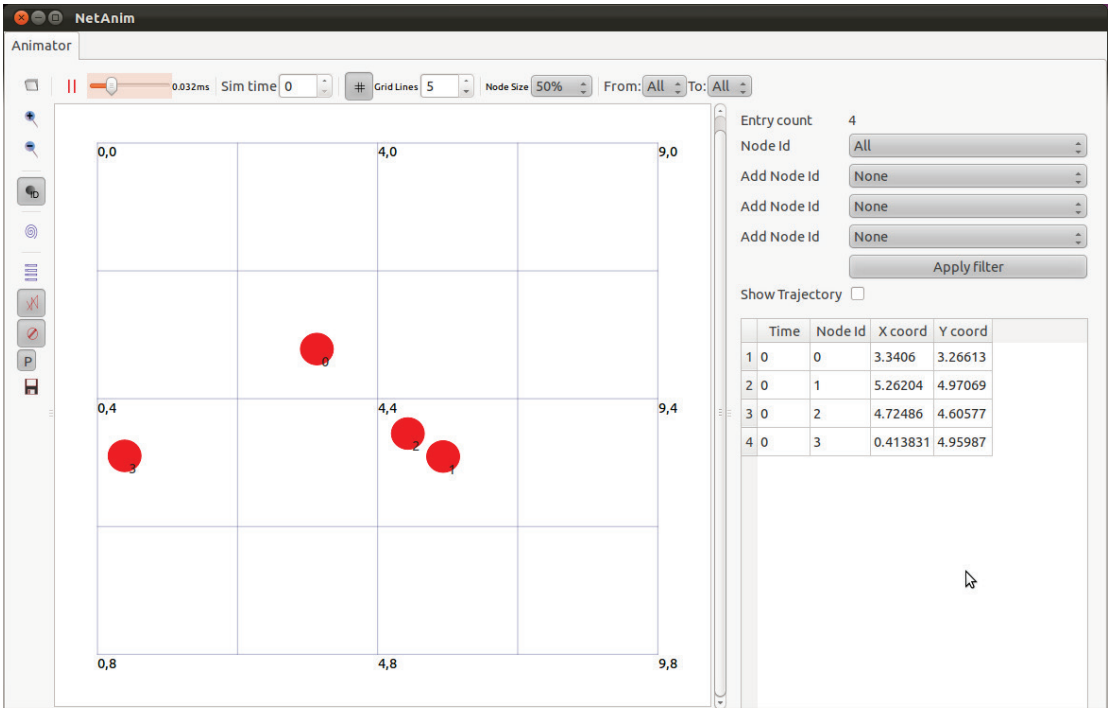

Figure C4. NetAnim, busModbusTcp scenario

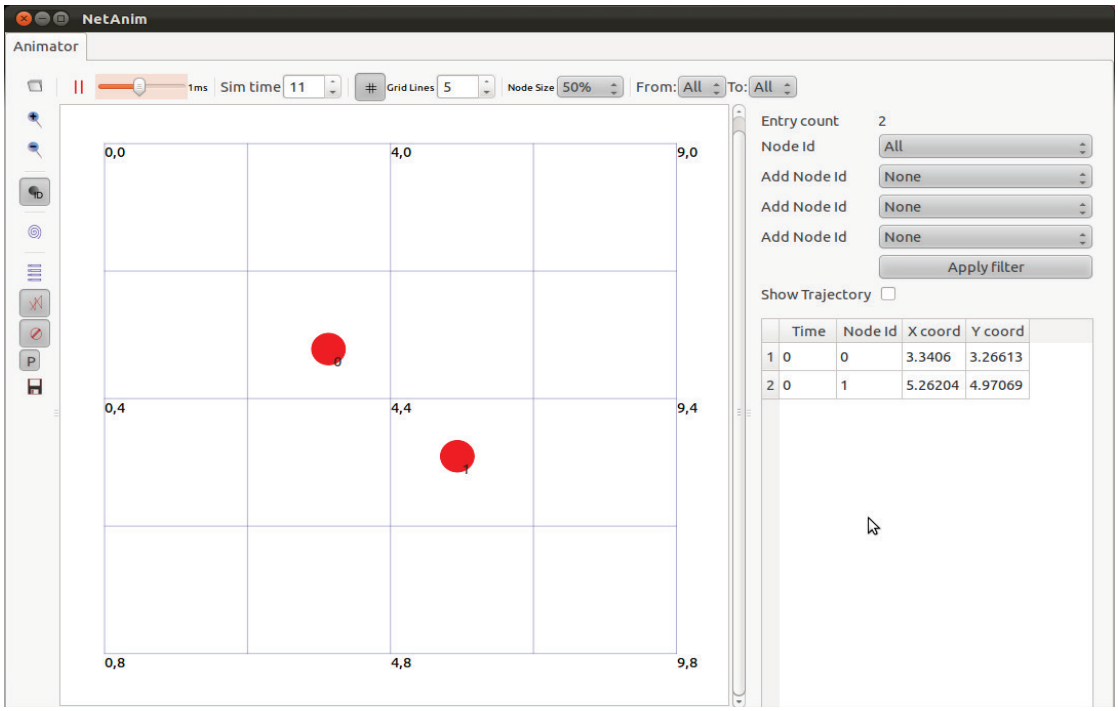

Figure C5. NetAnim, busModbusTcpScenario1 scenario

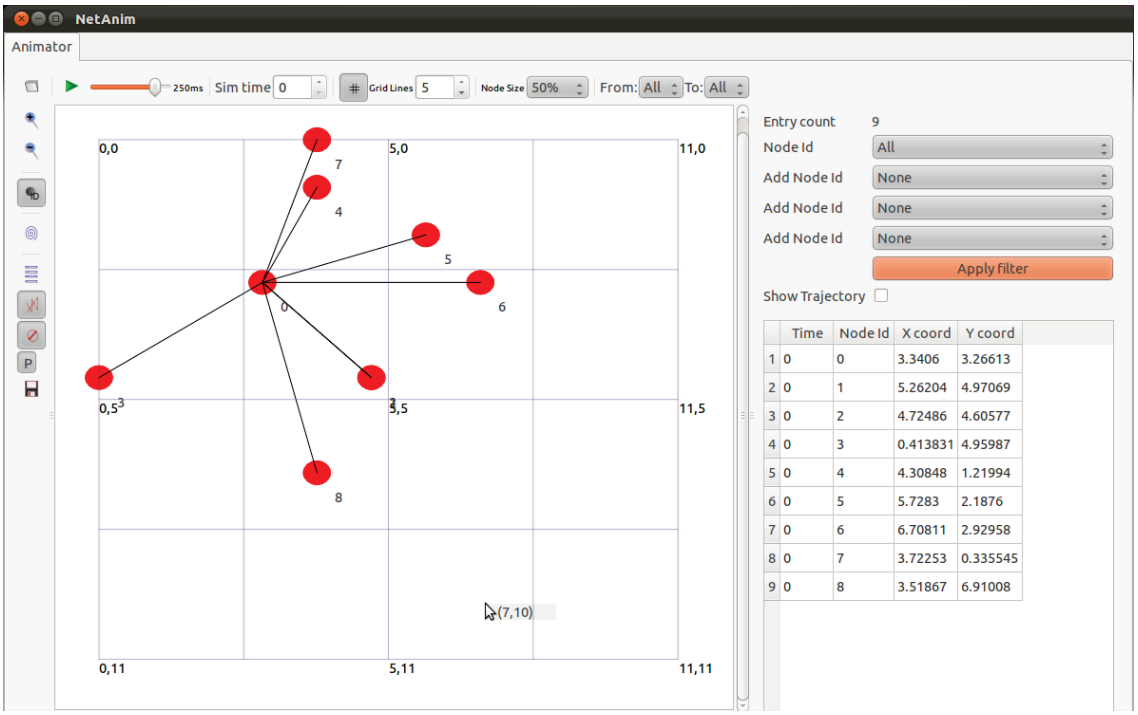

Figure C6. NetAnim, starModbusUdp scenario

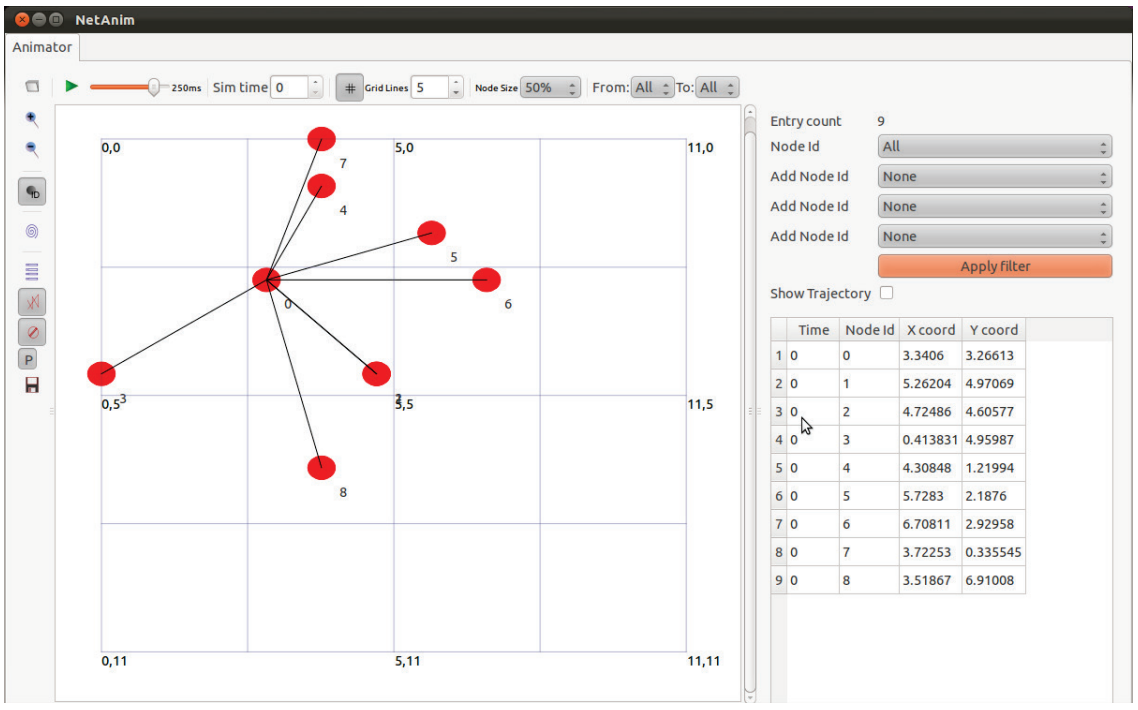

Figure C7. NetAnim, starModbusTcp scenario

Appendix D.

Simulation Environment

The development and simulation environment, which was used for this project, is as follows:

Dell Inspiron
1. Intel Pentium® Dual-Core CPU T4200 @ 2.00 GHz
2. 3 GB DIMM 800 MHz DDR, 4.5 GB Swap
3. Hitachi 250 GB, Dual Port Cache, 5400 RPM HDD

Ubuntu 11.10, with Linux 3.0.0-12-generic
SVN (version 1.6.12), RapidSVN (version 0.12.0)
Mercurial Distributed SCM (version 1.9.1)
GCC (Ubuntu/Linaro 4.6.1-9ubuntu3) 4.6.1
Python 2.7.2+
Gedit 3.2.0
Ns-3.13 (released Dec 2011)
NetAnim 3.0

Appendix E.

Scenario Scripts

The source code of the scenario scripts implemented for ns-Modbus are provided in this Appendix.

myFirstModbusUdp

This is the first scenario that was implemented to model a bus Modbus UDP for ns-Modbus. The scenario is located at *~/ns-3-allinone/ns-3-dev/scratch/* in a C++ file called *myFirstModbusUdp.cc*.

```
/* -*- Mode:C++; c-file-style:"gnu"; indent-tabs-mode:nil; -*- */
/*
 * Copyright (c) 2012 Simon Fraser University
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License version 2 as
 * published by the Free Software Foundation;
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
 *
 * Author: Reza Sahraei <mrs16@sfu.ca>
 */

#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/netanim-module.h"
#include "ns3/internet-module.h"
#include "ns3/point-to-point-module.h"
#include "ns3/applications-module.h"
#include "ns3/modbus-mgr.h"

// Network Topology
```

```
//
//    n0   n1
//      |    |
//    ======
//  LAN 10.1.1.0

//
// How to Run
//
// Example 1: Run MODBUS UDP with verbose logging:
// ./waf --run "scratch/myfirstModbusUdp --verbose=1"
//

// MODBUS requests
uint8_t ENCAP_INTERFACE_TRANS_CANOPEN_GENERAL_REF_REQ_RSP_PDU[] = {0x2B,
0x0D};
uint8_t ENCAP_INTERFACE_TRANS_READ_DEVICE_ID__BASIC_DEVICE_ID_OBJ_ID_00[] =
{0x2B, 0x0E, 0x01, 0x00};
uint8_t
ENCAP_INTERFACE_TRANS_READ_DEVICE_ID__REGULAR_DEVICE_ID_OBJ_ID_00[]      =
{0x2B, 0x0E, 0x02, 0x00};
uint8_t
ENCAP_INTERFACE_TRANS_READ_DEVICE_ID__EXTENDED_DEVICE_ID_OBJ_ID_00[]    =
{0x2B, 0x0E, 0x03, 0x00};
uint8_t
ENCAP_INTERFACE_TRANS_READ_DEVICE_ID__SPECIFIC_DEVICE_ID_OBJ_ID_00[]      =
{0x2B, 0x0E, 0x04, 0x00};



using namespace ns3;

NS_LOG_COMPONENT_DEFINE ("FirstModbusUdpScriptExample");

int
main (int argc, char *argv[])
{
  bool verbose = false;
  std::string animFile = "first-modbus-udp-animfile.xml";

  // under ns-3-dev type ./waf --run 'scratch/myfirst --PrintHelp'
  // to see a list of parameters can be passed to the parser
  CommandLine cmd;
  cmd.AddValue ("verbose", "Tell MODBUS applications to log if true", verbose);
  cmd.AddValue ("animFile",  "File Name for Animation Output", animFile);
  cmd.Parse (argc, argv);
```

```
if (verbose)
 {
   LogComponentEnable ("ModbusUdpServer", LOG_LEVEL_INFO);
   LogComponentEnable ("ModbusUdpClient", LOG_LEVEL_INFO);
 }

NodeContainer nodes;
nodes.Create (2);

PointToPointHelper pointToPoint;
pointToPoint.SetDeviceAttribute ("DataRate", StringValue ("5Mbps"));
pointToPoint.SetChannelAttribute ("Delay", StringValue ("2ms"));

NetDeviceContainer devices;
devices = pointToPoint.Install (nodes);

InternetStackHelper stack;
stack.Install (nodes);

Ipv4AddressHelper address;
address.SetBase ("10.1.1.0", "255.255.255.0");

Ipv4InterfaceContainer interfaces = address.Assign (devices);

ModbusUdpServerHelper cModbusUdpServerHelper;

ApplicationContainer serverApps = cModbusUdpServerHelper.Install (nodes.Get (1));
serverApps.Start (Seconds (1.0));
serverApps.Stop (Seconds (10.0));

ModbusUdpClientHelper cModbusUdpClientHelper (interfaces.GetAddress (1));
cModbusUdpClientHelper.SetAttribute ("MaxPackets", UintegerValue (1));
cModbusUdpClientHelper.SetAttribute ("Interval", TimeValue (Seconds (1.0)));

ApplicationContainer clientApps = cModbusUdpClientHelper.Install (nodes.Get (0));

// set MODBUS requests
cModbusUdpClientHelper.SetRequest (clientApps.Get(0), Seconds (0.1),
  ENCAP_INTERFACE_TRANS_CANOPEN_GENERAL_REF_REQ_RSP_PDU);

cModbusUdpClientHelper.SetRequest (clientApps.Get(0), Seconds (0.2),
  ENCAP_INTERFACE_TRANS_READ_DEVICE_ID__BASIC_DEVICE_ID_OBJ_ID_00);
```

```
  cModbusUdpClientHelper.SetRequest (clientApps.Get(0), Seconds (0.1),
    ENCAP_INTERFACE_TRANS_READ_DEVICE_ID__SPECIFIC_DEVICE_ID_OBJ_ID_00);

  cModbusUdpClientHelper.SetCount (clientApps.Get(0), 100);


  clientApps.Start (Seconds (2.0));
  clientApps.Stop (Seconds (10.0));

  //configure tracing
  AsciiTraceHelper ascii;
  pointToPoint.EnableAsciiAll (ascii.CreateFileStream ("first-modbus-udp.tr"));
  pointToPoint.EnablePcapAll ("first-modbus-udp");

  // Create the animation object and configure for specified output
  AnimationInterface anim (animFile);
  anim.SetXMLOutput ();
  //anim.StartAnimation ();

  Simulator::Run ();
  Simulator::Destroy ();
  return 0;
}
```

myFirstModbusTcp

This is the first scenario that was implemented to model a bus Modbus TCP for ns-Modbus. The scenario is located at *~/ns-3-allinone/ns-3-dev/scratch/* in a C++ file called *myFirstModbusTcp.cc*.

```
/* -*- Mode:C++; c-file-style:"gnu"; indent-tabs-mode:nil; -*- */
/*
 * Copyright (c) 2012 Simon Fraser University
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License version 2 as
 * published by the Free Software Foundation;
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
```

```
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
 *
 * Author: Reza Sahraei <mrs16@sfu.ca>
 */

#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/netanim-module.h"
#include "ns3/internet-module.h"
#include "ns3/point-to-point-module.h"
#include "ns3/applications-module.h"
#include "ns3/modbus-mgr.h"

// Network Topology
//
//    n0   n1
//      |    |
//    ======
//  LAN 10.1.1.0

//
// How to Run
//
// Example 1: Run MODBUS TCP with verbose logging:
// ./waf --run "scratch/myfirstModbusTcp --verbose=1"
//

// MODBUS requests
uint8_t ENCAP_INTERFACE_TRANS_CANOPEN_GENERAL_REF_REQ_RSP_PDU[] = {0x2B,
0x0D};
uint8_t ENCAP_INTERFACE_TRANS_READ_DEVICE_ID__BASIC_DEVICE_ID_OBJ_ID_00[] =
{0x2B, 0x0E, 0x01, 0x00};
uint8_t
ENCAP_INTERFACE_TRANS_READ_DEVICE_ID__REGULAR_DEVICE_ID_OBJ_ID_00[]     =
{0x2B, 0x0E, 0x02, 0x00};
uint8_t
ENCAP_INTERFACE_TRANS_READ_DEVICE_ID__EXTENDED_DEVICE_ID_OBJ_ID_00[]    =
{0x2B, 0x0E, 0x03, 0x00};
uint8_t
ENCAP_INTERFACE_TRANS_READ_DEVICE_ID__SPECIFIC_DEVICE_ID_OBJ_ID_00[]     =
{0x2B, 0x0E, 0x04, 0x00};



using namespace ns3;
```

```
NS_LOG_COMPONENT_DEFINE ("FirstModbusTcpScriptExample");

int
main (int argc, char *argv[])
{
  bool verbose = false;
  std::string animFile = "first-modbus-tcp-animfile.xml";

  // under ns-3-dev type ./waf --run 'scratch/myfirst --PrintHelp'
  // to see a list of parameters can be passed to the parser
  CommandLine cmd;
  cmd.AddValue ("verbose", "Tell MODBUS applications to log if true", verbose);
  cmd.AddValue ("animFile",  "File Name for Animation Output", animFile);
  cmd.Parse (argc, argv);

  if (verbose)
    {
      LogComponentEnable ("ModbusTcpServer", LOG_LEVEL_INFO);
      LogComponentEnable ("ModbusTcpClient", LOG_LEVEL_INFO);
    }

  NodeContainer nodes;
  nodes.Create (2);

  PointToPointHelper pointToPoint;
  pointToPoint.SetDeviceAttribute ("DataRate", StringValue ("5Mbps"));
  pointToPoint.SetChannelAttribute ("Delay", StringValue ("2ms"));

  NetDeviceContainer devices;
  devices = pointToPoint.Install (nodes);

  InternetStackHelper stack;
  stack.Install (nodes);

  Ipv4AddressHelper address;
  address.SetBase ("10.1.1.0", "255.255.255.0");

  Ipv4InterfaceContainer interfaces = address.Assign (devices);

  ModbusTcpServerHelper cModbusTcpServerHelper;

  ApplicationContainer serverApps = cModbusTcpServerHelper.Install (nodes.Get (1));
```

```
  serverApps.Start (Seconds (1.0));
  serverApps.Stop (Seconds (10.0));

  ModbusTcpClientHelper cModbusTcpClientHelper (interfaces.GetAddress (1));
  cModbusTcpClientHelper.SetAttribute ("MaxPackets", UintegerValue (1));
  cModbusTcpClientHelper.SetAttribute ("Interval", TimeValue (Seconds (1.0)));

  ApplicationContainer clientApps = cModbusTcpClientHelper.Install (nodes.Get (0));

  // set MODBUS requests
  cModbusTcpClientHelper.SetRequest (clientApps.Get(0), Seconds (0.1),
    ENCAP_INTERFACE_TRANS_CANOPEN_GENERAL_REF_REQ_RSP_PDU);

  cModbusTcpClientHelper.SetRequest (clientApps.Get(0), Seconds (0.2),
    ENCAP_INTERFACE_TRANS_READ_DEVICE_ID__BASIC_DEVICE_ID_OBJ_ID_00);

  cModbusTcpClientHelper.SetRequest (clientApps.Get(0), Seconds (0.1),
    ENCAP_INTERFACE_TRANS_READ_DEVICE_ID__SPECIFIC_DEVICE_ID_OBJ_ID_00);

  cModbusTcpClientHelper.SetCount (clientApps.Get(0), 100);

  clientApps.Start (Seconds (2.0));
  clientApps.Stop (Seconds (10.0));

  //configure tracing
  AsciiTraceHelper ascii;
  pointToPoint.EnableAsciiAll (ascii.CreateFileStream ("first-modbus-tcp.tr"));
  pointToPoint.EnablePcapAll ("first-modbus-tcp");

  // Create the animation object and configure for specified output
  AnimationInterface anim (animFile);
  anim.SetXMLOutput ();
  //anim.StartAnimation ();

  Simulator::Run ();
  Simulator::Destroy ();
  return 0;
}
```

busModbusUdp

This scenario was implemented to model a bus Modbus UDP for ns-Modbus. The scenario is located at *~/ns-3-allinone/ns-3-dev/scratch/* in a C++ file called *busModbusUdp.cc*.

```
/* -*- Mode:C++; c-file-style:"gnu"; indent-tabs-mode:nil; -*- */
/*
 * Copyright (c) 2012 Simon Fraser University
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License version 2 as
 * published by the Free Software Foundation;
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
 *
 * Author: Reza Sahraei <mrs16@sfu.ca>
 */

#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/netanim-module.h"
#include "ns3/internet-module.h"
#include "ns3/csma-helper.h"
#include "ns3/applications-module.h"
#include "ns3/modbus-mgr.h"
#include "sys/time.h"

// Network Topology (default)
//
//   n0   n1   n2   n3
//    |    |    |    |
//   ================
//     LAN 10.1.1.0

//
// How to Run
//
// set NS_LOG if required
// export NS_LOG=BusModbusUdp=level_all
//
// Example 1: Run bus topology MODBUS UDP with verbose logging:
```

```
// ./waf --run "scratch/busModbusUdp --verbose=1"
//
// Example 2: Run bus topology MODBUS UDP with 10 MODBUS clients:
// ./waf --run "scratch/busModbusUdp --verbose=0 --nCsma=10"
//

// MODBUS requests
uint8_t ENCAP_INTERFACE_TRANS_CANOPEN_GENERAL_REF_REQ_RSP_PDU[] = {0x2B,
0x0D};
uint8_t ENCAP_INTERFACE_TRANS_READ_DEVICE_ID__BASIC_DEVICE_ID_OBJ_ID_00[] =
{0x2B, 0x0E, 0x01, 0x00};
uint8_t
ENCAP_INTERFACE_TRANS_READ_DEVICE_ID__REGULAR_DEVICE_ID_OBJ_ID_00[]     =
{0x2B, 0x0E, 0x02, 0x00};
uint8_t
ENCAP_INTERFACE_TRANS_READ_DEVICE_ID__EXTENDED_DEVICE_ID_OBJ_ID_00[]    =
{0x2B, 0x0E, 0x03, 0x00};
uint8_t
ENCAP_INTERFACE_TRANS_READ_DEVICE_ID__SPECIFIC_DEVICE_ID_OBJ_ID_00[]     =
{0x2B, 0x0E, 0x04, 0x00};



using namespace ns3;

NS_LOG_COMPONENT_DEFINE ("BusModbusUdp");

int
main (int argc, char *argv[])
{
  struct timeval detail_time1, detail_time2, detail_time3;

  // to get current time
  gettimeofday(&detail_time1, NULL);
  double t1 = detail_time1.tv_sec + (detail_time1.tv_usec/1000000.0);

  bool verbose = false;
  std::string animFile = "bus-modbus-udp-animfile.xml";

  //
  // Default number of nodes.  Overridable by command line argument.
  //
  uint32_t nCsma = 4;

  // under ns-3-dev type ./waf --run 'scratch/myfirst --PrintHelp'
  // to see a list of parameters can be passed to the parser
```

```
CommandLine cmd;
cmd.AddValue ("nCsma", "Number of CSMA nodes/devices", nCsma);
cmd.AddValue ("verbose", "Tell MODBUS applications to log if true", verbose);
cmd.AddValue ("animFile",  "File Name for Animation Output", animFile);
cmd.Parse (argc, argv);

if (verbose)
 {
   LogComponentEnable ("ModbusUdpServer", LOG_LEVEL_INFO);
   LogComponentEnable ("ModbusUdpClient", LOG_LEVEL_INFO);
 }

nCsma = nCsma == 0 ? 1 : nCsma;

NS_LOG_INFO ("Build bus topology.");
NodeContainer csmaNodes;
csmaNodes.Create (nCsma);

CsmaHelper csma;
csma.SetChannelAttribute ("DataRate", StringValue ("100Mbps"));
csma.SetChannelAttribute ("Delay", TimeValue (NanoSeconds (6560)));

NetDeviceContainer csmaDevices;
csmaDevices = csma.Install (csmaNodes);

NS_LOG_INFO ("Install internet stack on all nodes.");
InternetStackHelper internet;
internet.Install (csmaNodes);


NS_LOG_INFO ("Assign IP Addresses.");
Ipv4AddressHelper address;
address.SetBase ("10.1.1.0", "255.255.255.0");
Ipv4InterfaceContainer csmaInterfaces;
csmaInterfaces = address.Assign (csmaDevices);

NS_LOG_INFO ("Create applications.");
//
// Create a MODBUS UDP server on the star "hub"
//
ModbusUdpServerHelper cModbusUdpServerHelper;
ApplicationContainer serverApps = cModbusUdpServerHelper.Install (csmaNodes.Get (0));
serverApps.Start (Seconds (1.0));
```

```
serverApps.Stop (Seconds (10.0));

//
// Create MODBUS UDP client applications send UDP to the hub
//
for (uint32_t i = 1; i < nCsma; ++i)
 {
   ModbusUdpClientHelper cModbusUdpClientHelper (csmaInterfaces.GetAddress (0));
   cModbusUdpClientHelper.SetAttribute ("MaxPackets", UintegerValue (1));
   cModbusUdpClientHelper.SetAttribute ("Interval", TimeValue (Seconds (1.0)));

   ApplicationContainer clientApps = cModbusUdpClientHelper.Install (csmaNodes.Get (i));

   // set MODBUS requests
   cModbusUdpClientHelper.SetRequest (clientApps.Get(0), Seconds (0.1),
     ENCAP_INTERFACE_TRANS_CANOPEN_GENERAL_REF_REQ_RSP_PDU);

   cModbusUdpClientHelper.SetRequest (clientApps.Get(0), Seconds (0.2),
     ENCAP_INTERFACE_TRANS_READ_DEVICE_ID__BASIC_DEVICE_ID_OBJ_ID_00);

   cModbusUdpClientHelper.SetRequest (clientApps.Get(0), Seconds (0.1),
     ENCAP_INTERFACE_TRANS_READ_DEVICE_ID__SPECIFIC_DEVICE_ID_OBJ_ID_00);

   cModbusUdpClientHelper.SetCount (clientApps.Get(0), 2);

   clientApps.Start (Seconds (2.0));
   clientApps.Stop (Seconds (10.0));
 }

NS_LOG_INFO ("Enable pcap tracing.");
//configure tracing
AsciiTraceHelper ascii;
csma.EnableAsciiAll (ascii.CreateFileStream ("bus-modbus-udp.tr"));
csma.EnablePcapAll ("bus-modbus-udp");

// Create the animation object and configure for specified output
AnimationInterface anim (animFile);
anim.SetXMLOutput ();
//anim.StartAnimation ();

// to get current time
gettimeofday(&detail_time2, NULL);
double t2 = detail_time2.tv_sec + (detail_time2.tv_usec/1000000.0);
```

```
  NS_LOG_INFO ("Run Simulation.");
  Simulator::Run ();
  Simulator::Destroy ();
  NS_LOG_INFO ("Done.");

  // print total time elapsed in microsecond
  gettimeofday(&detail_time3, NULL);
  double t3 = detail_time3.tv_sec + (detail_time3.tv_usec/1000000.0);
  NS_LOG_INFO ("Total (ms): " << (t3 - t1)*1000);
  NS_LOG_INFO ("Modbus Simulation time (ms): " << (t3 - t2)*1000);
  NS_LOG_INFO ("Node and Link creation (ms): " << (t2 - t1)*1000);

  return 0;
}
```

busModbusTcp

This scenario was implemented to model a bus Modbus TCP for ns-Modbus. The scenario is located at *~/ns-3-allinone/ns-3-dev/scratch/* in a C++ file called *myFirstModbusTcp.cc*.

```
/* -*- Mode:C++; c-file-style:"gnu"; indent-tabs-mode:nil; -*- */
/*
 * Copyright (c) 2012 Simon Fraser University
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License version 2 as
 * published by the Free Software Foundation;
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
 *
 * Author: Reza Sahraei <mrs16@sfu.ca>
 */

#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/netanim-module.h"
```

```
#include "ns3/internet-module.h"
#include "ns3/csma-helper.h"
#include "ns3/applications-module.h"
#include "ns3/modbus-mgr.h"

// Network Topology (default)
//
//   n0   n1   n2   n3
//    |    |    |    |
//   ================
//     LAN 10.1.1.0

//
// How to Run
//
// set NS_LOG if required
// export NS_LOG=BusModbusTcp=level_all
//
// Example 1: Run bus topology MODBUS TCP with verbose logging:
// ./waf --run "scratch/busModbusTcp --verbose=1"
//
// Example 2: Run bus topology MODBUS TCP with 9 MODBUS clients:
// ./waf --run "scratch/busModbusTcp --verbose=0 --nCsma=10"
//

// MODBUS requests
uint8_t ENCAP_INTERFACE_TRANS_CANOPEN_GENERAL_REF_REQ_RSP_PDU[] = {0x2B,
0x0D};
uint8_t ENCAP_INTERFACE_TRANS_READ_DEVICE_ID__BASIC_DEVICE_ID_OBJ_ID_00[] =
{0x2B, 0x0E, 0x01, 0x00};
uint8_t
ENCAP_INTERFACE_TRANS_READ_DEVICE_ID__REGULAR_DEVICE_ID_OBJ_ID_00[]     =
{0x2B, 0x0E, 0x02, 0x00};
uint8_t
ENCAP_INTERFACE_TRANS_READ_DEVICE_ID__EXTENDED_DEVICE_ID_OBJ_ID_00[]   =
{0x2B, 0x0E, 0x03, 0x00};
uint8_t
ENCAP_INTERFACE_TRANS_READ_DEVICE_ID__SPECIFIC_DEVICE_ID_OBJ_ID_00[]     =
{0x2B, 0x0E, 0x04, 0x00};



using namespace ns3;

NS_LOG_COMPONENT_DEFINE ("BusModbusTcp");
```

```
int
main (int argc, char *argv[])
{
  struct timeval detail_time1, detail_time2, detail_time3;

  // to get current time
  gettimeofday(&detail_time1, NULL);
  double t1 = detail_time1.tv_sec + (detail_time1.tv_usec/1000000.0);

  bool verbose = false;
  std::string animFile = "bus-modbus-tcp-animfile.xml";

  //
  // Default number of nodes.  Overridable by command line argument.
  //
  uint32_t nCsma = 4;

  // under ns-3-dev type ./waf --run 'scratch/myfirst --PrintHelp'
  // to see a list of parameters can be passed to the parser
  CommandLine cmd;
  cmd.AddValue ("nCsma", "Number of CSMA nodes/devices", nCsma);
  cmd.AddValue ("verbose", "Tell MODBUS applications to log if true", verbose);
  cmd.AddValue ("animFile",  "File Name for Animation Output", animFile);
  cmd.Parse (argc, argv);

  if (verbose)
    {
      LogComponentEnable ("ModbusTcpServer", LOG_LEVEL_INFO);
      LogComponentEnable ("ModbusTcpClient", LOG_LEVEL_INFO);
    }

  nCsma = nCsma == 0 ? 1 : nCsma;

  NS_LOG_INFO ("Build bus topology.");
  NodeContainer csmaNodes;
  csmaNodes.Create (nCsma);

  CsmaHelper csma;
  csma.SetChannelAttribute ("DataRate", StringValue ("100Mbps"));
  csma.SetChannelAttribute ("Delay", TimeValue (NanoSeconds (6560)));

  NetDeviceContainer csmaDevices;
  csmaDevices = csma.Install (csmaNodes);
```

```
NS_LOG_INFO ("Install internet stack on all nodes.");
InternetStackHelper internet;
internet.Install (csmaNodes);


NS_LOG_INFO ("Assign IP Addresses.");
Ipv4AddressHelper address;
address.SetBase ("10.1.1.0", "255.255.255.0");
Ipv4InterfaceContainer csmaInterfaces;
csmaInterfaces = address.Assign (csmaDevices);

NS_LOG_INFO ("Create applications.");
//
// Create a MODBUS TCP server on the star "hub"
//
ModbusTcpServerHelper cModbusTcpServerHelper;
ApplicationContainer serverApps = cModbusTcpServerHelper.Install (csmaNodes.Get (0));
serverApps.Start (Seconds (1.0));
serverApps.Stop (Seconds (10.0));

//
// Create MODBUS TCP client applications send UDP to the hub
//
for (uint32_t i = 1; i < nCsma; ++i)
  {
    ModbusTcpClientHelper cModbusTcpClientHelper (csmaInterfaces.GetAddress (0));
    cModbusTcpClientHelper.SetAttribute ("MaxPackets", UintegerValue (1));
    cModbusTcpClientHelper.SetAttribute ("Interval", TimeValue (Seconds (1.0)));

    ApplicationContainer clientApps = cModbusTcpClientHelper.Install (csmaNodes.Get (i));

    // set MODBUS requests
    cModbusTcpClientHelper.SetRequest (clientApps.Get(0), Seconds (0.01),
      ENCAP_INTERFACE_TRANS_CANOPEN_GENERAL_REF_REQ_RSP_PDU);

    cModbusTcpClientHelper.SetRequest (clientApps.Get(0), Seconds (0.2),
      ENCAP_INTERFACE_TRANS_READ_DEVICE_ID__BASIC_DEVICE_ID_OBJ_ID_00);

    cModbusTcpClientHelper.SetRequest (clientApps.Get(0), Seconds (0.1),
      ENCAP_INTERFACE_TRANS_READ_DEVICE_ID__SPECIFIC_DEVICE_ID_OBJ_ID_00);

    cModbusTcpClientHelper.SetCount (clientApps.Get(0), 2);
```

```
    clientApps.Start (Seconds (2.0));
    clientApps.Stop (Seconds (10.0));
  }

 NS_LOG_INFO ("Enable pcap tracing.");
 //configure tracing
 AsciiTraceHelper ascii;
 csma.EnableAsciiAll (ascii.CreateFileStream ("bus-modbus-tcp.tr"));
 csma.EnablePcapAll ("bus-modbus-tcp");

 // Create the animation object and configure for specified output
 AnimationInterface anim (animFile);
 anim.SetXMLOutput ();
 //anim.StartAnimation ();

 // to get current time
 gettimeofday(&detail_time2, NULL);
 double t2 = detail_time2.tv_sec + (detail_time2.tv_usec/1000000.0);

 NS_LOG_INFO ("Run Simulation.");
 Simulator::Run ();
 Simulator::Destroy ();
 NS_LOG_INFO ("Done.");

 // print total time elapsed in microsecond
 gettimeofday(&detail_time3, NULL);
 double t3 = detail_time3.tv_sec + (detail_time3.tv_usec/1000000.0);
 NS_LOG_INFO ("Total (ms): " << (t3 - t1)*1000);
 NS_LOG_INFO ("Modbus Simulation time (ms): " << (t3 - t2)*1000);
 NS_LOG_INFO ("Node and Link creation (ms): " << (t2 - t1)*1000);

 return 0;
}
```

