Chapter 9

ASSESSING THE INTEGRITY OF FIELD DEVICES IN MODBUS NETWORKS

Ryan Shayto, Brian Porter, Rodrigo Chandia, Mauricio Papa and Sujeet Shenoi

Abstract Pipeline control systems often incorporate thousands of widely dispersed sensors and actuators, many of them in remote locations. Information about the operational aspects (functionality) and integrity (state) of these field devices is critical because they perform vital measurement and control functions.

This paper describes a distributed scanner for remotely verifying the functionality and state of field devices in Modbus networks. The scanner is designed for the Modbus protocol and, therefore, accommodates the delicate TCP/IP stacks of field devices. Furthermore, field device scanning and data storage and retrieval operations are scheduled so as not to impact normal pipeline control operations. Experimental results and simulations demonstrate that the distributed scanner is scalable, distributable and operates satisfactorily in low bandwidth networks.

Keywords: Modbus networks, distributed scanner, field devices, integrity

1. Introduction

The oil and gas industry primarily uses distributed control systems implementing the Modbus protocol [6–8] for midstream and transport activities in pipeline operations. A typical midstream application may have 10,000 or more field devices dispersed over several thousand square miles, including offshore platforms and remote wells. On the other hand, a transport application may have 1,000 field devices located at various points along a 3,000 mile pipeline. Many of these devices are sensors that measure key process parameters such as pressure, temperature, flow and hydrogen sulfide content. Other field devices are actuators that perform various pipeline control actions.

Pipeline operators need accurate and timely information about the status and integrity of field devices [4, 5]. Operational and business decisions are

adversely affected when large numbers of field devices are non-operational or corrupted; such a situation could also lead to an industrial accident. Most energy companies employ technicians whose only job is to travel to distant sites to verify the condition of field devices and service them. Field devices may be audited once a month, often just once a quarter.

This paper describes the design and implementation of a distributed scanner that remotely verifies the functionality and state of field devices in Modbus networks. The scanner accommodates the delicate TCP/IP stacks of field devices and scanning activities can be scheduled to minimize the impact on control operations. Tests on laboratory-scale and virtual environments indicate that the distributed scanner is scalable, distributable and operates satisfactorily in low bandwidth environments. These features make it an attractive tool for providing situational awareness in pipeline control networks, including those incorporating legacy systems.

2. Modbus Protocol

The Modbus protocol is widely used in the oil and gas sector, especially for controlling natural gas and liquids pipelines. The original Modbus protocol [6] was designed in 1979 for serial communications between the control center (master unit) and field devices (slaves). The Modbus TCP protocol [7], which was published in 1999, extends its serial counterpart for use in IP-interconnected networks. The extended protocol enables a master to have multiple outstanding transactions, and a slave to engage in concurrent communications with multiple masters.

2.1 Modbus Serial Protocol

Modbus implements a strict request/response messaging system between a master unit and slave devices. The master uses unicast or broadcast messages to communicate with slave devices. In a unicast transaction, the master sends a request to a single slave device and waits for a response message from the slave. If a response message is not returned, the master assumes that the request has failed. In a broadcast transaction, the master sends a request to all the slave devices in the network; the slaves perform the requested action but do not send response messages to the master.

A Modbus message has three parts (Figure 1). The first is a header, which includes the slave's address and control information for the slave. The second part contains a protocol data unit (PDU), which specifies an application-level operation. PDUs have two fields: a function code describing the purpose of the message and function parameters associated with the request or reply aspect of the message. The third part of a Modbus message is used for error-checking.

The maximum length of a Modbus message is 256 bytes. The slave address and function code fields use one byte each and the error checking field uses two bytes; this leaves 252 bytes for the function parameters.

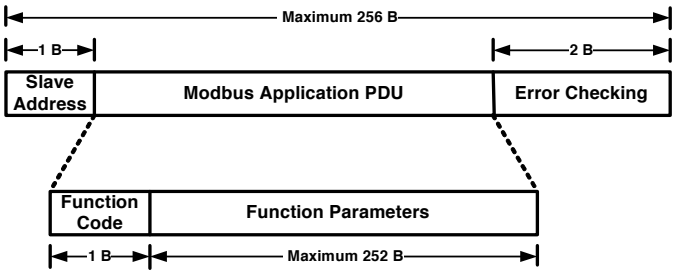

Figure 1. Modbus Serial message format.

