



UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE
CENTRO DE TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA E
DE COMPUTAÇÃO



Uma Ferramenta de Manipulação de Pacotes para Análise de Protocolos de Redes Industriais Baseados em TCP/IP

Tiago Hiroshi Kobayashi

Orientador: Prof. Dr. Paulo Sérgio da Motta Pires

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Engenharia Elétrica e de Computação da UFRN (área de concentração: Engenharia de Computação) como parte dos requisitos para obtenção do título de Mestre em Ciências.

Número de ordem PPgEE: M236
Natal, RN, julho de 2009

Uma Ferramenta de Manipulação de Pacotes para Análise de Protocolos de Redes Industriais Baseados em TCP/IP

Tiago Hiroshi Kobayashi

Dissertação de Mestrado aprovada em 07 de julho de 2009 pela banca examinadora composta pelos seguintes membros:

Prof. Dr. Paulo Sérgio da Motta Pires (orientador) DCA/UFRN

Prof. Dr. José Sérgio da Rocha Neto (examinador externo) DEE/UFCG

Prof. Dr. Agostinho de Medeiros Brito Júnior DCA/UFRN

Prof. Dr. Jorge Dantas de Melo DCA/UFRN

Divisão de Serviços Técnicos

Catálogo da publicação na fonte. UFRN / Biblioteca Central Zila Mamede

Kobayashi, Tiago Hiroshi.

Uma ferramenta de manipulação de pacotes para análise de protocolos de redes industriais baseados em TCP/IP / Tiago Hiroshi Kobayashi - Natal, RN, 2009

59 f. : il.

Orientador: Paulo Sérgio da Motta Pires

Dissertação (Mestrado) - Universidade Federal do Rio Grande do Norte. Centro de Tecnologia. Programa de Pós-Graduação em Engenharia Elétrica e de Computação.

1. Protocolos industriais - Dissertação. 2. Redes industriais - segurança - Dissertação. 3. Modbus/TCP (Protocolo) - Dissertação. I. Pires, Paulo Sérgio da Motta. II. Universidade Federal do Rio Grande do Norte. III. Título.

RN/UF/BCZM

CDU 004.057.4:65(043.3)

*Aos meus pais, Eliana e Osamu,
pela educação e apoio prestados
durante minha vida.*

Agradecimentos

A minha namorada, agradeço o auxílio prestado no decorrer da dissertação e dos anos que convivemos.

Aos amigos, que contribuíram, direta ou indiretamente, para minha formação como pessoa e como profissional gostaria de ressaltar um agradecimento.

Ao Prof. Agostinho, que fez considerações relevantes para o desenvolvimento dessa dissertação.

Ao meu orientador, Prof. Paulo Motta, sou grato por contribuir e engrandecer essa dissertação.

Ao LabSIN (Laboratório de Segurança da Informação), através do projeto Petrobrás-SERI (Segurança de Redes Industriais), agradeço por fornecerem recursos essenciais ao desenvolvimento desta dissertação.

Resumo

Neste trabalho é apresentada uma ferramenta de manipulação de pacotes destinada à realização de testes em dispositivos que implementam protocolos de comunicação baseados em TCP/IP utilizados em redes industriais. A ferramenta foi desenvolvida em linguagem de programação Python, como uma extensão ao Scapy. Esta ferramenta, denominada **IndPM** - *Industrial Packet Manipulator*, permite testar os dispositivos presentes em redes industriais em relação a possíveis vulnerabilidades, realizar testes de conformidade de protocolos, coletar respostas de servidores existentes nas redes e utilizar os recursos do interpretador Python para compor testes. Como prova de conceito, foi implementado o protocolo Modbus/TCP. O protocolo DNP3 sobre TCP também foi implementado, mas não foi testado por indisponibilidade de recursos. Os resultados dos testes obtidos com a manipulação de pacotes Modbus/TCP mostram falhas de implementação em um módulo de comunicação para um Controlador Lógico Programável bastante utilizado na indústria.

Palavras-chave: Segurança em Redes Industriais, Segurança em Infraestrutura Crítica, Testes de Conformidade de Protocolos Industriais, Modbus/TCP, DNP3 sobre TCP.