busModbusScenario1

This scenario was implemented to model a bus Modbus TCP for ns-Modbus. The scenario is located at *~/ns-3-allinone/ns-3-dev/scratch/* in a C++ file called *busModbusScenario1.cc*.

```
/* -*- Mode:C++; c-file-style:"gnu"; indent-tabs-mode:nil; -*- */
/*
 * Copyright (c) 2012 Simon Fraser University
 *
```

```
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License version 2 as
 * published by the Free Software Foundation;
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
 *
 * Author: Reza Sahraei <mrs16@sfu.ca>
 */

#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/netanim-module.h"
#include "ns3/internet-module.h"
#include "ns3/csma-helper.h"
#include "ns3/applications-module.h"
#include "ns3/modbus-mgr.h"

// Network Topology (default)
//
//   n0   n1
//    |    |
//   ======
// LAN 10.1.1.0

//
// How to Run
//
// set NS_LOG if required
// export NS_LOG=BusModbusTcpScenario1=level_all
//
// Example 1: Run bus topology MODBUS TCP with verbose logging:
// ./waf --run "scratch/BusModbusTcpScenario1 --verbose=1"
//
// Example 2: Run bus topology MODBUS TCP with 9 MODBUS clients:
// ./waf --run "scratch/BusModbusTcpScenario1 --verbose=0 --nCsma=10"
//
```

```
// MODBUS requests
uint8_t READ_COILS_REQ[] = {0x01, 0x00, 0x13, 0x00, 0x13};
uint8_t ENCAP_INTERFACE_TRANS_CANOPEN_GENERAL_REF_REQ_RSP_PDU[] = {0x2B,
0x0D};
uint8_t ENCAP_INTERFACE_TRANS_READ_DEVICE_ID__BASIC_DEVICE_ID_OBJ_ID_00[] =
{0x2B, 0x0E, 0x01, 0x00};
uint8_t
ENCAP_INTERFACE_TRANS_READ_DEVICE_ID__REGULAR_DEVICE_ID_OBJ_ID_00[]     =
{0x2B, 0x0E, 0x02, 0x00};
uint8_t
ENCAP_INTERFACE_TRANS_READ_DEVICE_ID__EXTENDED_DEVICE_ID_OBJ_ID_00[]    =
{0x2B, 0x0E, 0x03, 0x00};
uint8_t
ENCAP_INTERFACE_TRANS_READ_DEVICE_ID__SPECIFIC_DEVICE_ID_OBJ_ID_00[]     =
{0x2B, 0x0E, 0x04, 0x00};



using namespace ns3;

NS_LOG_COMPONENT_DEFINE ("BusModbusTcpScenario1");

int
main (int argc, char *argv[])
{
  struct timeval detail_time1, detail_time2, detail_time3;

  // to get current time
  gettimeofday(&detail_time1, NULL);
  double t1 = detail_time1.tv_sec + (detail_time1.tv_usec/1000000.0);

  bool verbose = false;
  std::string animFile = "bus-modbus-tcp-scenario1-animfile.xml";

  //
  // Default number of nodes.  Overridable by command line argument.
  //
  uint32_t nCsma = 2;

  // under ns-3-dev type ./waf --run 'scratch/myfirst --PrintHelp'
  // to see a list of parameters can be passed to the parser
  CommandLine cmd;
  cmd.AddValue ("nCsma", "Number of CSMA nodes/devices", nCsma);
  cmd.AddValue ("verbose", "Tell MODBUS applications to log if true", verbose);
  cmd.AddValue ("animFile",  "File Name for Animation Output", animFile);
```

```
cmd.Parse (argc, argv);

if (verbose)
 {
   LogComponentEnable ("ModbusTcpServer", LOG_LEVEL_INFO);
   LogComponentEnable ("ModbusTcpClient", LOG_LEVEL_INFO);
 }

nCsma = nCsma == 0 ? 1 : nCsma;

NS_LOG_INFO ("Build bus topology.");
NodeContainer csmaNodes;
csmaNodes.Create (nCsma);

CsmaHelper csma;
csma.SetChannelAttribute ("DataRate", StringValue ("100Mbps"));
csma.SetChannelAttribute ("Delay", TimeValue (NanoSeconds (6560)));

NetDeviceContainer csmaDevices;
csmaDevices = csma.Install (csmaNodes);

NS_LOG_INFO ("Install internet stack on all nodes.");
InternetStackHelper internet;
internet.Install (csmaNodes);


NS_LOG_INFO ("Assign IP Addresses.");
Ipv4AddressHelper address;
address.SetBase ("10.1.1.0", "255.255.255.0");
Ipv4InterfaceContainer csmaInterfaces;
csmaInterfaces = address.Assign (csmaDevices);

NS_LOG_INFO ("Create applications.");
//
// Create a MODBUS TCP server on the star "hub"
//
ModbusTcpServerHelper cModbusTcpServerHelper;
ApplicationContainer serverApps = cModbusTcpServerHelper.Install (csmaNodes.Get (0));
serverApps.Start (Seconds (1.0));
serverApps.Stop (Seconds (10.0));

//
// Create MODBUS TCP client applications send TCP to the hub
```

```
//
for (uint32_t i = 1; i < nCsma; ++i)
  {
    ModbusTcpClientHelper cModbusTcpClientHelper (csmaInterfaces.GetAddress (0));
    cModbusTcpClientHelper.SetAttribute ("MaxPackets", UintegerValue (1));
    cModbusTcpClientHelper.SetAttribute ("Interval", TimeValue (Seconds (1.0)));

    ApplicationContainer clientApps = cModbusTcpClientHelper.Install (csmaNodes.Get (i));

    // set MODBUS requests
    cModbusTcpClientHelper.SetRequest (clientApps.Get(0), Seconds (1),
      ENCAP_INTERFACE_TRANS_CANOPEN_GENERAL_REF_REQ_RSP_PDU);

    cModbusTcpClientHelper.SetRequest (clientApps.Get(0), Seconds (1),
      READ_COILS_REQ);

    cModbusTcpClientHelper.SetCount (clientApps.Get(0), 1);

    clientApps.Start (Seconds (2.0));
    clientApps.Stop (Seconds (10.0));
  }

NS_LOG_INFO ("Enable pcap tracing.");
//configure tracing
AsciiTraceHelper ascii;
csma.EnableAsciiAll (ascii.CreateFileStream ("bus-modbus-tcp-scenario1.tr"));
csma.EnablePcapAll ("bus-modbus-tcp-scenario1");

// Create the animation object and configure for specified output
AnimationInterface anim (animFile);
anim.SetXMLOutput ();
//anim.StartAnimation ();

// to get current time
gettimeofday(&detail_time2, NULL);
double t2 = detail_time2.tv_sec + (detail_time2.tv_usec/1000000.0);

NS_LOG_INFO ("Run Simulation.");
Simulator::Run ();
Simulator::Destroy ();
NS_LOG_INFO ("Done.");

// print total time elapsed in microsecond
```

```
  gettimeofday(&detail_time3, NULL);
  double t3 = detail_time3.tv_sec + (detail_time3.tv_usec/1000000.0);
  NS_LOG_INFO ("Total (ms): " << (t3 - t1)*1000);
  NS_LOG_INFO ("Modbus Simulation time (ms): " << (t3 - t2)*1000);
  NS_LOG_INFO ("Node and Link creation (ms): " << (t2 - t1)*1000);

  return 0;
}
```

busModbusScenario2

This scenario was implemented to model a bus Modbus TCP for ns-Modbus. The scenario is located at *~/ns-3-allinone/ns-3-dev/scratch/* in a C++ file called *busModbusScenario2.cc*.

```
/* -*- Mode:C++; c-file-style:"gnu"; indent-tabs-mode:nil; -*- */
/*
 * Copyright (c) 2012 Simon Fraser University
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License version 2 as
 * published by the Free Software Foundation;
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
 *
 * Author: Reza Sahraei <mrs16@sfu.ca>
 */

#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/netanim-module.h"
#include "ns3/internet-module.h"
#include "ns3/csma-helper.h"
#include "ns3/applications-module.h"
#include "ns3/modbus-mgr.h"

// Network Topology (default)
//
```

```
//   n0   n1
//   |    |
//   ======
// LAN 10.1.1.0

//
// How to Run
//
// set NS_LOG if required
// export NS_LOG=BusModbusTcpScenario1=level_all
//
// Example 1: Run bus topology MODBUS TCP with verbose logging:
// ./waf --run "scratch/BusModbusTcpScenario1 --verbose=1"
//
// Example 2: Run bus topology MODBUS TCP with 9 MODBUS clients:
// ./waf --run "scratch/BusModbusTcpScenario1 --verbose=0 --nCsma=10"
//

// MODBUS requests
uint8_t WRITE_MULTIPLE_COILS_REQ[] =
  {0x0F, 0x00, 0x00, 0x00, 0x10, 0x02, 0xA5, 0xF0};
uint8_t READ_COILS_REQ[] =
  {0x01, 0x00, 0x00, 0x00, 0x10};

using namespace ns3;

NS_LOG_COMPONENT_DEFINE ("BusModbusTcpScenario2");

int
main (int argc, char *argv[])
{
  struct timeval detail_time1, detail_time2, detail_time3;

  // to get current time
  gettimeofday(&detail_time1, NULL);
  double t1 = detail_time1.tv_sec + (detail_time1.tv_usec/1000000.0);

  bool verbose = false;
  std::string animFile = "bus-modbus-tcp-scenario1-animfile.xml";

  //
  // Default number of nodes.  Overridable by command line argument.
  //
```

```
uint32_t nCsma = 2;

// under ns-3-dev type ./waf --run 'scratch/myfirst --PrintHelp'
// to see a list of parameters can be passed to the parser
CommandLine cmd;
cmd.AddValue ("nCsma", "Number of CSMA nodes/devices", nCsma);
cmd.AddValue ("verbose", "Tell MODBUS applications to log if true", verbose);
cmd.AddValue ("animFile",  "File Name for Animation Output", animFile);
cmd.Parse (argc, argv);

if (verbose)
 {
   LogComponentEnable ("ModbusTcpServer", LOG_LEVEL_INFO);
   LogComponentEnable ("ModbusTcpClient", LOG_LEVEL_INFO);
 }

nCsma = nCsma == 0 ? 1 : nCsma;

NS_LOG_INFO ("Build bus topology.");
NodeContainer csmaNodes;
csmaNodes.Create (nCsma);

CsmaHelper csma;
csma.SetChannelAttribute ("DataRate", StringValue ("100Mbps"));
csma.SetChannelAttribute ("Delay", TimeValue (NanoSeconds (6560)));

NetDeviceContainer csmaDevices;
csmaDevices = csma.Install (csmaNodes);

NS_LOG_INFO ("Install internet stack on all nodes.");
InternetStackHelper internet;
internet.Install (csmaNodes);


NS_LOG_INFO ("Assign IP Addresses.");
Ipv4AddressHelper address;
address.SetBase ("10.1.1.0", "255.255.255.0");
Ipv4InterfaceContainer csmaInterfaces;
csmaInterfaces = address.Assign (csmaDevices);

NS_LOG_INFO ("Create applications.");
//
// Create a MODBUS TCP server on the star "hub"
```

```
//
ModbusTcpServerHelper cModbusTcpServerHelper;
ApplicationContainer serverApps = cModbusTcpServerHelper.Install (csmaNodes.Get (0));
serverApps.Start (Seconds (1.0));
serverApps.Stop (Seconds (10.0));

//
// Create MODBUS TCP client applications send TCP to the hub
//
for (uint32_t i = 1; i < nCsma; ++i)
 {
   ModbusTcpClientHelper cModbusTcpClientHelper (csmaInterfaces.GetAddress (0));
   cModbusTcpClientHelper.SetAttribute ("MaxPackets", UintegerValue (1));
   cModbusTcpClientHelper.SetAttribute ("Interval", TimeValue (Seconds (1.0)));

   ApplicationContainer clientApps = cModbusTcpClientHelper.Install (csmaNodes.Get (i));

   // set MODBUS requests
   cModbusTcpClientHelper.SetRequest (clientApps.Get(0), Seconds (1),
     WRITE_MULTIPLE_COILS_REQ);

   cModbusTcpClientHelper.SetRequest (clientApps.Get(0), Seconds (1),
     READ_COILS_REQ);

   cModbusTcpClientHelper.SetCount (clientApps.Get(0), 1);

   clientApps.Start (Seconds (2.0));
   clientApps.Stop (Seconds (10.0));
 }

NS_LOG_INFO ("Enable pcap tracing.");
//configure tracing
AsciiTraceHelper ascii;
csma.EnableAsciiAll (ascii.CreateFileStream ("bus-modbus-tcp-scenario1.tr"));
csma.EnablePcapAll ("bus-modbus-tcp-scenario1");

// Create the animation object and configure for specified output
AnimationInterface anim (animFile);
anim.SetXMLOutput ();
//anim.StartAnimation ();

// to get current time
gettimeofday(&detail_time2, NULL);
```

```
  double t2 = detail_time2.tv_sec + (detail_time2.tv_usec/1000000.0);

  NS_LOG_INFO ("Run Simulation.");
  Simulator::Run ();
  Simulator::Destroy ();
  NS_LOG_INFO ("Done.");

  // print total time elapsed in microsecond
  gettimeofday(&detail_time3, NULL);
  double t3 = detail_time3.tv_sec + (detail_time3.tv_usec/1000000.0);
  NS_LOG_INFO ("Total (ms): " << (t3 - t1)*1000);
  NS_LOG_INFO ("Modbus Simulation time (ms): " << (t3 - t2)*1000);
  NS_LOG_INFO ("Node and Link creation (ms): " << (t2 - t1)*1000);

  return 0;
}
```

busModbusScenario3

This scenario was implemented to model a bus Modbus TCP for ns-Modbus. The scenario is located at *~/ns-3-allinone/ns-3-dev/scratch/* in a C++ file called *busModbusScenario3.cc*.

```
/* -*- Mode:C++; c-file-style:"gnu"; indent-tabs-mode:nil; -*- */
/*
 * Copyright (c) 2012 Simon Fraser University
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License version 2 as
 * published by the Free Software Foundation;
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
 *
 * Author: Reza Sahraei <mrs16@sfu.ca>
 */

#include "ns3/core-module.h"
#include "ns3/network-module.h"
```

```
#include "ns3/netanim-module.h"
#include "ns3/internet-module.h"
#include "ns3/csma-helper.h"
#include "ns3/applications-module.h"
#include "ns3/modbus-mgr.h"

// Network Topology (default)
//
//   n0   n1
//   |    |
//   ======
// LAN 10.1.1.0

//
// How to Run
//
// set NS_LOG if required
// export NS_LOG=BusModbusTcpScenario1=level_all
//
// Example 1: Run bus topology MODBUS TCP with verbose logging:
// ./waf --run "scratch/BusModbusTcpScenario1 --verbose=1"
//
// Example 2: Run bus topology MODBUS TCP with 9 MODBUS clients:
// ./waf --run "scratch/BusModbusTcpScenario1 --verbose=0 --nCsma=10"
//

// Invalid MODBUS requests
//
// Function code is not supported: exception code = 1
uint8_t WRONG_FUNCTION_CODE_REQ[] = {0x00};
//
// Illegal data value: exception code = 2
uint8_t WRITE_MULTIPLE_COILS_REQ_WRONG_START_ADR_PLUS_QUANTITY[] =
  {0x0F, 0xFF, 0xFF, 0x00, 0x10, 0x02, 0xA5, 0xF0};
//
// Illegal data value: exception code = 3
uint8_t WRITE_MULTIPLE_COILS_REQ_WRONG_QUANTITY_OF_OUTPUT[] =
  {0x0F, 0x00, 0x00, 0xFF, 0xFF, 0x02, 0xA5, 0xF0};

using namespace ns3;

NS_LOG_COMPONENT_DEFINE ("BusModbusTcpScenario3");
```

```
int
main (int argc, char *argv[])
{
  struct timeval detail_time1, detail_time2, detail_time3;

  // to get current time
  gettimeofday(&detail_time1, NULL);
  double t1 = detail_time1.tv_sec + (detail_time1.tv_usec/1000000.0);

  bool verbose = false;
  std::string animFile = "bus-modbus-tcp-scenario1-animfile.xml";

  //
  // Default number of nodes.  Overridable by command line argument.
  //
  uint32_t nCsma = 2;

  // under ns-3-dev type ./waf --run 'scratch/myfirst --PrintHelp'
  // to see a list of parameters can be passed to the parser
  CommandLine cmd;
  cmd.AddValue ("nCsma", "Number of CSMA nodes/devices", nCsma);
  cmd.AddValue ("verbose", "Tell MODBUS applications to log if true", verbose);
  cmd.AddValue ("animFile",  "File Name for Animation Output", animFile);
  cmd.Parse (argc, argv);

  if (verbose)
    {
      LogComponentEnable ("ModbusTcpServer", LOG_LEVEL_INFO);
      LogComponentEnable ("ModbusTcpClient", LOG_LEVEL_INFO);
    }

  nCsma = nCsma == 0 ? 1 : nCsma;

  NS_LOG_INFO ("Build bus topology.");
  NodeContainer csmaNodes;
  csmaNodes.Create (nCsma);

  CsmaHelper csma;
  csma.SetChannelAttribute ("DataRate", StringValue ("100Mbps"));
  csma.SetChannelAttribute ("Delay", TimeValue (NanoSeconds (6560)));

  NetDeviceContainer csmaDevices;
  csmaDevices = csma.Install (csmaNodes);
```

```
NS_LOG_INFO ("Install internet stack on all nodes.");
InternetStackHelper internet;
internet.Install (csmaNodes);


NS_LOG_INFO ("Assign IP Addresses.");
Ipv4AddressHelper address;
address.SetBase ("10.1.1.0", "255.255.255.0");
Ipv4InterfaceContainer csmaInterfaces;
csmaInterfaces = address.Assign (csmaDevices);

NS_LOG_INFO ("Create applications.");
//
// Create a MODBUS TCP server on the star "hub"
//
ModbusTcpServerHelper cModbusTcpServerHelper;
ApplicationContainer serverApps = cModbusTcpServerHelper.Install (csmaNodes.Get (0));
serverApps.Start (Seconds (1.0));
serverApps.Stop (Seconds (10.0));

//
// Create MODBUS TCP client applications send TCP to the hub
//
for (uint32_t i = 1; i < nCsma; ++i)
  {
    ModbusTcpClientHelper cModbusTcpClientHelper (csmaInterfaces.GetAddress (0));
    cModbusTcpClientHelper.SetAttribute ("MaxPackets", UintegerValue (1));
    cModbusTcpClientHelper.SetAttribute ("Interval", TimeValue (Seconds (1.0)));

    ApplicationContainer clientApps = cModbusTcpClientHelper.Install (csmaNodes.Get (i));

    // set MODBUS requests
    cModbusTcpClientHelper.SetRequest (clientApps.Get(0), Seconds (1),
      WRONG_FUNCTION_CODE_REQ);

    cModbusTcpClientHelper.SetRequest (clientApps.Get(0), Seconds (1),
      WRITE_MULTIPLE_COILS_REQ_WRONG_START_ADR_PLUS_QUANTITY);

    cModbusTcpClientHelper.SetRequest (clientApps.Get(0), Seconds (1),
      WRITE_MULTIPLE_COILS_REQ_WRONG_QUANTITY_OF_OUTPUT);

    cModbusTcpClientHelper.SetCount (clientApps.Get(0), 1);
```

```
    clientApps.Start (Seconds (2.0));
    clientApps.Stop (Seconds (10.0));
  }

 NS_LOG_INFO ("Enable pcap tracing.");
 //configure tracing
 AsciiTraceHelper ascii;
 csma.EnableAsciiAll (ascii.CreateFileStream ("bus-modbus-tcp-scenario1.tr"));
 csma.EnablePcapAll ("bus-modbus-tcp-scenario1");

 // Create the animation object and configure for specified output
 AnimationInterface anim (animFile);
 anim.SetXMLOutput ();
 //anim.StartAnimation ();

 // to get current time
 gettimeofday(&detail_time2, NULL);
 double t2 = detail_time2.tv_sec + (detail_time2.tv_usec/1000000.0);

 NS_LOG_INFO ("Run Simulation.");
 Simulator::Run ();
 Simulator::Destroy ();
 NS_LOG_INFO ("Done.");

 // print total time elapsed in microsecond
 gettimeofday(&detail_time3, NULL);
 double t3 = detail_time3.tv_sec + (detail_time3.tv_usec/1000000.0);
 NS_LOG_INFO ("Total (ms): " << (t3 - t1)*1000);
 NS_LOG_INFO ("Modbus Simulation time (ms): " << (t3 - t2)*1000);
 NS_LOG_INFO ("Node and Link creation (ms): " << (t2 - t1)*1000);

 return 0;
}
```

starModbusUdp

This scenario was implemented to model a star (P2P) Modbus UDP for ns-Modbus. The scenario is located at *~/ns-3-allinone/ns-3-dev/scratch/* in a C++ file called *starModbusUdp.cc*.

```
/* -*- Mode:C++; c-file-style:"gnu"; indent-tabs-mode:nil; -*- */
/*
 * Copyright (c) 2012 Simon Fraser University
 *
```

```
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License version 2 as
 * published by the Free Software Foundation;
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
 *
 * Author: Reza Sahraei <mrs16@sfu.ca>
 */

#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/netanim-module.h"
#include "ns3/internet-module.h"
#include "ns3/point-to-point-module.h"
#include "ns3/applications-module.h"
#include "ns3/modbus-mgr.h"
#include "ns3/point-to-point-layout-module.h"

// Network topology (default)
//
//        n2 n3 n4              .
//            \ | /             .
//             \|/              .
//     n1--- n0---n5            .
//             /|\              .
//            / | \             .
//        n8 n7 n6              .
//

//
// How to Run
//
// set NS_LOG if required
// export NS_LOG=StarModbusUdp=level_all
//
// Example 1: Run star topology MODBUS UDP with verbose logging:
```

```
// ./waf --run "scratch/starModbusUdp --verbose=1"
//
// Example 2: Run star topology MODBUS UDP with 10 MODBUS clients:
// ./waf --run "scratch/starModbusUdp --verbose=0 --nSpokes=10"
//

// MODBUS requests
uint8_t ENCAP_INTERFACE_TRANS_CANOPEN_GENERAL_REF_REQ_RSP_PDU[] = {0x2B,
0x0D};
uint8_t ENCAP_INTERFACE_TRANS_READ_DEVICE_ID__BASIC_DEVICE_ID_OBJ_ID_00[] =
{0x2B, 0x0E, 0x01, 0x00};
uint8_t
ENCAP_INTERFACE_TRANS_READ_DEVICE_ID__REGULAR_DEVICE_ID_OBJ_ID_00[]     =
{0x2B, 0x0E, 0x02, 0x00};
uint8_t
ENCAP_INTERFACE_TRANS_READ_DEVICE_ID__EXTENDED_DEVICE_ID_OBJ_ID_00[]    =
{0x2B, 0x0E, 0x03, 0x00};
uint8_t
ENCAP_INTERFACE_TRANS_READ_DEVICE_ID__SPECIFIC_DEVICE_ID_OBJ_ID_00[]     =
{0x2B, 0x0E, 0x04, 0x00};



using namespace ns3;

NS_LOG_COMPONENT_DEFINE ("StarModbusUdp");

int
main (int argc, char *argv[])
{
  struct timeval detail_time1, detail_time2, detail_time3;

  // to get current time
  gettimeofday(&detail_time1, NULL);
  double t1 = detail_time1.tv_sec + (detail_time1.tv_usec/1000000.0);

  bool verbose = false;
  std::string animFile = "star-modbus-udp-animfile.xml";

  //
  // Default number of nodes in the star.  Overridable by command line argument.
  //
  uint32_t nSpokes = 8;

  // under ns-3-dev type ./waf --run 'scratch/myfirst --PrintHelp'
  // to see a list of parameters can be passed to the parser
```

```
CommandLine cmd;
cmd.AddValue ("verbose", "Tell MODBUS applications to log if true", verbose);
cmd.AddValue ("nSpokes", "Number of MODBUS star clients", nSpokes);
cmd.AddValue ("animFile",  "File Name for Animation Output", animFile);
cmd.Parse (argc, argv);

if (verbose)
 {
   LogComponentEnable ("ModbusUdpServer", LOG_LEVEL_INFO);
   LogComponentEnable ("ModbusUdpClient", LOG_LEVEL_INFO);
 }

NS_LOG_INFO ("Build star topology.");
PointToPointHelper pointToPoint;
pointToPoint.SetDeviceAttribute ("DataRate", StringValue ("5Mbps"));
pointToPoint.SetChannelAttribute ("Delay", StringValue ("2ms"));
PointToPointStarHelper star (nSpokes, pointToPoint);

NS_LOG_INFO ("Install internet stack on all nodes.");
InternetStackHelper internet;
star.InstallStack (internet);

NS_LOG_INFO ("Assign IP Addresses.");
star.AssignIpv4Addresses (Ipv4AddressHelper ("10.1.1.0", "255.255.255.0"));

//Ipv4InterfaceContainer interfaces = address.Assign (devices);

NS_LOG_INFO ("Create applications.");
//
// Create a MODBUS UDP server on the star "hub"
//
ModbusUdpServerHelper cModbusUdpServerHelper;
ApplicationContainer serverApps = cModbusUdpServerHelper.Install (star.GetHub ());
serverApps.Start (Seconds (1.0));
serverApps.Stop (Seconds (10.0));

//
// Create MODBUS UDP client applications send UDP to the hub,
// one on each spoke node
//
ApplicationContainer spokeApps;

for (uint32_t i = 0; i < star.SpokeCount (); ++i)
```

```
 {
   ModbusUdpClientHelper cModbusUdpClientHelper (star.GetHubIpv4Address (i));
   cModbusUdpClientHelper.SetAttribute ("MaxPackets", UintegerValue (1));
   cModbusUdpClientHelper.SetAttribute ("Interval", TimeValue (Seconds (1.0)));

   ApplicationContainer clientApps = cModbusUdpClientHelper.Install (star.GetSpokeNode (i));
   spokeApps.Add (clientApps);

   // set MODBUS requests
   cModbusUdpClientHelper.SetRequest (clientApps.Get(0), Seconds (0.1),
    ENCAP_INTERFACE_TRANS_CANOPEN_GENERAL_REF_REQ_RSP_PDU);

   cModbusUdpClientHelper.SetRequest (clientApps.Get(0), Seconds (0.2),
    ENCAP_INTERFACE_TRANS_READ_DEVICE_ID__BASIC_DEVICE_ID_OBJ_ID_00);

   cModbusUdpClientHelper.SetRequest (clientApps.Get(0), Seconds (0.1),
    ENCAP_INTERFACE_TRANS_READ_DEVICE_ID__SPECIFIC_DEVICE_ID_OBJ_ID_00);

   cModbusUdpClientHelper.SetCount (clientApps.Get(0), 2);
 }
spokeApps.Start (Seconds (2.0));
spokeApps.Stop (Seconds (10.0));

NS_LOG_INFO ("Enable static global routing.");
//
// Turn on global static routing so we can actually be routed across the star.
//
Ipv4GlobalRoutingHelper::PopulateRoutingTables ();

NS_LOG_INFO ("Enable pcap tracing.");
//configure tracing
AsciiTraceHelper ascii;
pointToPoint.EnableAsciiAll (ascii.CreateFileStream ("start-modbus-udp.tr"));
pointToPoint.EnablePcapAll ("star-modbus-udp");

// Create the animation object and configure for specified output
AnimationInterface anim (animFile);
anim.SetXMLOutput ();
//anim.StartAnimation ();

// to get current time
gettimeofday(&detail_time2, NULL);
double t2 = detail_time2.tv_sec + (detail_time2.tv_usec/1000000.0);
```

```
  NS_LOG_INFO ("Run Simulation.");
  Simulator::Run ();
  Simulator::Destroy ();
  NS_LOG_INFO ("Done.");

  // print total time elapsed in microsecond
  gettimeofday(&detail_time3, NULL);
  double t3 = detail_time3.tv_sec + (detail_time3.tv_usec/1000000.0);
  NS_LOG_INFO ("Total (ms): " << (t3 - t1)*1000);
  NS_LOG_INFO ("Modbus Simulation time (ms): " << (t3 - t2)*1000);
  NS_LOG_INFO ("Node and Link creation (ms): " << (t2 - t1)*1000);

  return 0;
}
```

starModbusTcp

This scenario was implemented to model a star (P2P) Modbus TCP for ns-Modbus. The scenario is located at *~/ns-3-allinone/ns-3-dev/scratch/* in a C++ file called *starModbusTcp.cc*.

```
/* -*- Mode:C++; c-file-style:"gnu"; indent-tabs-mode:nil; -*- */
/*
 * Copyright (c) 2012 Simon Fraser University
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License version 2 as
 * published by the Free Software Foundation;
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
 *
 * Author: Reza Sahraei <mrs16@sfu.ca>
 */

#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/netanim-module.h"
```

```
#include "ns3/internet-module.h"
#include "ns3/point-to-point-module.h"
#include "ns3/applications-module.h"
#include "ns3/modbus-mgr.h"
#include "ns3/point-to-point-layout-module.h"

// Network topology (default)
//
//        n2 n3 n4              .
//             \ | /            .
//              \|/             .
//     n1--- n0---n5            .
//              /|\             .
//             / | \            .
//        n8 n7 n6              .
//


//
// How to Run
//
// set NS_LOG if required
// export NS_LOG=StarModbusTcp=level_all
//
// Example 1: Run star topology MODBUS TCP with verbose logging:
// ./waf --run "scratch/starModbusTcp --verbose=1"
//
// Example 2: Run star topology MODBUS TCP with 10 MODBUS clients:
// ./waf --run "scratch/starModbusTcp --verbose=0 --nSpokes=10"
//


// MODBUS requests
uint8_t ENCAP_INTERFACE_TRANS_CANOPEN_GENERAL_REF_REQ_RSP_PDU[] = {0x2B,
0x0D};
uint8_t ENCAP_INTERFACE_TRANS_READ_DEVICE_ID__BASIC_DEVICE_ID_OBJ_ID_00[] =
{0x2B, 0x0E, 0x01, 0x00};
uint8_t
ENCAP_INTERFACE_TRANS_READ_DEVICE_ID__REGULAR_DEVICE_ID_OBJ_ID_00[]     =
{0x2B, 0x0E, 0x02, 0x00};
uint8_t
ENCAP_INTERFACE_TRANS_READ_DEVICE_ID__EXTENDED_DEVICE_ID_OBJ_ID_00[]    =
{0x2B, 0x0E, 0x03, 0x00};
uint8_t
ENCAP_INTERFACE_TRANS_READ_DEVICE_ID__SPECIFIC_DEVICE_ID_OBJ_ID_00[]     =
{0x2B, 0x0E, 0x04, 0x00};
```

```
using namespace ns3;

NS_LOG_COMPONENT_DEFINE ("StarModbusTcp");

int
main (int argc, char *argv[])
{
 struct timeval detail_time1, detail_time2, detail_time3;

 // to get current time
 gettimeofday(&detail_time1, NULL);
 double t1 = detail_time1.tv_sec + (detail_time1.tv_usec/1000000.0);

 bool verbose = false;
 std::string animFile = "star-modbus-tcp-animfile.xml";

 //
 // Default number of nodes in the star.  Overridable by command line argument.
 //
 uint32_t nSpokes = 8;

 // under ns-3-dev type ./waf --run 'scratch/myfirst --PrintHelp'
 // to see a list of parameters can be passed to the parser
 CommandLine cmd;
 cmd.AddValue ("verbose", "Tell MODBUS applications to log if true", verbose);
 cmd.AddValue ("nSpokes", "Number of MODBUS star clients", nSpokes);
 cmd.AddValue ("animFile",  "File Name for Animation Output", animFile);
 cmd.Parse (argc, argv);

 if (verbose)
  {
    LogComponentEnable ("ModbusTcpServer", LOG_LEVEL_INFO);
    LogComponentEnable ("ModbusTcpClient", LOG_LEVEL_INFO);
  }

 NS_LOG_INFO ("Build star topology.");
 PointToPointHelper pointToPoint;
 pointToPoint.SetDeviceAttribute ("DataRate", StringValue ("5Mbps"));
 pointToPoint.SetChannelAttribute ("Delay", StringValue ("2ms"));
 PointToPointStarHelper star (nSpokes, pointToPoint);

 NS_LOG_INFO ("Install internet stack on all nodes.");
```

```
InternetStackHelper internet;
star.InstallStack (internet);

NS_LOG_INFO ("Assign IP Addresses.");
star.AssignIpv4Addresses (Ipv4AddressHelper ("10.1.1.0", "255.255.255.0"));

//Ipv4InterfaceContainer interfaces = address.Assign (devices);

NS_LOG_INFO ("Create applications.");
//
// Create a MODBUS TCP server on the star "hub"
//
ModbusTcpServerHelper cModbusTcpServerHelper;
ApplicationContainer serverApps = cModbusTcpServerHelper.Install (star.GetHub ());
serverApps.Start (Seconds (1.0));
serverApps.Stop (Seconds (15.0));

//
// Create MODBUS TCP client applications send TCP to the hub,
// one on each spoke node
//
ApplicationContainer spokeApps;

for (uint32_t i = 0; i < star.SpokeCount (); ++i)
 {
   ModbusTcpClientHelper cModbusTcpClientHelper (star.GetHubIpv4Address (i));
   cModbusTcpClientHelper.SetAttribute ("MaxPackets", UintegerValue (1));
   cModbusTcpClientHelper.SetAttribute ("Interval", TimeValue (Seconds (1.0)));

   ApplicationContainer clientApps = cModbusTcpClientHelper.Install (star.GetSpokeNode (i));
   spokeApps.Add (clientApps);

   // set MODBUS requests
   cModbusTcpClientHelper.SetRequest (clientApps.Get(0), Seconds (0.1),
     ENCAP_INTERFACE_TRANS_CANOPEN_GENERAL_REF_REQ_RSP_PDU);

   cModbusTcpClientHelper.SetRequest (clientApps.Get(0), Seconds (0.2),
     ENCAP_INTERFACE_TRANS_READ_DEVICE_ID__BASIC_DEVICE_ID_OBJ_ID_00);

   cModbusTcpClientHelper.SetRequest (clientApps.Get(0), Seconds (0.1),
     ENCAP_INTERFACE_TRANS_READ_DEVICE_ID__SPECIFIC_DEVICE_ID_OBJ_ID_00);

   cModbusTcpClientHelper.SetCount (clientApps.Get(0), 2);
```

```
  }
 spokeApps.Start (Seconds (2.0));
 spokeApps.Stop (Seconds (10.0));

 NS_LOG_INFO ("Enable static global routing.");
 //
 // Turn on global static routing so we can actually be routed across the star.
 //
 Ipv4GlobalRoutingHelper::PopulateRoutingTables ();

 NS_LOG_INFO ("Enable pcap tracing.");
 //configure tracing
 AsciiTraceHelper ascii;
 pointToPoint.EnableAsciiAll (ascii.CreateFileStream ("star-modbus-tcp.tr"));
 pointToPoint.EnablePcapAll ("star-modbus-tcp");

 // Create the animation object and configure for specified output
 AnimationInterface anim (animFile);
 anim.SetXMLOutput ();
 //anim.StartAnimation ();

 // to get current time
 gettimeofday(&detail_time2, NULL);
 double t2 = detail_time2.tv_sec + (detail_time2.tv_usec/1000000.0);

 NS_LOG_INFO ("Run Simulation.");
 Simulator::Run ();
 Simulator::Destroy ();
 NS_LOG_INFO ("Done.");

 // print total time elapsed in microsecond
 gettimeofday(&detail_time3, NULL);
 double t3 = detail_time3.tv_sec + (detail_time3.tv_usec/1000000.0);
 NS_LOG_INFO ("Total (ms): " << (t3 - t1)*1000);
 NS_LOG_INFO ("Modbus Simulation time (ms): " << (t3 - t2)*1000);
 NS_LOG_INFO ("Node and Link creation (ms): " << (t2 - t1)*1000);

 return 0;
}
```

Appendix F.