Function codes are used by the master to perform diagnostic operations, configure slaves, perform control operations or obtain process data. Three types of function codes are defined in the Modbus protocol: public, user-defined and reserved codes. Public codes in the ranges [1, 64], [73, 99] and [111, 127] correspond to functions documented in the Modbus specification. Because there is no minimum set of function codes that is required to be implemented, vendors incorporate function codes in their products as needed. User-defined function codes in the ranges [65, 72] and [100, 110] are designated for vendors who wish to implement specialized functions. Reserved codes are used to support legacy systems; they overlap with public codes but cannot be used in new Modbus implementations.

Function codes in the [128, 255] range are used to denote error conditions. Specifically, if an error condition occurs for a function code $x \in [0, 127]$ in a request from a master to a slave, the corresponding error function code in the slave's response message is given by $y = x + 128$. Details about the error are indicated using an exception response in the data field of the message. Nine exception codes are specified: 1..6, 8 and 10..11. The exception codes 1..3 are useful when implementing Modbus network scanners. These codes are generated during the pre-processing of Modbus requests, i.e., before any action is taken by a slave to execute the master's request. Thus, malformed messages may be used by a Modbus scanner to obtain information about slave devices without affecting their state.

Table 1. Modbus memory table types.

Name	**Data Size**	**Usage**
Discrete Input	1 bit	Read-Only; Digital Input
Coil	1 bit	Read-Write; Digital Output
Input Register	16 bits	Read-Only; Analog Input
Holding Register	16 bits	Read-Write; Analog Output

Modbus devices store data in four types of tables: discrete inputs, coils, input registers and holding registers (Table 1). The maximum memory available for

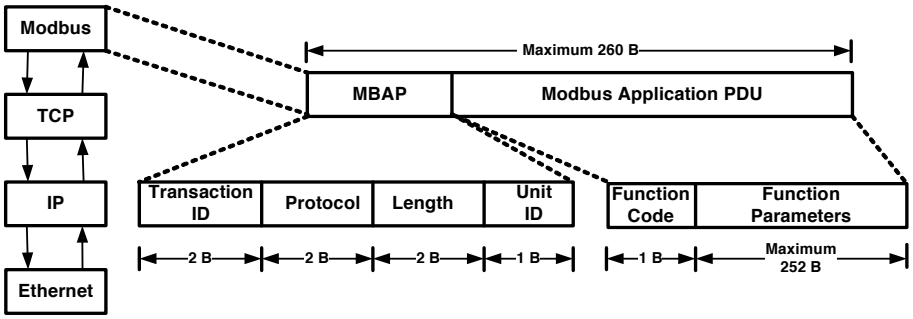

Figure 2. Modbus TCP message format.

each table type is 65,536 addressable items. A device may implement the tables using separate or overlapping memory spaces.

2.2 Modbus TCP Protocol

Modbus TCP extends the serial version of the protocol by wrapping messages with TCP/IP headers. A master is defined as a "client" because it initiates a TCP connection while a slave device is a "server" that passively listens for a connection on port 502 (or other optional ports). The client and server roles cannot be changed after a TCP connection is established. However, a Modbus device can establish a new connection in which it may assume a different role.

A Modbus TCP message uses the Modbus application protocol (MBAP) header instead of the serial message header (Figure 2). The MBAP has four fields: transaction identifier (2 bytes), protocol identifier (2 bytes), length (2 bytes) and unit identifier (1 byte). Since the MBAP takes up seven bytes, the maximum size of a Modbus TCP packet is 260 bytes. This length restriction arises from legacy implementations of the serial protocol.

The transaction identifier uniquely marks each transaction to permit the matching of paired request and reply messages. The protocol identifier specifies the protocol used for the message (this is set to zero corresponding to the protocol identifier for Modbus). The length field gives the size of the remaining portion of the Modbus message, which includes the unit identifier and the PDU. The unit identifier is used for addressing a slave located behind a gateway that bridges an IP network and a legacy serial network. The PDU is largely unchanged from the serial version. It incorporates the function code and data fields; however, error-checking is provided by the TCP layer.