Abstract

This work presents a packet manipulation tool developed to realize tests in industrial devices that implements TCP/IP-based communication protocols. The tool was developed in Python programming language, as a Scapy extension. This tool, named **IndPM** - Industrial Packet Manipulator, can realize vulnerability tests in devices of industrial networks, industrial protocol compliance tests, receive server replies and utilize the Python interpreter to build tests. The Modbus/TCP protocol was implemented as proof-of-concept. The DNP3 over TCP protocol was also implemented but tests could not be realized because of the lack of resources. The **IndPM** results with Modbus/TCP protocol show some implementation faults in a Programmable Logic Controller communication module frequently utilized in automation companies.

Keywords: Industrial Network Security, Critical Infrastructure Security, Industrial Protocols Compliance Tests, Modbus/TCP, DNP3 over TCP.

Sumário

Sumário	i
Lista de Figuras	iii
Lista de Tabelas	v
Lista de Símbolos e Abreviaturas	vii
1 Introdução	1
1.1 Manipulação de Pacotes	3
1.2 Organização da Dissertação	4
2 O Protocolo Modbus	5
2.1 Modbus	5
2.1.1 Modos de Comunicação Serial e Formato dos Pacotes	5
2.1.2 Funções do Modbus	6
2.1.3 Modbus/TCP	7
3 Implementação	11
3.1 Descrição do Scapy	11
3.2 Arquitetura do IndPM	12
3.2.1 Módulo Modbus/TCP	13
3.2.2 Interface Gráfica	13
4 Testes Realizados	17
4.1 Ambientes de Testes	17
4.2 Metodologia e Resultados	18
4.2.1 Testes de Conformidades	18
4.2.2 Teste de Segurança	28
5 Conclusões	31
Referências Bibliográficas	32
A DNP3 sobre TCP	35
A.1 DNP3	35
A.2 Módulo DNP3 sobre TCP do IndPM	36

Lista de Figuras

1.1	Modelo de quatro camadas TCP/IP.	1
1.2	Exemplo de uma rede de automação.	2
1.3	Exemplo de manipulação de pacotes utilizando o Scapy.	4
2.1	Formato do Pacote RTU.	6
2.2	Formato do pacote ASCII.	6
2.3	Fluxograma para pacotes Modbus com código de função igual a 1.	8
2.4	Pacote Modbus/TCP e encapsulamento dentro de um segmento TCP.	9
2.5	Conteúdo de um pacote Modbus/TCP.	9
3.1	Arquitetura do IndPM	12
3.2	Exemplo de utilização do Módulo Modbus/TCP do IndPM	14
3.3	Tela da interface da manipulação de pacotes do protocolo Modbus/TCP.	15
4.1	Ambientes de testes utilizados.	17
4.2	Teste realizado com pacote contendo o código de função igual a 1 e quantidade de registradores igual a 0.	19
4.3	Teste realizado com pacote contendo o código de função igual a 1 e quantidade de registradores maior que 2000.	19
4.4	Teste realizado com pacote contendo o código de função igual a 2 e quantidade de registradores igual a 0.	20
4.5	Teste realizado com pacote contendo o código de função igual a 3 e quantidade de registradores igual a 0.	20
4.6	Teste realizado com pacote contendo o código de função igual a 4 e quantidade de registradores igual a 0.	21
4.7	Teste realizado com pacote contendo o código de função igual a 15 e quantidade de registradores igual a 0.	21
4.8	Teste realizado com pacote contendo o código de função igual a 16 e quantidade de registradores maior que 120.	22
4.9	Teste realizado com pacote contendo o código de função igual a 23 e quantidade de registradores leitura igual a 0.	22
4.10	Teste realizado com pacote contendo o código de função igual a 23 e quantidade de registradores de escrita igual a 0.	22
4.11	Teste realizado com pacote contendo o código de função igual a 2 e quantidade de registradores maior que 2000.	23
4.12	Teste realizado com pacote contendo o código de função igual a 3 e quantidade de registradores maior que 125.	23