Source Code

Wscript

Wscript indicates the list of files to be included in the build. Wscript is located at *~/ns-3-allinone/ns-3-dev/src/applications/*.

```
## -*- Mode: python; py-indent-offset: 4; indent-tabs-mode: nil; coding: utf-8; -*-

def build(bld):
    module = bld.create_ns3_module('applications', ['internet', 'config-store', 'tools'])
    module.source = [
        'model/bulk-send-application.cc',
        'model/onoff-application.cc',
        'model/packet-sink.cc',
        'model/ping6.cc',
        'model/radvd.cc',
        'model/radvd-interface.cc',
        'model/radvd-prefix.cc',
        'model/udp-client.cc',
        'model/udp-server.cc',
        'model/seq-ts-header.cc',
        'model/udp-trace-client.cc',
        'model/packet-loss-counter.cc',
        'model/udp-echo-client.cc',
        'model/udp-echo-server.cc',
        'model/v4ping.cc',
        'model/modbus-udp-client.cc',
        'model/modbus-udp-server.cc',
        'model/modbus-tcp-client.cc',
        'model/modbus-tcp-server.cc',
        'model/modbus-pdu.cc',
        'model/modbus-mgr.cc',
        'model/bgp-router.cc',
        'model/bgp-pdu.cc',
        'helper/bulk-send-helper.cc',
        'helper/on-off-helper.cc',
        'helper/packet-sink-helper.cc',
        'helper/ping6-helper.cc',
```

```
    'helper/udp-client-server-helper.cc',
    'helper/udp-echo-helper.cc',
    'helper/v4ping-helper.cc',
    'helper/modbus-udp-helper.cc',
    'helper/modbus-tcp-helper.cc',
    'helper/bgp-router-helper.cc',
    ]

applications_test = bld.create_ns3_module_test_library('applications')
applications_test.source = [
    'test/udp-client-server-test.cc',
    ]

headers = bld.new_task_gen(features=['ns3header'])
headers.module = 'applications'
headers.source = [
    'model/bulk-send-application.h',
    'model/onoff-application.h',
    'model/packet-sink.h',
    'model/ping6.h',
    'model/radvd.h',
    'model/radvd-interface.h',
    'model/radvd-prefix.h',
    'model/udp-client.h',
    'model/udp-server.h',
    'model/seq-ts-header.h',
    'model/udp-trace-client.h',
    'model/packet-loss-counter.h',
    'model/udp-echo-client.h',
    'model/udp-echo-server.h',
    'model/v4ping.h',
    'model/modbus-udp-client.h',
    'model/modbus-udp-server.h',
    'model/modbus-tcp-client.h',
    'model/modbus-tcp-server.h',
    'model/modbus-pdu.h',
    'model/modbus-mgr.h',
    'model/bgp-router.h',
    'model/bgp-pdu.h',
    'helper/bulk-send-helper.h',
    'helper/on-off-helper.h',
    'helper/packet-sink-helper.h',
    'helper/ping6-helper.h',
```

```
        'helper/udp-client-server-helper.h',
        'helper/udp-echo-helper.h',
        'helper/v4ping-helper.h',
        'helper/modbus-udp-helper.h',
        'helper/modbus-tcp-helper.h',
        'helper/bgp-router-helper.h',
        ]

    bld.ns3_python_bindings()
```

modbus-udp-helper.h

This header file is located at *~/ns-3-allinone/ns-3-dev/src/applications/helper*.

```
/* -*- Mode:C++; c-file-style:"gnu"; indent-tabs-mode:nil; -*- */
/*
 * Copyright (c) 2012 Simon Fraser University
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License version 2 as
 * published by the Free Software Foundation;
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
 *
 * Author: Reza Sahraei <mrs16@sfu.ca>
 */
#ifndef MODBUS_UDP_HELPER_H
#define MODBUS_UDP_HELPER_H

#include <stdint.h>
#include "ns3/application-container.h"
#include "ns3/node-container.h"
#include "ns3/object-factory.h"
#include "ns3/ipv4-address.h"
#include "ns3/modbus-udp-server.h"
#include "ns3/modbus-udp-client.h"
```

```
namespace ns3 {
/**
 * \brief Create a server application which waits for input MODBUS UDP request
 *        and generates appropriate response according to the function code.
 *        It also uses the information carried into their payload of the packet
 *        to compute delay and to determine if some packets are lost.
 */
class ModbusUdpServerHelper
{
public:
  /**
   * Create ModbusUdpServerHelper which will make life easier for people trying
   * to set up simulations with MODBUS UDP application.
   *
   */
  ModbusUdpServerHelper ();

  /**
   * Record an attribute to be set in each Application after it is created.
   *
   * \param name the name of the attribute to set
   * \param value the value of the attribute to set
   */
  void SetAttribute (std::string name, const AttributeValue &value);

  /**
   * Create a MODBUS UDP server application on the specified node.  The Node
   * is provided as a Ptr<Node>.
   *
   * \param node The Ptr<Node> on which to create the ModbusUdpClientApplication.
   *
   * \returns An ApplicationContainer that holds a Ptr<Application> to the
   *          application created
   */
  ApplicationContainer Install (Ptr<Node> node) const;

  /**
   * Create a MODBUS UDP server application on the specified node.  The Node
   * is provided as a string name of a Node that has been previously
   * associated using the Object Name Service.
   *
   * \param nodeName The name of the node on which to create the ModbusUdpClientApplication
```

```
   *
   * \returns An ApplicationContainer that holds a Ptr<Application> to the
   *          application created
   */
  ApplicationContainer Install (std::string nodeName) const;

  /**
   * Create one MODBUS UDP server application on each of the Nodes in the
   * NodeContainer.
   *
   * \param c The nodes on which to create the Applications.  The nodes
   *          are specified by a NodeContainer.
   * \returns The applications created, one Application per Node in the
   *          NodeContainer.
   */
  ApplicationContainer Install (NodeContainer c) const;

  Ptr<ModbusUdpServer> GetServer (void);

private:
  Ptr<Application> InstallPriv (Ptr<Node> node) const;
  ObjectFactory m_factory;
  Ptr<ModbusUdpServer> m_server;
};

/**
 * \brief Create a client application which sends MODBUS UDP requests.
 * The packets carrying a 32bit sequence number and a 64 bit time stamp.
 *
 */
class ModbusUdpClientHelper
{
public:
  /**
   *  Create ModbusUdpClientHelper which will make life easier for people trying
   * to set up simulations with udp-client-server.
   *
   * \param ip The IP address of the remote MODBUS UDP server
   */
  ModbusUdpClientHelper (Ipv4Address ip);

  /**
   * Record an attribute to be set in each Application after it is created.
```

```
   *
   * \param name the name of the attribute to set
   * \param value the value of the attribute to set
   */
  void SetAttribute (std::string name, const AttributeValue &value);

  /**
   * \param c the nodes
   *
   * Create one MODBUS UDP client application on each of the input nodes
   *
   * \returns the applications created, one application per input node.
   */
  ApplicationContainer Install (NodeContainer c);

  Ptr<ModbusUdpClient> GetServer (void);
  void SetRequest (Ptr<Application> app, Time requestInterval, uint8_t *pdu);
  void SetCount (Ptr<Application> app, uint32_t count);

private:
  ObjectFactory m_factory;
  Ptr<ModbusUdpClient> m_cModbusUdpClient;
};
/**
 * Create udpTraceClient application which sends udp packets based on a trace
 * file of an MPEG4 stream. Trace files could be downloaded form :
 * http://www.tkn.tu-berlin.de/research/trace/ltvt.html (the 2 first lines of
 * the file should be removed)
 * A valid trace file is a file with 4 columns:
 * -1- the first one represents the frame index
 * -2- the second one indicates the type of the frame: I, P or B
 * -3- the third one indicates the time on which the frame was generated by the encoder
 * -4- the fourth one indicates the frame size in byte
*/
class ModbusUdpTraceClientHelper
{
public:
  /**
   * Create ModbusUdpTraceClientHelper which will make life easier for people trying
   * to set up simulations with MODBUS UDP server.
   *
   */
  ModbusUdpTraceClientHelper ();
```

```
  /**
   * Create ModbusUdpTraceClientHelper which will make life easier for people trying
   * to set up simulations with MODBUS UDP server.
   *
   * \param ip The IP address of the remote MODBUS UDP server
   * \param filename the file from which packet traces will be loaded
   */
  ModbusUdpTraceClientHelper (Ipv4Address ip, std::string filename);

  /**
    * Record an attribute to be set in each Application after it is created.
    *
    * \param name the name of the attribute to set
    * \param value the value of the attribute to set
    */
  void SetAttribute (std::string name, const AttributeValue &value);

  /**
    * \param c the nodes
    *
    * Create one udp trace client application on each of the input nodes
    *
    * \returns the applications created, one application per input node.
    */
  ApplicationContainer Install (NodeContainer c);

private:
  ObjectFactory m_factory;
};

} // namespace ns3

#endif /* MODBUS_UDP_HELPER_H */
```

modbus-udp-helper.cc

This C++ file is located at *~/ns-3-allinone/ns-3-dev/src/applications/helper*.

```
/* -*- Mode:C++; c-file-style:"gnu"; indent-tabs-mode:nil; -*- */
/*
 * Copyright (c) 2012 Simon Fraser University
 *
 * This program is free software; you can redistribute it and/or modify
```

```
 * it under the terms of the GNU General Public License version 2 as
 * published by the Free Software Foundation;
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
 *
 * Author: Reza Sahraei <mrs16@sfu.ca>
 */

#include "modbus-udp-helper.h"

//#include "udp-client-server-helper.h"

#include "ns3/udp-trace-client.h"
#include "ns3/uinteger.h"
#include "ns3/string.h"
#include "ns3/names.h"
#include <iostream>

namespace ns3 {

ModbusUdpServerHelper::ModbusUdpServerHelper ()
{
  //std::cout << "ModbusUdpServerHelper - Constructor" << std::endl;
  uint16_t port = 502; // standard MODBUS UDP port number
  m_factory.SetTypeId (ModbusUdpServer::GetTypeId ());
  SetAttribute ("Port", UintegerValue (port));
}

void
ModbusUdpServerHelper::SetAttribute (std::string name, const AttributeValue &value)
{
  m_factory.Set (name, value);
}

ApplicationContainer
ModbusUdpServerHelper::Install (Ptr<Node> node) const
```

```
{
  //std::cout << "ModbusUdpServerHelper - Install (Ptr<Node> node)" << std::endl;
  return ApplicationContainer (InstallPriv (node));
}

ApplicationContainer
ModbusUdpServerHelper::Install (std::string nodeName) const
{
  //std::cout << "ModbusUdpServerHelper - Install (std::string nodeName)" << std::endl;
  Ptr<Node> node = Names::Find<Node> (nodeName);
  return ApplicationContainer (InstallPriv (node));
}

ApplicationContainer
ModbusUdpServerHelper::Install (NodeContainer c) const
{
  //std::cout << "ModbusUdpServerHelper - Install (NodeContainer c)" << std::endl;
  ApplicationContainer apps;
  for (NodeContainer::Iterator i = c.Begin (); i != c.End (); ++i)
    {
      apps.Add (InstallPriv (*i));
    }

  return apps;
}

Ptr<Application>
ModbusUdpServerHelper::InstallPriv (Ptr<Node> node) const
{
  //std::cout << "ModbusUdpServerHelper - InstallPriv (Ptr<Node> node)" << std::endl;
  Ptr<Application> app = m_factory.Create<ModbusUdpServer> ();
  node->AddApplication (app);
  return app;
}

Ptr<ModbusUdpServer>
ModbusUdpServerHelper::GetServer (void)
{
  return m_server;
}

ModbusUdpClientHelper::ModbusUdpClientHelper (Ipv4Address address)
{
```

```
  //std::cout << "ModbusUdpClientHelper - Constructor" << std::endl;
  uint16_t port = 502;
  m_factory.SetTypeId (ModbusUdpClient::GetTypeId ());
  SetAttribute ("RemoteAddress", Ipv4AddressValue (address));
  SetAttribute ("RemotePort", UintegerValue (port));
}

void
ModbusUdpClientHelper::SetAttribute (std::string name, const AttributeValue &value)
{
  m_factory.Set (name, value);
}

ApplicationContainer
ModbusUdpClientHelper::Install (NodeContainer c)
{
  //std::cout << "ModbusUdpClientHelper - Install" << std::endl;
  ApplicationContainer apps;
  for (NodeContainer::Iterator i = c.Begin (); i != c.End (); ++i)
    {
      Ptr<Node> node = *i;
      Ptr<ModbusUdpClient> client = m_factory.Create<ModbusUdpClient> ();
      node->AddApplication (client);
      apps.Add (client);
    }
  return apps;
}

void
ModbusUdpClientHelper::SetRequest (Ptr<Application> app, Time requestInterval, uint8_t *pdu)
{
  //std::cout << "ModbusUdpClientHelper - SetRequest" << std::endl;
  app->GetObject<ModbusUdpClient>()->SetRequest (requestInterval, pdu);
}

void
ModbusUdpClientHelper::SetCount (Ptr<Application> app, uint32_t count)
{
  //std::cout << "ModbusUdpClientHelper - SetCount" << std::endl;
  app->GetObject<ModbusUdpClient>()->SetCount (count);
}

ModbusUdpTraceClientHelper::ModbusUdpTraceClientHelper ()
```

```
{
}

ModbusUdpTraceClientHelper::ModbusUdpTraceClientHelper (Ipv4Address address, std::string
filename)
{
  uint16_t port = 502;
  m_factory.SetTypeId (UdpTraceClient::GetTypeId ());
  SetAttribute ("RemoteAddress", Ipv4AddressValue (address));
  SetAttribute ("RemotePort", UintegerValue (port));
  SetAttribute ("TraceFilename", StringValue (filename));
}

void
ModbusUdpTraceClientHelper::SetAttribute (std::string name, const AttributeValue &value)
{
  m_factory.Set (name, value);
}

ApplicationContainer
ModbusUdpTraceClientHelper::Install (NodeContainer c)
{
  ApplicationContainer apps;
  for (NodeContainer::Iterator i = c.Begin (); i != c.End (); ++i)
    {
      Ptr<Node> node = *i;
      Ptr<UdpTraceClient> client = m_factory.Create<UdpTraceClient> ();
      node->AddApplication (client);
      apps.Add (client);
    }
  return apps;
}

} // namespace ns3
```

modbus-tcp-helper.h

This header file is located at *~/ns-3-allinone/ns-3-dev/src/applications/helper*.

```
/* -*- Mode:C++; c-file-style:"gnu"; indent-tabs-mode:nil; -*- */
/*
 * Copyright (c) 2012 Simon Fraser University
 *
 * This program is free software; you can redistribute it and/or modify
```

```
 * it under the terms of the GNU General Public License version 2 as
 * published by the Free Software Foundation;
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
 *
 * Author: Reza Sahraei <mrs16@sfu.ca>
 */
#ifndef MODBUS_TCP_HELPER_H
#define MODBUS_TCP_HELPER_H

#include <stdint.h>
#include "ns3/application-container.h"
#include "ns3/node-container.h"
#include "ns3/object-factory.h"
#include "ns3/ipv4-address.h"
#include "ns3/modbus-tcp-server.h"
#include "ns3/modbus-tcp-client.h"

namespace ns3 {
/**
 * \brief Create a server application which waits for input MODBUS TCP request
 *        and generates appropriate response according to the function code.
 *        It also uses the information carried into their payload of the packet
 *        to compute delay and to determine if some packets are lost.
 */
class ModbusTcpServerHelper
{
public:
  /**
   * Create ModbusTcpServerHelper which will make life easier for people trying
   * to set up simulations with MODBUS TCP application.
   *
   */
  ModbusTcpServerHelper ();

  /**
```

```
 * Record an attribute to be set in each Application after it is created.
 *
 * \param name the name of the attribute to set
 * \param value the value of the attribute to set
 */
void SetAttribute (std::string name, const AttributeValue &value);

/**
 * Create a MODBUS TCP server application on the specified node.  The Node
 * is provided as a Ptr<Node>.
 *
 * \param node The Ptr<Node> on which to create the ModbusTcpClientApplication.
 *
 * \returns An ApplicationContainer that holds a Ptr<Application> to the
 *          application created
 */
ApplicationContainer Install (Ptr<Node> node) const;

/**
 * Create a MODBUS TCP server application on the specified node.  The Node
 * is provided as a string name of a Node that has been previously
 * associated using the Object Name Service.
 *
 * \param nodeName The name of the node on which to create the ModbusTcpClientApplication
 *
 * \returns An ApplicationContainer that holds a Ptr<Application> to the
 *          application created
 */
ApplicationContainer Install (std::string nodeName) const;

/**
 * Create one MODBUS TCP server application on each of the Nodes in the
 * NodeContainer.
 *
 * \param c The nodes on which to create the Applications.  The nodes
 *          are specified by a NodeContainer.
 * \returns The applications created, one Application per Node in the
 *          NodeContainer.
 */
ApplicationContainer Install (NodeContainer c) const;

Ptr<ModbusTcpServer> GetServer (void);
```

```
private:
  Ptr<Application> InstallPriv (Ptr<Node> node) const;
  ObjectFactory m_factory;
  Ptr<ModbusTcpServer> m_server;
};

/**
 * \brief Create a client application which sends MODBUS TCP requests.
 * The packets carrying a 32bit sequence number and a 64 bit time stamp.
 *
 */
class ModbusTcpClientHelper
{
public:
  /**
   *  Create ModbusTcpClientHelper which will make life easier for people trying
   * to set up simulations with udp-client-server.
   *
   * \param ip The IP address of the remote MODBUS TCP server
   */
  ModbusTcpClientHelper (Ipv4Address ip);

  /**
   * Record an attribute to be set in each Application after it is created.
   *
   * \param name the name of the attribute to set
   * \param value the value of the attribute to set
   */
  void SetAttribute (std::string name, const AttributeValue &value);

  /**
   * \param c the nodes
   *
   * Create one MODBUS TCP client application on each of the input nodes
   *
   * \returns the applications created, one application per input node.
   */
  ApplicationContainer Install (NodeContainer c);

  Ptr<ModbusTcpClient> GetServer (void);
  void SetRequest (Ptr<Application> app, Time requestInterval, uint8_t *pdu);
  void SetCount (Ptr<Application> app, uint32_t count);
```

```
private:
  ObjectFactory m_factory;
  Ptr<ModbusTcpClient> m_cModbusTcpClient;
};

} // namespace ns3

#endif /* MODBUS_TCP_HELPER_H */
```

modbus-tcp-helper.cc

This C++ file is located at *~/ns-3-allinone/ns-3-dev/src/applications/helper*.

```
/* -*- Mode:C++; c-file-style:"gnu"; indent-tabs-mode:nil; -*- */
/*
 * Copyright (c) 2012 Simon Fraser University
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License version 2 as
 * published by the Free Software Foundation;
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
 *
 * Author: Reza Sahraei <mrs16@sfu.ca>
 */

#include "modbus-tcp-helper.h"

//#include "ns3/tcp-trace-client.h"
#include "ns3/uinteger.h"
#include "ns3/string.h"
#include "ns3/names.h"
#include <iostream>

namespace ns3 {

ModbusTcpServerHelper::ModbusTcpServerHelper ()
```

```
{
  //std::cout << "ModbusTcpServerHelper - Constructor" << std::endl;
  uint16_t port = 502; // standard MODBUS TCP port number
  m_factory.SetTypeId (ModbusTcpServer::GetTypeId ());
  SetAttribute ("Port", UintegerValue (port));
}

void
ModbusTcpServerHelper::SetAttribute (std::string name, const AttributeValue &value)
{
  m_factory.Set (name, value);
}

ApplicationContainer
ModbusTcpServerHelper::Install (Ptr<Node> node) const
{
  //std::cout << "ModbusTcpServerHelper - Install (Ptr<Node> node)" << std::endl;
  return ApplicationContainer (InstallPriv (node));
}

ApplicationContainer
ModbusTcpServerHelper::Install (std::string nodeName) const
{
  //std::cout << "ModbusTcpServerHelper - Install (std::string nodeName)" << std::endl;
  Ptr<Node> node = Names::Find<Node> (nodeName);
  return ApplicationContainer (InstallPriv (node));
}

ApplicationContainer
ModbusTcpServerHelper::Install (NodeContainer c) const
{
  //std::cout << "ModbusTcpServerHelper - Install (NodeContainer c)" << std::endl;
  ApplicationContainer apps;
  for (NodeContainer::Iterator i = c.Begin (); i != c.End (); ++i)
    {
      apps.Add (InstallPriv (*i));
    }

  return apps;
}

Ptr<Application>
ModbusTcpServerHelper::InstallPriv (Ptr<Node> node) const
```

```
{
  //std::cout << "ModbusTcpServerHelper - InstallPriv (Ptr<Node> node)" << std::endl;
  Ptr<Application> app = m_factory.Create<ModbusTcpServer> ();
  node->AddApplication (app);
  return app;
}

Ptr<ModbusTcpServer>
ModbusTcpServerHelper::GetServer (void)
{
  return m_server;
}

ModbusTcpClientHelper::ModbusTcpClientHelper (Ipv4Address address)
{
  //std::cout << "ModbusTcpClientHelper - Constructor" << std::endl;
  uint16_t port = 502;
  m_factory.SetTypeId (ModbusTcpClient::GetTypeId ());
  SetAttribute ("RemoteAddress", Ipv4AddressValue (address));
  SetAttribute ("RemotePort", UintegerValue (port));
}

void
ModbusTcpClientHelper::SetAttribute (std::string name, const AttributeValue &value)
{
  m_factory.Set (name, value);
}

ApplicationContainer
ModbusTcpClientHelper::Install (NodeContainer c)
{
  //std::cout << "ModbusTcpClientHelper - Install" << std::endl;
  ApplicationContainer apps;
  for (NodeContainer::Iterator i = c.Begin (); i != c.End (); ++i)
    {
      Ptr<Node> node = *i;
      Ptr<ModbusTcpClient> client = m_factory.Create<ModbusTcpClient> ();
      node->AddApplication (client);
      apps.Add (client);
    }
  return apps;
}
```

```
void
ModbusTcpClientHelper::SetRequest (Ptr<Application> app, Time requestInterval, uint8_t *pdu)
{
  //std::cout << "ModbusTcpClientHelper - SetRequest" << std::endl;
  app->GetObject<ModbusTcpClient>()->SetRequest (requestInterval, pdu);
}

void
ModbusTcpClientHelper::SetCount (Ptr<Application> app, uint32_t count)
{
  //std::cout << "ModbusTcpClientHelper - SetCount" << std::endl;
  app->GetObject<ModbusTcpClient>()->SetCount (count);
}
} // namespace ns3
```

modbus-pdu.h

This header file is located at *~/ns-3-allinone/ns-3-dev/src/applications/model*.

```
/* -*- Mode:C++; c-file-style:"gnu"; indent-tabs-mode:nil; -*- */
/*
 * Copyright (c) 2012 Simon Fraser University
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License version 2 as
 * published by the Free Software Foundation;
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
 *
 * Author: Reza Sahraei <mrs16@sfu.ca>
 */
#ifndef MODBUS_PDU_H
#define MODBUS_PDU_H

#include "ns3/header.h"
#include "ns3/nstime.h"
const uint32_t PDU_SIZE = 253;
```

```
namespace ns3 {
/**
 * \ingroup modbus
 * \class ModbusPdu
 * \brief Packet header for MODBUS application
 * The header is made of a pointer to uint8_t.
 */
class ModbusPdu : public Header
{
public:
  ModbusPdu ();
  virtual ~ModbusPdu ();

  /**
   * \return the pdu
   */
  uint8_t* GetPdu (void) const;

  static TypeId GetTypeId (void);
private:
  virtual TypeId GetInstanceTypeId (void) const;
  virtual void Print (std::ostream &os) const;
  virtual uint32_t GetSerializedSize (void) const;
  virtual void Serialize (Buffer::Iterator start) const;
  virtual uint32_t Deserialize (Buffer::Iterator start);

  uint8_t *m_pdu;
};

} // namespace ns3

#endif /* MODBUS_PDU_H */
```

modbus-pdu.cc

This C++ file is located at *~/ns-3-allinone/ns-3-dev/src/applications/model*.

```
/* -*- Mode:C++; c-file-style:"gnu"; indent-tabs-mode:nil; -*- */
/*
 * Copyright (c) 2012 Simon Fraser University
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License version 2 as
```

```
 * published by the Free Software Foundation;
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
 *
 * Author: Reza Sahraei <mrs16@sfu.ca>
 */
#include "ns3/assert.h"
#include "ns3/log.h"
#include "ns3/header.h"
#include "ns3/simulator.h"
#include "modbus-pdu.h"

NS_LOG_COMPONENT_DEFINE ("ModbusPdu");

namespace ns3 {

NS_OBJECT_ENSURE_REGISTERED (ModbusPdu);

ModbusPdu::ModbusPdu ()
{
  NS_LOG_FUNCTION_NOARGS ();
  m_pdu = new uint8_t [PDU_SIZE];
}

ModbusPdu::~ModbusPdu ()
{
  NS_LOG_FUNCTION_NOARGS ();
  delete [] m_pdu;
  m_pdu = 0;
}

uint8_t*
ModbusPdu::GetPdu (void) const
{
  NS_LOG_FUNCTION_NOARGS ();
  return m_pdu;
```

```
}

TypeId
ModbusPdu::GetTypeId (void)
{
  static TypeId tid = TypeId ("ns3::ModbusPdu")
    .SetParent<Header> ()
    .AddConstructor<ModbusPdu> ()
  ;
  return tid;
}

TypeId
ModbusPdu::GetInstanceTypeId (void) const
{
  return GetTypeId ();
}

void
ModbusPdu::Print (std::ostream &os) const
{
  for (uint32_t i = 0; i < PDU_SIZE; ++i)
    {
      os << m_pdu[i] << " ";
    }
  os << std::endl;
}
uint32_t
ModbusPdu::GetSerializedSize (void) const
{
  NS_LOG_FUNCTION_NOARGS ();
  return PDU_SIZE;
}

void
ModbusPdu::Serialize (Buffer::Iterator start) const
{
  NS_LOG_FUNCTION_NOARGS ();

  Buffer::Iterator cIterator = start;
  for (uint32_t i = 0; i < PDU_SIZE; ++i)
    {
      cIterator.WriteU8 (m_pdu[i]);
```

```
    }
}

uint32_t
ModbusPdu::Deserialize (Buffer::Iterator start)
{
  NS_LOG_FUNCTION_NOARGS ();

  Buffer::Iterator cIterator = start;
  for (uint32_t i = 0; i < PDU_SIZE; ++i)
    {
      m_pdu[i] = cIterator.ReadU8 ();
    }
  return GetSerializedSize ();
}

} // namespace ns3
```

modbus-mgr.h

This header file is located at *~/ns-3-allinone/ns-3-dev/src/applications/model*.

```
/* -*- Mode:C++; c-file-style:"gnu"; indent-tabs-mode:nil; -*- */
/*
 * Copyright (c) 2012 Simon Fraser University
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License version 2 as
 * published by the Free Software Foundation;
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
 *
 * Author: Reza Sahraei <mrs16@sfu.ca>
 */
#ifndef MODBUS_MGR_H
#define MODBUS_MGR_H
```

```
#include <stdint.h>
#include <string>
#include "modbus-pdu.h"

const uint32_t MAX_MODBUS_PDU = 253; // Protocol Data Unit

const uint32_t MAX_COILS_NUM = 65536;
const uint32_t MAX_INPUTS_NUM = 65536;
const uint32_t MAX_HOLDING_REGISTERS = 65536;
const uint32_t MAX_INPUT_REGISTERS = 65536;
const uint32_t MAX_EVENTS = 64;
const uint32_t MAX_FILE_NUMBER = 10; // although can be as high as 0xFFFF,
                                     // but some legacy equipment may be compromised
const uint32_t MAX_FIFO_QUEUE_SIZE = 65536;
const uint32_t MAX_FIFO_COUNT = 31;

const uint8_t READ_COILS = 0x01;
const uint8_t READ_DISCRETE_INPUTS = 0x02;
const uint8_t READ_HOLDING_REGISTERS = 0x03;
const uint8_t READ_INPUT_REGISTERS = 0x04;
const uint8_t WRITE_SINGLE_COIL = 0x05;
const uint8_t WRITE_SINGLE_REGISTER = 0x06;
const uint8_t READ_EXCEPTION_STATUS = 0x07;
const uint8_t DIAGNOSTICS = 0x08;
const uint8_t GET_COMM_EVENT_COUNTER = 0x0B;
const uint8_t GET_COMM_EVENT_LOG = 0x0C;
const uint8_t WRITE_MULTIPLE_COILS = 0x0F;
const uint8_t WRITE_MULTIPLE_REGISTERS = 0x10;
const uint8_t REPORT_SLAVE_ID = 0x11;
const uint8_t READ_FILE_RECORD = 0x14;
const uint8_t WRITE_FILE_RECORD = 0x15;
const uint8_t MASK_WRITE_REGISTER = 0x16;
const uint8_t READ_WRITE_MULTIPLE_REGISTERS = 0x17;
const uint8_t READ_FIFO_QUEUE = 0x18;
const uint8_t ENCAPSULATED_INTERFACE_TRANSPORT = 0x2B;

//
// Diagnostics sub-function codes
//
const uint16_t RETURN_QUERY_DATA = 0x0000;
const uint16_t RESTART_COMMUNICATION_OPTION = 0x0001;
const uint16_t RETURN_DIAGNOSTICS_REGISTER = 0x0002;
const uint16_t CHANGE_ASCII_INPUT_DELIMITER = 0x0003;
```

```
const uint16_t FORCE_LISTEN_ONLY = 0x0004;
// 0X05 to 0X09 RESERVED
const uint16_t CLEAR_COUNTERS_AND_DIAGNOSTICS_REGISTER = 0x000A;
const uint16_t RETURN_BUS_MESSAGE_COUNT = 0x000B;
const uint16_t RETURN_BUS_COMMUNICATION_ERROR_COUNT = 0x000C;
const uint16_t RETURN_BUS_EXCEPTION_ERROR_COUNT = 0x000D;
const uint16_t RETURN_SLAVE_MESSAGE_COUNT = 0x000E;
const uint16_t RETURN_SLAVE_NO_RESPONSE_COUNT = 0x000F;
const uint16_t RETURN_SLAVE_NAK_COUNT = 0x0010;
const uint16_t RETURN_SLAVE_BUSY_COUNT = 0x0011;
const uint16_t RETURN_BUS_CHARACTER_OVERRUN_COUNT = 0x0012;
// 0X13 RESERVED
const uint16_t CLEAR_OVERRUN_COUNTER_AND_FLAG = 0x0014;
// 0X15 to 0XFFFF RESERVED

//
// Events code
//
// Remote device MODBUS receive event
const uint8_t REMOTE_DEVICE_MODBUS_RECEIVE_EVENT = 0x80; // bit 7=1
const uint8_t COMMUNICATION_ERROR = 0x02;
const uint8_t CHARACTER_OVERRUN = 0x10;
const uint8_t CURRENTLY_IN_LISTEN_MODE_ONLY = 0x20;
const uint8_t BROADCAST_RECEIVED = 0x40;
// Remote device MODBUS send event
const uint8_t REMOTE_DEVICE_MODBUS_SEND_EVENT = 0x40; // bit 7=0, bit 6=1
const uint8_t READ_EXCEPTION_SENT = 0x01;  // exception codes 1-3
const uint8_t SLAVE_ABORT_EXCEPTION_SENT = 0x02; // exception code 4
const uint8_t SLAVE_BUSY_EXCEPTION_SENT = 0x04; // exception code 5-6
const uint8_t SLAVE_PROGRAM_NAK_EXCEPTION_SENT = 0x08; // exception code 7
const uint8_t WRITE_TIMEOUT_ERROR_OCCURRED = 0x10;
const uint8_t CURRENTLY_IN_LISTEN_MODE = 0x20;
// Remote device entered listen only mode
const uint8_t REMOTE_DEVICE_ENTERED_LISTEM_ONLY_MODE = 0x04;
// Remote device initiated communication restart
const uint8_t REMOTE_DEVICE_INITIATED_COMMUNICATION_RESTART = 0x00;

//
// Encapsulated interface transport codes (MEI type)
//
const uint8_t CANOPEN_GENERAL_REFERENCE_REQUEST_AND_RESPONSE_PDU =
0x0D;
const uint8_t READ_DEVICE_IDENTIFICATION = 0x0E;
```

```
//
// Read device ID codes
//
const uint8_t READ_BASIC_DEVICE_ID = 0x01;
const uint8_t READ_REGULAR_DEVICE_ID = 0x02;
const uint8_t READ_EXTENDED_DEVICE_ID = 0x03;
const uint8_t READ_SPECIFIC_DEVICE_ID = 0x04;
//
// Basic object IDs
//
const uint8_t VENDOR_NAME_ID = 0x00;
const uint8_t PRODUCT_CODE_ID = 0x01;
const uint8_t MAJOR_MINOR_REVISION_ID = 0x02;
const std::string VENDOR_NAME = "School of Engineering Science - Simon Fraser University"; //
Object ID: 0x00
const std::string PRODUCT_CODE = "Product Code XX"; // Object ID: 0x01
const std::string MAJOR_MINOR_REVISION = "V1.0"; // Object ID: 0x02
//
// Conformity level
//
const uint8_t CL_BASIC_ID_STREAM_ACCESS_ONLY = 0x01;
const uint8_t CL_REGULAR_ID_STREAM_ACCESS_ONLY = 0x02;
const uint8_t CL_EXTENDED_ID_STREAM_ACCESS_ONLY = 0x03;
const uint8_t CL_BASIC_ID_STREAM_ACCESS_AND_INDIVIDUAL_ACCESS = 0x81;
const uint8_t CL_REGULAR_ID_STREAM_ACCESS_AND_INDIVIDUAL_ACCESS = 0x82;
const uint8_t CL_EXTENDED_ID_STREAM_ACCESS_AND_INDIVIDUAL_ACCESS = 0x83;

namespace ns3 {
/**
 * \ingroup modbus
 * \class ModbusMgr
 * \brief Manager for MODBUS application
 */
class ModbusMgr
{
public:
  ModbusMgr ();
  virtual ~ModbusMgr ();
  uint8_t* DoMain (uint8_t * pdu);

private:
  void PushEventLog (uint8_t event);
  uint8_t* ReadCoils (uint16_t startAdr, uint16_t quantityOfCoils);
```

```
uint8_t* ReadDiscreteInputs (uint16_t startAdr, uint16_t quantityOfInputs);
uint8_t* ReadHoldingRegisters (uint16_t startAdr, uint16_t quantityOfRegisters);
uint8_t* ReadInputRegisters (uint16_t startAdr, uint16_t quantityOfRegisters);
uint8_t* WriteSingleCoil (uint16_t outputAdr, uint16_t outputValue);
uint8_t* WriteSingleRegister (uint16_t registerAdr, uint16_t registerValue);
uint8_t* ReadExceptionStatus ();
uint8_t* Diagnostics (uint16_t subfunctionCode, uint8_t * pdu);
uint8_t* GetCommEventCounter ();
uint8_t* GetCommEventLog ();
uint8_t* WriteMultipleCoils (uint16_t startAdr, uint16_t quantityOfOutputs,
      uint8_t byteCount, uint8_t * pdu);
uint8_t* WriteMultipleRegisters (uint16_t startAdr, uint16_t quantityOfRegisters,
      uint8_t byteCount, uint8_t * pdu);
uint8_t* ReportSlaveID ();
uint8_t* ReadFileRecord (uint8_t byteCount, uint8_t * pdu);
uint8_t* WriteFileRecord (uint8_t requestDataLength, uint8_t * pdu);
uint8_t* MaskWriteRegister (uint16_t referenceAdr, uint16_t and_mask, uint16_t or_mask);
uint8_t* ReadWriteMultipleRegisters (uint16_t readStartAdr, uint16_t quantityToRead,
      uint16_t writeStartAdr, uint16_t quantityToWrite, uint8_t writeByteCount,
      uint8_t * pdu);
uint8_t* ReadFifoQueue (uint16_t fifoPointerAdr);
uint8_t* encapsulatedInterfaceTransport (uint8_t meiType, uint8_t * pdu);

uint8_t *m_rspPdu;
uint8_t *m_coilsStatus;
uint8_t *m_inputsStatus;
uint16_t *m_holdingRegisters;
uint16_t *m_inputRegisters;
uint8_t m_exceptionStatusRegister;
bool m_listenOnlyMode;
std::string m_communicationEventLog;
uint16_t m_diagnosticRegister;
std::string m_endOfMessageDelimiter;
uint16_t m_busMessageCounter;
uint16_t m_busCommunicationErrorCounter;
uint16_t m_busExceptionErrorCounter;
uint16_t m_slaveMessageCounter;
uint16_t m_slaveNoResponseCounter;
uint16_t m_slaveNAKCounter;
uint16_t m_slaveBusyCounter;
uint16_t m_busCharacterOverrunCounter;
uint16_t m_commEventCounter;
uint16_t m_EventCounter; // to count the events upto MAX_EVENTS
```

```
  uint8_t *m_events;
  uint16_t **m_fileRecord;
  uint16_t *m_fifo;
  uint16_t m_fifoCount;
};

} // namespace ns3

#endif /* MODBUS_MGR_H */
```