3. Distributed Modbus Network Scanner

The distributed Modbus network scanner is designed to gather information about the functionality and state of field devices. It consists of a master scanner, remote sensors and a database, which stores data gathered during network scanning for further processing and applications support [10]. The master scanner controls the remote sensors that perform passive and/or active scans of local

Table 2. Remote sensor actions.

Commands	**Action**
Configure Network Interface	Set network interface for listening to traffic
Configure Database	Set IP address, user name and password for accessing database
Start Passive Scanning	Open network interface; Process traffic
Stop Passive Scanning	Flush queue of packets; Close network interface
Start Active Scanning	Retrieve assigned records; Begin scanning
Stop Active Scanning	Finish current scan; Close connections

networks. In addition, the master scanner may passively or actively scan its own network.

3.1 Master Scanner

The master scanner controls remote sensors, schedules active scans, examines the network topology and reviews device records. These actions rely heavily on a relational database, which is ideally located on the same host or subnet as the master scanner.

The master scanner constantly listens for connections from remote sensors. After establishing a connection with a sensor, the master scanner sends commands to initiate passive or active scans or to update sensor configuration.

3.2 Remote Sensors

Remote sensors deployed at field sites serve as collectors and filters of Modbus messages. Upon receiving the appropriate command from the master scanner, a remote sensor may configure its network interface or database connection, start/stop passive scanning or start/stop active scanning. Table 2 summarizes the actions performed by remote sensors upon receiving commands from the master scanner.

3.3 Database

A relational database is used to maintain data gathered by the master scanner and remote sensors, and to store system status information. The database contains tables that model field devices, and store data about device fingerprints and the status of active scans.

Three tables are used to hold information about Modbus field devices. The primary table, *Modbus Device*, stores identifying information about Modbus devices. The other two tables, *Function Codes* and *Memory Contents*, hold information about the functionality and state of field devices. In particular, *Function Codes* maintains records of the implemented function codes and *Memory Contents* stores the memory maps of field devices.

The *Device Fingerprint* table stores known fingerprints, including the components that define the signature. A device signature is based on the implemented function codes (device functionality) and the memory map (device state). This data enables the comparison of field devices against known fingerprints. The comparison is done using SQL commands, which offload processing from the scanner to the database.

The *Active Scan State* table stores the scanning status of each field device. An active scan involves several steps and may be performed over several days to conserve network bandwidth. The *Active Scan State* table provides information about the progress of a scan and enables scanning to resume in the event of sensor interruption or failure.

4. Modbus Network Scanning

This section describes the passive and active scanning modes used by the distributed scanner to collect and store information about Modbus devices.

4.1 Passive Scanning

Passive scanning uses a packet parser to examine the contents of Modbus messages. Messages are analyzed to determine the identity of the master unit and field devices involved in transactions, and message PDUs are parsed to discover the state of field devices. Note that traffic generated by the active scanning process (described in Section 4.2) is ignored to prevent the scanning process from being designated as a master unit in the database. All the information gathered is stored in the database tables described in the previous section.

Eventually, passive scanning discovers all the active devices in a Modbus network. The database is updated only when a network packet contains new information. Nevertheless, information retrieval and database updates consume significant network bandwidth, possibly impacting control system availability. This problem is alleviated by queuing database updates that involve data transfers exceeding 1 KB per second.

4.2 Active Scanning

Active scanning determines the functionality and state of devices in a Modbus network. In addition, it discovers inactive and disabled devices.

The active scanning algorithm exploits the Modbus protocol to safely determine if function codes are implemented by field devices. In the case of a function code that implements a read operation, the active scanning process sends a request message with the function code; a valid response from the addressed device implies that the function code is implemented by the device. For a function code corresponding to a write operation, a special malformed packet is sent so that a response is received without altering device memory.

Table 3. Sample *Active Scan State* data.