4.13	Teste realizado com pacote contendo o código de função igual a 4 e quantidade de registradores maior que 125.	24
4.14	Teste realizado com pacote contendo o código de função igual a 3 e a soma do endereço inicial e a quantidade de registradores maior que 10.000.	24
4.15	Teste realizado com pacote contendo o código de função igual a 4 e a soma do endereço inicial e a quantidade de registradores maior que 10.000.	24
4.16	Teste realizado com pacote contendo o código de função igual a 6 e o endereço inicial maior que 10.000.	25
4.17	Teste realizado com pacote contendo o código de função igual a 15 e a quantidade de saídas maior que 800.	25
4.18	Teste realizado com pacote contendo o código de função igual a 16 e a soma do endereço inicial e a quantidade de registradores maior que 10.000.	26
4.19	Teste realizado com pacote contendo o código de função igual a 22 e o endereço inicial maior que 10.000.	27
4.20	Teste realizado com pacote contendo o código de função igual a 23 e a soma do endereço inicial e a quantidade de registradores de leitura maior que 10.000.	27
4.21	Teste realizado com pacote contendo o código de função igual a 23 e a soma do endereço inicial e a quantidade de registradores de escrita maior que 10.000.	27
4.22	Teste realizado com pacote contendo o código de função igual a 7.	28
4.23	Teste realizado com pacote contendo o código de função igual a 8 e sub-função igual a 1.	28
4.24	Teste realizado com pacote contendo o código de função igual a 8 e sub-função igual a 65.5353.	29
4.25	Tráfego entre o módulo do CLP e o IndPM	29
4.26	Monitoramento do módulo Modbus/TCP.	30
A.1	Encapsulamento do DNP3 sobre TCP.	36
A.2	Tela da interface da manipulação de pacotes do protocolo DNP3 sobre TCP.	37

Lista de Tabelas

2.1	Principais funções especificadas no protocolo Modbus.	7
-----	---	---

Lista de Símbolos e Abreviaturas

APCI:	Application Control Protocol Information
ASCII:	American Standard Code for Information Interchange
CLP:	Controlador Lógico Programável
CR:	Carriage Return
CRC:	Cyclic Redundancy Check
DoS:	Denial of Service
GPL:	General Public License
GTK:	GIMP ToolKit
IP:	Internet Protocol
LF:	Line Feed
LRC:	Longitudinal Redundancy Check
RTU:	Remote Terminal Unit
SCADA:	Supervisory Control And Data Acquisition
TA:	Tecnologia de Automação
TCP/IP:	Transmission Control Protocol/Internet Protocol
TCP:	Transmission Control Protocol
XML:	eXtensible Markup Language

Capítulo 1

Introdução

A comunicação entre dispositivos interligados em rede, geralmente, é feita utilizando-se protocolos enquadrados em modelos multiníveis. Um desses modelos, apresentado na Figura 1.1, é chamado de pilha TCP/IP (*Transmission Control Protocol/Internet Protocol*)[Stevens 2001]. O modelo é composto por quatro camadas: Enlace, Internet (Rede), Transporte e Aplicação. Este modelo de comunicação entre dispositivos será utilizado no decorrer dessa dissertação.

Pilha TCP/IP

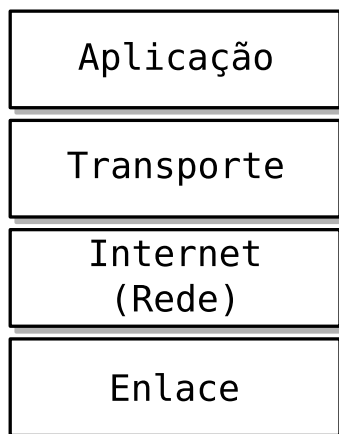


Figura 1.1: Modelo de quatro camadas TCP/IP.

As redes industriais ou redes de automação possuem protocolos de comunicação específicos com os quais os dispositivos trocam informações. Essas redes são compostas por diversos dispositivos, sendo comum encontrar os Controladores Lógicos Programáveis (CLPs) e os dispositivos sensores e/ou atuadores. Os sensores são responsáveis por adquirir informações, como temperatura, pressão ou vazão, e enviá-las ao CLP. Os atuadores possuem a função de modificar os estados dos dispositivos que atuam diretamente em um processo, como válvulas ou motores. Os CLPs, por sua vez, são dispositivos importantes num sistema de controle, uma vez que são responsáveis por controlar o processo

através do recebimento de informações dos sensores e do envio de comandos/ações para os atuadores. Estes dispositivos constituem os níveis mais baixos das redes de automação [Sauter 2005, Lobashov & Sauter 2006], sendo ilustrados na Figura 1.2.