modbus-mgr.cc

This C++ file is located at *~/ns-3-allinone/ns-3-dev/src/applications/model*.

```
/* -*- Mode:C++; c-file-style:"gnu"; indent-tabs-mode:nil; -*- */
/*
 * Copyright (c) 2012 Simon Fraser University
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License version 2 as
 * published by the Free Software Foundation;
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
 *
 * Author: Reza Sahraei <mrs16@sfu.ca>
 */

#include "modbus-mgr.h"

namespace ns3 {

ModbusMgr::ModbusMgr ()
{
  m_rspPdu = new uint8_t [PDU_SIZE];
  m_coilsStatus = new uint8_t [MAX_COILS_NUM];
  m_inputsStatus = new uint8_t [MAX_INPUTS_NUM];
  m_holdingRegisters = new uint16_t [MAX_HOLDING_REGISTERS];
```

```
  m_inputRegisters = new uint16_t [MAX_INPUT_REGISTERS];
  m_exceptionStatusRegister = 0;
  m_listenOnlyMode = false;
  m_communicationEventLog = std::string ();
  m_diagnosticRegister = 0;
  m_endOfMessageDelimiter = std::string (1, 0x0A);
  m_busMessageCounter = 0;
  m_busCommunicationErrorCounter = 0;
  m_busExceptionErrorCounter = 0;
  m_slaveMessageCounter = 0;
  m_slaveNoResponseCounter = 0;
  m_slaveNAKCounter = 0;
  m_slaveBusyCounter = 0;
  m_busCharacterOverrunCounter = 0;
  m_commEventCounter = 0;
  m_EventCounter = 0;
  m_events = new uint8_t [MAX_EVENTS];
  m_fileRecord = new uint16_t *[MAX_FILE_NUMBER]; // rows
  for (uint32_t i = 0; i < MAX_FILE_NUMBER; ++i)
    {
      m_fileRecord[i] = new uint16_t [1000]; // cols
    }
  m_fifo = new uint16_t [MAX_FIFO_QUEUE_SIZE];
  m_fifoCount = 0;
}

ModbusMgr::~ModbusMgr ()
{
  delete [] m_rspPdu; // release response PDU
  m_rspPdu = 0;
  delete [] m_coilsStatus; // release coils status
  m_coilsStatus = 0;
  delete [] m_inputsStatus; // release inputs status
  m_inputsStatus = 0;
  delete [] m_holdingRegisters; // release holding registers
  m_holdingRegisters = 0;
  delete [] m_inputRegisters; // release holding registers
  m_inputRegisters = 0;
  delete [] m_events; // release event logs
  m_events = 0;

  for (uint32_t i = 0; i < MAX_FILE_NUMBER; ++i)
    {
```

```
      delete [] m_fileRecord[i];
    }
  delete [] m_fileRecord;
  m_fileRecord = 0;

  delete [] m_fifo;
  m_fifo = 0;

  //delete [] READ_DEVICE_ID__BASIC_DEVICE_ID_OBJ_ID_00;
}

/**
 * \ingroup modbus
 * \class ModbusMgr
 * \brief Generates MODBUS response. Returns NULL if no response generated
 */
uint8_t*
ModbusMgr::DoMain (uint8_t * pdu)
{
  // increment bus message count
  ++m_busMessageCounter;

  // increment slave message count
  ++m_slaveMessageCounter;

  // increment slave no response count
  if (m_listenOnlyMode)
    {
      ++m_slaveNoResponseCounter;
    }

  // store remote device MODBUS receive event
  (m_listenOnlyMode)?
    PushEventLog                    (REMOTE_DEVICE_MODBUS_RECEIVE_EVENT                    |
CURRENTLY_IN_LISTEN_MODE_ONLY):
    PushEventLog (REMOTE_DEVICE_MODBUS_RECEIVE_EVENT);

  uint8_t functionCode = pdu[0];

  switch (functionCode)
    {
      case READ_COILS:
        {
```

```
    std::cout << "Read Coils " << std::endl;
    uint16_t startAdr;
    uint16_t quantityOfCoils;
    startAdr = (pdu[1] << 8) + pdu[2];
    quantityOfCoils = (pdu[3] << 8) + pdu[4];
    return (m_listenOnlyMode)? NULL: ReadCoils (startAdr, quantityOfCoils);
  }

case READ_DISCRETE_INPUTS:
  {
    std::cout << "Read Discrete Inputs " << std::endl;
    uint16_t startAdr;
    uint16_t quantityOfInputs;
    startAdr = (pdu[1] << 8) + pdu[2];
    quantityOfInputs = (pdu[3] << 8) + pdu[4];
    return (m_listenOnlyMode)? NULL: ReadDiscreteInputs (startAdr, quantityOfInputs);
  }

case READ_HOLDING_REGISTERS:
  {
    std::cout << "Read Holding Registers " << std::endl;
    uint16_t startAdr;
    uint16_t quantityOfRegisters;
    startAdr = (pdu[1] << 8) + pdu[2];
    quantityOfRegisters = (pdu[3] << 8) + pdu[4];
    return (m_listenOnlyMode)? NULL: ReadHoldingRegisters (startAdr, quantityOfRegisters);
  }

case READ_INPUT_REGISTERS:
  {
    std::cout << "Read Input Registers " << std::endl;
    uint16_t startAdr;
    uint16_t quantityOfRegisters;
    startAdr = (pdu[1] << 8) + pdu[2];
    quantityOfRegisters = (pdu[3] << 8) + pdu[4];
    return (m_listenOnlyMode)? NULL: ReadInputRegisters (startAdr, quantityOfRegisters);
  }

case WRITE_SINGLE_COIL:
  {
    std::cout << "Write Single Coil " << std::endl;
    uint16_t outputAdr;
    uint16_t outputValue;
```

```
    outputAdr = (pdu[1] << 8) + pdu[2];
    outputValue = (pdu[3] << 8) + pdu[4];
    return (m_listenOnlyMode)? NULL: WriteSingleCoil (outputAdr, outputValue);
  }

case WRITE_SINGLE_REGISTER:
  {
    std::cout << "Write Single Register " << std::endl;
    uint16_t registerAdr;
    uint16_t registerValue;
    registerAdr = (pdu[1] << 8) + pdu[2];
    registerValue = (pdu[3] << 8) + pdu[4];
    return (m_listenOnlyMode)? NULL: WriteSingleRegister (registerAdr, registerValue);
  }

case READ_EXCEPTION_STATUS:
  {
    std::cout << "Read Exception Status " << std::endl;
    return (m_listenOnlyMode)? NULL: ReadExceptionStatus ();
  }

case DIAGNOSTICS:
  {
    std::cout << "Diagnostics, ";
    uint16_t subfunctionCode;
    subfunctionCode = (pdu[1] << 8) + pdu[2];
    bool listenOnlyMode;
    listenOnlyMode = m_listenOnlyMode;
    return (listenOnlyMode)? NULL: Diagnostics (subfunctionCode, pdu);
  }

case GET_COMM_EVENT_COUNTER:
  {
    std::cout << "Get Comm Event Counter " << std::endl;
    return (m_listenOnlyMode)? NULL: GetCommEventCounter ();
  }

case GET_COMM_EVENT_LOG:
  {
    std::cout << "Get Comm Event Log " << std::endl;
    return (m_listenOnlyMode)? NULL: GetCommEventLog ();
  }
```

```
    case WRITE_MULTIPLE_COILS:
     {
       std::cout << "Write Multiple Coils " << std::endl;
       uint16_t startAdr;
       uint16_t quantityOfOutputs;
       uint8_t byteCount;
       startAdr = (pdu[1] << 8) + pdu[2];
       quantityOfOutputs = (pdu[3] << 8) + pdu[4];
       byteCount = pdu[5];
       return (m_listenOnlyMode)? NULL: WriteMultipleCoils (startAdr, quantityOfOutputs,
byteCount, pdu);
     }

    case WRITE_MULTIPLE_REGISTERS:
     {
       std::cout << "Write Multiple Registers " << std::endl;
       uint16_t startAdr;
       uint16_t quantityOfRegisters;
       uint8_t byteCount;
       startAdr = (pdu[1] << 8) + pdu[2];
       quantityOfRegisters = (pdu[3] << 8) + pdu[4];
       byteCount = pdu[5];
       return (m_listenOnlyMode)? NULL: WriteMultipleRegisters (startAdr, quantityOfRegisters,
byteCount, pdu);
     }

    case REPORT_SLAVE_ID:
     {
       std::cout << "Report Slave ID " << std::endl;
       return (m_listenOnlyMode)? NULL: ReportSlaveID ();
     }

    case READ_FILE_RECORD:
     {
       std::cout << "Read File Record " << std::endl;
       uint8_t byteCount;
       byteCount = pdu[1];
       return (m_listenOnlyMode)? NULL: ReadFileRecord (byteCount, pdu);
     }

    case WRITE_FILE_RECORD:
     {
       std::cout << "Write File Record " << std::endl;
```

```
      uint8_t requestDataLength;
      requestDataLength = pdu[1];
      return (m_listenOnlyMode)? NULL: WriteFileRecord (requestDataLength, pdu);
    }

  case MASK_WRITE_REGISTER:
    {
      std::cout << "Mask Write Register " << std::endl;
      uint16_t referenceAdr;
      uint16_t and_mask;
      uint16_t or_mask;
      referenceAdr = (pdu[1] << 8) + pdu[2];
      and_mask = (pdu[3] << 8) + pdu[4];
      or_mask = (pdu[5] << 8) + pdu[6];
      return (m_listenOnlyMode)? NULL: MaskWriteRegister (referenceAdr, and_mask,
or_mask);
    }

  case READ_WRITE_MULTIPLE_REGISTERS:
    {
      std::cout << "Read/Write Multiple Registers " << std::endl;
      uint16_t readStartAdr;
      uint16_t quantityToRead;
      uint16_t writeStartAdr;
      uint16_t quantityToWrite;
      uint8_t writeByteCount;
      readStartAdr = (pdu[1] << 8) + pdu[2];
      quantityToRead = (pdu[3] << 8) + pdu[4];
      writeStartAdr = (pdu[5] << 8) + pdu[6];
      quantityToWrite = (pdu[7] << 8) + pdu[8];
      writeByteCount = pdu[9];
      return (m_listenOnlyMode)? NULL: ReadWriteMultipleRegisters (readStartAdr,
quantityToRead,
        writeStartAdr, quantityToWrite, writeByteCount, pdu);
    }

  case READ_FIFO_QUEUE:
    {
      std::cout << "Read FIFO Queue " << std::endl;
      uint16_t fifoPointerAdr;
      fifoPointerAdr = (pdu[1] << 8) + pdu[2];
      return (m_listenOnlyMode)? NULL: ReadFifoQueue (fifoPointerAdr);
    }
```

```
    case ENCAPSULATED_INTERFACE_TRANSPORT:
     {
       std::cout << "Encapsulated Interface Transport " << std::endl;
       uint8_t meiType;
       meiType = pdu[1];
       return (m_listenOnlyMode)? NULL: encapsulatedInterfaceTransport (meiType, pdu);
     }

    default:
     {
       std::cout << "error: undefined function code " << std::endl;
       m_rspPdu[0] = pdu[0] + 0x80;
       m_rspPdu[1] = 0x01; // exception code 0x01, function code not supported
       return (m_listenOnlyMode)? NULL: m_rspPdu;
     }
   }

  return m_rspPdu;
}

void
ModbusMgr::PushEventLog (uint8_t event)
{
  if (m_EventCounter < MAX_EVENTS)
    {
      ++m_EventCounter;
    }

  // push down the old events
  for (uint16_t i = m_EventCounter-1; i >= 1; --i)
    {
      m_events[i] = m_events[i-1];
    }

  m_events[0] = event;
}

uint8_t*
ModbusMgr::ReadCoils (uint16_t startAdr, uint16_t quantityOfCoils)
{
  std::cout << "startAdr: " << startAdr << std::endl;
  std::cout << "quantityOfCoils: " << quantityOfCoils << std::endl;
```

```
 if ((0x0001 > quantityOfCoils) || (0x07D0 < quantityOfCoils))
  {
    m_rspPdu[0] = READ_COILS + 0x80;
    m_rspPdu[1] = 0x03; // exception code 0x03

    (m_listenOnlyMode)?
      PushEventLog                  (REMOTE_DEVICE_MODBUS_SEND_EVENT                  |
READ_EXCEPTION_SENT |
        CURRENTLY_IN_LISTEN_MODE_ONLY):
      PushEventLog                  (REMOTE_DEVICE_MODBUS_SEND_EVENT                  |
READ_EXCEPTION_SENT);

    return m_rspPdu;
  }

 if (0xFFFF < startAdr + quantityOfCoils)
  {
    m_rspPdu[0] = READ_COILS + 0x80;
    m_rspPdu[1] = 0x02; // exception code 0x02

    (m_listenOnlyMode)?
      PushEventLog                  (REMOTE_DEVICE_MODBUS_SEND_EVENT                  |
READ_EXCEPTION_SENT |
        CURRENTLY_IN_LISTEN_MODE_ONLY):
      PushEventLog                  (REMOTE_DEVICE_MODBUS_SEND_EVENT                  |
READ_EXCEPTION_SENT);

    return m_rspPdu;
  }

 m_rspPdu[0] = READ_COILS;
 uint8_t byteCount = quantityOfCoils / 8;
 if (0 != (quantityOfCoils % 8))
  {
    ++byteCount;
  }
 m_rspPdu[1] = byteCount;

 // read coils
 try
  {
    for (uint16_t i = startAdr; i < (quantityOfCoils + startAdr); ++i)
     {
       uint8_t coilsStatusRspByteIndex = i / 8;
```

```
uint8_t coilsStatusRspBitIndex = i % 8;

if (0 == (0x01 & m_coilsStatus[i])) // LSb: 0 = OFF and 1 = ON
 {
   switch (coilsStatusRspBitIndex)
    {
      case 0:
        m_rspPdu[2 + coilsStatusRspByteIndex] &= 0xFE;
        break;
      case 1:
        m_rspPdu[2 + coilsStatusRspByteIndex] &= 0xFD;
        break;
      case 2:
        m_rspPdu[2 + coilsStatusRspByteIndex] &= 0xFB;
        break;
      case 3:
        m_rspPdu[2 + coilsStatusRspByteIndex] &= 0xF7;
        break;
      case 4:
        m_rspPdu[2 + coilsStatusRspByteIndex] &= 0xEF;
        break;
      case 5:
        m_rspPdu[2 + coilsStatusRspByteIndex] &= 0xDF;
        break;
      case 6:
        m_rspPdu[2 + coilsStatusRspByteIndex] &= 0xBF;
        break;
      case 7:
        m_rspPdu[2 + coilsStatusRspByteIndex] &= 0x7F;
        break;

    }
 }
else
 {
   switch (coilsStatusRspBitIndex)
    {
      case 0:
        m_rspPdu[2 + coilsStatusRspByteIndex] |= 0x01;
        break;
      case 1:
        m_rspPdu[2 + coilsStatusRspByteIndex] |= 0x02;
        break;
```

```
            case 2:
              m_rspPdu[2 + coilsStatusRspByteIndex] |= 0x04;
              break;
            case 3:
              m_rspPdu[2 + coilsStatusRspByteIndex] |= 0x08;
              break;
            case 4:
              m_rspPdu[2 + coilsStatusRspByteIndex] |= 0x10;
              break;
            case 5:
              m_rspPdu[2 + coilsStatusRspByteIndex] |= 0x20;
              break;
            case 6:
              m_rspPdu[2 + coilsStatusRspByteIndex] |= 0x40;
              break;
            case 7:
              m_rspPdu[2 + coilsStatusRspByteIndex] |= 0x80;
              break;

          }
        }
      }
    }
   catch (...)
    {
      // exception
      m_rspPdu[0] = READ_COILS + 0x80;
      m_rspPdu[1] = 0x04; // exception code 0x04

      (m_listenOnlyMode)?
        PushEventLog                    (REMOTE_DEVICE_MODBUS_SEND_EVENT                    |
SLAVE_ABORT_EXCEPTION_SENT |
          CURRENTLY_IN_LISTEN_MODE_ONLY):
        PushEventLog                    (REMOTE_DEVICE_MODBUS_SEND_EVENT                    |
SLAVE_ABORT_EXCEPTION_SENT);

      return m_rspPdu;
    }

   ++m_commEventCounter; // increment for each successful message completion

   (m_listenOnlyMode)?
     PushEventLog                    (REMOTE_DEVICE_MODBUS_SEND_EVENT                    |
CURRENTLY_IN_LISTEN_MODE_ONLY):
```

```
   PushEventLog (REMOTE_DEVICE_MODBUS_SEND_EVENT);

  return m_rspPdu;
}

uint8_t*
ModbusMgr::ReadDiscreteInputs (uint16_t startAdr, uint16_t quantityOfInputs)
{
  std::cout << "startAdr: " << startAdr << std::endl;
  std::cout << "quantityOfInputs: " << quantityOfInputs << std::endl;
  if ((0x0001 > quantityOfInputs) || (0x07D0 < quantityOfInputs))
    {
      m_rspPdu[0] = READ_DISCRETE_INPUTS + 0x80;
      m_rspPdu[1] = 0x03; // exception code 0x03

      (m_listenOnlyMode)?
        PushEventLog                    (REMOTE_DEVICE_MODBUS_SEND_EVENT                    |
READ_EXCEPTION_SENT |
          CURRENTLY_IN_LISTEN_MODE_ONLY):
        PushEventLog                    (REMOTE_DEVICE_MODBUS_SEND_EVENT                    |
READ_EXCEPTION_SENT);

      return m_rspPdu;
    }

  if (0xFFFF < startAdr + quantityOfInputs)
    {
      m_rspPdu[0] = READ_DISCRETE_INPUTS + 0x80;
      m_rspPdu[1] = 0x02; // exception code 0x02

      (m_listenOnlyMode)?
        PushEventLog                    (REMOTE_DEVICE_MODBUS_SEND_EVENT                    |
READ_EXCEPTION_SENT |
          CURRENTLY_IN_LISTEN_MODE_ONLY):
        PushEventLog                    (REMOTE_DEVICE_MODBUS_SEND_EVENT                    |
READ_EXCEPTION_SENT);

      return m_rspPdu;
    }

  m_rspPdu[0] = READ_DISCRETE_INPUTS;
  uint8_t byteCount = quantityOfInputs / 8;
  if (0 != (quantityOfInputs % 8))
    {
```

```
    ++byteCount;
  }
m_rspPdu[1] = byteCount;

// read discrete inputs
try
  {
    for (uint16_t i = startAdr; i < (quantityOfInputs + startAdr); ++i)
      {
        uint8_t inputStatusRspByteIndex = i / 8;
        uint8_t inputStatusRspBitIndex = i % 8;
        if (0 == (0x01 & m_inputsStatus[i])) // LSb: 0 = OFF and 1 = ON
          {
            switch (inputStatusRspBitIndex)
              {
                case 0:
                  m_rspPdu[2 + inputStatusRspByteIndex] &= 0x7F;
                  break;
                case 1:
                  m_rspPdu[2 + inputStatusRspByteIndex] &= 0xBF;
                  break;
                case 2:
                  m_rspPdu[2 + inputStatusRspByteIndex] &= 0xDF;
                  break;
                case 3:
                  m_rspPdu[2 + inputStatusRspByteIndex] &= 0xEF;
                  break;
                case 4:
                  m_rspPdu[2 + inputStatusRspByteIndex] &= 0xF7;
                  break;
                case 5:
                  m_rspPdu[2 + inputStatusRspByteIndex] &= 0xFB;
                  break;
                case 6:
                  m_rspPdu[2 + inputStatusRspByteIndex] &= 0xFD;
                  break;
                case 7:
                  m_rspPdu[2 + inputStatusRspByteIndex] &= 0xFE;
                  break;
              }
          }
        else
          {
```

```
          switch (inputStatusRspBitIndex)
           {
             case 0:
               m_rspPdu[2 + inputStatusRspByteIndex] |= 0x80;
               break;
             case 1:
               m_rspPdu[2 + inputStatusRspByteIndex] |= 0x40;
               break;
             case 2:
               m_rspPdu[2 + inputStatusRspByteIndex] |= 0x20;
               break;
             case 3:
               m_rspPdu[2 + inputStatusRspByteIndex] |= 0x10;
               break;
             case 4:
               m_rspPdu[2 + inputStatusRspByteIndex] |= 0x08;
               break;
             case 5:
               m_rspPdu[2 + inputStatusRspByteIndex] |= 0x04;
               break;
             case 6:
               m_rspPdu[2 + inputStatusRspByteIndex] |= 0x02;
               break;
             case 7:
               m_rspPdu[2 + inputStatusRspByteIndex] |= 0x01;
               break;
           }
         }
       }
     }
    catch (...)
     {
       // exception
       m_rspPdu[0] = READ_DISCRETE_INPUTS + 0x80;
       m_rspPdu[1] = 0x04; // exception code 0x04

       (m_listenOnlyMode)?
         PushEventLog                (REMOTE_DEVICE_MODBUS_SEND_EVENT                |
SLAVE_ABORT_EXCEPTION_SENT |
          CURRENTLY_IN_LISTEN_MODE_ONLY):
         PushEventLog                (REMOTE_DEVICE_MODBUS_SEND_EVENT                |
SLAVE_ABORT_EXCEPTION_SENT);
```

```
    return m_rspPdu;
   }

  ++m_commEventCounter; // increment for each successful message completion

  (m_listenOnlyMode)?
    PushEventLog                    (REMOTE_DEVICE_MODBUS_SEND_EVENT                    |
CURRENTLY_IN_LISTEN_MODE_ONLY):
    PushEventLog (REMOTE_DEVICE_MODBUS_SEND_EVENT);

  return m_rspPdu;
}

uint8_t*
ModbusMgr::ReadHoldingRegisters (uint16_t startAdr, uint16_t quantityOfRegisters)
{
  std::cout << "startAdr: " << startAdr << std::endl;
  std::cout << "quantityOfRegisters: " << quantityOfRegisters << std::endl;
  if ((0x0001 > quantityOfRegisters) || (0x007D < quantityOfRegisters))
    {
      m_rspPdu[0] = READ_HOLDING_REGISTERS + 0x80;
      m_rspPdu[1] = 0x03; // exception code 0x03

      (m_listenOnlyMode)?
        PushEventLog                    (REMOTE_DEVICE_MODBUS_SEND_EVENT                    |
READ_EXCEPTION_SENT |
          CURRENTLY_IN_LISTEN_MODE_ONLY):
        PushEventLog                    (REMOTE_DEVICE_MODBUS_SEND_EVENT                    |
READ_EXCEPTION_SENT);

      return m_rspPdu;
    }

  if (0xFFFF < startAdr + quantityOfRegisters)
    {
      m_rspPdu[0] = READ_HOLDING_REGISTERS + 0x80;
      m_rspPdu[1] = 0x02; // exception code 0x02

      (m_listenOnlyMode)?
        PushEventLog                    (REMOTE_DEVICE_MODBUS_SEND_EVENT                    |
READ_EXCEPTION_SENT |
          CURRENTLY_IN_LISTEN_MODE_ONLY):
        PushEventLog                    (REMOTE_DEVICE_MODBUS_SEND_EVENT                    |
READ_EXCEPTION_SENT);
```

```
    return m_rspPdu;
  }

 // read holding registers
 try
  {
    for (uint16_t i = startAdr, rspIndex = 2; i < (quantityOfRegisters + startAdr); ++i, rspIndex+=2)
      {
        m_rspPdu[rspIndex] = (m_holdingRegisters[i] & 0xff00) >> 8;
        m_rspPdu[rspIndex+1] = (m_holdingRegisters[i] & 0xff);
      }
  }
 catch (...)
  {
    // exception
    m_rspPdu[0] = READ_HOLDING_REGISTERS + 0x80;
    m_rspPdu[1] = 0x04; // exception code 0x04

    (m_listenOnlyMode)?
      PushEventLog                  (REMOTE_DEVICE_MODBUS_SEND_EVENT                  |
SLAVE_ABORT_EXCEPTION_SENT |
       CURRENTLY_IN_LISTEN_MODE_ONLY):
      PushEventLog                  (REMOTE_DEVICE_MODBUS_SEND_EVENT                  |
SLAVE_ABORT_EXCEPTION_SENT);

    return m_rspPdu;
  }

 m_rspPdu[0] = READ_HOLDING_REGISTERS;
 m_rspPdu[1] = quantityOfRegisters * 2;

 ++m_commEventCounter; // increment for each successful message completion

 (m_listenOnlyMode)?
   PushEventLog                  (REMOTE_DEVICE_MODBUS_SEND_EVENT                  |
CURRENTLY_IN_LISTEN_MODE_ONLY):
   PushEventLog (REMOTE_DEVICE_MODBUS_SEND_EVENT);

 return m_rspPdu;
}

uint8_t*
ModbusMgr::ReadInputRegisters (uint16_t startAdr, uint16_t quantityOfRegisters)
```

```
{
  std::cout << "startAdr: " << startAdr << std::endl;
  std::cout << "quantityOfRegisters: " << quantityOfRegisters << std::endl;
  if ((0x0001 > quantityOfRegisters) || (0x007D < quantityOfRegisters))
    {
      m_rspPdu[0] = READ_INPUT_REGISTERS + 0x80;
      m_rspPdu[1] = 0x03; // exception code 0x03

      (m_listenOnlyMode)?
        PushEventLog                    (REMOTE_DEVICE_MODBUS_SEND_EVENT                    |
READ_EXCEPTION_SENT |
          CURRENTLY_IN_LISTEN_MODE_ONLY):
        PushEventLog                    (REMOTE_DEVICE_MODBUS_SEND_EVENT                    |
READ_EXCEPTION_SENT);

      return m_rspPdu;
    }

  if (0xFFFF < startAdr + quantityOfRegisters)
    {
      m_rspPdu[0] = READ_INPUT_REGISTERS + 0x80;
      m_rspPdu[1] = 0x02; // exception code 0x02

      (m_listenOnlyMode)?
        PushEventLog                    (REMOTE_DEVICE_MODBUS_SEND_EVENT                    |
READ_EXCEPTION_SENT |
          CURRENTLY_IN_LISTEN_MODE_ONLY):
        PushEventLog                    (REMOTE_DEVICE_MODBUS_SEND_EVENT                    |
READ_EXCEPTION_SENT);

      return m_rspPdu;
    }

  // read input registers
  try
    {
      for (uint16_t i = startAdr, rspIndex = 2; i < (quantityOfRegisters + startAdr); ++i, rspIndex+=2)
        {
          m_rspPdu[rspIndex] = (m_inputRegisters[i] & 0xff00) >> 8;
          m_rspPdu[rspIndex+1] = (m_inputRegisters[i] & 0xff);
        }
    }
  catch (...)
    {
```

```
    // exception
    m_rspPdu[0] = READ_INPUT_REGISTERS + 0x80;
    m_rspPdu[1] = 0x04; // exception code 0x04

    (m_listenOnlyMode)?
      PushEventLog                (REMOTE_DEVICE_MODBUS_SEND_EVENT                |
SLAVE_ABORT_EXCEPTION_SENT |
        CURRENTLY_IN_LISTEN_MODE_ONLY):
      PushEventLog                (REMOTE_DEVICE_MODBUS_SEND_EVENT                |
SLAVE_ABORT_EXCEPTION_SENT);

    return m_rspPdu;
   }

  m_rspPdu[0] = READ_INPUT_REGISTERS;
  m_rspPdu[1] = quantityOfRegisters * 2;

  ++m_commEventCounter; // increment for each successful message completion

  (m_listenOnlyMode)?
    PushEventLog                (REMOTE_DEVICE_MODBUS_SEND_EVENT                |
CURRENTLY_IN_LISTEN_MODE_ONLY):
    PushEventLog (REMOTE_DEVICE_MODBUS_SEND_EVENT);

  return m_rspPdu;
}

uint8_t*
ModbusMgr::WriteSingleCoil (uint16_t outputAdr, uint16_t outputValue)
{
  std::cout << "outputAdr: " << outputAdr << std::endl;
  std::cout << "outputValue: " << outputValue << std::endl;
  if ((0x0000 != outputValue) && (0xFF00 != outputValue))
    {
     m_rspPdu[0] = WRITE_SINGLE_COIL + 0x80;
     m_rspPdu[1] = 0x03; // exception code 0x03

     (m_listenOnlyMode)?
       PushEventLog                (REMOTE_DEVICE_MODBUS_SEND_EVENT                |
READ_EXCEPTION_SENT |
         CURRENTLY_IN_LISTEN_MODE_ONLY):
       PushEventLog                (REMOTE_DEVICE_MODBUS_SEND_EVENT                |
READ_EXCEPTION_SENT);
```

```
    return m_rspPdu;
   }

  if (0xFFFF < outputAdr) // added for completeness (uint16_t)
   {
    m_rspPdu[0] = WRITE_SINGLE_COIL + 0x80;
    m_rspPdu[1] = 0x02; // exception code 0x02

    (m_listenOnlyMode)?
      PushEventLog                  (REMOTE_DEVICE_MODBUS_SEND_EVENT                  |
READ_EXCEPTION_SENT |
        CURRENTLY_IN_LISTEN_MODE_ONLY):
      PushEventLog                  (REMOTE_DEVICE_MODBUS_SEND_EVENT                  |
READ_EXCEPTION_SENT);

    return m_rspPdu;
   }

  // set the value. LSb: 0 = OFF and 1 = ON
  try
   {
    if (0x0000 == outputValue)
     {
      // set OFF
      m_coilsStatus[outputAdr] = 0;
     }
    else
     {
      // set ON
      m_coilsStatus[outputAdr] = 1;
     }
   }
  catch (...)
   {
    // exception
    m_rspPdu[0] = WRITE_SINGLE_COIL + 0x80;
    m_rspPdu[1] = 0x04; // exception code 0x04

    (m_listenOnlyMode)?
      PushEventLog                  (REMOTE_DEVICE_MODBUS_SEND_EVENT                  |
SLAVE_ABORT_EXCEPTION_SENT |
        CURRENTLY_IN_LISTEN_MODE_ONLY):
      PushEventLog                  (REMOTE_DEVICE_MODBUS_SEND_EVENT                  |
SLAVE_ABORT_EXCEPTION_SENT);
```

```
    return m_rspPdu;
   }

  // the response is an echo of the request
  m_rspPdu[0] = WRITE_SINGLE_COIL;
  m_rspPdu[1] = (outputAdr & 0xff00) >> 8;
  m_rspPdu[2] = (outputAdr & 0xff);
  m_rspPdu[3] = (outputValue & 0xff00) >> 8;
  m_rspPdu[4] = (outputValue & 0xff);

  ++m_commEventCounter; // increment for each successful message completion

  (m_listenOnlyMode)?
    PushEventLog                    (REMOTE_DEVICE_MODBUS_SEND_EVENT                    |
CURRENTLY_IN_LISTEN_MODE_ONLY):
    PushEventLog (REMOTE_DEVICE_MODBUS_SEND_EVENT);

  return m_rspPdu;
}

uint8_t*
ModbusMgr::WriteSingleRegister (uint16_t registerAdr, uint16_t registerValue)
{
  std::cout << "registerAdr: " << registerAdr << std::endl;
  std::cout << "registerValue: " << registerValue << std::endl;
  if ((0x0000 > registerValue) || (0xFFFF < registerValue)) // added for completeness (uint16_t)
    {
      m_rspPdu[0] = WRITE_SINGLE_REGISTER + 0x80;
      m_rspPdu[1] = 0x03; // exception code 0x03

      (m_listenOnlyMode)?
        PushEventLog                    (REMOTE_DEVICE_MODBUS_SEND_EVENT                    |
READ_EXCEPTION_SENT |
          CURRENTLY_IN_LISTEN_MODE_ONLY):
        PushEventLog                    (REMOTE_DEVICE_MODBUS_SEND_EVENT                    |
READ_EXCEPTION_SENT);

      return m_rspPdu;
    }

  if (0xFFFF < registerAdr) // added for completeness (uint16_t)
    {
      m_rspPdu[0] = WRITE_SINGLE_REGISTER + 0x80;
```

```
    m_rspPdu[1] = 0x02; // exception code 0x02

    (m_listenOnlyMode)?
      PushEventLog                (REMOTE_DEVICE_MODBUS_SEND_EVENT                |
READ_EXCEPTION_SENT |
        CURRENTLY_IN_LISTEN_MODE_ONLY):
      PushEventLog                (REMOTE_DEVICE_MODBUS_SEND_EVENT                |
READ_EXCEPTION_SENT);

    return m_rspPdu;
  }

 // set the register value
 try
  {
    m_holdingRegisters[registerAdr] = registerValue;
  }
 catch (...)
  {
    // exception
    m_rspPdu[0] = WRITE_SINGLE_REGISTER + 0x80;
    m_rspPdu[1] = 0x04; // exception code 0x04

    (m_listenOnlyMode)?
      PushEventLog                (REMOTE_DEVICE_MODBUS_SEND_EVENT                |
SLAVE_ABORT_EXCEPTION_SENT |
        CURRENTLY_IN_LISTEN_MODE_ONLY):
      PushEventLog                (REMOTE_DEVICE_MODBUS_SEND_EVENT                |
SLAVE_ABORT_EXCEPTION_SENT);

    return m_rspPdu;
  }

 // the response is an echo of the request
 m_rspPdu[0] = WRITE_SINGLE_REGISTER;
 m_rspPdu[1] = (registerAdr & 0xff00) >> 8;
 m_rspPdu[2] = (registerAdr & 0xff);
 m_rspPdu[3] = (registerValue & 0xff00) >> 8;
 m_rspPdu[4] = (registerValue & 0xff);

 ++m_commEventCounter; // increment for each successful message completion

 (m_listenOnlyMode)?
```

```
   PushEventLog                    (REMOTE_DEVICE_MODBUS_SEND_EVENT                    |
CURRENTLY_IN_LISTEN_MODE_ONLY):
   PushEventLog (REMOTE_DEVICE_MODBUS_SEND_EVENT);

  return m_rspPdu;
}

uint8_t*
ModbusMgr::ReadExceptionStatus ()
{
  // read exception status
  try
    {
      m_rspPdu[0] = READ_EXCEPTION_STATUS;
      m_rspPdu[1] = m_exceptionStatusRegister;

      ++m_commEventCounter; // increment for each successful message completion

      (m_listenOnlyMode)?
        PushEventLog                    (REMOTE_DEVICE_MODBUS_SEND_EVENT                    |
CURRENTLY_IN_LISTEN_MODE_ONLY):
        PushEventLog (REMOTE_DEVICE_MODBUS_SEND_EVENT);

      return m_rspPdu;
    }
  catch (...)
    {
      // exception
      m_rspPdu[0] = READ_EXCEPTION_STATUS + 0x80;
      m_rspPdu[1] = 0x04; // exception code 0x04

      (m_listenOnlyMode)?
        PushEventLog                    (REMOTE_DEVICE_MODBUS_SEND_EVENT                    |
SLAVE_ABORT_EXCEPTION_SENT |
          CURRENTLY_IN_LISTEN_MODE_ONLY):
        PushEventLog                    (REMOTE_DEVICE_MODBUS_SEND_EVENT                    |
SLAVE_ABORT_EXCEPTION_SENT);

      return m_rspPdu;
    }
}

uint8_t*
ModbusMgr::Diagnostics (uint16_t subfunctionCode, uint8_t * pdu)
```

```
{
  std::cout << "subfunctionCode: 0x" << std::hex << subfunctionCode << " " << std::dec;

  try
    {
      switch (subfunctionCode)
        {
          case RETURN_QUERY_DATA:
            {
              std::cout << "Return query data " << std::endl;
              for (uint16_t i = 0; i < MAX_MODBUS_PDU; ++i)
                {
                  m_rspPdu[i] = pdu[i];
                }
              ++m_commEventCounter; // increment for each successful message completion

              (m_listenOnlyMode)?
                PushEventLog                (REMOTE_DEVICE_MODBUS_SEND_EVENT                |
CURRENTLY_IN_LISTEN_MODE_ONLY):
                PushEventLog (REMOTE_DEVICE_MODBUS_SEND_EVENT);

              return m_rspPdu;
            }

          case RESTART_COMMUNICATION_OPTION:
            {
              std::cout << "Restart communication option " << std::endl;
              uint16_t requestDataField;
              requestDataField = (pdu[3] << 8) + pdu[4];

              switch (requestDataField)
                {
                  case 0xFF00:
                    {
                      m_communicationEventLog.clear ();
                      m_commEventCounter = 0;

                      // create an echo response
                      for (uint16_t i = 0; i < 5; ++i)
                        {
                          m_rspPdu[i] = pdu[i];
                        }
                      break;
```

```
            }
          case 0x0000:
            {
              // create an echo response
              for (uint16_t i = 0; i < 5; ++i)
                {
                  m_rspPdu[i] = pdu[i];
                }
              ++m_commEventCounter; // increment for each successful message completion