Device IP Address	Unit ID	Next Phase	Next Number	Next Scan	Time Between
192.168.30.1	1	Function Code	10	2007-03-28 16:40:00	00:05:00
192.168.30.2	3	Coils Maximum	0, 32767	2007-03-28 11:40:00	00:09:00

A slave device always checks the function code before acting on a Modbus message [6]. It returns an exception code of 1 if the function code is not implemented; an exception code of 2 (resp. 3) is returned when the function code is implemented, but a malformed request with an illegal data address (resp. illegal data value) was received by the device. The active scanning process interprets the exception response and updates the corresponding device record in the database. The function codes implemented by a Modbus device are determined by systematically querying the device for every function code in the range [0, 127].

Upon obtaining information about the functionality of a device, the active scanning process determines the range and size of each memory table in the device. Read-only requests are used for this purpose and a search is conducted using the starting and ending addresses to determine the valid addresses in each memory table. Note that although the Modbus documentation specifies tables with 65,536 addresses, some devices may not implement such large tables.

Scanning actions are scheduled so as not to interfere with normal operations. In fact, scanning may be scheduled over several days in the case of a large Modbus network.

An active scan of a device involves fourteen phases. The first phase tests whether or not the device implements function codes from 0 to 127. The next twelve phases determine the minimum, maximum and offset values for each of the four types of device memory. The final phase tests diagnostic sub-functions implemented under function code 8.

Table 3 presents two sample entries from the *Active Scan State* table. The first entry indicates that the function codes of Device 1 at the IP address 192.168.30.1 are being examined; and function code 10 is the next one to be tested. The second entry indicates that the maximum value of the coil memory table of Device 3 at IP address 192.168.30.2 is being identified. The *Next Number* field of the table stores the minimum and maximum values of the memory addresses used in the search. The *Next Scan* value indicates when the next scan operation will be performed, and the *Time Between* value specifies the interval between successive scan operations.

Our experiments have shown that a complete scan of a device involves an average of 300 request messages. To reduce bandwidth, consecutive messages

are sent after a relatively long time interval (e.g., four minutes apart in our laboratory testbed). The time interval should be increased for networks with large numbers of field devices.

5. Experimental Results

A major design goal is to ensure that the distributed scanner has minimal impact on normal Modbus network operations. This section evaluates the impact of the scanner on a laboratory testbed and a virtual environment intended to model a small-scale industrial environment. Scanner performance is evaluated with respect to three metrics: scalability, distributability and link utilization.

5.1 Experimental Setup

The distributed Modbus network scanner was evaluated using a laboratory-scale SCADA testbed as well as a larger virtual environment. The SCADA testbed incorporated two Allen-Bradley PLCs with Logix 5555 processors [1]; both devices used Prosoft MVI56-MNET Modbus TCP/IP interface modules for communication [9]. The testbed also incorporated two Direct Logic 205 PLCs with DL260 processors using D2-DCM Ethernet communication modules [2, 3].

The distributed scanner components used with the SCADA testbed were executed on two computers running Windows XP Service Pack 2. The master scanner and database were hosted on an Intel Xeon 3 GHz machine with 2 GB RAM. The remote sensor was hosted on an Intel Pentium III 1 GHz machine with 496 MB RAM.

A virtual experimental facility was created to evaluate the scalability of the distributed scanner in industrial environments. Figure 3 illustrates the virtual SCADA environment. Virtual PLCs were created using Jamod Java libraries [12]. The master and slave devices were located on separate machines; otherwise, traffic between the master and slaves is transmitted at the operating system level and is not visible to a scanner. Each group of slaves was placed on a separate VMWare guest [11] to ensure that they would only see their network adapter. Each guest, which was assigned 512 MB RAM, hosted up to 50 slave devices. The VMWare guests used Windows XP Service Pack 2.

The distributed scanner for the virtual environment incorporated three computers (Table 4). One computer created a virtual router using IMUNES, a popular network simulator based on Unix FreeBSD [13]. The other two computers, which ran Windows XP Service Pack 2, were configured to communicate through the virtual router.