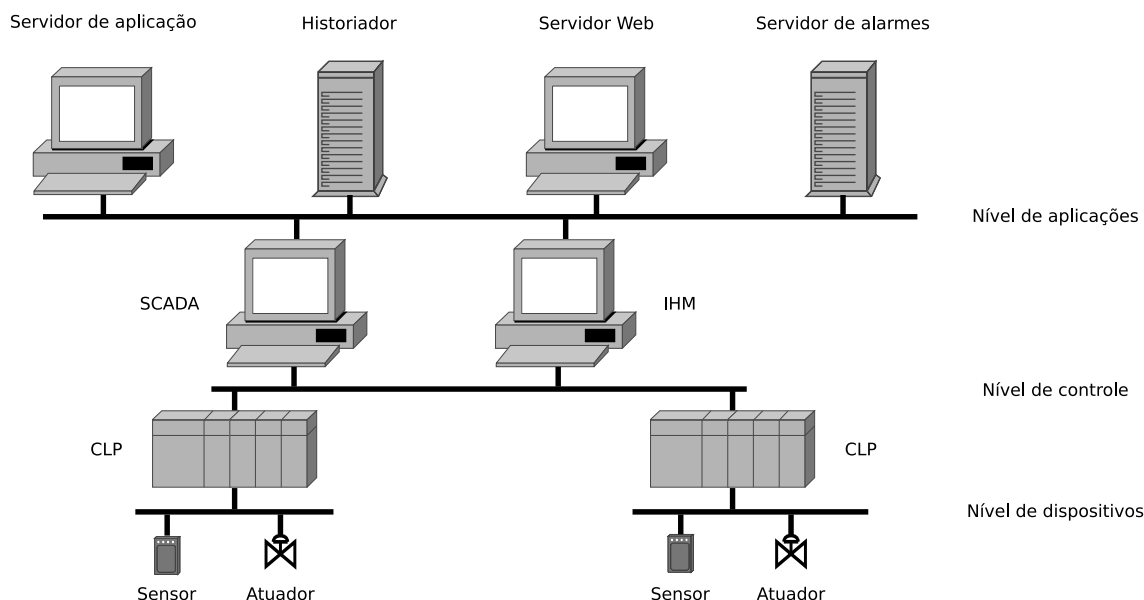


Figura 1.2: Exemplo de uma rede de automação.

Os vários níveis das redes industriais utilizam protocolos de comunicação diversificados, sendo alguns destes baseados na pilha TCP/IP. Esses protocolos são utilizados para permitir a interconexão entre os dispositivos da rede, que facilita a troca de informações entre esses dispositivos. Esta interconexão pode trazer desvantagem também, sendo necessário analisar mecanismos de segurança dessas informações, que podem ser transmitidas através de meios não confiáveis [Bailey & Wright 2003].

Os ataques às redes de automação possuem motivações e implicações semelhantes a qualquer ação maliciosa que ocorre em outros sistemas computacionais. Os aspectos chave utilizados para garantir a segurança da informação, como integridade, disponibilidade, confidencialidade e autenticidade, também são considerados em estudos na área de segurança das redes industriais. Um destes trabalhos [Shifflet 2004] apresenta uma taxonomia para os sistemas de controle levando em consideração as peculiaridades desses aspectos chave de segurança existentes nos ambientes de tecnologia de automação (TA). Um exemplo destas particularidades seria o aspecto da disponibilidade da informação. No caso de sistemas de controle, a disponibilidade pode ser de extrema importância para o funcionamento normal de um sistema.

As vulnerabilidades existentes nas redes de automação também podem ocorrer no nível de arquitetura, devido à utilização de sistemas antigos (legados). Muitos dos sistemas antigos foram desenvolvidos sem preocupação com aspectos de segurança e com poucos testes de segurança comprovados [Shifflet 2004].

As possibilidades de inserção de vulnerabilidades podem ser expandidas com a interconexão das redes corporativas e redes industriais [Pires & Oliveira 2006]. Esta interco-

nexão modifica a arquitetura dos sistemas de controle, interligando diferentes sistemas e dispositivos computacionais às redes de automação. Assim, é importante definir alguns aspectos relativos à segurança, para que se possam obter as vantagens da interconexão entre as redes.