              break;
            }
          default:
            {
              // incorrect data value
              m_rspPdu[0] = DIAGNOSTICS + 0x80;
              m_rspPdu[1] = 0x03; // exception code 0x03

              (m_listenOnlyMode)?
                PushEventLog          (REMOTE_DEVICE_MODBUS_SEND_EVENT          |
READ_EXCEPTION_SENT |
                  CURRENTLY_IN_LISTEN_MODE_ONLY):
                PushEventLog          (REMOTE_DEVICE_MODBUS_SEND_EVENT          |
READ_EXCEPTION_SENT);

              return m_rspPdu;
            }
        }

      m_listenOnlyMode = false; // the only function that brings the port out of Listen Only
Mode
      PushEventLog (REMOTE_DEVICE_INITIATED_COMMUNICATION_RESTART);
      return m_rspPdu;
    }

  case RETURN_DIAGNOSTICS_REGISTER:
    {
      std::cout << "Return diagnostics registers " << std::endl;
      uint16_t requestDataField;
      requestDataField = (pdu[3] << 8) + pdu[4];

      switch (requestDataField)
        {
          case 0x0000:
            {
```

```
            // get diagnostic register contents
            m_rspPdu[0] = DIAGNOSTICS;
            m_rspPdu[1] = (RETURN_DIAGNOSTICS_REGISTER & 0xff00) >> 8;
            m_rspPdu[2] = (RETURN_DIAGNOSTICS_REGISTER & 0xff);
            m_rspPdu[3] = (m_diagnosticRegister & 0xff00) >> 8;
            m_rspPdu[4] = (m_diagnosticRegister & 0xff);
            ++m_commEventCounter; // increment for each successful message completion

            (m_listenOnlyMode)?
              PushEventLog          (REMOTE_DEVICE_MODBUS_SEND_EVENT          |
CURRENTLY_IN_LISTEN_MODE_ONLY):
              PushEventLog (REMOTE_DEVICE_MODBUS_SEND_EVENT);

            break;
           }
          default:
           {
            // incorrect data value
            m_rspPdu[0] = DIAGNOSTICS + 0x80;
            m_rspPdu[1] = 0x03; // exception code 0x03

            (m_listenOnlyMode)?
              PushEventLog          (REMOTE_DEVICE_MODBUS_SEND_EVENT          |
READ_EXCEPTION_SENT |
                CURRENTLY_IN_LISTEN_MODE_ONLY):
              PushEventLog          (REMOTE_DEVICE_MODBUS_SEND_EVENT          |
READ_EXCEPTION_SENT);

            break;
           }
         }
        return m_rspPdu;
       }

      case CHANGE_ASCII_INPUT_DELIMITER:
       {
        std::cout << "Change ASCII input delimiter " << std::endl;

        if (0x00 != pdu[4])
         {
          // incorrect data value
          m_rspPdu[0] = DIAGNOSTICS + 0x80;
          m_rspPdu[1] = 0x03; // exception code 0x03
```

```
            (m_listenOnlyMode)?
              PushEventLog              (REMOTE_DEVICE_MODBUS_SEND_EVENT              |
READ_EXCEPTION_SENT |
                CURRENTLY_IN_LISTEN_MODE_ONLY):
              PushEventLog              (REMOTE_DEVICE_MODBUS_SEND_EVENT              |
READ_EXCEPTION_SENT);

            return m_rspPdu;
           }

          // change end of message delimiter
          m_endOfMessageDelimiter = std::string (1, pdu[3]);

          // echo request data
          for (uint16_t i = 0; i < 5; ++i)
           {
            m_rspPdu[i] = pdu[i];
           }
          ++m_commEventCounter; // increment for each successful message completion

          (m_listenOnlyMode)?
            PushEventLog              (REMOTE_DEVICE_MODBUS_SEND_EVENT              |
CURRENTLY_IN_LISTEN_MODE_ONLY):
            PushEventLog (REMOTE_DEVICE_MODBUS_SEND_EVENT);

          return m_rspPdu;
         }

        case FORCE_LISTEN_ONLY:
         {
          std::cout << "Force listen only " << std::endl;
          uint16_t requestDataField;
          requestDataField = (pdu[3] << 8) + pdu[4];

          if (0x0000 != requestDataField)
           {
            // incorrect data value
            m_rspPdu[0] = DIAGNOSTICS + 0x80;
            m_rspPdu[1] = 0x03; // exception code 0x03

            (m_listenOnlyMode)?
              PushEventLog              (REMOTE_DEVICE_MODBUS_SEND_EVENT              |
READ_EXCEPTION_SENT |
                CURRENTLY_IN_LISTEN_MODE_ONLY):
```

```
            PushEventLog            (REMOTE_DEVICE_MODBUS_SEND_EVENT            |
READ_EXCEPTION_SENT);

            return m_rspPdu;
           }

          m_listenOnlyMode = true;
          ++m_commEventCounter; // increment for each successful message completion
          PushEventLog (REMOTE_DEVICE_ENTERED_LISTEM_ONLY_MODE);
          return NULL;
         }

        case CLEAR_COUNTERS_AND_DIAGNOSTICS_REGISTER:
         {
          std::cout << "Clear counters and diagnostics register " << std::endl;
          uint16_t requestDataField;
          requestDataField = (pdu[3] << 8) + pdu[4];

          if (0x0000 != requestDataField)
           {
            // incorrect data value
            m_rspPdu[0] = DIAGNOSTICS + 0x80;
            m_rspPdu[1] = 0x03; // exception code 0x03

            (m_listenOnlyMode)?
             PushEventLog            (REMOTE_DEVICE_MODBUS_SEND_EVENT            |
READ_EXCEPTION_SENT |
              CURRENTLY_IN_LISTEN_MODE_ONLY):
             PushEventLog            (REMOTE_DEVICE_MODBUS_SEND_EVENT            |
READ_EXCEPTION_SENT);

            return m_rspPdu;
           }

          m_diagnosticRegister = 0;
          m_busMessageCounter = 0;
          m_busCommunicationErrorCounter = 0;
          m_busExceptionErrorCounter = 0;
          m_slaveMessageCounter = 0;
          m_slaveNoResponseCounter = 0;
          m_slaveNAKCounter = 0;
          m_slaveBusyCounter = 0;
          m_busCharacterOverrunCounter = 0;
          m_commEventCounter = 0;
```

```
          // echo request data
          for (uint16_t i = 0; i < 5; ++i)
           {
             m_rspPdu[i] = pdu[i];
           }
          ++m_commEventCounter; // increment for each successful message completion

          (m_listenOnlyMode)?
            PushEventLog                (REMOTE_DEVICE_MODBUS_SEND_EVENT                |
CURRENTLY_IN_LISTEN_MODE_ONLY):
            PushEventLog (REMOTE_DEVICE_MODBUS_SEND_EVENT);

          return m_rspPdu;
         }

        case RETURN_BUS_MESSAGE_COUNT:
         {
          std::cout << "Return bus message count " << std::endl;
          uint16_t requestDataField;
          requestDataField = (pdu[3] << 8) + pdu[4];

          if (0x0000 != requestDataField)
           {
             // incorrect data value
             m_rspPdu[0] = DIAGNOSTICS + 0x80;
             m_rspPdu[1] = 0x03; // exception code 0x03

             (m_listenOnlyMode)?
               PushEventLog              (REMOTE_DEVICE_MODBUS_SEND_EVENT              |
READ_EXCEPTION_SENT |
                 CURRENTLY_IN_LISTEN_MODE_ONLY):
               PushEventLog              (REMOTE_DEVICE_MODBUS_SEND_EVENT              |
READ_EXCEPTION_SENT);

             return m_rspPdu;
           }

          // get bus message count
          m_rspPdu[0] = DIAGNOSTICS;
          m_rspPdu[1] = (RETURN_BUS_MESSAGE_COUNT & 0xff00) >> 8;
          m_rspPdu[2] = (RETURN_BUS_MESSAGE_COUNT & 0xff);
          m_rspPdu[3] = (m_busMessageCounter & 0xff00) >> 8;
          m_rspPdu[4] = (m_busMessageCounter & 0xff);
```

```
        ++m_commEventCounter; // increment for each successful message completion

        (m_listenOnlyMode)?
          PushEventLog              (REMOTE_DEVICE_MODBUS_SEND_EVENT              |
CURRENTLY_IN_LISTEN_MODE_ONLY):
          PushEventLog (REMOTE_DEVICE_MODBUS_SEND_EVENT);

        return m_rspPdu;
       }

      case RETURN_BUS_COMMUNICATION_ERROR_COUNT:
       {
        std::cout << "Return bus communication error count " << std::endl;
        uint16_t requestDataField;
        requestDataField = (pdu[3] << 8) + pdu[4];

        if (0x0000 != requestDataField)
         {
          // incorrect data value
          m_rspPdu[0] = DIAGNOSTICS + 0x80;
          m_rspPdu[1] = 0x03; // exception code 0x03

          (m_listenOnlyMode)?
            PushEventLog              (REMOTE_DEVICE_MODBUS_SEND_EVENT              |
READ_EXCEPTION_SENT |
              CURRENTLY_IN_LISTEN_MODE_ONLY):
            PushEventLog              (REMOTE_DEVICE_MODBUS_SEND_EVENT              |
READ_EXCEPTION_SENT);

          return m_rspPdu;
         }

        // get bus communication error count
        m_rspPdu[0] = DIAGNOSTICS;
        m_rspPdu[1] = (RETURN_BUS_COMMUNICATION_ERROR_COUNT & 0xff00) >> 8;
        m_rspPdu[2] = (RETURN_BUS_COMMUNICATION_ERROR_COUNT & 0xff);
        m_rspPdu[3] = (m_busCommunicationErrorCounter & 0xff00) >> 8;
        m_rspPdu[4] = (m_busCommunicationErrorCounter & 0xff);
        ++m_commEventCounter; // increment for each successful message completion

        (m_listenOnlyMode)?
          PushEventLog              (REMOTE_DEVICE_MODBUS_SEND_EVENT              |
CURRENTLY_IN_LISTEN_MODE_ONLY):
          PushEventLog (REMOTE_DEVICE_MODBUS_SEND_EVENT);
```

```
        return m_rspPdu;
      }

    case RETURN_BUS_EXCEPTION_ERROR_COUNT:
      {
        std::cout << "Return bus exception error count " << std::endl;
        uint16_t requestDataField;
        requestDataField = (pdu[3] << 8) + pdu[4];

        if (0x0000 != requestDataField)
          {
            // incorrect data value
            m_rspPdu[0] = DIAGNOSTICS + 0x80;
            m_rspPdu[1] = 0x03; // exception code 0x03

            (m_listenOnlyMode)?
              PushEventLog              (REMOTE_DEVICE_MODBUS_SEND_EVENT              |
READ_EXCEPTION_SENT |
                CURRENTLY_IN_LISTEN_MODE_ONLY):
              PushEventLog              (REMOTE_DEVICE_MODBUS_SEND_EVENT              |
READ_EXCEPTION_SENT);

            return m_rspPdu;
          }

        // get bus exception error count
        m_rspPdu[0] = DIAGNOSTICS;
        m_rspPdu[1] = (RETURN_BUS_EXCEPTION_ERROR_COUNT & 0xff00) >> 8;
        m_rspPdu[2] = (RETURN_BUS_EXCEPTION_ERROR_COUNT & 0xff);
        m_rspPdu[3] = (m_busExceptionErrorCounter & 0xff00) >> 8;
        m_rspPdu[4] = (m_busExceptionErrorCounter & 0xff);
        ++m_commEventCounter; // increment for each successful message completion

        (m_listenOnlyMode)?
          PushEventLog              (REMOTE_DEVICE_MODBUS_SEND_EVENT              |
CURRENTLY_IN_LISTEN_MODE_ONLY):
          PushEventLog (REMOTE_DEVICE_MODBUS_SEND_EVENT);

        return m_rspPdu;
      }

    case RETURN_SLAVE_MESSAGE_COUNT:
      {
```

```
            std::cout << "Return slave message count " << std::endl;
            uint16_t requestDataField;
            requestDataField = (pdu[3] << 8) + pdu[4];

            if (0x0000 != requestDataField)
             {
               // incorrect data value
               m_rspPdu[0] = DIAGNOSTICS + 0x80;
               m_rspPdu[1] = 0x03; // exception code 0x03

               (m_listenOnlyMode)?
                 PushEventLog               (REMOTE_DEVICE_MODBUS_SEND_EVENT               |
READ_EXCEPTION_SENT |
                   CURRENTLY_IN_LISTEN_MODE_ONLY):
                 PushEventLog               (REMOTE_DEVICE_MODBUS_SEND_EVENT               |
READ_EXCEPTION_SENT);

               return m_rspPdu;
             }

            // get slave message count
            m_rspPdu[0] = DIAGNOSTICS;
            m_rspPdu[1] = (RETURN_SLAVE_MESSAGE_COUNT & 0xff00) >> 8;
            m_rspPdu[2] = (RETURN_SLAVE_MESSAGE_COUNT & 0xff);
            m_rspPdu[3] = (m_slaveMessageCounter & 0xff00) >> 8;
            m_rspPdu[4] = (m_slaveMessageCounter & 0xff);
            ++m_commEventCounter; // increment for each successful message completion

            (m_listenOnlyMode)?
              PushEventLog                 (REMOTE_DEVICE_MODBUS_SEND_EVENT               |
CURRENTLY_IN_LISTEN_MODE_ONLY):
              PushEventLog (REMOTE_DEVICE_MODBUS_SEND_EVENT);

            return m_rspPdu;
           }

         case RETURN_SLAVE_NO_RESPONSE_COUNT:
          {
            std::cout << "Return slave no response count " << std::endl;
            uint16_t requestDataField;
            requestDataField = (pdu[3] << 8) + pdu[4];

            if (0x0000 != requestDataField)
             {
```

```
            // incorrect data value
            m_rspPdu[0] = DIAGNOSTICS + 0x80;
            m_rspPdu[1] = 0x03; // exception code 0x03

            (m_listenOnlyMode)?
              PushEventLog              (REMOTE_DEVICE_MODBUS_SEND_EVENT              |
READ_EXCEPTION_SENT |
                CURRENTLY_IN_LISTEN_MODE_ONLY):
              PushEventLog              (REMOTE_DEVICE_MODBUS_SEND_EVENT              |
READ_EXCEPTION_SENT);

            return m_rspPdu;
           }

          // get slave no response count
          m_rspPdu[0] = DIAGNOSTICS;
          m_rspPdu[1] = (RETURN_SLAVE_NO_RESPONSE_COUNT & 0xff00) >> 8;
          m_rspPdu[2] = (RETURN_SLAVE_NO_RESPONSE_COUNT & 0xff);
          m_rspPdu[3] = (m_slaveNoResponseCounter & 0xff00) >> 8;
          m_rspPdu[4] = (m_slaveNoResponseCounter & 0xff);
          ++m_commEventCounter; // increment for each successful message completion

          (m_listenOnlyMode)?
            PushEventLog              (REMOTE_DEVICE_MODBUS_SEND_EVENT              |
CURRENTLY_IN_LISTEN_MODE_ONLY):
            PushEventLog (REMOTE_DEVICE_MODBUS_SEND_EVENT);

          return m_rspPdu;
         }

        case RETURN_SLAVE_NAK_COUNT:
         {
          std::cout << "Return slave NAK count " << std::endl;
          uint16_t requestDataField;
          requestDataField = (pdu[3] << 8) + pdu[4];

          if (0x0000 != requestDataField)
           {
            // incorrect data value
            m_rspPdu[0] = DIAGNOSTICS + 0x80;
            m_rspPdu[1] = 0x03; // exception code 0x03

            (m_listenOnlyMode)?
```

```
            PushEventLog              (REMOTE_DEVICE_MODBUS_SEND_EVENT              |
READ_EXCEPTION_SENT |
              CURRENTLY_IN_LISTEN_MODE_ONLY):
            PushEventLog              (REMOTE_DEVICE_MODBUS_SEND_EVENT              |
READ_EXCEPTION_SENT);

           return m_rspPdu;
          }

         // get slave NAL count
         m_rspPdu[0] = DIAGNOSTICS;
         m_rspPdu[1] = (RETURN_SLAVE_NAK_COUNT & 0xff00) >> 8;
         m_rspPdu[2] = (RETURN_SLAVE_NAK_COUNT & 0xff);
         m_rspPdu[3] = (m_slaveNAKCounter & 0xff00) >> 8;
         m_rspPdu[4] = (m_slaveNAKCounter & 0xff);
         ++m_commEventCounter; // increment for each successful message completion

         (m_listenOnlyMode)?
          PushEventLog              (REMOTE_DEVICE_MODBUS_SEND_EVENT              |
CURRENTLY_IN_LISTEN_MODE_ONLY):
          PushEventLog (REMOTE_DEVICE_MODBUS_SEND_EVENT);

         return m_rspPdu;
        }

       case RETURN_SLAVE_BUSY_COUNT:
        {
         std::cout << "Return slave busy count " << std::endl;
         uint16_t requestDataField;
         requestDataField = (pdu[3] << 8) + pdu[4];

         if (0x0000 != requestDataField)
          {
           // incorrect data value
           m_rspPdu[0] = DIAGNOSTICS + 0x80;
           m_rspPdu[1] = 0x03; // exception code 0x03

           (m_listenOnlyMode)?
            PushEventLog              (REMOTE_DEVICE_MODBUS_SEND_EVENT              |
READ_EXCEPTION_SENT |
              CURRENTLY_IN_LISTEN_MODE_ONLY):
            PushEventLog              (REMOTE_DEVICE_MODBUS_SEND_EVENT              |
READ_EXCEPTION_SENT);
```

```
            return m_rspPdu;
          }

        // get slave busy count
        m_rspPdu[0] = DIAGNOSTICS;
        m_rspPdu[1] = (RETURN_SLAVE_BUSY_COUNT & 0xff00) >> 8;
        m_rspPdu[2] = (RETURN_SLAVE_BUSY_COUNT & 0xff);
        m_rspPdu[3] = (m_slaveBusyCounter & 0xff00) >> 8;
        m_rspPdu[4] = (m_slaveBusyCounter & 0xff);
        ++m_commEventCounter; // increment for each successful message completion

        (m_listenOnlyMode)?
          PushEventLog              (REMOTE_DEVICE_MODBUS_SEND_EVENT              |
CURRENTLY_IN_LISTEN_MODE_ONLY):
          PushEventLog (REMOTE_DEVICE_MODBUS_SEND_EVENT);

        return m_rspPdu;
       }

      case RETURN_BUS_CHARACTER_OVERRUN_COUNT:
       {
        std::cout << "Return bus character overrun count " << std::endl;
        uint16_t requestDataField;
        requestDataField = (pdu[3] << 8) + pdu[4];

        if (0x0000 != requestDataField)
         {
          // incorrect data value
          m_rspPdu[0] = DIAGNOSTICS + 0x80;
          m_rspPdu[1] = 0x03; // exception code 0x03

          (m_listenOnlyMode)?
            PushEventLog              (REMOTE_DEVICE_MODBUS_SEND_EVENT              |
READ_EXCEPTION_SENT |
              CURRENTLY_IN_LISTEN_MODE_ONLY):
            PushEventLog              (REMOTE_DEVICE_MODBUS_SEND_EVENT              |
READ_EXCEPTION_SENT);

          return m_rspPdu;
         }

        // get bus character overrun count
        m_rspPdu[0] = DIAGNOSTICS;
        m_rspPdu[1] = (RETURN_BUS_CHARACTER_OVERRUN_COUNT & 0xff00) >> 8;
```

```
          m_rspPdu[2] = (RETURN_BUS_CHARACTER_OVERRUN_COUNT & 0xff);
          m_rspPdu[3] = (m_busCharacterOverrunCounter & 0xff00) >> 8;
          m_rspPdu[4] = (m_busCharacterOverrunCounter & 0xff);
          ++m_commEventCounter; // increment for each successful message completion

          (m_listenOnlyMode)?
            PushEventLog                (REMOTE_DEVICE_MODBUS_SEND_EVENT                |
CURRENTLY_IN_LISTEN_MODE_ONLY):
            PushEventLog (REMOTE_DEVICE_MODBUS_SEND_EVENT);

          return m_rspPdu;
         }

        case CLEAR_OVERRUN_COUNTER_AND_FLAG:
         {
          std::cout << "Clear overrun counter and flag " << std::endl;
          uint16_t requestDataField;
          requestDataField = (pdu[3] << 8) + pdu[4];

          if (0x0000 != requestDataField)
           {
            // incorrect data value
            m_rspPdu[0] = DIAGNOSTICS + 0x80;
            m_rspPdu[1] = 0x03; // exception code 0x03

            (m_listenOnlyMode)?
              PushEventLog                (REMOTE_DEVICE_MODBUS_SEND_EVENT                |
READ_EXCEPTION_SENT |
                CURRENTLY_IN_LISTEN_MODE_ONLY):
              PushEventLog                (REMOTE_DEVICE_MODBUS_SEND_EVENT                |
READ_EXCEPTION_SENT);

            return m_rspPdu;
           }

          // clears the overrun error counter
          m_busCharacterOverrunCounter = 0;

          // echo request data
          m_rspPdu[0] = DIAGNOSTICS;
          m_rspPdu[1] = (CLEAR_OVERRUN_COUNTER_AND_FLAG & 0xff00) >> 8;
          m_rspPdu[2] = (CLEAR_OVERRUN_COUNTER_AND_FLAG & 0xff);
          m_rspPdu[3] = 0;
          m_rspPdu[4] = 0;
```

```
          ++m_commEventCounter; // increment for each successful message completion

          (m_listenOnlyMode)?
            PushEventLog                (REMOTE_DEVICE_MODBUS_SEND_EVENT                |
CURRENTLY_IN_LISTEN_MODE_ONLY):
            PushEventLog (REMOTE_DEVICE_MODBUS_SEND_EVENT);

          return m_rspPdu;
         }

        default:
         {
          std::cout << "error: undefined sub-function code " << std::endl;
          m_rspPdu[0] = DIAGNOSTICS + 0x80;
          m_rspPdu[1] = 0x01; // exception code 0x01, sub-function code not supported

          (m_listenOnlyMode)?
            PushEventLog                (REMOTE_DEVICE_MODBUS_SEND_EVENT                |
READ_EXCEPTION_SENT |
              CURRENTLY_IN_LISTEN_MODE_ONLY):
            PushEventLog                (REMOTE_DEVICE_MODBUS_SEND_EVENT                |
READ_EXCEPTION_SENT);

          return m_rspPdu;
         }
       }
    }
   catch (...)
    {
     // exception
     m_rspPdu[0] = DIAGNOSTICS + 0x80;
     m_rspPdu[1] = 0x04; // exception code 0x04

     (m_listenOnlyMode)?
       PushEventLog                    (REMOTE_DEVICE_MODBUS_SEND_EVENT                |
SLAVE_ABORT_EXCEPTION_SENT |
        CURRENTLY_IN_LISTEN_MODE_ONLY):
       PushEventLog                    (REMOTE_DEVICE_MODBUS_SEND_EVENT                |
SLAVE_ABORT_EXCEPTION_SENT);

     return m_rspPdu;
    }
}
```

```
uint8_t*
ModbusMgr::GetCommEventCounter ()
{
  // read exception status
  try
    {
      m_rspPdu[0] = GET_COMM_EVENT_COUNTER;
      m_rspPdu[1] = 0;
      m_rspPdu[2] = 0;
      m_rspPdu[3] = (m_commEventCounter & 0xff00) >> 8;
      m_rspPdu[4] = (m_commEventCounter & 0xff);

      (m_listenOnlyMode)?
        PushEventLog                    (REMOTE_DEVICE_MODBUS_SEND_EVENT                    |
CURRENTLY_IN_LISTEN_MODE_ONLY):
        PushEventLog (REMOTE_DEVICE_MODBUS_SEND_EVENT);

      return m_rspPdu;
    }
  catch (...)
    {
      // exception
      m_rspPdu[0] = GET_COMM_EVENT_COUNTER + 0x80;
      m_rspPdu[1] = 0x04; // exception code 0x04

      (m_listenOnlyMode)?
        PushEventLog                    (REMOTE_DEVICE_MODBUS_SEND_EVENT                    |
SLAVE_ABORT_EXCEPTION_SENT |
          CURRENTLY_IN_LISTEN_MODE_ONLY):
        PushEventLog                    (REMOTE_DEVICE_MODBUS_SEND_EVENT                    |
SLAVE_ABORT_EXCEPTION_SENT);

      return m_rspPdu;
    }
}

uint8_t*
ModbusMgr::GetCommEventLog ()
{
  // read exception status
  try
    {
      m_rspPdu[0] = GET_COMM_EVENT_LOG;
      m_rspPdu[1] = 6 + m_EventCounter; // byte count
```

```
    m_rspPdu[2] = 0;
    m_rspPdu[3] = 0;
    m_rspPdu[4] = (m_commEventCounter & 0xff00) >> 8;
    m_rspPdu[5] = (m_commEventCounter & 0xff);
    m_rspPdu[6] = (m_busMessageCounter & 0xff00) >> 8;
    m_rspPdu[7] = (m_busMessageCounter & 0xff);

    // copy the events in the response
    for (uint16_t i = 0; i < m_EventCounter; ++i)
     {
       m_rspPdu[8+i] = m_events[i];
     }

    (m_listenOnlyMode)?
      PushEventLog                    (REMOTE_DEVICE_MODBUS_SEND_EVENT                    |
CURRENTLY_IN_LISTEN_MODE_ONLY):
      PushEventLog (REMOTE_DEVICE_MODBUS_SEND_EVENT);

    return m_rspPdu;
   }
  catch (...)
   {
    // exception
    m_rspPdu[0] = GET_COMM_EVENT_LOG + 0x80;
    m_rspPdu[1] = 0x04; // exception code 0x04

    (m_listenOnlyMode)?
      PushEventLog                    (REMOTE_DEVICE_MODBUS_SEND_EVENT                    |
SLAVE_ABORT_EXCEPTION_SENT |
       CURRENTLY_IN_LISTEN_MODE_ONLY):
      PushEventLog                    (REMOTE_DEVICE_MODBUS_SEND_EVENT                    |
SLAVE_ABORT_EXCEPTION_SENT);

    return m_rspPdu;
   }
}

uint8_t*
ModbusMgr::WriteMultipleCoils (uint16_t startAdr, uint16_t quantityOfOutputs, uint8_t byteCount,
uint8_t * pdu)
{
  std::cout << "startAdr: " << startAdr << std::endl;
  std::cout << "quantityOfOutputs: " << quantityOfOutputs << std::endl;
  std::cout << "byteCount: " << (byteCount + 0) << std::endl;
```

```
 if ((0x0001 > quantityOfOutputs) || (0x07B0 < quantityOfOutputs))
  {
    m_rspPdu[0] = WRITE_MULTIPLE_COILS + 0x80;
    m_rspPdu[1] = 0x03; // exception code 0x03

    (m_listenOnlyMode)?
      PushEventLog                  (REMOTE_DEVICE_MODBUS_SEND_EVENT                  |
READ_EXCEPTION_SENT |
        CURRENTLY_IN_LISTEN_MODE_ONLY):
      PushEventLog                  (REMOTE_DEVICE_MODBUS_SEND_EVENT                  |
READ_EXCEPTION_SENT);

    return m_rspPdu;
  }

 uint8_t byteCountTemp = quantityOfOutputs / 8;
 if (0 != (quantityOfOutputs % 8))
  {
    ++byteCountTemp;
  }

 if (byteCountTemp != byteCount)
  {
    m_rspPdu[0] = WRITE_MULTIPLE_COILS + 0x80;
    m_rspPdu[1] = 0x03; // exception code 0x03

    (m_listenOnlyMode)?
      PushEventLog                  (REMOTE_DEVICE_MODBUS_SEND_EVENT                  |
READ_EXCEPTION_SENT |
        CURRENTLY_IN_LISTEN_MODE_ONLY):
      PushEventLog                  (REMOTE_DEVICE_MODBUS_SEND_EVENT                  |
READ_EXCEPTION_SENT);

    return m_rspPdu;
  }

 if (0xFFFF < startAdr + quantityOfOutputs)
  {
    m_rspPdu[0] = WRITE_MULTIPLE_COILS + 0x80;
    m_rspPdu[1] = 0x02; // exception code 0x02

    (m_listenOnlyMode)?
      PushEventLog                  (REMOTE_DEVICE_MODBUS_SEND_EVENT                  |
READ_EXCEPTION_SENT |
```

```
        CURRENTLY_IN_LISTEN_MODE_ONLY):
      PushEventLog                    (REMOTE_DEVICE_MODBUS_SEND_EVENT                    |
READ_EXCEPTION_SENT);

    return m_rspPdu;
  }

 // write multiple coils
 try
  {
    uint8_t bitMask = 0x01;
    uint16_t pduRequestOutputByteIndex = 6;
    for (uint16_t i = 0, coilIndex = startAdr; i < quantityOfOutputs; ++i, ++coilIndex)
     {
       if (pdu[pduRequestOutputByteIndex] & bitMask)
        {
          // turn on the coil
          m_coilsStatus[coilIndex] = 1; // LSb: 0 = OFF and 1 = ON
        }
       else
        {
          // turn off the coil
          m_coilsStatus[coilIndex] = 0; // LSb: 0 = OFF and 1 = ON
        }

       // shift the bit mask
       bitMask <<= 1;

       if (0 == bitMask)
        {
          bitMask = 0x1;
          ++pduRequestOutputByteIndex;
        }
     }
  }
 catch (...)
  {
    // exception
    m_rspPdu[0] = WRITE_MULTIPLE_COILS + 0x80;
    m_rspPdu[1] = 0x04; // exception code 0x04

    (m_listenOnlyMode)?
```

```
      PushEventLog                    (REMOTE_DEVICE_MODBUS_SEND_EVENT                    |
SLAVE_ABORT_EXCEPTION_SENT |
       CURRENTLY_IN_LISTEN_MODE_ONLY):
      PushEventLog                    (REMOTE_DEVICE_MODBUS_SEND_EVENT                    |
SLAVE_ABORT_EXCEPTION_SENT);

    return m_rspPdu;
   }

  m_rspPdu[0] = WRITE_MULTIPLE_COILS;
  m_rspPdu[1] = (startAdr & 0xff00) >> 8;
  m_rspPdu[2] = (startAdr & 0xff);
  m_rspPdu[3] = (quantityOfOutputs & 0xff00) >> 8;
  m_rspPdu[4] = (quantityOfOutputs & 0xff);

  ++m_commEventCounter; // increment for each successful message completion

  (m_listenOnlyMode)?
   PushEventLog                    (REMOTE_DEVICE_MODBUS_SEND_EVENT                    |
CURRENTLY_IN_LISTEN_MODE_ONLY):
   PushEventLog (REMOTE_DEVICE_MODBUS_SEND_EVENT);

  return m_rspPdu;
}

uint8_t*
ModbusMgr::WriteMultipleRegisters (uint16_t startAdr, uint16_t quantityOfRegisters, uint8_t
byteCount, uint8_t * pdu)
{
  std::cout << "startAdr: " << startAdr << std::endl;
  std::cout << "quantityOfRegisters: " << quantityOfRegisters << std::endl;
  std::cout << "byteCount: " << (byteCount + 0) << std::endl;
  if ((0x0001 > quantityOfRegisters) || (0x007B < quantityOfRegisters))
   {
    m_rspPdu[0] = WRITE_MULTIPLE_REGISTERS + 0x80;
    m_rspPdu[1] = 0x03; // exception code 0x03

    (m_listenOnlyMode)?
      PushEventLog                    (REMOTE_DEVICE_MODBUS_SEND_EVENT                    |
READ_EXCEPTION_SENT |
       CURRENTLY_IN_LISTEN_MODE_ONLY):
      PushEventLog                    (REMOTE_DEVICE_MODBUS_SEND_EVENT                    |
READ_EXCEPTION_SENT);
```

```
    return m_rspPdu;
   }

  if ((quantityOfRegisters * 2) != byteCount)
   {
    m_rspPdu[0] = WRITE_MULTIPLE_REGISTERS + 0x80;
    m_rspPdu[1] = 0x03; // exception code 0x03

    (m_listenOnlyMode)?
      PushEventLog                (REMOTE_DEVICE_MODBUS_SEND_EVENT                |
READ_EXCEPTION_SENT |
        CURRENTLY_IN_LISTEN_MODE_ONLY):
      PushEventLog                (REMOTE_DEVICE_MODBUS_SEND_EVENT                |
READ_EXCEPTION_SENT);

    return m_rspPdu;
   }

  if (0xFFFF < startAdr + quantityOfRegisters)
   {
    m_rspPdu[0] = WRITE_MULTIPLE_REGISTERS + 0x80;
    m_rspPdu[1] = 0x02; // exception code 0x02

    (m_listenOnlyMode)?
      PushEventLog                (REMOTE_DEVICE_MODBUS_SEND_EVENT                |
READ_EXCEPTION_SENT |
        CURRENTLY_IN_LISTEN_MODE_ONLY):
      PushEventLog                (REMOTE_DEVICE_MODBUS_SEND_EVENT                |
READ_EXCEPTION_SENT);

    return m_rspPdu;
   }

  // write multiple registers
  try
   {
    uint16_t pduRequestOutputByteIndex = 6;
    for (uint16_t i = 0, registerIndex = startAdr; i < quantityOfRegisters; ++i, ++registerIndex)
     {
      m_holdingRegisters[registerIndex]    =    (pdu[pduRequestOutputByteIndex]    <<    8)    +
pdu[pduRequestOutputByteIndex + 1];
      pduRequestOutputByteIndex += 2;
     }
   }
```

```
  catch (...)
    {
      // exception
      m_rspPdu[0] = WRITE_MULTIPLE_REGISTERS + 0x80;
      m_rspPdu[1] = 0x04; // exception code 0x04

      (m_listenOnlyMode)?
        PushEventLog                    (REMOTE_DEVICE_MODBUS_SEND_EVENT                    |
SLAVE_ABORT_EXCEPTION_SENT |
          CURRENTLY_IN_LISTEN_MODE_ONLY):
        PushEventLog                    (REMOTE_DEVICE_MODBUS_SEND_EVENT                    |
SLAVE_ABORT_EXCEPTION_SENT);

      return m_rspPdu;
    }

  m_rspPdu[0] = WRITE_MULTIPLE_REGISTERS;
  m_rspPdu[1] = (startAdr & 0xff00) >> 8;
  m_rspPdu[2] = (startAdr & 0xff);
  m_rspPdu[3] = (quantityOfRegisters & 0xff00) >> 8;
  m_rspPdu[4] = (quantityOfRegisters & 0xff);

  ++m_commEventCounter; // increment for each successful message completion

  (m_listenOnlyMode)?
    PushEventLog                    (REMOTE_DEVICE_MODBUS_SEND_EVENT                    |
CURRENTLY_IN_LISTEN_MODE_ONLY):
    PushEventLog (REMOTE_DEVICE_MODBUS_SEND_EVENT);

  return m_rspPdu;
}

uint8_t*
ModbusMgr::ReportSlaveID ()
{
  // report slave id
  try
    {
      m_rspPdu[0] = REPORT_SLAVE_ID;
      m_rspPdu[1] = 3; // byte count
      m_rspPdu[2] = 0; // slave id
      m_rspPdu[3] = 0xFF; // run indicator status. 0x00 = OFF, 0xFF = ON
      m_rspPdu[4] = 0; // additional information
```

```
   }
  catch (...)
   {
     // exception
     m_rspPdu[0] = REPORT_SLAVE_ID + 0x80;
     m_rspPdu[1] = 0x04; // exception code 0x04

     (m_listenOnlyMode)?
       PushEventLog                    (REMOTE_DEVICE_MODBUS_SEND_EVENT                    |
SLAVE_ABORT_EXCEPTION_SENT |
         CURRENTLY_IN_LISTEN_MODE_ONLY):
       PushEventLog                    (REMOTE_DEVICE_MODBUS_SEND_EVENT                    |
SLAVE_ABORT_EXCEPTION_SENT);

     return m_rspPdu;
   }

  ++m_commEventCounter; // increment for each successful message completion

  (m_listenOnlyMode)?
    PushEventLog                    (REMOTE_DEVICE_MODBUS_SEND_EVENT                    |