The traffic volumes generated during passive and active scanning of the physical and virtual environments are summarized in Table 5. Passive scanning produces traffic volumes that are within a few hundred bytes for the two environments. The active scanning results are also similar. The minor differences seen for the two environments may be attributed to lost packets and other network transmission errors. Therefore, it appears that the virtual envi-

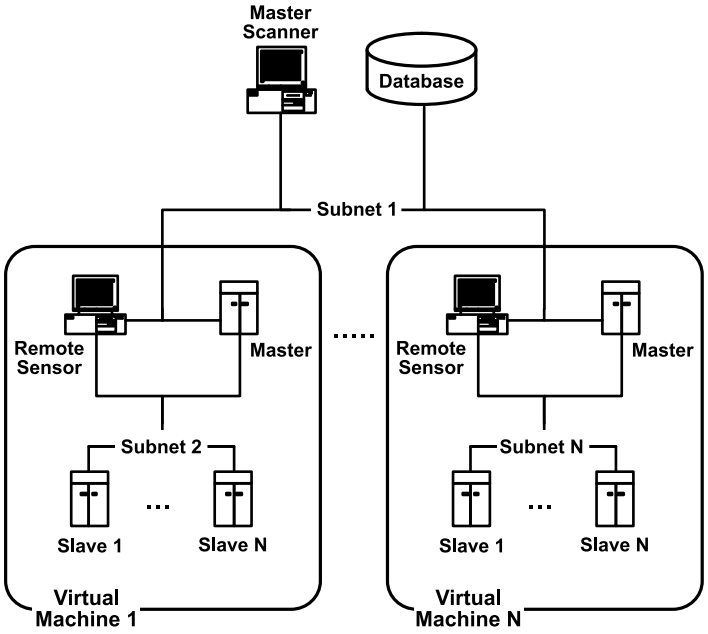

Figure 3. Virtual environment.

Table 4. Virtual environment components.

Host	Role	Hardware
Computer 1	Master Scanner and Database	Intel Xeon 3 GHz, 2 GB RAM
Computer 2	3 VMWare guests with Remote Sensor, Master and Slaves	Intel Xeon 3 GHz, 2 GB RAM
Computer 3	IMUNES virtual router	Pentium III 800 MHz, 768 MB RAM

Table 5. Scanning results for the physical and virtual environments.

	Subnets	Passive Scan KB Sent	Active Scan KB Sent
Physical	1	15.720	528.191
	2	31.911	1056.350
Virtual	1	15.952	528.511
	2	31.788	1057.051

Table 6. Bytes sent during active and passive scanning.

Slave Devices	Passive Scan Traffic (KB)	Active Scan Traffic (KB)
1	15.95	528.51
10	33.23	3674.05
20	52.38	7166.56
30	71.63	10531.38
40	90.82	13928.15
50	107.64	17324.92
Incremental Traffic per Device	1.89	342.31

ronment closely models the physical environment, and the results of tests on the virtual environment are applicable to real-world Modbus networks.

5.2 Performance Metrics

The three performance metrics considered in this study were scalability, distributability and link utilization. The scanner is deemed to be scalable if the volume of scanning traffic increases linearly with the number of devices in a subnet. The scanner is distributable if the traffic increases linearly as the number of subnets grows given that each subnet has the same number of devices. Link utilization is measured as the fraction of the available bandwidth used by the scanner, i.e., bytes per second divided by link speed.

Scalability The scalability of the distributed scanner was evaluated using the virtual environment. Traffic volumes generated during passive and active scanning were measured for subnets with numbers of devices ranging from 1 to 50.

Table 6 presents the results obtained for passive and active scanning. In both cases, the traffic volume grows linearly with the number of devices, which shows that the scanner is scalable. In the case of passive scanning, each additional device causes an average of 1.89 KB of incremental traffic to be sent from the remote sensor to the master scanner. For active scanning, each device adds an average of 342.31 KB of traffic.

Distributability The scanner satisfies the distributability metric when the volume of traffic it generates grows linearly with the number of subnets. In the tests, the number of subnets ranged from 1 to 3 and the number of devices per subnet were 1, 5 and 10.

The results in Tables 7 and 8 show that the traffic volume generated during passive and active scanning is proportional to the number of subnets. Note that the relationship holds regardless of the number of devices per subnet.