Os protocolos de comunicação das redes industriais que são baseados na pilha TCP/IP são comumente utilizados na comunicação entre CLP's e sistemas SCADA (*Supervisory Control And Data Acquisition*) (Nível de controle, na Figura 1.2). Como os protocolos podem apresentar vulnerabilidades [Byres et al. 2006, Mander et al. 2007, Grzelak 2008], é importante analisá-los com o objetivo de identificar essas vulnerabilidades, evitando que problemas de segurança possam ocorrer quando de suas utilizações.

Na Seção 1.1, apresenta-se a utilização de manipuladores de pacotes, expondo as utilidades deste tipo de ferramenta no teste de dispositivo e na análise de protocolos. Finalizando este capítulo, tem-se a organização desta dissertação na Seção 1.2.

1.1 Manipulação de Pacotes

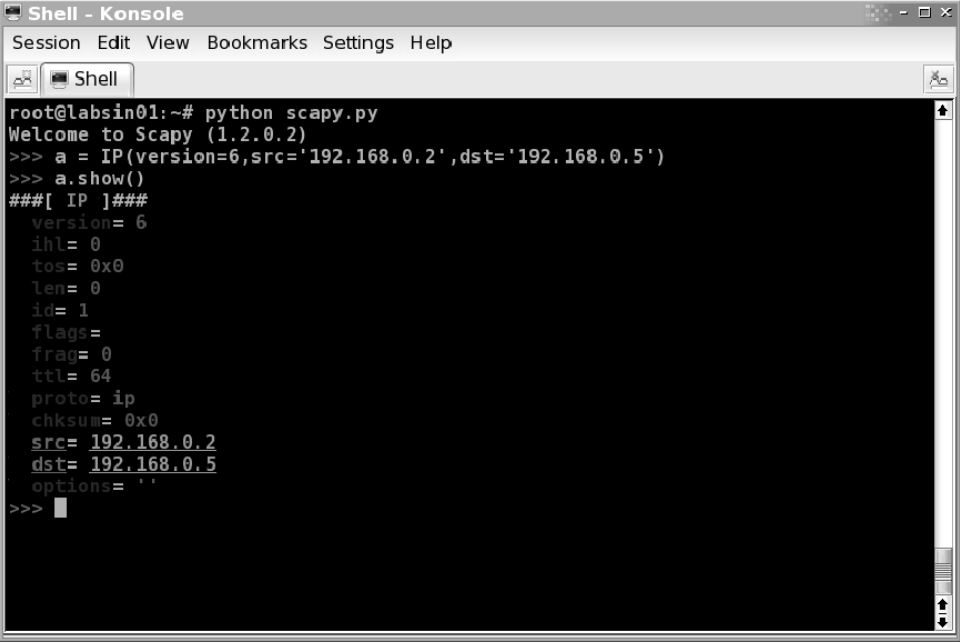
Os programas manipuladores de pacotes têm como principal objetivo construir pacotes modificando o conteúdo de seus campos, de acordo com a necessidade do usuário. Um exemplo de uma ferramenta que tem essa funcionalidade em redes TCP/IP é o Scapy¹. Na Figura 1.3, mostra-se um exemplo de manipulação utilizando o protocolo IP com o Scapy. Neste exemplo, um datagrama IP é manipulado, pela definição do campo de versão, do endereço de origem e do endereço de destino.

Os manipuladores de pacotes podem ser utilizados com diferentes objetivos. Um exemplo de uso de um manipulador de pacotes seria testar dispositivos de segurança, como *firewalls*. As *firewalls* são mecanismos de defesa que impedem o acesso indevido a uma rede ou a um sistema computacional, analisando o tráfego destinado à rede ou ao sistema computacional [Tanenbaum 2003]. Outra utilização de manipuladores de pacotes é na realização de testes de conformidades de dispositivos em relação à implementação de determinado protocolo.

O objetivo neste trabalho é apresentar uma ferramenta de manipulação de pacotes, chamada **IndPM** (*Industrial Packet Manipulator*), mostrando os testes e resultados que podem ser obtidos com este tipo de ferramenta. O **IndPM** permite testar os dispositivos que implementam os protocolos industriais que têm como base a pilha TCP/IP. Como prova de conceito, foi implementado o protocolo Modbus/TCP e foram realizados testes em um CLP real, comumente utilizado em redes de automação que utiliza esse protocolo, e em um emulador do protocolo Modbus/TCP.