CURRENTLY_IN_LISTEN_MODE_ONLY):
    PushEventLog (REMOTE_DEVICE_MODBUS_SEND_EVENT);

  return m_rspPdu;
}

uint8_t*
ModbusMgr::ReadFileRecord (uint8_t byteCount, uint8_t * pdu)
{
  std::cout << "byteCount: " << (byteCount + 0) << std::endl;
  if ((0x0007 > byteCount) || (0x00F5 < byteCount))
    {
     m_rspPdu[0] = READ_FILE_RECORD + 0x80;
     m_rspPdu[1] = 0x03; // exception code 0x03

     (m_listenOnlyMode)?
       PushEventLog                    (REMOTE_DEVICE_MODBUS_SEND_EVENT                    |
READ_EXCEPTION_SENT |
         CURRENTLY_IN_LISTEN_MODE_ONLY):
       PushEventLog                    (REMOTE_DEVICE_MODBUS_SEND_EVENT                    |
READ_EXCEPTION_SENT);

     return m_rspPdu;
```

```
  }

 uint32_t pduRequestInputByteIndex = 2;
 uint32_t pduRequestOutputByteIndex = 2;
 uint32_t subRequestNumber = 1;
 for (uint16_t i = 0; i < byteCount; i+=7)
  {
    uint8_t referenceType = pdu[pduRequestInputByteIndex++];
    uint16_t fileNumber = (pdu[pduRequestInputByteIndex] << 8) +
pdu[pduRequestInputByteIndex + 1]; // one based
    pduRequestInputByteIndex += 2;
    uint16_t recordNumber = (pdu[pduRequestInputByteIndex] << 8) +
pdu[pduRequestInputByteIndex + 1]; // zero based
    pduRequestInputByteIndex += 2;
    uint16_t recordLength = (pdu[pduRequestInputByteIndex] << 8) +
pdu[pduRequestInputByteIndex + 1];
    pduRequestInputByteIndex += 2;

    std::cout << "Sub-Req: " << (subRequestNumber + 0) << std::endl;
    std::cout << "referenceType: " << (referenceType + 0) << std::endl;
    std::cout << "fileNumber: " << (fileNumber + 0) << std::endl; // one based
    std::cout << "recordNumber: " << (recordNumber + 0) << std::endl; // zero based
    std::cout << "recordLength: " << (recordLength + 0) << std::endl;
    ++subRequestNumber;

    if ((0x06 != referenceType) ||
      (0x0001 > fileNumber || MAX_FILE_NUMBER < fileNumber) ||
      (0x0000 > recordNumber || 0x270F < recordNumber) ||
      (0x270f < recordNumber + recordLength))
     {
       m_rspPdu[0] = READ_FILE_RECORD + 0x80;
       m_rspPdu[1] = 0x02; // exception code 0x02

       (m_listenOnlyMode)?
         PushEventLog (REMOTE_DEVICE_MODBUS_SEND_EVENT |
READ_EXCEPTION_SENT |
          CURRENTLY_IN_LISTEN_MODE_ONLY):
         PushEventLog (REMOTE_DEVICE_MODBUS_SEND_EVENT |
READ_EXCEPTION_SENT);

       return m_rspPdu;
     }

    // read file record
```

```
    try
     {
      m_rspPdu[pduRequestOutputByteIndex++] = (recordLength << 1) + 1; // Sub-Req. x, File
Resp. length
      m_rspPdu[pduRequestOutputByteIndex++] = 6; // Sub-Req. x, Reference Type

      for (uint16_t iRecCount = 0; iRecCount < recordLength; ++iRecCount)
       {
        m_rspPdu[pduRequestOutputByteIndex++]      =      (m_fileRecord[fileNumber-
1][recordNumber+iRecCount] & 0xff00) >> 8;
        m_rspPdu[pduRequestOutputByteIndex++]      =      (m_fileRecord[fileNumber-
1][recordNumber+iRecCount] & 0xff);
       }
     }
    catch (...)
     {
      // exception
      m_rspPdu[0] = READ_FILE_RECORD + 0x80;
      m_rspPdu[1] = 0x04; // exception code 0x04

      (m_listenOnlyMode)?
       PushEventLog          (REMOTE_DEVICE_MODBUS_SEND_EVENT          |
SLAVE_ABORT_EXCEPTION_SENT |
        CURRENTLY_IN_LISTEN_MODE_ONLY):
       PushEventLog          (REMOTE_DEVICE_MODBUS_SEND_EVENT          |
SLAVE_ABORT_EXCEPTION_SENT);

      return m_rspPdu;
     }
  }

 m_rspPdu[0] = READ_FILE_RECORD;
 m_rspPdu[1] = pduRequestOutputByteIndex - 2; // data length

 ++m_commEventCounter; // increment for each successful message completion

 (m_listenOnlyMode)?
  PushEventLog          (REMOTE_DEVICE_MODBUS_SEND_EVENT          |
CURRENTLY_IN_LISTEN_MODE_ONLY):
  PushEventLog (REMOTE_DEVICE_MODBUS_SEND_EVENT);

 return m_rspPdu;
}
```

```
uint8_t*
ModbusMgr::WriteFileRecord (uint8_t requestDataLength, uint8_t * pdu)
{
  std::cout << "requestDataLength: " << (requestDataLength + 0) << std::endl;
  if ((0x0007 > requestDataLength) || (0x00F5 < requestDataLength))
    {
      m_rspPdu[0] = WRITE_FILE_RECORD + 0x80;
      m_rspPdu[1] = 0x03; // exception code 0x03

      (m_listenOnlyMode)?
        PushEventLog                    (REMOTE_DEVICE_MODBUS_SEND_EVENT                    |
READ_EXCEPTION_SENT |
          CURRENTLY_IN_LISTEN_MODE_ONLY):
        PushEventLog                    (REMOTE_DEVICE_MODBUS_SEND_EVENT                    |
READ_EXCEPTION_SENT);

      return m_rspPdu;
    }

  uint32_t pduRequestInputByteIndex = 2;
  uint32_t subRequestNumber = 1;
  while (pduRequestInputByteIndex <= (requestDataLength + 1))
    {
      uint8_t referenceType = pdu[pduRequestInputByteIndex++];
      uint16_t        fileNumber        =        (pdu[pduRequestInputByteIndex]        <<        8)        +
pdu[pduRequestInputByteIndex + 1]; // one based
      pduRequestInputByteIndex += 2;
      uint16_t        recordNumber        =        (pdu[pduRequestInputByteIndex]        <<        8)        +
pdu[pduRequestInputByteIndex + 1]; // zero based
      pduRequestInputByteIndex += 2;
      uint16_t        recordLength        =        (pdu[pduRequestInputByteIndex]        <<        8)        +
pdu[pduRequestInputByteIndex + 1];
      pduRequestInputByteIndex += 2;

      std::cout << "Sub-Req: " << (subRequestNumber + 0) << std::endl;
      std::cout << "referenceType: " << (referenceType + 0) << std::endl;
      std::cout << "fileNumber: " << (fileNumber + 0) << std::endl; // one based
      std::cout << "recordNumber: " << (recordNumber + 0) << std::endl; // zero based
      std::cout << "recordLength: " << (recordLength + 0) << std::endl;
      ++subRequestNumber;

      if ((0x06 != referenceType) ||
        (0x0001 > fileNumber || MAX_FILE_NUMBER < fileNumber) ||
        (0x0000 > recordNumber || 0x270F < recordNumber) ||
```

```
      (0x270f < recordNumber + recordLength))
     {
       m_rspPdu[0] = WRITE_FILE_RECORD + 0x80;
       m_rspPdu[1] = 0x02; // exception code 0x02

       (m_listenOnlyMode)?
         PushEventLog                (REMOTE_DEVICE_MODBUS_SEND_EVENT                |
READ_EXCEPTION_SENT |
           CURRENTLY_IN_LISTEN_MODE_ONLY):
         PushEventLog                (REMOTE_DEVICE_MODBUS_SEND_EVENT                |
READ_EXCEPTION_SENT);

       return m_rspPdu;
     }

    // write file record
    try
     {
       for (uint16_t iRecCount = 0; iRecCount < recordLength; ++iRecCount)
         {
           uint8_t regDataHigh = pdu[pduRequestInputByteIndex++];
           uint8_t regDataLow = pdu[pduRequestInputByteIndex++];
           m_fileRecord[fileNumber-1][recordNumber+iRecCount] = ((regDataHigh << 8) & 0xff00)
+ regDataLow;
         }
     }
    catch (...)
     {
       // exception
       m_rspPdu[0] = WRITE_FILE_RECORD + 0x80;
       m_rspPdu[1] = 0x04; // exception code 0x04

       (m_listenOnlyMode)?
         PushEventLog                (REMOTE_DEVICE_MODBUS_SEND_EVENT                |
SLAVE_ABORT_EXCEPTION_SENT |
           CURRENTLY_IN_LISTEN_MODE_ONLY):
         PushEventLog                (REMOTE_DEVICE_MODBUS_SEND_EVENT                |
SLAVE_ABORT_EXCEPTION_SENT);

       return m_rspPdu;
     }
  }

 // return sucessful response, which is an echo of the request
```

```
  for (uint16_t i = 0; i < (requestDataLength + 2); ++i)
    {
      m_rspPdu[i] = pdu[i];
    }

  ++m_commEventCounter; // increment for each successful message completion

  (m_listenOnlyMode)?
    PushEventLog                    (REMOTE_DEVICE_MODBUS_SEND_EVENT                    |
CURRENTLY_IN_LISTEN_MODE_ONLY):
    PushEventLog (REMOTE_DEVICE_MODBUS_SEND_EVENT);

  return m_rspPdu;
}

uint8_t*
ModbusMgr::MaskWriteRegister (uint16_t referenceAdr, uint16_t and_mask, uint16_t or_mask)
{
  std::cout << "referenceAdr: " << referenceAdr << std::endl;
  std::cout << "and_mask: " << and_mask << std::endl;
  std::cout << "or_mask: " << or_mask << std::endl;
  if ((0x0000 > referenceAdr) || (0xFFFF < referenceAdr)) // added for completeness
    {
      m_rspPdu[0] = MASK_WRITE_REGISTER + 0x80;
      m_rspPdu[1] = 0x02; // exception code 0x02

      (m_listenOnlyMode)?
        PushEventLog                    (REMOTE_DEVICE_MODBUS_SEND_EVENT                    |
READ_EXCEPTION_SENT |
          CURRENTLY_IN_LISTEN_MODE_ONLY):
        PushEventLog                    (REMOTE_DEVICE_MODBUS_SEND_EVENT                    |
READ_EXCEPTION_SENT);

      return m_rspPdu;
    }

  if ((0x0000 > and_mask) || (0xFFFF < and_mask)) // added for completeness
    {
      m_rspPdu[0] = MASK_WRITE_REGISTER + 0x80;
      m_rspPdu[1] = 0x03; // exception code 0x03

      (m_listenOnlyMode)?
        PushEventLog                    (REMOTE_DEVICE_MODBUS_SEND_EVENT                    |
READ_EXCEPTION_SENT |
```

```
        CURRENTLY_IN_LISTEN_MODE_ONLY):
      PushEventLog                (REMOTE_DEVICE_MODBUS_SEND_EVENT                |
READ_EXCEPTION_SENT);

    return m_rspPdu;
   }

  if ((0x0000 > or_mask) || (0xFFFF < or_mask)) // added for completeness
   {
    m_rspPdu[0] = MASK_WRITE_REGISTER + 0x80;
    m_rspPdu[1] = 0x03; // exception code 0x03

    (m_listenOnlyMode)?
      PushEventLog                (REMOTE_DEVICE_MODBUS_SEND_EVENT                |
READ_EXCEPTION_SENT |
        CURRENTLY_IN_LISTEN_MODE_ONLY):
      PushEventLog                (REMOTE_DEVICE_MODBUS_SEND_EVENT                |
READ_EXCEPTION_SENT);

    return m_rspPdu;
   }

  // mask write register
  try
   {
    uint16_t result = (m_holdingRegisters[referenceAdr] and and_mask) or (or_mask and (not
and_mask));
    m_holdingRegisters[referenceAdr] = result;
   }
  catch (...)
   {
    // exception
    m_rspPdu[0] = MASK_WRITE_REGISTER + 0x80;
    m_rspPdu[1] = 0x04; // exception code 0x04

    (m_listenOnlyMode)?
      PushEventLog                (REMOTE_DEVICE_MODBUS_SEND_EVENT                |
SLAVE_ABORT_EXCEPTION_SENT |
        CURRENTLY_IN_LISTEN_MODE_ONLY):
      PushEventLog                (REMOTE_DEVICE_MODBUS_SEND_EVENT                |
SLAVE_ABORT_EXCEPTION_SENT);

    return m_rspPdu;
   }
```

```
  // successful response is an echo of the request
  m_rspPdu[0] = MASK_WRITE_REGISTER;
  m_rspPdu[1] = (referenceAdr & 0xff00) >> 8;
  m_rspPdu[2] = (referenceAdr & 0xff);
  m_rspPdu[3] = (and_mask & 0xff00) >> 8;
  m_rspPdu[4] = (and_mask & 0xff);
  m_rspPdu[5] = (or_mask & 0xff00) >> 8;
  m_rspPdu[6] = (or_mask & 0xff);

  ++m_commEventCounter; // increment for each successful message completion

  (m_listenOnlyMode)?
    PushEventLog                    (REMOTE_DEVICE_MODBUS_SEND_EVENT                    |
CURRENTLY_IN_LISTEN_MODE_ONLY):
    PushEventLog (REMOTE_DEVICE_MODBUS_SEND_EVENT);

  return m_rspPdu;
}

uint8_t*
ModbusMgr::ReadWriteMultipleRegisters (uint16_t readStartAdr, uint16_t quantityToRead,
  uint16_t writeStartAdr, uint16_t quantityToWrite, uint8_t writeByteCount, uint8_t * pdu)
{
  std::cout << "readStartAdr: " << readStartAdr << std::endl;
  std::cout << "quantityToRead: " << quantityToRead << std::endl;
  std::cout << "writeStartAdr: " << writeStartAdr << std::endl;
  std::cout << "quantityToWrite: " << quantityToWrite << std::endl;
  std::cout << "writeByteCount: " << (writeByteCount + 0) << std::endl;
  if ((0x0001 > quantityToRead) || (0x007D < quantityToRead))
    {
      m_rspPdu[0] = READ_WRITE_MULTIPLE_REGISTERS + 0x80;
      m_rspPdu[1] = 0x03; // exception code 0x03

      (m_listenOnlyMode)?
        PushEventLog                    (REMOTE_DEVICE_MODBUS_SEND_EVENT                    |
READ_EXCEPTION_SENT |
          CURRENTLY_IN_LISTEN_MODE_ONLY):
        PushEventLog                    (REMOTE_DEVICE_MODBUS_SEND_EVENT                    |
READ_EXCEPTION_SENT);

      return m_rspPdu;
    }
```

```
 if ((0x0001 > quantityToWrite) || (0x007D < quantityToWrite))
  {
    m_rspPdu[0] = READ_WRITE_MULTIPLE_REGISTERS + 0x80;
    m_rspPdu[1] = 0x03; // exception code 0x03

    (m_listenOnlyMode)?
      PushEventLog                  (REMOTE_DEVICE_MODBUS_SEND_EVENT                  |
READ_EXCEPTION_SENT |
        CURRENTLY_IN_LISTEN_MODE_ONLY):
      PushEventLog                  (REMOTE_DEVICE_MODBUS_SEND_EVENT                  |
READ_EXCEPTION_SENT);

    return m_rspPdu;
  }

 if (writeByteCount != (2 * quantityToWrite))
  {
    m_rspPdu[0] = READ_WRITE_MULTIPLE_REGISTERS + 0x80;
    m_rspPdu[1] = 0x03; // exception code 0x03

    (m_listenOnlyMode)?
      PushEventLog                  (REMOTE_DEVICE_MODBUS_SEND_EVENT                  |
READ_EXCEPTION_SENT |
        CURRENTLY_IN_LISTEN_MODE_ONLY):
      PushEventLog                  (REMOTE_DEVICE_MODBUS_SEND_EVENT                  |
READ_EXCEPTION_SENT);

    return m_rspPdu;
  }

 if ((0x0000 > readStartAdr) || (0xFFFF < readStartAdr)) // added for completeness
  {
    m_rspPdu[0] = READ_WRITE_MULTIPLE_REGISTERS + 0x80;
    m_rspPdu[1] = 0x02; // exception code 0x02

    (m_listenOnlyMode)?
      PushEventLog                  (REMOTE_DEVICE_MODBUS_SEND_EVENT                  |
READ_EXCEPTION_SENT |
        CURRENTLY_IN_LISTEN_MODE_ONLY):
      PushEventLog                  (REMOTE_DEVICE_MODBUS_SEND_EVENT                  |
READ_EXCEPTION_SENT);

    return m_rspPdu;
  }
```

```
 if (0xFFFF < readStartAdr + quantityToRead)
  {
    m_rspPdu[0] = READ_WRITE_MULTIPLE_REGISTERS + 0x80;
    m_rspPdu[1] = 0x02; // exception code 0x02

    (m_listenOnlyMode)?
      PushEventLog                (REMOTE_DEVICE_MODBUS_SEND_EVENT                |
READ_EXCEPTION_SENT |
        CURRENTLY_IN_LISTEN_MODE_ONLY):
      PushEventLog                (REMOTE_DEVICE_MODBUS_SEND_EVENT                |
READ_EXCEPTION_SENT);

    return m_rspPdu;
  }


 if ((0x0000 > writeStartAdr) || (0xFFFF < writeStartAdr)) // added for completeness
  {
    m_rspPdu[0] = READ_WRITE_MULTIPLE_REGISTERS + 0x80;
    m_rspPdu[1] = 0x02; // exception code 0x02

    (m_listenOnlyMode)?
      PushEventLog                (REMOTE_DEVICE_MODBUS_SEND_EVENT                |
READ_EXCEPTION_SENT |
        CURRENTLY_IN_LISTEN_MODE_ONLY):
      PushEventLog                (REMOTE_DEVICE_MODBUS_SEND_EVENT                |
READ_EXCEPTION_SENT);

    return m_rspPdu;
  }

 if (0xFFFF < writeStartAdr + quantityToWrite)
  {
    m_rspPdu[0] = READ_WRITE_MULTIPLE_REGISTERS + 0x80;
    m_rspPdu[1] = 0x02; // exception code 0x02

    (m_listenOnlyMode)?
      PushEventLog                (REMOTE_DEVICE_MODBUS_SEND_EVENT                |
READ_EXCEPTION_SENT |
        CURRENTLY_IN_LISTEN_MODE_ONLY):
      PushEventLog                (REMOTE_DEVICE_MODBUS_SEND_EVENT                |
READ_EXCEPTION_SENT);
```

```
    return m_rspPdu;
  }

 // read/write multiple registers
 try
  {
    // write multiple registers
    uint16_t pduRequestInputByteIndex = 10;
    for (uint16_t i = 0, registerIndex = writeStartAdr; i < quantityToWrite; ++i, ++registerIndex)
     {
       m_holdingRegisters[registerIndex]    =    (pdu[pduRequestInputByteIndex]    <<    8)    +
pdu[pduRequestInputByteIndex + 1];
       pduRequestInputByteIndex += 2;
     }

    // read multiple registers
    for (uint16_t i = readStartAdr, rspIndex = 2; i < (readStartAdr + quantityToRead); ++i,
rspIndex+=2)
     {
       m_rspPdu[rspIndex] = (m_holdingRegisters[i] & 0xff00) >> 8;
       m_rspPdu[rspIndex+1] = (m_holdingRegisters[i] & 0xff);
     }

  }
 catch (...)
  {
    // exception
    m_rspPdu[0] = READ_WRITE_MULTIPLE_REGISTERS + 0x80;
    m_rspPdu[1] = 0x04; // exception code 0x04

    (m_listenOnlyMode)?
      PushEventLog                  (REMOTE_DEVICE_MODBUS_SEND_EVENT                  |
SLAVE_ABORT_EXCEPTION_SENT |
       CURRENTLY_IN_LISTEN_MODE_ONLY):
      PushEventLog                  (REMOTE_DEVICE_MODBUS_SEND_EVENT                  |
SLAVE_ABORT_EXCEPTION_SENT);

    return m_rspPdu;
  }

 // successful response is an echo of the request
 m_rspPdu[0] = READ_WRITE_MULTIPLE_REGISTERS;
 m_rspPdu[1] = quantityToRead * 2;
```

```
  ++m_commEventCounter; // increment for each successful message completion

  (m_listenOnlyMode)?
    PushEventLog                    (REMOTE_DEVICE_MODBUS_SEND_EVENT                    |
CURRENTLY_IN_LISTEN_MODE_ONLY):
    PushEventLog (REMOTE_DEVICE_MODBUS_SEND_EVENT);

  return m_rspPdu;
}

uint8_t*
ModbusMgr::ReadFifoQueue (uint16_t fifoPointerAdr)
{
  std::cout << "fifoPointerAdr: " << fifoPointerAdr << std::endl;
  if ((0x0000 > fifoPointerAdr) || (0xFFFF < fifoPointerAdr)) // added for completeness
    {
      m_rspPdu[0] = READ_FIFO_QUEUE + 0x80;
      m_rspPdu[1] = 0x02; // exception code 0x02

      (m_listenOnlyMode)?
        PushEventLog                    (REMOTE_DEVICE_MODBUS_SEND_EVENT                    |
READ_EXCEPTION_SENT |
          CURRENTLY_IN_LISTEN_MODE_ONLY):
        PushEventLog                    (REMOTE_DEVICE_MODBUS_SEND_EVENT                    |
READ_EXCEPTION_SENT);

      return m_rspPdu;
    }

  if (MAX_FIFO_COUNT < m_fifoCount)
    {
      m_rspPdu[0] = READ_FIFO_QUEUE + 0x80;
      m_rspPdu[1] = 0x03; // exception code 0x03

      (m_listenOnlyMode)?
        PushEventLog                    (REMOTE_DEVICE_MODBUS_SEND_EVENT                    |
READ_EXCEPTION_SENT |
          CURRENTLY_IN_LISTEN_MODE_ONLY):
        PushEventLog                    (REMOTE_DEVICE_MODBUS_SEND_EVENT                    |
READ_EXCEPTION_SENT);

      return m_rspPdu;
    }
```

```
  // read fifo queue
  uint32_t pduRequestOutputByteIndex = 3;
  try
   {
     m_rspPdu[pduRequestOutputByteIndex++] = (m_fifoCount & 0xff00) >> 8;
     m_rspPdu[pduRequestOutputByteIndex++] = (m_fifoCount & 0xff);

     for (uint16_t i = fifoPointerAdr; i < (fifoPointerAdr + m_fifoCount) && i <
MAX_FIFO_QUEUE_SIZE; ++i)
       {
         m_rspPdu[pduRequestOutputByteIndex++] = (m_fifo[i] & 0xff00) >> 8;
         m_rspPdu[pduRequestOutputByteIndex++] = (m_fifo[i] & 0xff);
       }
   }
  catch (...)
   {
     // exception
     m_rspPdu[0] = READ_FIFO_QUEUE + 0x80;
     m_rspPdu[1] = 0x04; // exception code 0x04

     (m_listenOnlyMode)?
       PushEventLog                (REMOTE_DEVICE_MODBUS_SEND_EVENT                |
SLAVE_ABORT_EXCEPTION_SENT |
         CURRENTLY_IN_LISTEN_MODE_ONLY):
       PushEventLog                (REMOTE_DEVICE_MODBUS_SEND_EVENT                |
SLAVE_ABORT_EXCEPTION_SENT);

     return m_rspPdu;
   }

  m_rspPdu[0] = READ_FIFO_QUEUE;
  m_rspPdu[1] = ((pduRequestOutputByteIndex - 3) & 0xff00) >> 8; // byte count high
  m_rspPdu[2] = ((pduRequestOutputByteIndex - 3) & 0xff); // byte count low

  ++m_commEventCounter; // increment for each successful message completion

  (m_listenOnlyMode)?
    PushEventLog                (REMOTE_DEVICE_MODBUS_SEND_EVENT                |
CURRENTLY_IN_LISTEN_MODE_ONLY):
    PushEventLog (REMOTE_DEVICE_MODBUS_SEND_EVENT);

  return m_rspPdu;
}
```

```
uint8_t*
ModbusMgr::encapsulatedInterfaceTransport (uint8_t meiType, uint8_t * pdu)
{
  std::cout << "meiType: 0x" << std::hex << (meiType + 0) << std::dec << std::endl;

  try
    {
      switch (meiType)
        {
          case CANOPEN_GENERAL_REFERENCE_REQUEST_AND_RESPONSE_PDU:
           {
             std::cout << "CANopen General Reference Request and Response PDU " << std::endl;

             // return sucessful response, which is an echo of the request
             for (uint16_t i = 0; i < MAX_MODBUS_PDU; ++i)
              {
                m_rspPdu[i] = pdu[i];
              }

             ++m_commEventCounter; // increment for each successful message completion

             (m_listenOnlyMode)?
               PushEventLog                (REMOTE_DEVICE_MODBUS_SEND_EVENT                |
CURRENTLY_IN_LISTEN_MODE_ONLY):
               PushEventLog (REMOTE_DEVICE_MODBUS_SEND_EVENT);

             return m_rspPdu;
           }

          case READ_DEVICE_IDENTIFICATION:
           {
             std::cout << "Read Device Identification " << std::endl;
             uint8_t readDeviceIdCode;
             uint8_t objectId;
             readDeviceIdCode = pdu[2];
             objectId = pdu[3];
             std::cout << "readDeviceIdCode " << (readDeviceIdCode + 0) << std::endl;
             std::cout << "objectId " << (objectId + 0) << std::endl;

             switch (readDeviceIdCode)
              {
                case READ_BASIC_DEVICE_ID:
                 {
```

```
            uint32_t iIndex = 0;
            m_rspPdu[iIndex++] = ENCAPSULATED_INTERFACE_TRANSPORT;
            m_rspPdu[iIndex++] = READ_DEVICE_IDENTIFICATION;
            m_rspPdu[iIndex++] = READ_BASIC_DEVICE_ID;
            m_rspPdu[iIndex++]                                                    =
CL_BASIC_ID_STREAM_ACCESS_AND_INDIVIDUAL_ACCESS;
            m_rspPdu[iIndex++] = 0x00; // more follows. 0x00 no more objects follows, 0xFF
follows
            m_rspPdu[iIndex++] = 0x00; // Next Object ID
            m_rspPdu[iIndex++] = 0x03; // Number of objects
            m_rspPdu[iIndex++] = VENDOR_NAME_ID;
            m_rspPdu[iIndex++] = VENDOR_NAME.length(); // Object length
            for (uint32_t i = 0; i < VENDOR_NAME.length(); ++i)
             {
               m_rspPdu[iIndex++] = VENDOR_NAME.at(i);
             }

            m_rspPdu[iIndex++] = PRODUCT_CODE_ID;
            m_rspPdu[iIndex++] = PRODUCT_CODE.length(); // Object length
            for (uint32_t i = 0; i < PRODUCT_CODE.length(); ++i)
             {
               m_rspPdu[iIndex++] = PRODUCT_CODE.at(i);
             }

            m_rspPdu[iIndex++] = MAJOR_MINOR_REVISION_ID;
            m_rspPdu[iIndex++] = MAJOR_MINOR_REVISION.length(); // Object length
            for (uint32_t i = 0; i < MAJOR_MINOR_REVISION.length(); ++i)
             {
               m_rspPdu[iIndex++] = MAJOR_MINOR_REVISION.at(i);
             }

            ++m_commEventCounter; // increment for each successful message completion

            (m_listenOnlyMode)?
              PushEventLog             (REMOTE_DEVICE_MODBUS_SEND_EVENT             |
CURRENTLY_IN_LISTEN_MODE_ONLY):
              PushEventLog (REMOTE_DEVICE_MODBUS_SEND_EVENT);

            return m_rspPdu;
           }

          case READ_REGULAR_DEVICE_ID:
           {
            m_rspPdu[0] = ENCAPSULATED_INTERFACE_TRANSPORT;
```

```
            m_rspPdu[1] = READ_DEVICE_IDENTIFICATION;
            m_rspPdu[2] = READ_REGULAR_DEVICE_ID;
            m_rspPdu[3] = CL_BASIC_ID_STREAM_ACCESS_AND_INDIVIDUAL_ACCESS;
            m_rspPdu[4] = 0x00; // more follows. 0x00 no more objects follows, 0xFF follows
            m_rspPdu[5] = 0x00; // Next Object ID
            m_rspPdu[6] = 0x00; // Number of objects

            ++m_commEventCounter; // increment for each successful message completion

            (m_listenOnlyMode)?
              PushEventLog          (REMOTE_DEVICE_MODBUS_SEND_EVENT          |
CURRENTLY_IN_LISTEN_MODE_ONLY):
              PushEventLog (REMOTE_DEVICE_MODBUS_SEND_EVENT);

            return m_rspPdu;
           }

          case READ_EXTENDED_DEVICE_ID:
           {
            m_rspPdu[0] = ENCAPSULATED_INTERFACE_TRANSPORT;
            m_rspPdu[1] = READ_DEVICE_IDENTIFICATION;
            m_rspPdu[2] = READ_EXTENDED_DEVICE_ID;
            m_rspPdu[3] = CL_BASIC_ID_STREAM_ACCESS_AND_INDIVIDUAL_ACCESS;
            m_rspPdu[4] = 0x00; // more follows. 0x00 no more objects follows, 0xFF follows
            m_rspPdu[5] = 0x00; // Next Object ID
            m_rspPdu[6] = 0x00; // Number of objects

            ++m_commEventCounter; // increment for each successful message completion

            (m_listenOnlyMode)?
              PushEventLog          (REMOTE_DEVICE_MODBUS_SEND_EVENT          |
CURRENTLY_IN_LISTEN_MODE_ONLY):
              PushEventLog (REMOTE_DEVICE_MODBUS_SEND_EVENT);

            return m_rspPdu;
           }

          case READ_SPECIFIC_DEVICE_ID: // individual access
           {
            // just object id of VENDOR_NAME_ID, PRODUCT_CODE_ID, and
            // MAJOR_MINOR_REVISION_ID are supported
            if (VENDOR_NAME_ID == objectId)
             {
```

```
                m_rspPdu[0] = ENCAPSULATED_INTERFACE_TRANSPORT;
                m_rspPdu[1] = READ_DEVICE_IDENTIFICATION;
                m_rspPdu[2] = READ_SPECIFIC_DEVICE_ID;
                m_rspPdu[3]                                                    =
CL_BASIC_ID_STREAM_ACCESS_AND_INDIVIDUAL_ACCESS;
                m_rspPdu[4] = 0x00; // more follows. 0x00 no more objects follows, 0xFF follows
                m_rspPdu[5] = 0x00; // Next Object ID
                m_rspPdu[6] = 0x01; // Number of objects
                m_rspPdu[7] = VENDOR_NAME_ID;
                m_rspPdu[8] = VENDOR_NAME.length(); // Object length
                for (uint32_t i = 0, iIndex = 9; i < VENDOR_NAME.length(); ++i, ++iIndex)
                 {
                   m_rspPdu[iIndex] = VENDOR_NAME.at(i);
                 }

                ++m_commEventCounter; // increment for each successful message completion

                (m_listenOnlyMode)?
                  PushEventLog          (REMOTE_DEVICE_MODBUS_SEND_EVENT          |
CURRENTLY_IN_LISTEN_MODE_ONLY):
                  PushEventLog (REMOTE_DEVICE_MODBUS_SEND_EVENT);

                return m_rspPdu;
               }
              else if (PRODUCT_CODE_ID == objectId)
               {
                m_rspPdu[0] = ENCAPSULATED_INTERFACE_TRANSPORT;
                m_rspPdu[1] = READ_DEVICE_IDENTIFICATION;
                m_rspPdu[2] = READ_SPECIFIC_DEVICE_ID;
                m_rspPdu[3]                                                    =
CL_BASIC_ID_STREAM_ACCESS_AND_INDIVIDUAL_ACCESS;
                m_rspPdu[4] = 0x00; // more follows. 0x00 no more objects follows, 0xFF follows
                m_rspPdu[5] = 0x00; // Next Object ID
                m_rspPdu[6] = 0x01; // Number of objects
                m_rspPdu[7] = PRODUCT_CODE_ID;
                m_rspPdu[8] = PRODUCT_CODE.length(); // Object length
                for (uint32_t i = 0, iIndex = 9; i < PRODUCT_CODE.length(); ++i, ++iIndex)
                 {
                   m_rspPdu[iIndex] = PRODUCT_CODE.at(i);
                 }

                ++m_commEventCounter; // increment for each successful message completion

                (m_listenOnlyMode)?
```

```
                PushEventLog          (REMOTE_DEVICE_MODBUS_SEND_EVENT          |
CURRENTLY_IN_LISTEN_MODE_ONLY):
                PushEventLog (REMOTE_DEVICE_MODBUS_SEND_EVENT);

                return m_rspPdu;
               }
              else if (MAJOR_MINOR_REVISION_ID == objectId)
               {
                m_rspPdu[0] = ENCAPSULATED_INTERFACE_TRANSPORT;
                m_rspPdu[1] = READ_DEVICE_IDENTIFICATION;
                m_rspPdu[2] = READ_SPECIFIC_DEVICE_ID;
                m_rspPdu[3]                                                    =
CL_BASIC_ID_STREAM_ACCESS_AND_INDIVIDUAL_ACCESS;
                m_rspPdu[4] = 0x00; // more follows. 0x00 no more objects follows, 0xFF follows
                m_rspPdu[5] = 0x00; // Next Object ID
                m_rspPdu[6] = 0x01; // Number of objects
                m_rspPdu[7] = MAJOR_MINOR_REVISION_ID;
                m_rspPdu[8] = MAJOR_MINOR_REVISION.length(); // Object length
                for (uint32_t i = 0, iIndex = 9; i < MAJOR_MINOR_REVISION.length(); ++i,
++iIndex)
                 {
                  m_rspPdu[iIndex] = MAJOR_MINOR_REVISION.at(i);
                 }

                ++m_commEventCounter; // increment for each successful message completion

                (m_listenOnlyMode)?
                 PushEventLog          (REMOTE_DEVICE_MODBUS_SEND_EVENT          |
CURRENTLY_IN_LISTEN_MODE_ONLY):
                 PushEventLog (REMOTE_DEVICE_MODBUS_SEND_EVENT);

                return m_rspPdu;
               }
              else
               {
                m_rspPdu[0] = ENCAPSULATED_INTERFACE_TRANSPORT + 0x80;
                m_rspPdu[1] = 0x02; // exception code 0x02

                (m_listenOnlyMode)?
                 PushEventLog          (REMOTE_DEVICE_MODBUS_SEND_EVENT          |
READ_EXCEPTION_SENT |
                  CURRENTLY_IN_LISTEN_MODE_ONLY):
                 PushEventLog          (REMOTE_DEVICE_MODBUS_SEND_EVENT          |
READ_EXCEPTION_SENT);
```

```
                return m_rspPdu;
              }
            }

          default:
            {
              m_rspPdu[0] = ENCAPSULATED_INTERFACE_TRANSPORT + 0x80;
              m_rspPdu[1] = 0x03; // exception code 0x03

              (m_listenOnlyMode)?
                PushEventLog          (REMOTE_DEVICE_MODBUS_SEND_EVENT          |
READ_EXCEPTION_SENT |
                  CURRENTLY_IN_LISTEN_MODE_ONLY):
                PushEventLog          (REMOTE_DEVICE_MODBUS_SEND_EVENT          |
READ_EXCEPTION_SENT);

              return m_rspPdu;
            }
          }
        }

      default:
        {
          std::cout << "error: undefined MEI type " << std::endl;
          m_rspPdu[0] = ENCAPSULATED_INTERFACE_TRANSPORT + 0x80;
          m_rspPdu[1] = 0x01; // exception code 0x01, sub-function code not supported

          (m_listenOnlyMode)?
            PushEventLog          (REMOTE_DEVICE_MODBUS_SEND_EVENT          |
READ_EXCEPTION_SENT |
              CURRENTLY_IN_LISTEN_MODE_ONLY):
            PushEventLog          (REMOTE_DEVICE_MODBUS_SEND_EVENT          |
READ_EXCEPTION_SENT);

          return m_rspPdu;
        }
      }
    }
  catch (...)
    {
      // exception
      m_rspPdu[0] = ENCAPSULATED_INTERFACE_TRANSPORT + 0x80;
      m_rspPdu[1] = 0x04; // exception code 0x04
```

```
    (m_listenOnlyMode)?
      PushEventLog                (REMOTE_DEVICE_MODBUS_SEND_EVENT                |
SLAVE_ABORT_EXCEPTION_SENT |
       CURRENTLY_IN_LISTEN_MODE_ONLY):
      PushEventLog                (REMOTE_DEVICE_MODBUS_SEND_EVENT                |
SLAVE_ABORT_EXCEPTION_SENT);

    return m_rspPdu;
  }
}

} // namespace ns3
```