Table 7. Traffic generated during passive scanning.

1 Subnet		2 Subnets		3 Subnets	
Slaves	**KB Sent**	**Slaves**	**KB Sent**	**Slaves**	**KB Sent**
1	15.952	2	31.788	3	46.992
5	23.817	10	47.320	15	69.645
10	33.360	20	66.140	30	100.930

Table 8. Traffic generated during active scanning.

1 Subnet		2 Subnets		3 Subnets	
Slaves	**KB Sent**	**Slaves**	**KB Sent**	**Slaves**	**KB Sent**
1	528.511	2	1057.051	3	1584.456
5	1926.827	10	3855.527	15	5778.640
10	3674.047	20	7347.702	30	11022.781

Link Utilization Link utilization measures the impact of scanner transmissions on network bandwidth. In the case of passive scanning, traffic is generated by three types of events: (i) when the remote sensor establishes a connection to the master scanner, (ii) when a new device is detected in the network, and (iii) when the function codes implemented in previously detected devices are being determined.

Table 9. Link utilization during passive scanning.

Link Speed	Connection	New Device	Update FCs
52 Kbps	25.38%	1.63%	2.54%
128 Kbps	10.31%	0.66%	1.03%
384 Kbps	3.44%	0.22%	0.34%
768 Kbps	1.72%	0.11%	0.17%
1.5 Mbps	0.88%	0.06%	0.09%
10 Mbps	0.13%	0.01%	0.01%

The results in Table 9 show that passive scanning uses only a small fraction of the available bandwidth after the initial connection phase. Since the initial connection occurs only during start up, the operator of the distributed scanner can plan for this situation.

To forestall link flooding, especially during the initial scanning of a network, the distributed scanner imposes a limit on the number of packets sent per second. Figure 4 shows a graphic of the number of packets transmitted during

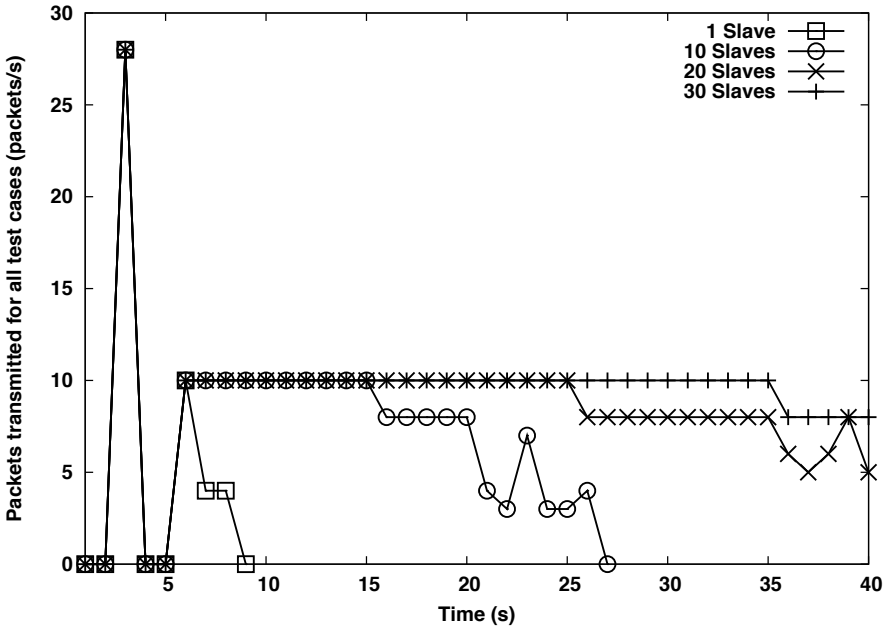

Figure 4. Device discovery with limited transmissions.

passive scanning of a subnet with the number of slaves ranging from 1 to 30. As expected, there is an initial spike as new devices are detected. However, the limits imposed on the packet rate soon take effect, helping prevent excessive link utilization.

Table 10. Link utilization during one active scanning step.