A ferramenta também pode ser utilizada para coletar respostas dos dispositivos testados, criar e utilizar pacotes específicos para realizar testes de vulnerabilidades, bem como realizar a composição de testes usando o ambiente de programação Python.

¹Scapy. <http://www.secdev.org/projects/scapy/>



```
root@labsin01:~# python scapy.py
Welcome to Scapy (1.2.0.2)
>>> a = IP(version=6,src='192.168.0.2',dst='192.168.0.5')
>>> a.show()
###[ IP ]###
version= 6
ihl= 0
tos= 0x0
len= 0
id= 1
flags=
frag= 0
ttl= 64
proto= ip
chksum= 0x0
src= 192.168.0.2
dst= 192.168.0.5
options= ''
>>>
```

Figura 1.3: Exemplo de manipulação de pacotes utilizando o Scapy.

1.2 Organização da Dissertação

Este trabalho contém mais quatro capítulos. No Capítulo 2, apresenta-se o Modbus/TCP, que foi implementado como prova de conceito. Serão apresentados os detalhes necessários para o entendimento do protocolo Modbus/TCP e sua utilização.

No Capítulo 3, são apresentados os aspectos relativos às implementações do **IndPM**. São descritos como o módulo referente ao protocolo industrial foi implementado, bem como a interface gráfica desenvolvida.

No Capítulo 4, os resultados obtidos com a manipulação de pacotes realizados com o **IndPM** serão apresentados. A manipulação de pacotes foi testada em dois dispositivos diferentes, mostrando as falhas existentes nesses dispositivos.

No último Capítulo, apresentam-se as conclusões referentes aos resultados obtidos com a ferramenta desenvolvida e perspectivas para trabalhos futuros.

No Apêndice A, é apresentado o protocolo DNP3 sobre TCP e sua implementação. Apesar de totalmente implementado na ferramenta, seu funcionamento não pode ser validado por indisponibilidade de dispositivos.

Por fim, as publicações realizadas com este trabalho estão no Apêndice B.

Capítulo 2

O Protocolo Modbus

Nas seções a seguir, apresentam-se alguns detalhes do protocolo industrial que foi implementado, o Modbus/TCP. A implementação desse protocolo foi baseada na especificação ([Modbus-IDA 2006a], [Modbus-IDA 2006b]).

2.1 Modbus

O protocolo Modbus é baseado no paradigma de comunicação mestre-escravo, onde o mestre é o responsável por coordenar diversos escravos. O Modbus é utilizado no nível da camada de aplicação [Modbus-IDA 2006a] e foi inicialmente desenvolvido para ser utilizado na comunicação dos CLPs com os sensores e os atuadores. O protocolo Modbus foi desenvolvido pela *Modicon Industrial Automation Systems* (atualmente *Schneider Electric*) nos anos 70 e recentemente tornou-se GPL (*General Public License*). Atualmente, o Modbus é mantido pela Modbus-IDA que é formada por um grupo de usuários e fornecedores independentes.

2.1.1 Modos de Comunicação Serial e Formato dos Pacotes

Inicialmente, o Modbus foi desenvolvido para dispositivos que utilizavam meios de comunicação serial. A comunicação Modbus que utiliza linhas seriais pode ser realizada de duas formas: RTU (*Remote Terminal Unit*) e ASCII (*American Standard Code for Information Interchange*). Esses dois modos de comunicação do Modbus possuem um formato de pacote específico, que é o mesmo para requisição ou para resposta. As formas RTU e ASCII diferenciam-se na representação dos seus dados.

Cada pacote Modbus serial é composto de seis campos que são representados por dados binários, no modo RTU, e por caracteres, no modo ASCII. Estes campos são definidos como:

- **Início:** indica o começo do pacote;
- **Endereço:** indica qual dispositivo receberá ou enviará o pacote. A faixa de endereços válidos dos dispositivos varia de 0 a 247, sendo o endereço 0 (zero) utilizado para mensagens que são enviadas para todos os escravos;
- **Função:** este campo representa o objetivo do pacote;

- **Dados:** campo onde os dados serão alocados. Este campo tem o seu conteúdo de acordo com o campo **Função** do pacote;
- **Controle:** é responsável pela detecção de erros no pacote. Para versões que possuam camadas de protocolos subjacentes que contenham mecanismos de detecção de erros próprios, este campo é dispensado;
- **Fim:** sinaliza que a comunicação entre os dispositivos foi terminada.