modbus-udp-client.h

This header file is located at *~/ns-3-allinone/ns-3-dev/src/applications/model*.

```
/* -*- Mode:C++; c-file-style:"gnu"; indent-tabs-mode:nil; -*- */
/*
 * Copyright (c) 2012 Simon Fraser University
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License version 2 as
 * published by the Free Software Foundation;
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
 *
 * Author: Reza Sahraei <mrs16@sfu.ca>
 */

#ifndef MODBUS_UDP_CLIENT_H
#define MODBUS_UDP_CLIENT_H

#include "ns3/application.h"
#include "ns3/event-id.h"
#include "ns3/ptr.h"
```

```
#include "ns3/ipv4-address.h"
#include "ns3/traced-callback.h"
#include "sys/time.h"

const uint32_t MAX_NUM_MODBUS_REQUEST = 10; // maximum number of MODBUS request
to store

namespace ns3 {

class Socket;
class Packet;

/**
 * \ingroup modbusudpclientserver
 * \class ModbusUdpClient
 * \brief A MODBUS UDP client. Sends MODBUS UDP requests.
 *
 */
class ModbusUdpClient : public Application
{
public:
  static TypeId GetTypeId (void);

  ModbusUdpClient ();

  virtual ~ModbusUdpClient ();

  /**
   * \brief set the remote address and port
   * \param ip remote IP address
   * \param port remote port
   */
  void SetRemote (Ipv4Address ip, uint16_t port);

  /**
   * Set a MODBUS UDP request.
   */
  void SetRequest (Time requestInterval, uint8_t *pdu);

  /**
   * Set number of iteration for sending the request(s).
   */
  void SetCount (uint32_t count);
```

```
protected:
  virtual void DoDispose (void);

private:

  virtual void StartApplication (void);
  virtual void StopApplication (void);

  void ScheduleTransmit (Time dt);
  void Send (void);

  void HandleRead (Ptr<Socket> socket);

  uint32_t m_count;
  Time m_interval;
  uint32_t m_size;

  uint32_t m_dataSize;
  uint8_t *m_data;

  uint8_t **m_modbusRequests;
  Time *m_modbusRequestsInterval;
  uint32_t m_modbusRequestsIndex;
  uint32_t m_modbusSendRequestsIndex;

  uint32_t m_sent;
  Ptr<Socket> m_socket;
  Ipv4Address m_peerAddress;
  uint16_t m_peerPort;
  EventId m_sendEvent;

  /// Callbacks for tracing the packet Tx events
  TracedCallback<Ptr<const Packet> > m_txTrace;

  struct timeval m_sent_time, m_received_time;
};

} // namespace ns3

#endif /* MODBUS_UDP_CLIENT_H */
```

modbus-udp-client.cc

This C++ file is located at *~/ns-3-allinone/ns-3-dev/src/applications/model*.

```
/* -*- Mode:C++; c-file-style:"gnu"; indent-tabs-mode:nil; -*- */
/*
 * Copyright (c) 2012 Simon Fraser University
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License version 2 as
 * published by the Free Software Foundation;
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
 *
 * Author: Reza Sahraei <mrs16@sfu.ca>
 */

#include "ns3/log.h"
#include "ns3/ipv4-address.h"
#include "ns3/nstime.h"
#include "ns3/inet-socket-address.h"
#include "ns3/socket.h"
#include "ns3/simulator.h"
#include "ns3/socket-factory.h"
#include "ns3/packet.h"
#include "ns3/uinteger.h"
#include "ns3/trace-source-accessor.h"
#include "modbus-udp-client.h"
#include "seq-ts-header.h"
#include "modbus-pdu.h"
#include "modbus-mgr.h"
#include <stdlib.h>
#include <stdio.h>

namespace ns3 {

NS_LOG_COMPONENT_DEFINE ("ModbusUdpClient");
```

```
NS_OBJECT_ENSURE_REGISTERED (ModbusUdpClient);

TypeId
ModbusUdpClient::GetTypeId (void)
{
  static TypeId tid = TypeId ("ns3::ModbusUdpClient")
    .SetParent<Application> ()
    .AddConstructor<ModbusUdpClient> ()
    .AddAttribute ("MaxPackets",
                   "The maximum number of packets the application will send",
                   UintegerValue (100),
                   MakeUintegerAccessor (&ModbusUdpClient::m_count),
                   MakeUintegerChecker<uint32_t> ())
    .AddAttribute ("Interval",
                   "The time to wait between packets",
                   TimeValue (Seconds (1.0)),
                   MakeTimeAccessor (&ModbusUdpClient::m_interval),
                   MakeTimeChecker ())
    .AddAttribute ("RemoteAddress",
                   "The destination Ipv4Address of the outbound packets",
                   Ipv4AddressValue (),
                   MakeIpv4AddressAccessor (&ModbusUdpClient::m_peerAddress),
                   MakeIpv4AddressChecker ())
    .AddAttribute ("RemotePort", "The destination port of the outbound packets",
                   UintegerValue (502),
                   MakeUintegerAccessor (&ModbusUdpClient::m_peerPort),
                   MakeUintegerChecker<uint16_t> ())
    .AddAttribute ("PacketSize",
                   "Size of packets generated. The minimum packet size is 12 bytes which is the size of
the header carrying the sequence number and the time stamp.",
                   UintegerValue (253),
                   MakeUintegerAccessor (&ModbusUdpClient::m_size),
                   MakeUintegerChecker<uint32_t> (12,1500))
    .AddTraceSource ("Tx", "A new packet is created and is sent",
                     MakeTraceSourceAccessor (&ModbusUdpClient::m_txTrace))
  ;
  return tid;
}

ModbusUdpClient::ModbusUdpClient ()
{
  NS_LOG_FUNCTION_NOARGS ();
  m_sent = 0;
```

```
  m_socket = 0;
  m_sendEvent = EventId ();
  m_data = new uint8_t [MAX_MODBUS_PDU];
  m_dataSize = 0;
  m_count = 1;

  m_modbusRequests = new uint8_t *[MAX_NUM_MODBUS_REQUEST]; // rows
  for (uint32_t i = 0; i < MAX_NUM_MODBUS_REQUEST; ++i)
    {
      m_modbusRequests[i] = new uint8_t [MAX_MODBUS_PDU]; // cols
    }

  m_modbusRequestsInterval = new Time [MAX_NUM_MODBUS_REQUEST];
  m_modbusRequestsIndex = 0;
  m_modbusSendRequestsIndex = 0;
}

ModbusUdpClient::~ModbusUdpClient ()
{
  NS_LOG_FUNCTION_NOARGS ();
  m_socket = 0;
  delete [] m_data;
  m_data = 0;
  m_dataSize = 0;

  for (uint32_t i = 0; i < MAX_NUM_MODBUS_REQUEST; ++i)
    {
      delete [] m_modbusRequests[i];
    }
  delete [] m_modbusRequests;
  m_modbusRequests = 0;

  delete [] m_modbusRequestsInterval;
}

void
ModbusUdpClient::SetRemote (Ipv4Address ip, uint16_t port)
{
  m_peerAddress = ip;
  m_peerPort = port;
}

void
```

```
ModbusUdpClient::SetRequest (Time requestInterval, uint8_t *pdu)
{
  NS_LOG_INFO ("ModbusUdpClient::SetRequest ()");
  memcpy (m_modbusRequests[m_modbusRequestsIndex], pdu, MAX_MODBUS_PDU);
  m_modbusRequestsInterval[m_modbusRequestsIndex++] = requestInterval;
}

void
ModbusUdpClient::SetCount (uint32_t count)
{
  NS_LOG_INFO ("ModbusUdpClient::SetCount ()");
  m_count = count;
}

void
ModbusUdpClient::DoDispose (void)
{
  NS_LOG_FUNCTION_NOARGS ();
  Application::DoDispose ();
}

void
ModbusUdpClient::StartApplication (void)
{
  NS_LOG_FUNCTION_NOARGS ();

  if (m_socket == 0)
    {
      TypeId tid = TypeId::LookupByName ("ns3::UdpSocketFactory");
      m_socket = Socket::CreateSocket (GetNode (), tid);
      m_socket->Bind ();
      m_socket->Connect (InetSocketAddress (m_peerAddress, m_peerPort));
    }

  m_socket->SetRecvCallback (MakeCallback (&ModbusUdpClient::HandleRead, this));

  if (m_modbusSendRequestsIndex < m_modbusRequestsIndex)
    {
      m_sendEvent = Simulator::Schedule (m_modbusRequestsInterval[0],
&ModbusUdpClient::Send, this);
    }
}
```

```
void
ModbusUdpClient::StopApplication ()
{
  NS_LOG_FUNCTION_NOARGS ();

  if (m_socket != 0)
    {
      m_socket->Close ();
      m_socket->SetRecvCallback (MakeNullCallback<void, Ptr<Socket> > ());
      m_socket = 0;
    }

  Simulator::Cancel (m_sendEvent);
}

void
ModbusUdpClient::Send (void)
{
  NS_LOG_INFO ("ModbusUdpClient::Send ()");
  NS_LOG_FUNCTION_NOARGS ();
  NS_ASSERT (m_sendEvent.IsExpired ());
  SeqTsHeader seqTs;
  seqTs.SetSeq (m_sent);

  m_size = 0;
  Ptr<Packet> p = Create<Packet> (m_size);

  // call to the trace sinks before the packet is actually sent,
  // so that tags added to the packet can be sent as well
  m_txTrace (p);

  p->AddHeader (seqTs);

  ModbusPdu cModbusRequestPdu;
  uint8_t *pdu = cModbusRequestPdu.GetPdu ();
  memcpy (pdu, m_modbusRequests[m_modbusSendRequestsIndex++], MAX_MODBUS_PDU);
  p->AddHeader (cModbusRequestPdu);

  if ((m_socket->Send (p)) >= 0)
    {
      // to get current time
      gettimeofday(&m_sent_time, NULL);
```

```
    ++m_sent;
    NS_LOG_INFO ("TraceDelay TX: " << (p->GetSize()-12) << " bytes to " // 8+4 : the size of the
seqTs header
                          << m_peerAddress << " Uid: " << p->GetUid ()
                          << " Time: " << (Simulator::Now ()).GetSeconds ());

   }
  else
   {
    NS_LOG_INFO ("Error while sending " << m_size << " bytes to "
                                 << m_peerAddress);
   }

  if (m_modbusSendRequestsIndex < m_modbusRequestsIndex)
   {
    m_sendEvent                                =                                Simulator::Schedule
(m_modbusRequestsInterval[m_modbusSendRequestsIndex],
     &ModbusUdpClient::Send, this);
   }

  if ((m_modbusSendRequestsIndex >= m_modbusRequestsIndex) && (m_sent < m_count))
   {
    m_modbusSendRequestsIndex = 0;
    m_sendEvent          =          Simulator::Schedule          (m_modbusRequestsInterval[0],
&ModbusUdpClient::Send, this);
   }
}

void
ModbusUdpClient::HandleRead (Ptr<Socket> socket)
{
  NS_LOG_INFO ("ModbusUdpClient::HandleRead ()");
  NS_LOG_FUNCTION (this << socket);
  Ptr<Packet> packet;
  Address from;
  while (packet = socket->RecvFrom (from))
   {
    if (InetSocketAddress::IsMatchingType (from))
     {
       // to get current time
       gettimeofday(&m_received_time, NULL);

       // log round-trip Modbus request/response
       double t1 = m_sent_time.tv_sec + (m_sent_time.tv_usec/1000000.0);
```

```
      double t2 = m_received_time.tv_sec + (m_received_time.tv_usec/1000000.0);
      //NS_LOG_INFO ("Round-trip Modbus request/response (ms): " << (t2 - t1)*1000);
      std::cout << "Round-trip Modbus request/response (ms): " << (t2 - t1)*1000 << std::endl;

      NS_LOG_INFO ("Received " << packet->GetSize () << " bytes from " <<
                   InetSocketAddress::ConvertFrom (from).GetIpv4 ());

      // read the first byte to see it is an exception or successful
      ModbusPdu cModbusResponsePdu;
      packet->RemoveHeader (cModbusResponsePdu);
      uint8_t *pdu = cModbusResponsePdu.GetPdu ();

      if (0x80 <= pdu[0])
        {
          NS_LOG_INFO ("MODBUS response error - function code: 0x" << std::hex << (pdu[0] -
0x80)
            << " exception code: 0x" << std::hex << (pdu[1] + 0));
          switch (pdu[1])
            {
              case 0x01:
                {
                  NS_LOG_INFO ("Invalid funcion or sub-function code");
                  break;
                }
              case 0x02:
                {
                  NS_LOG_INFO ("Invalid data address");
                  break;
                }
              case 0x03:
                {
                  NS_LOG_INFO ("Invalid data value");
                  break;
                }
              case 0x04:
              case 0x05:
              case 0x06:
                {
                  NS_LOG_INFO ("MODBUS function execution failed");
                  break;
                }
              default:
                {
```

```
                NS_LOG_INFO ("Unknown exception code!");
                break;
              }
            }

          for (uint32_t i = 0; i < 5; ++i)
            {
              NS_LOG_INFO ("pdu[" << std::dec << i << "]: 0x" << std::hex << (pdu[i] + 0));
            }
        }
       else
        {
          NS_LOG_INFO ("Success - function code: 0x" << std::hex << (pdu[0] + 0));
          for (uint32_t i = 0; i < 10; ++i)
            {
              NS_LOG_INFO ("pdu[" << std::dec << i << "]: 0x" << std::hex << (pdu[i] + 0));
            }
        }
        NS_LOG_INFO (std::dec);
      }
  }
}

} // Namespace ns3
```

modbus-udp-server.h

This header file is located at *~/ns-3-allinone/ns-3-dev/src/applications/model*.

```
/* -*- Mode:C++; c-file-style:"gnu"; indent-tabs-mode:nil; -*- */
/*
 * Copyright (c) 2012 Simon Fraser University
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License version 2 as
 * published by the Free Software Foundation;
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
```

```
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
 *
 * Author: Reza Sahraei <mrs16@sfu.ca>
 */

#ifndef MODBUS_TCP_SERVER_H
#define MODBUS_TCP_SERVER_H

#include "ns3/application.h"
#include "ns3/event-id.h"
#include "ns3/ptr.h"
#include "ns3/address.h"
#include "packet-loss-counter.h"
#include "modbus-mgr.h"

namespace ns3 {
/**
 * \ingroup applications
 * \defgroup modbustcpclientserver modbustcpclientserver
 */

/**
 * \ingroup tcpclientserver
 * \class ModbusTcpServer
 * \brief Create a server application which waits for input MODBUS TCP request
 *        and generates appropriate response according to the function code.
 *        It also uses the information carried into their payload of the packet
 *        to compute delay and to determine if some packets are lost.

 * \brief A MODBUS TCP server. Receives MODBUS TCP request from a remote host
 *        and generates appropriate response according to the function code.
 *        It also uses the information carried into their payload of the packet
 *        to compute delay and to determine if some packets are lost.
 */
class ModbusTcpServer : public Application
{
public:
  static TypeId GetTypeId (void);
  ModbusTcpServer ();
  virtual ~ModbusTcpServer ();
  /**
   * returns the number of lost packets
   * \return the number of lost packets
```

```
   */
  uint32_t GetLost (void) const;

  /**
   * \brief returns the number of received packets
   * \return the number of received packets
   */
  uint32_t GetReceived (void) const;

  /**
   * \return the size of the window used for checking loss.
   */
  uint16_t GetPacketWindowSize () const;

  /**
   * \brief Set the size of the window used for checking loss. This value should
   *  be a multiple of 8
   * \param size the size of the window used for checking loss. This value should
   *  be a multiple of 8
   */
  void SetPacketWindowSize (uint16_t size);

  /**
   * \return pointer to listening socket
   */
  Ptr<Socket> GetListeningSocket (void) const;

  /**
   * \return list of pointers to accepted sockets
   */
  std::list<Ptr<Socket> > GetAcceptedSockets (void) const;

protected:
  virtual void DoDispose (void);

private:

  virtual void StartApplication (void);
  virtual void StopApplication (void);

  void HandleRead (Ptr<Socket> socket);

  void HandleAccept (Ptr<Socket>, const Address& from);
```

```
  void HandlePeerClose (Ptr<Socket>);
  void HandlePeerError (Ptr<Socket>);

  uint16_t m_port;
  Ptr<Socket> m_socket;

  ModbusMgr m_cModbusMgr;

  std::list<Ptr<Socket> > m_socketList; // the accepted sockets

  Address m_local;
  uint32_t m_received;
  PacketLossCounter m_lossCounter;
};

} // namespace ns3

#endif /* MODBUS_TCP_SERVER_H */
```

modbus-udp-server.cc

This C++ file is located at *~/ns-3-allinone/ns-3-dev/src/applications/model*.

```
/* -*- Mode:C++; c-file-style:"gnu"; indent-tabs-mode:nil; -*- */
/*
 * Copyright (c) 2012 Simon Fraser University
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License version 2 as
 * published by the Free Software Foundation;
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
 *
 * Author: Reza Sahraei <mrs16@sfu.ca>
 */

#include "ns3/log.h"
```

```
#include "ns3/ipv4-address.h"
#include "ns3/nstime.h"
#include "ns3/inet-socket-address.h"
#include "ns3/socket.h"
#include "ns3/simulator.h"
#include "ns3/socket-factory.h"
#include "ns3/packet.h"
#include "ns3/uinteger.h"
#include "packet-loss-counter.h"

#include "seq-ts-header.h"
#include "modbus-tcp-server.h"
#include "modbus-pdu.h"

namespace ns3 {

NS_LOG_COMPONENT_DEFINE ("ModbusTcpServer");
NS_OBJECT_ENSURE_REGISTERED (ModbusTcpServer);


TypeId
ModbusTcpServer::GetTypeId (void)
{
  static TypeId tid = TypeId ("ns3::ModbusTcpServer")
    .SetParent<Application> ()
    .AddConstructor<ModbusTcpServer> ()
    .AddAttribute ("Port",
                   "Port on which we listen for incoming packets.",
                   UintegerValue (502),
                   MakeUintegerAccessor (&ModbusTcpServer::m_port),
                   MakeUintegerChecker<uint16_t> ())
    .AddAttribute ("PacketWindowSize",
                   "The size of the window used to compute the packet loss. This value should be a
multiple of 8.",
                   UintegerValue (32),
                   MakeUintegerAccessor (&ModbusTcpServer::GetPacketWindowSize,
                                         &ModbusTcpServer::SetPacketWindowSize),
                   MakeUintegerChecker<uint16_t> (8,256))
  ;
  return tid;
}

ModbusTcpServer::ModbusTcpServer ()
```

```
  : m_lossCounter (0)
{
  NS_LOG_FUNCTION (this);
  m_received=0;
}

ModbusTcpServer::~ModbusTcpServer ()
{
  NS_LOG_FUNCTION (this);
}

uint16_t
ModbusTcpServer::GetPacketWindowSize () const
{
  return m_lossCounter.GetBitMapSize ();
}

void
ModbusTcpServer::SetPacketWindowSize (uint16_t size)
{
  m_lossCounter.SetBitMapSize (size);
}

uint32_t
ModbusTcpServer::GetLost (void) const
{
  return m_lossCounter.GetLost ();
}

uint32_t
ModbusTcpServer::GetReceived (void) const
{
  return m_received;
}

Ptr<Socket>
ModbusTcpServer::GetListeningSocket (void) const
{
  NS_LOG_FUNCTION (this);
  return m_socket;
}

std::list<Ptr<Socket> >
```

```
ModbusTcpServer::GetAcceptedSockets (void) const
{
  NS_LOG_FUNCTION (this);
  return m_socketList;
}

void
ModbusTcpServer::DoDispose (void)
{
  NS_LOG_FUNCTION (this);
  m_socket = 0;
  m_socketList.clear ();
  Application::DoDispose ();
}

void
ModbusTcpServer::StartApplication (void)
{
  NS_LOG_FUNCTION_NOARGS ();

  if (m_socket == 0)
    {
      TypeId tid = TypeId::LookupByName ("ns3::TcpSocketFactory");
      m_socket = Socket::CreateSocket (GetNode (), tid);
      m_socket->SetAttribute ("DelAckCount", UintegerValue (1));
      InetSocketAddress local = InetSocketAddress (Ipv4Address::GetAny (), m_port);
      m_socket->Bind (local);
      //std::cout << "ModbusTcpServer::StartApplication Ipv4Address::GetAny () " <<
Ipv4Address::GetAny () << " m_port " << m_port << std::endl;
      //std::cout << "ModbusTcpServer::StartApplication local " << local << std::endl;

      m_socket->Listen ();
    }

  m_socket->SetRecvCallback (MakeCallback (&ModbusTcpServer::HandleRead, this));

  m_socket->SetAcceptCallback (
    MakeNullCallback<bool, Ptr<Socket>, const Address &> (),
    MakeCallback (&ModbusTcpServer::HandleAccept, this));

  m_socket->SetCloseCallbacks (
    MakeCallback (&ModbusTcpServer::HandlePeerClose, this),
    MakeCallback (&ModbusTcpServer::HandlePeerError, this));
```

```
}

void
ModbusTcpServer::StopApplication ()
{
  NS_LOG_FUNCTION (this);

/*
  if (m_socket != 0)
    {
      m_socket->SetRecvCallback (MakeNullCallback<void, Ptr<Socket> > ());
    }
*/

  while (!m_socketList.empty ()) // these are accepted sockets, close them
    {
      Ptr<Socket> acceptedSocket = m_socketList.front ();
      m_socketList.pop_front ();
      acceptedSocket->Close ();
    }
  if (m_socket)
    {
      m_socket->Close ();
      m_socket->SetRecvCallback (MakeNullCallback<void, Ptr<Socket> > ());
    }
}

void
ModbusTcpServer::HandleRead (Ptr<Socket> socket)
{
  NS_LOG_FUNCTION (this << socket);
  Ptr<Packet> packet;
  Address from;
  while (packet = socket->RecvFrom (from))
    {
      if (InetSocketAddress::IsMatchingType (from))
        {
          if (packet->GetSize () > 0)
            {
              NS_LOG_INFO ("Parsing MODBUS TCP function code");

              uint32_t packetSize = packet->GetSize() - 12; // 8+4 : the size of the seqTs header
              ModbusPdu cModbusRequestPdu;
```

```
          packet->RemoveHeader (cModbusRequestPdu);
          uint8_t *pdu = cModbusRequestPdu.GetPdu ();
          uint8_t *rspPdu = m_cModbusMgr.DoMain(pdu);

          SeqTsHeader seqTs;
          packet->RemoveHeader (seqTs);
          uint32_t currentSequenceNumber = seqTs.GetSeq ();
          NS_LOG_INFO ("TraceDelay: RX " << packetSize <<
                  " bytes from "<< InetSocketAddress::ConvertFrom (from).GetIpv4 () <<
                  " Sequence Number: " << currentSequenceNumber <<
                  " Uid: " << packet->GetUid () <<
                  " TXtime: " << seqTs.GetTs () <<
                  " RXtime: " << Simulator::Now () <<
                  " Delay: " << Simulator::Now () - seqTs.GetTs ());

          m_lossCounter.NotifyReceived (currentSequenceNumber);
          m_received++;

          if (NULL != rspPdu)
           {
             // send MODBUS TCP response back
             NS_LOG_INFO ("Sending MODBUS TCP response packet");

             ModbusPdu cModbusResponsePdu;
             uint8_t *pduRspTemp = cModbusResponsePdu.GetPdu ();
             for (uint16_t i = 0; i < MAX_MODBUS_PDU; ++i)
              {
                pduRspTemp[i] = rspPdu[i];
              }

             Ptr<Packet> sendPacket = Create<Packet> (0);
             sendPacket->AddHeader (cModbusResponsePdu);
             socket->SendTo (sendPacket, 0, from);
           }
        }
      }
    }
}

void
ModbusTcpServer::HandlePeerClose (Ptr<Socket> socket)
{
  NS_LOG_INFO ("ModbusTcpServer, peerClose");
```

```
}

void
ModbusTcpServer::HandlePeerError (Ptr<Socket> socket)
{
  NS_LOG_INFO ("ModbusTcpServer, peerError");
}

void
ModbusTcpServer::HandleAccept (Ptr<Socket> s, const Address& from)
{
  NS_LOG_FUNCTION (this << s << from);
  s->SetRecvCallback (MakeCallback (&ModbusTcpServer::HandleRead, this));
  m_socketList.push_back (s);
}
} // Namespace ns3
```

modbus-tcp-client.h

This header file is located at *~/ns-3-allinone/ns-3-dev/src/applications/model*.

```
/* -*- Mode:C++; c-file-style:"gnu"; indent-tabs-mode:nil; -*- */
/*
 * Copyright (c) 2012 Simon Fraser University
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License version 2 as
 * published by the Free Software Foundation;
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
 *
 * Author: Reza Sahraei <mrs16@sfu.ca>
 */

#ifndef MODBUS_TCP_CLIENT_H
#define MODBUS_TCP_CLIENT_H
```

```
#include "ns3/application.h"
#include "ns3/event-id.h"
#include "ns3/ptr.h"
#include "ns3/ipv4-address.h"
#include "ns3/traced-callback.h"
#include "sys/time.h"

const uint32_t MAX_NUM_MODBUS_TCP_REQUEST = 10; // maximum number of MODBUS request to store

namespace ns3 {

class Socket;
class Packet;

/**
 * \ingroup modbustcpclientserver
 * \class ModbusTcpClient
 * \brief A MODBUS TCP client. Sends MODBUS TCP requests.
 *
 */
class ModbusTcpClient : public Application
{
public:
  static TypeId GetTypeId (void);

  ModbusTcpClient ();

  virtual ~ModbusTcpClient ();

  /**
   * \brief set the remote address and port
   * \param ip remote IP address
   * \param port remote port
   */
  void SetRemote (Ipv4Address ip, uint16_t port);

  /**
   * Set a MODBUS TCP request.
   */
  void SetRequest (Time requestInterval, uint8_t *pdu);

  /**
```

```
   * Set number of iteration for sending the request(s).
   */
  void SetCount (uint32_t count);

  /**
   * \return list of pointers to accepted sockets
   */
  //std::list<Ptr<Socket> > GetAcceptedSockets (void) const;

protected:
  virtual void DoDispose (void);

private:

  virtual void StartApplication (void);
  virtual void StopApplication (void);

  void ScheduleTransmit (Time dt);
  void Send (void);

  void HandleRead (Ptr<Socket> socket);

  void HandleAccept (Ptr<Socket>, const Address& from);
//  void HandlePeerClose (Ptr<Socket>);
//  void HandlePeerError (Ptr<Socket>);

  uint32_t m_count;
  Time m_interval;
  uint32_t m_size;

  uint32_t m_dataSize;
  uint8_t *m_data;

  uint8_t **m_modbusRequests;
  Time *m_modbusRequestsInterval;
  uint32_t m_modbusRequestsIndex;
  uint32_t m_modbusSendRequestsIndex;

  uint32_t m_sent;
  Ptr<Socket> m_socket;

//  std::list<Ptr<Socket> > m_socketList; // the accepted sockets
```

```
  Ipv4Address m_peerAddress;
  uint16_t m_peerPort;
  EventId m_sendEvent;

  /// Callbacks for tracing the packet Tx events
  TracedCallback<Ptr<const Packet> > m_txTrace;

  struct timeval m_sent_time, m_received_time;
};

} // namespace ns3

#endif /* MODBUS_TCP_CLIENT_H */
```