Link Speed	% Link Used
52 Kbps	3.32%
128 Kbps	1.35%
384 Kbps	0.45%
768 Kbps	0.22%
1.5 Mbps	0.12%
10 Mbps	0.02%

Unlike passive scanning, the active scanning process executes its scan steps following a schedule that prevents excessive link utilization. A typical scanning schedule spreads the active scan over a twenty-four hour period. Measurements show that a single scan step produces 1,725 bytes of traffic directed at the master scanner. Table 10 shows link utilization during active scanning with an

unreasonably high rate of one scan step per second. In fact, active scanning consumes only 3.32% of the bandwidth even for a slow (52 Kbps) link.

6. Conclusions

Despite the scale, cost and significance of the oil and gas pipeline infrastructure, asset owners and operators do not currently have the means to remotely verify the integrity of the thousands of field devices that are vital to pipeline operations. Our distributed scanner addresses this need by remotely conducting stateful analyses of Modbus devices, verifying their configurations and assessing their integrity. Tests on a laboratory system and a virtual environment that models a small-scale industrial facility indicate that the distributed scanner is scalable, distributable and operates satisfactorily in low bandwidth environments. Equally important is the fact that the scanner is designed to have minimal impact on normal pipeline control operations. In particular, the scanner accommodates the delicate TCP/IP stacks of field devices and scanning activities can be scheduled based on network size and bandwidth.

Our future work will focus on testing the distributed scanner in simulated moderate-scale and large-scale pipeline control environments. A demonstration project involving an operating pipeline is also planned. We hope this work will spur the development of sophisticated situational awareness systems that provide control center operators with a comprehensive view of network topology along with detailed information about the configuration, status and integrity of field devices, communications links and control center software.

Acknowledgements

This work was partially supported by the Institute for Information Infrastructure Protection (I3P) at Dartmouth College, Hanover, New Hampshire, under Award 2003-TK-TX-0003 and Award 2006-CS-001-000001 from the U.S. Department of Homeland Security.

References

[1] Allen-Bradley, Logix5000 Controllers Common Procedures Programming Manual, Milwaukee, Wisconsin, 2004.

[2] Automation Direct, D2-DCM Data Communications Module User Manual (web2.automationdirect.com/static/manuals/d2dcm/d2dcm.pdf), 2003.

[3] Automation Direct, DL205 User Manual (Volumes 1 and 2) (www.automationdirect.com), 2003.

[4] J. Gonzalez, Security Strategies for Process Control Networks, Ph.D. Dissertation, Department of Computer Science, University of Tulsa, Tulsa, Oklahoma, 2006.

[5] J. Gonzalez and M. Papa, Passive scanning in Modbus networks, in *Critical Infrastructure Protection*, E. Goetz and S. Shenoi (Eds.), Springer, Boston, Massachusetts, pp. 175–187, 2007.

[6] Modbus IDA, Modbus Application Protocol Specification v1.1a, North Grafton, Massachusetts (www.modbus.org/specs.php), 2004.

[7] Modbus IDA, Modbus Messaging on TCP/IP Implementation Guide v1.0a, North Grafton, Massachusetts (www.modbus.org/specs.php), 2004.

[8] Modbus.org, Modbus over Serial Line Specification and Implementation Guide v1.0, North Grafton, Massachusetts (www.modbus.org/specs.php), 2002.

[9] ProSoft Technology, MVI56-MNET ControlLogix Platform Modbus TCP/IP Interface Module User Manual (www.prosoft-technology.com/content/download/2801/26796/file/mvi56_mnet_user_manual1.pdf), 2007.

[10] R. Shayto, Industry-Scale Distributed Scanners for Modbus Networks, Ph.D. Dissertation, Department of Computer Science, University of Tulsa, Tulsa, Oklahoma, 2007.

[11] VMWare, VMWare Server Virtual Machine Guide (www.vmware.com/pdf/server_vm_manual.pdf), 2007.

[12] D. Wimberger, Jamod – Java Modbus Implementation (jamod.sourceforge.net), 2004.

[13] M. Zec, Implementing a clonable network stack in the FreeBSD Kernel, *Proceedings of the 2003 USENIX Annual Technical Conference*, pp. 137–150, 2003.