modbus-tcp-client.cc

This C++ file is located at *~/ns-3-allinone/ns-3-dev/src/applications/model*.

```
/* -*- Mode:C++; c-file-style:"gnu"; indent-tabs-mode:nil; -*- */
/*
 * Copyright (c) 2012 Simon Fraser University
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License version 2 as
 * published by the Free Software Foundation;
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
 *
 * Author: Reza Sahraei <mrs16@sfu.ca>
 */

#include "ns3/log.h"
#include "ns3/ipv4-address.h"
#include "ns3/nstime.h"
#include "ns3/inet-socket-address.h"
#include "ns3/socket.h"
#include "ns3/simulator.h"
```

```
#include "ns3/socket-factory.h"
#include "ns3/packet.h"
#include "ns3/uinteger.h"
#include "ns3/trace-source-accessor.h"
#include "modbus-tcp-client.h"
#include "seq-ts-header.h"
#include "modbus-pdu.h"
#include "modbus-mgr.h"
#include <stdlib.h>
#include <stdio.h>

namespace ns3 {

NS_LOG_COMPONENT_DEFINE ("ModbusTcpClient");
NS_OBJECT_ENSURE_REGISTERED (ModbusTcpClient);

TypeId
ModbusTcpClient::GetTypeId (void)
{
  static TypeId tid = TypeId ("ns3::ModbusTcpClient")
    .SetParent<Application> ()
    .AddConstructor<ModbusTcpClient> ()
    .AddAttribute ("MaxPackets",
                   "The maximum number of packets the application will send",
                   UintegerValue (100),
                   MakeUintegerAccessor (&ModbusTcpClient::m_count),
                   MakeUintegerChecker<uint32_t> ())
    .AddAttribute ("Interval",
                   "The time to wait between packets",
                   TimeValue (Seconds (1.0)),
                   MakeTimeAccessor (&ModbusTcpClient::m_interval),
                   MakeTimeChecker ())
    .AddAttribute ("RemoteAddress",
                   "The destination Ipv4Address of the outbound packets",
                   Ipv4AddressValue (),
                   MakeIpv4AddressAccessor (&ModbusTcpClient::m_peerAddress),
                   MakeIpv4AddressChecker ())
    .AddAttribute ("RemotePort", "The destination port of the outbound packets",
                   UintegerValue (502),
                   MakeUintegerAccessor (&ModbusTcpClient::m_peerPort),
                   MakeUintegerChecker<uint16_t> ())
    .AddAttribute ("PacketSize",
```

```
                "Size of packets generated. The minimum packet size is 12 bytes which is the size of
the header carrying the sequence number and the time stamp.",
                UintegerValue (253),
                MakeUintegerAccessor (&ModbusTcpClient::m_size),
                MakeUintegerChecker<uint32_t> (12,1500))
    .AddTraceSource ("Tx", "A new packet is created and is sent",
                     MakeTraceSourceAccessor (&ModbusTcpClient::m_txTrace))
  ;
  return tid;
}

ModbusTcpClient::ModbusTcpClient ()
{
  NS_LOG_FUNCTION_NOARGS ();
  m_sent = 0;
  m_socket = 0;
  m_sendEvent = EventId ();
  m_data = new uint8_t [MAX_MODBUS_PDU];
  m_dataSize = 0;
  m_count = 1;

  m_modbusRequests = new uint8_t *[MAX_NUM_MODBUS_TCP_REQUEST]; // rows
  for (uint32_t i = 0; i < MAX_NUM_MODBUS_TCP_REQUEST; ++i)
    {
      m_modbusRequests[i] = new uint8_t [MAX_MODBUS_PDU]; // cols
    }

  m_modbusRequestsInterval = new Time [MAX_NUM_MODBUS_TCP_REQUEST];
  m_modbusRequestsIndex = 0;
  m_modbusSendRequestsIndex = 0;
}

ModbusTcpClient::~ModbusTcpClient ()
{
  NS_LOG_FUNCTION_NOARGS ();
  m_socket = 0;
  delete [] m_data;
  m_data = 0;
  m_dataSize = 0;

  for (uint32_t i = 0; i < MAX_NUM_MODBUS_TCP_REQUEST; ++i)
    {
      delete [] m_modbusRequests[i];
```

```
    }
  delete [] m_modbusRequests;
  m_modbusRequests = 0;

  delete [] m_modbusRequestsInterval;
}

void
ModbusTcpClient::SetRemote (Ipv4Address ip, uint16_t port)
{
  m_peerAddress = ip;
  m_peerPort = port;
}

void
ModbusTcpClient::SetRequest (Time requestInterval, uint8_t *pdu)
{
  NS_LOG_INFO ("ModbusTcpClient::SetRequest ()");
  memcpy (m_modbusRequests[m_modbusRequestsIndex], pdu, MAX_MODBUS_PDU);
  m_modbusRequestsInterval[m_modbusRequestsIndex++] = requestInterval;
}

void
ModbusTcpClient::SetCount (uint32_t count)
{
  NS_LOG_INFO ("ModbusTcpClient::SetCount ()");
  m_count = count;
}

/*
std::list<Ptr<Socket> >
ModbusTcpClient::GetAcceptedSockets (void) const
{
  NS_LOG_FUNCTION (this);
  return m_socketList;
}
*/
void
ModbusTcpClient::DoDispose (void)
{
  NS_LOG_FUNCTION_NOARGS ();
  //m_socket = 0;
  //m_socketList.clear ();
```

```
  Application::DoDispose ();
}

void
ModbusTcpClient::StartApplication (void)
{
  NS_LOG_FUNCTION_NOARGS ();
  //std::cout << "ModbusTcpClient::StartApplication " << std::endl;

  if (m_socket == 0)
    {
      TypeId tid = TypeId::LookupByName ("ns3::TcpSocketFactory");
      m_socket = Socket::CreateSocket (GetNode (), tid);
      m_socket->SetAttribute ("DelAckCount", UintegerValue (1));
      m_socket->Bind ();
      m_socket->Connect (InetSocketAddress (m_peerAddress, m_peerPort));
      m_socket->Listen ();
    }

  m_socket->SetRecvCallback (MakeCallback (&ModbusTcpClient::HandleRead, this));

  m_socket->SetAcceptCallback (
    MakeNullCallback<bool, Ptr<Socket>, const Address &> (),
    MakeCallback (&ModbusTcpClient::HandleAccept, this));

/*
  m_socket->SetCloseCallbacks (
    MakeCallback (&ModbusTcpClient::HandlePeerClose, this),
    MakeCallback (&ModbusTcpClient::HandlePeerError, this));
*/

  if (m_modbusSendRequestsIndex < m_modbusRequestsIndex)
    {
      m_sendEvent = Simulator::Schedule (m_modbusRequestsInterval[0],
&ModbusTcpClient::Send, this);
    }
}

void
ModbusTcpClient::StopApplication ()
{
  NS_LOG_FUNCTION_NOARGS ();
```

```
  if (m_socket != 0)
    {
      m_socket->Close ();
      m_socket->SetRecvCallback (MakeNullCallback<void, Ptr<Socket> > ());
      m_socket = 0;
    }

  Simulator::Cancel (m_sendEvent);
}

void
ModbusTcpClient::Send (void)
{
  NS_LOG_INFO ("ModbusTcpClient::Send ()");

  NS_LOG_FUNCTION_NOARGS ();
  NS_ASSERT (m_sendEvent.IsExpired ());
  SeqTsHeader seqTs;
  seqTs.SetSeq (m_sent);

  m_size = 0;
  Ptr<Packet> p = Create<Packet> (m_size);

  // call to the trace sinks before the packet is actually sent,
  // so that tags added to the packet can be sent as well
  m_txTrace (p);

  p->AddHeader (seqTs);

  ModbusPdu cModbusRequestPdu;
  uint8_t *pdu = cModbusRequestPdu.GetPdu ();
  memcpy (pdu, m_modbusRequests[m_modbusSendRequestsIndex++], MAX_MODBUS_PDU);
  p->AddHeader (cModbusRequestPdu);

  if ((m_socket->Send (p)) >= 0)
    {
      // to get current time
      gettimeofday(&m_sent_time, NULL);

      ++m_sent;
      NS_LOG_INFO ("TraceDelay TX: " << (p->GetSize()-12) << " bytes to " // 8+4 : the size of the
seqTs header
                                << m_peerAddress << " Uid: " << p->GetUid ()
```

```
                    << " Time: " << (Simulator::Now ()).GetSeconds ());

    }
  else
    {
      NS_LOG_INFO ("Error while sending " << m_size << " bytes to "
                                          << m_peerAddress);
    }

  if (m_modbusSendRequestsIndex < m_modbusRequestsIndex)
    {
      m_sendEvent                                    =                                    Simulator::Schedule
(m_modbusRequestsInterval[m_modbusSendRequestsIndex],
       &ModbusTcpClient::Send, this);
    }
/*
  if (m_sent < m_count)
    {
      m_sendEvent = Simulator::Schedule (m_interval, &ModbusTcpClient::Send, this);
    }
*/

  if ((m_modbusSendRequestsIndex >= m_modbusRequestsIndex) && (m_sent < m_count))
    {
      m_modbusSendRequestsIndex = 0;
      m_sendEvent          =          Simulator::Schedule          (m_modbusRequestsInterval[0],
&ModbusTcpClient::Send, this);
    }
}

void
ModbusTcpClient::HandleRead (Ptr<Socket> socket)
{
  NS_LOG_INFO ("ModbusTcpClient::HandleRead ()");
  NS_LOG_FUNCTION (this << socket);
  Ptr<Packet> packet;
  Address from;
  while (packet = socket->RecvFrom (from))
    {
      if (InetSocketAddress::IsMatchingType (from))
        {
          // to get current time
          gettimeofday(&m_received_time, NULL);
```

```
        // log round-trip Modbus request/response
        //double t1 = m_sent_time.tv_sec + (m_sent_time.tv_usec/1000000.0);
        //double t2 = m_received_time.tv_sec + (m_received_time.tv_usec/1000000.0);
        //NS_LOG_INFO ("Round-trip Modbus request/response (ms): " << (t2 - t1)*1000);
        //std::cout << "Round-trip Modbus request/response (ms): " << (t2 - t1)*1000 << std::endl;

        NS_LOG_INFO ("Received " << packet->GetSize () << " bytes from " <<
                     InetSocketAddress::ConvertFrom (from).GetIpv4 ());

        // read the first byte to see it is an exception or successful
        ModbusPdu cModbusResponsePdu;
        packet->RemoveHeader (cModbusResponsePdu);
        uint8_t *pdu = cModbusResponsePdu.GetPdu ();

        if (0x80 <= pdu[0])
          {
            NS_LOG_INFO ("MODBUS response error - function code: 0x" << std::hex << (pdu[0] -
0x80)
              << " exception code: 0x" << std::hex << (pdu[1] + 0));
            switch (pdu[1])
              {
                case 0x01:
                  {
                    NS_LOG_INFO ("Invalid funcion or sub-function code");
                    break;
                  }
                case 0x02:
                  {
                    NS_LOG_INFO ("Invalid data address");
                    break;
                  }
                case 0x03:
                  {
                    NS_LOG_INFO ("Invalid data value");
                    break;
                  }
                case 0x04:
                case 0x05:
                case 0x06:
                  {
                    NS_LOG_INFO ("MODBUS function execution failed");
                    break;
```

```
                }
              default:
                {
                  NS_LOG_INFO ("Unknown exception code!");
                  break;
                }
            }

          for (uint32_t i = 0; i < 5; ++i)
            {
              NS_LOG_INFO ("pdu[" << std::dec << i << "]: 0x" << std::hex << (pdu[i] + 0));
            }
        }
      else
        {
          NS_LOG_INFO ("Success - function code: 0x" << std::hex << (pdu[0] + 0));
          for (uint32_t i = 0; i < 10; ++i)
            {
              NS_LOG_INFO ("pdu[" << std::dec << i << "]: 0x" << std::hex << (pdu[i] + 0));
            }
        }
        NS_LOG_INFO (std::dec);
    }
  }
}
/*
void
ModbusTcpClient::HandlePeerClose (Ptr<Socket> socket)
{
  NS_LOG_INFO ("ModbusTcpClient, peerClose");
}

void
ModbusTcpClient::HandlePeerError (Ptr<Socket> socket)
{
  NS_LOG_INFO ("ModbusTcpClient, peerError");
}
*/
void
ModbusTcpClient::HandleAccept (Ptr<Socket> s, const Address& from)
{
  NS_LOG_FUNCTION (this << s << from);
  s->SetRecvCallback (MakeCallback (&ModbusTcpClient::HandleRead, this));
```

```
  //m_socketList.push_back (s);
}

} // Namespace ns3
```

modbus-tcp-server.h

This header file is located at *~/ns-3-allinone/ns-3-dev/src/applications/model*.

```
/* -*- Mode:C++; c-file-style:"gnu"; indent-tabs-mode:nil; -*- */
/*
 * Copyright (c) 2012 Simon Fraser University
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License version 2 as
 * published by the Free Software Foundation;
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
 *
 * Author: Reza Sahraei <mrs16@sfu.ca>
 */

#ifndef MODBUS_UDP_SERVER_H
#define MODBUS_UDP_SERVER_H

#include "ns3/application.h"
#include "ns3/event-id.h"
#include "ns3/ptr.h"
#include "ns3/address.h"
#include "packet-loss-counter.h"
#include "modbus-mgr.h"

namespace ns3 {
/**
 * \ingroup applications
 * \defgroup modbusudpclientserver modbusudpclientserver
 */
```

```
/**
 * \ingroup udpclientserver
 * \class ModbusUdpServer
 * \brief Create a server application which waits for input MODBUS UDP request
 *        and generates appropriate response according to the function code.
 *        It also uses the information carried into their payload of the packet
 *        to compute delay and to determine if some packets are lost.

 * \brief A MODBUS UDP server. Receives MODBUS UDP request from a remote host
 *        and generates appropriate response according to the function code.
 *        It also uses the information carried into their payload of the packet
 *        to compute delay and to determine if some packets are lost.
 */
class ModbusUdpServer : public Application
{
public:
  static TypeId GetTypeId (void);
  ModbusUdpServer ();
  virtual ~ModbusUdpServer ();
  /**
   * returns the number of lost packets
   * \return the number of lost packets
   */
  uint32_t GetLost (void) const;

  /**
   * \brief returns the number of received packets
   * \return the number of received packets
   */
  uint32_t GetReceived (void) const;

  /**
   * \return the size of the window used for checking loss.
   */
  uint16_t GetPacketWindowSize () const;

  /**
   * \brief Set the size of the window used for checking loss. This value should
   *  be a multiple of 8
   * \param size the size of the window used for checking loss. This value should
   *  be a multiple of 8
   */
```

```
  void SetPacketWindowSize (uint16_t size);
protected:
  virtual void DoDispose (void);

private:

  virtual void StartApplication (void);
  virtual void StopApplication (void);

  void HandleRead (Ptr<Socket> socket);

  uint16_t m_port;
  Ptr<Socket> m_socket;

  ModbusMgr m_cModbusMgr;

  Address m_local;
  uint32_t m_received;
  PacketLossCounter m_lossCounter;
};

} // namespace ns3

#endif /* MODBUS_UDP_SERVER_H */
```

modbus-tcp-server.cc

This C++ file is located at *~/ns-3-allinone/ns-3-dev/src/applications/model*.

```
/* -*- Mode:C++; c-file-style:"gnu"; indent-tabs-mode:nil; -*- */
/*
 * Copyright (c) 2012 Simon Fraser University
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License version 2 as
 * published by the Free Software Foundation;
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
```

```
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
 *
 * Author: Reza Sahraei <mrs16@sfu.ca>
 */

#include "ns3/log.h"
#include "ns3/ipv4-address.h"
#include "ns3/nstime.h"
#include "ns3/inet-socket-address.h"
#include "ns3/socket.h"
#include "ns3/simulator.h"
#include "ns3/socket-factory.h"
#include "ns3/packet.h"
#include "ns3/uinteger.h"
#include "packet-loss-counter.h"

#include "seq-ts-header.h"
#include "modbus-udp-server.h"
#include "modbus-pdu.h"

namespace ns3 {

NS_LOG_COMPONENT_DEFINE ("ModbusUdpServer");
NS_OBJECT_ENSURE_REGISTERED (ModbusUdpServer);


TypeId
ModbusUdpServer::GetTypeId (void)
{
  static TypeId tid = TypeId ("ns3::ModbusUdpServer")
    .SetParent<Application> ()
    .AddConstructor<ModbusUdpServer> ()
    .AddAttribute ("Port",
                   "Port on which we listen for incoming packets.",
                   UintegerValue (502),
                   MakeUintegerAccessor (&ModbusUdpServer::m_port),
                   MakeUintegerChecker<uint16_t> ())
    .AddAttribute ("PacketWindowSize",
                   "The size of the window used to compute the packet loss. This value should be a
multiple of 8.",
                   UintegerValue (32),
                   MakeUintegerAccessor (&ModbusUdpServer::GetPacketWindowSize,
                                         &ModbusUdpServer::SetPacketWindowSize),
```

```
                MakeUintegerChecker<uint16_t> (8,256))
  ;
  return tid;
}

ModbusUdpServer::ModbusUdpServer ()
  : m_lossCounter (0)
{
  NS_LOG_FUNCTION (this);
  m_received=0;
}

ModbusUdpServer::~ModbusUdpServer ()
{
  NS_LOG_FUNCTION (this);
}

uint16_t
ModbusUdpServer::GetPacketWindowSize () const
{
  return m_lossCounter.GetBitMapSize ();
}

void
ModbusUdpServer::SetPacketWindowSize (uint16_t size)
{
  m_lossCounter.SetBitMapSize (size);
}

uint32_t
ModbusUdpServer::GetLost (void) const
{
  return m_lossCounter.GetLost ();
}

uint32_t
ModbusUdpServer::GetReceived (void) const
{
  return m_received;
}

void
ModbusUdpServer::DoDispose (void)
```

```
{
  NS_LOG_FUNCTION (this);
  Application::DoDispose ();
}

void
ModbusUdpServer::StartApplication (void)
{
  NS_LOG_FUNCTION_NOARGS ();

  if (m_socket == 0)
    {
      TypeId tid = TypeId::LookupByName ("ns3::UdpSocketFactory");
      m_socket = Socket::CreateSocket (GetNode (), tid);
      InetSocketAddress local = InetSocketAddress (Ipv4Address::GetAny (), m_port);
      m_socket->Bind (local);
    }

  m_socket->SetRecvCallback (MakeCallback (&ModbusUdpServer::HandleRead, this));

}

void
ModbusUdpServer::StopApplication ()
{
  NS_LOG_FUNCTION (this);

  if (m_socket != 0)
    {
      m_socket->SetRecvCallback (MakeNullCallback<void, Ptr<Socket> > ());
    }
}

void
ModbusUdpServer::HandleRead (Ptr<Socket> socket)
{
  NS_LOG_FUNCTION (this << socket);
  Ptr<Packet> packet;
  Address from;
  while (packet = socket->RecvFrom (from))
    {
      if (InetSocketAddress::IsMatchingType (from))
        {
```

```
 if (packet->GetSize () > 0)
  {
    NS_LOG_INFO ("Parsing MODBUS UDP function code");

    uint32_t packetSize = packet->GetSize() - 12; // 8+4 : the size of the seqTs header
    ModbusPdu cModbusRequestPdu;
    packet->RemoveHeader (cModbusRequestPdu);
    uint8_t *pdu = cModbusRequestPdu.GetPdu ();
    uint8_t *rspPdu = m_cModbusMgr.DoMain(pdu);

    SeqTsHeader seqTs;
    packet->RemoveHeader (seqTs);
    uint32_t currentSequenceNumber = seqTs.GetSeq ();
    NS_LOG_INFO ("TraceDelay: RX " << packetSize <<
                 " bytes from "<< InetSocketAddress::ConvertFrom (from).GetIpv4 () <<
                 " Sequence Number: " << currentSequenceNumber <<
                 " Uid: " << packet->GetUid () <<
                 " TXtime: " << seqTs.GetTs () <<
                 " RXtime: " << Simulator::Now () <<
                 " Delay: " << Simulator::Now () - seqTs.GetTs ());

    m_lossCounter.NotifyReceived (currentSequenceNumber);
    m_received++;

    if (NULL != rspPdu)
     {
       // send MODBUS UDP response back
       NS_LOG_INFO ("Sending MODBUS UDP response packet");

       packet->RemoveAllPacketTags ();
       packet->RemoveAllByteTags ();

       ModbusPdu cModbusResponsePdu;
       uint8_t *pduRspTemp = cModbusResponsePdu.GetPdu ();
       for (uint16_t i = 0; i < MAX_MODBUS_PDU; ++i)
        {
          pduRspTemp[i] = rspPdu[i];
        }
       packet->AddHeader (cModbusResponsePdu);
       socket->SendTo (packet, 0, from);
     }
  }
}
```

```
    }
}

} // Namespace ns3
```

Appendix G.

Debugging Techniques

The debugging techniques employed in this project are a combination of *C++ std<<cout* print instructions, ns-3 logging functions and macros, Wireshark network traffic analyzer, and NetAnim network animator.

I mostly used the C++ print instructions in the Modbus protocol stack, which is majorly located under *modbus-mgr.cc*. Later, I have left some of these print instructions in the code to show the sequence of sending the requests, evaluating, and generating the responses.

The following is a list of ns-3 logging functions and macros that are used in the project:

LogComponentEnable

NS_LOG_COMPONENT_DEFINE

NS_OBJECT_ENSURE_REGISTERED

NS_LOG_FUNCTION

NS_LOG_FUNCTION_NOARGS

The ns-3 logging macros can be turned on or off using an environment variable called *NS_LOG*. The Linux command *export* is used to enable logging in ns-3 for a the running scenario script. The *export* command assigns the environment variable *NS_LOG* to one or more names, which are identified in the code using *NS_LOG_COMPONENT_DEFINE* macro. For example, the following command is used in *busModbusTcpScenario3.cc*:

```
NS_LOG_COMPONENT_DEFINE ("BusModbusTcpScenario3");
```

And also:

```
if (verbose)
{
  LogComponentEnable ("ModbusTcpServer", LOG_LEVEL_INFO);
  LogComponentEnable ("ModbusTcpClient", LOG_LEVEL_INFO);
}
```

In the condition that the following Linux command is entered in the command prompt before the execution of the scenario, all *NS_LOG_INFO* macros in the busModbusTcpScenario3.cc will print the associated messages:

```
export NS_LOG=BusModbusTcpScenario3=level_all
./waf --run "scratch/busModbusTcpScenario3"
```

If the scenario is executed with the *verbose* parameter turned on, the *ModbusTcpServer* and *ModbusTcpClient* names, which are respectively located in *modbus-tcp-server.cc* and *modbus-tcp-client.cc*, cause the logging macros in the files to be enabled as *LOG_LEVEL_INFO* level.

Appendix H.

How to Install

I this Appendix, I describe how to install ns-3 software and incorporate ns-Modbus code.

Mercurial code management is required to download ns-3. The following is the command to install Mercurial:

$apt-get install mercurial

The Mercurial command to download ns-3 code is:

hg clone http://code.nsnam.org/ns-3-dev

Alternately, the following command can be used to install the all-in-one distribution of ns-3, which supports Python biniding [10]:

hg clone http://code.nsnam.org/ns-3-allinone

cd ns-3-allinone

./download.py

ns-Modbus files are zipped in a tar file, which is called *ns-Modbus.tar.gz (*see Table 3 for a complete list of the files). The following are the steps to incorporate the ns-Modbus under the ns-3 installation:

Make a copy of the *wscript* file under ~*ns-3-allinone**ns-3-dev**src**applications* before extracting the tar file, since the tar file has a modified wscript file.

Extract the tar file under ~*ns-3-allinone* directory. This should add 25 files and override the *wscript*.

In a fresh installation of ns-3 no further step is required. Otherwise, manually add the highlighted statements in the appendix F under *wscript* section to the original ns-3 *wscript*, which was backed up, and store under ~*ns-3-allinone**ns-3-dev**src**applications.* For more information refer to section 3.4.3 ns-Modbus Incorporation into ns Distribution.

Appendix I.

How to Run the Scenarios

In this Appendix, the instructions to run the provided scenarios such as: myfirstModbusUdp, myfirstModbusTcp, busModbusUdp, busModbusTcp, busModbusTcpScenario1, starModbusUdp, and starModbusTcp are described. These scenarios accept certain parameters that I explain what they are and show how to use them. All of these instructions should run under the *~/ns-3/-allinone/ns-3-dev* path where ns-3 is installed. For example, if ns-3 is installed under */home/reza/temp* the following path should be used:

$cd /home/reza/temp/ns-3-allinone/ns-3-dev/

~/temp/ns-3-allinone/ns-3-dev$

myfirstModbusUdp

The simplest command to run the scenario is:

./waf --run "scratch/myfirstModbusUdp"

Set the following argument to enable the log info messages in the script before running it:

Export NS_LOG=FirstModbusUdpScriptExample=level_all

./waf --run "scratch/myfirstModbusUdp"

Set the following parameter to enable the log info messages in *ModbusUdpServer* and *ModbusUdpClient* in order to see verbose log messages (the default is off):

./waf --run "scratch/myfirstModbusUdp --verbose=1"

Set the following parameter to change the name of the animation file
(the default is first-modbus-udp-animfile.xml):

./waf --run "scratch/myfirstModbusUdp --animFile=yourAnimFilename.xml"

Multiple parameters setting is allowed:

./waf --run "scratch/myfirstModbusUdp --verbose=1 --animFile=yourAnimFilename.xml"

myfirstModbusTcp

The simplest command to run the scenario is:

./waf --run "scratch/myfirstModbusTcp"

Set the following argument to enable the log info messages in the script before running it:

Export NS_LOG=FirstModbusTcpScriptExample=level_all

./waf --run "scratch/myfirstModbusTcp"

Set the following parameter to enable the log info messages in *ModbusTcpServer* and *ModbusTcpClient* in order to see verbose log messages (the default is off):

./waf --run "scratch/myfirstModbusTcp --verbose=1"

Set the following parameter to change the name of the animation file
(the default is first-modbus-tcp-animfile.xml):

./waf --run "scratch/myfirstModbusTcp --animFile=yourAnimFilename.xml"

Multiple parameters setting is allowed:

./waf --run "scratch/myfirstModbusTcp --verbose=1 --animFile=yourAnimFilename.xml"

busModbusUdp

The simplest command to run the scenario is:

./waf --run "scratch/busModbusUdp"

Set the following argument to enable the log info messages in the script before running it:

Export NS_LOG=BusModbusUdp=level_all

./waf --run "scratch/busModbusUdp"

Set the following parameter to enable the log info messages in *ModbusUdpServer* and *ModbusUdpClient* in order to see verbose log messages (the default is off):

./waf --run "scratch/busModbusUdp --verbose=1"

Set the following parameter to change the name of the animation file
(the default is bus-modbus-udp-animfile.xml):

./waf --run "scratch/busModbusUdp --animFile=yourAnimFilename.xml"

Set the following parameter to change the number of nodes (the default is 4):

./waf --run "scratch/busModbusUdp --nCsma=10"

Multiple parameters setting is allowed:

./waf --run "scratch/busModbusUdp --verbose=1 --nCsma=5"

busModbusTcp

The simplest command to run the scenario is:

./waf --run "scratch/busModbusTcp"

Set the following argument to enable the log info messages in the script before running it:

Export NS_LOG=BusModbusTcp=level_all

./waf --run "scratch/busModbusTcp"

Set the following parameter to enable the log info messages in *ModbusTcpServer* and *ModbusTcpClient* in order to see the verbose log messages (the default is off):

./waf --run "scratch/busModbusTcp --verbose=1"

Set the following parameter to change the name of the animation file
(the default is bus-modbus-tcp-animfile.xml):

./waf --run "scratch/busModbusTcp --animFile=yourAnimFilename.xml"

Set the following parameter to change the number of nodes (the default is 4):

./waf --run "scratch/busModbusTcp --nCsma=10"

Multiple parameters setting is allowed:

./waf --run "scratch/busModbusTcp --verbose=1 --nCsma=5"

busModbusTcpScenario1

The simplest command to run the scenario is:

./waf --run "scratch/busModbusTcpScenario1"

Set the following argument to enable the log info messages in the script before running it:

Export NS_LOG=BusModbusTcpScenario1=level_all

./waf --run "scratch/busModbusTcpScenario1"

Set the following parameter to enable the log info messages in *ModbusTcpServer* and *ModbusTcpClient* in order to see the verbose log messages (the default is off):

./waf --run "scratch/busModbusTcpScenario1 --verbose=1"

Set the following parameter to change the name of the animation file
(the default is bus-modbus-tcp-scenario1-animfile.xml):

./waf --run "scratch/busModbusTcpScenario1 --animFile=yourAnimFilename.xml"

Set the following parameter to change the number of nodes (the default is 2):

./waf --run "scratch/busModbusTcpScenario1 --nCsma=10"

Multiple parameters setting is allowed:

./waf --run "scratch/busModbusTcpScenario1 --verbose=1 --nCsma=5"

starModbusUdp

The simplest command to run the scenario is:

./waf --run "scratch/starModbusUdp"

Set the following argument to enable the log info messages in the script before running it:

Export NS_LOG=StarModbusUdp=level_all

./waf --run "scratch/starModbusUdp"

Set the following parameter to enable the log info messages in *ModbusUdpServer* and *ModbusUdpClient* in order to see verbose log messages (the default is off):

./waf --run "scratch/starModbusUdp --verbose=1"

Set the following parameter to change the name of the animation file (the default is star-modbus-udp-animfile.xml):

./waf --run "scratch/starModbusUdp --animFile=yourAnimFilename.xml"

Set the following parameter to change the number of nodes around the star server (the default is 8):

./waf --run "scratch/starModbusUdp --nSpokes=10"

Multiple parameters setting is allowed:

./waf --run "scratch/starModbusUdp --verbose=1 --nSpokes=5"

starModbusTcp

The simplest command to run the scenario is:

./waf --run "scratch/starModbusTcp"

Set the following argument to enable the log info messages in the script before running it:

Export NS_LOG=StarModbusTcp=level_all

./waf --run "scratch/starModbusTcp"

Set the following parameter to enable the log info messages in *ModbusTcpServer* and *ModbusTcpClient* in order to see verbose log messages (the default is off):

./waf --run "scratch/starModbusTcp --verbose=1"

Set the following parameter to change the name of the animation file (the default is star-modbus-tcp-animfile.xml):

./waf --run "scratch/starModbusTcp --animFile=yourAnimFilename.xml"

Set the following parameter to change the number of nodes around the star server (the default is 8):

./waf --run "scratch/starModbusTcp --nSpokes=10"

Multiple parameters setting is allowed:

./waf --run "scratch/starModbusTcp --verbose=1 --nSpokes=5"

Appendix J.

Modbus Exception Codes

Table J1. Modbus Exception Codes.

Code	Name	Comment
01	ILLEGAL FUNCTION	The function code is not allowed in the server. The reasons are: 1 – The server is not supporting the function. 2 – The server is in a wrong state to perform the function (i.e. not configured but asked for a register).
02	ILLEGAL DATA ADDRESS	Invalid data address in the server (i.e. asking for a register number, which does not exist).
03	ILLEGAL DATA VALUE	The provided data value indicates a fault in a complex request (i.e. causing the length error). This code is NOT used for out-of-bound values.
04	SLAVE DEVICE FAILURE	An unrecoverable failure occurred while the server performing the function.
05	ACKNOWLEDGE	The server sends the exceptions acknowledging the client of receiving of the request when performing of the function is time consuming in order to avoid the client request timeout.
06	SLAVE DEVICE BUSY	The server is busy handling a lengthy function.
08	MEMORY PARITY ERROR	Used in Read/Write File recored and reference type 6, to show the file area failed a consistency check.
0A	GATEWAY PATH UNAVAILABLE	Gateway could not allocate resource from input to output ports. The reasons are: 1 – The gateway is misconfigured. 2 – The gateway is overloaded.
0B	GATEWAY TARGET DEVICE FAILED TO RESPOND	The target device is not responding.

Appendix K.

Modbus Round-trip Ratio

Table K1. Bus Modbus UDP Ratio.

Number of Nodes	Ratio Round-trip/Total time	Ratio Round-trip/Simulation time
2	14.58805745	34.21985816
5	25.64102564	50.58365759
10	37.72702256	62.9476584
25	55.93884376	77.9286475
50	69.38667045	85.14718031
75	75.51375767	86.77417438
100	79.36966429	88.26993777
125	71.11246365	77.46550467
150	69.5442143	74.34307976
175	68.99251167	73.24398067
200	69.96184817	73.47508537
225	67.60233599	70.66160972
253	65.00176352	67.45147456

Table K2. Bus Modbus TCP Ratio.

Number of Nodes	Ratio Round-trip/Total time	Ratio Round-trip/Simulation time
2	8.198683423	14.45147679
5	12.71017384	17.70543867
10	13.80618797	17.52176329
25	13.80545995	15.98864195
50	15.2189136	16.65967038
75	0.445784768	0.446558109
100	0.418507733	0.419010448
125	0.440359561	0.440757505
150	0.577952248	0.578399022
175	0.848086172	0.84875523
200	0.95961303	0.960317624
225	1.082367219	1.083140945
253	1.555895719	1.557060504

Table K3. Star Modbus UDP Ratio.

Number of Nodes	Ratio Round-trip/Total time	Ratio Round-trip/Simulation time
2	10.35197	28.90173
5	11.68898	34.76263
10	13.84036	40.11917
25	17.05625	49.91983
50	19.81646	57.42382
75	20.37918	60.03687
100	20.31985	62.36466
125	21.24625	65.36151
150	21.61027	66.51744
175	21.69155	69.99367
200	21.52557	70.55366
225	20.22631	71.64682
253	20.04022	72.77797
500	15.24273	80.51285
1000	8.673324	87.96734
1500	6.287035	91.51456
2000	4.987437	93.71308
2500	3.999804	94.71457
3000	3.385568	95.60426
3500	2.979247	96.37436

Table K4. Star Modbus TCP Ratio.

Number of Nodes	Ratio Round-trip/Total time	Ratio Round-trip/Simulation time
2	9.753645	18.38863
5	12.67427	21.18644
10	14.75892	23.04867
25	18.83952	28.70332
50	17.49236	25.74596
75	17.99806	26.25172
100	19.42271	27.61465
125	19.27656	27.54949
150	19.44579	27.87957
175	19.21759	27.6817
200	19.2062	27.82598
225	19.12966	28.23463
253	18.73535	28.13663
500	16.37647	28.43329
1000	12.08857	28.58869
1500	9.416448	28.47275
2000	7.760106	28.36035
2500	6.579581	28.2262
3000	5.872766	28.34672
3500	5.127755	28.06709