No modo RTU, o campo **Início** é representado por um intervalo de silêncio. Os campos, de **Endereço** e de **Função**, são representados por 8 *bits*. O campo **Dados** depende da função utilizada e cada informação é representada por 8 *bits* também, podendo conter inúmeras informações (n) dentro de um mesmo pacote. A verificação de erro neste modo é realizada através do algoritmo de CRC (*Cyclic Redundancy Check*), sendo representado por 16 *bits*. O campo **Fim** é representado pelo mesmo conteúdo do campo **Início**, um intervalo de silêncio.

O campo **Início**, no modo ASCII, é definido pelo caractere ":" (0x3A em hexadecimal). Os campos, de **Endereço** e de **Função**, são representados por dois caracteres. O campo **Dados**, no modo ASCII, é representado por uma quantidade variável (n) de caracteres. O campo **Controle** é representado por dois caracteres ASCII e utiliza o algoritmo de LRC (*Longitudinal Redundancy Check*) para detecção de erros. O fim do pacote é representado por um par de caracteres de CR (*Carriage Return*) e de LF (*Line Feed*) (0x0D e 0x0A em hexadecimal respectivamente).

Os campos dos pacotes RTU e ASCII podem ser visualizados na Figura 2.1 e na Figura 2.2, respectivamente.

Início	Endereço	Função	Dados	CRC	Fim
Silêncio	8 bits	8 bits	n x 8 bits	16 bits	Silêncio

Figura 2.1: Formato do Pacote RTU.

Início	Endereço	Função	Dados	LRC	Fim
: (0x3A)	2 caracteres	2 caracteres	n caracteres	2 caracteres	CRLF

Figura 2.2: Formato do pacote ASCII.

2.1.2 Funções do Modbus

O protocolo Modbus possui algumas funções que dependem do dispositivo que implementa o protocolo. As funções possuem diferentes objetivos, como a verificação da comunicação, a visualização da quantidade de eventos, dentre outras, sendo as mais utilizadas as funções de escrita e de leitura. Os dispositivos comunicam-se através de pacotes

de requisições e de respostas, onde os códigos de funções para estes dois tipos de pacotes podem ser iguais numa mesma transação.

As principais funções especificadas no protocolo Modbus, e suas descrições, são mostradas na Tabela 2.1. Mais detalhes das implementações das funções podem ser obtidos na especificação do protocolo [Modbus-IDA 2006a].

Código de função	Descrição
01	Ler o estado dos registradores booleanos de saída
02	Ler o estado de entradas discretas
03	Ler o conteúdo de um bloco de registradores de espera
04	Ler registradores de entrada
05	Escrever em um registrador booleano de saída
06	Escrever em um registrador de espera
07	Ler o conteúdo de oito estados de exceção
08	Prover uma série de testes para verificação da comunicação e erros internos
11	Obter o contador de eventos
12	Obter um relatório de eventos
15	Escrever em uma sequência de saídas
16	Escrever em um bloco de registradores
17	Ler algumas informações do dispositivo
20	Ler informações de um arquivo
21	Escrever informações em um arquivo
22	Modificar o conteúdo dos registradores de espera através de operações lógicas
23	Combina ler e escrever em registradores numa única transação
24	Ler o conteúdo da fila FIFO de registradores

Tabela 2.1: Principais funções especificadas no protocolo Modbus.

A especificação do protocolo define fluxogramas para cada função, detalhando o comportamento de cada uma delas. Esses fluxogramas apresentam como os pacotes devem ser respondidos em cada situação. Na Figura 2.3, apresenta-se um exemplo destes fluxogramas, mostrando o comportamento que um dispositivo deve apresentar quando recebe um pacote Modbus com o campo **Função** igual a 1.

Algumas funções do Modbus (7, 8, 11, 12 e 17) são definidas pelo protocolo como específicas para comunicações realizadas em canais de comunicação seriais, não sendo indicadas para uso em outros meios de comunicação.

2.1.3 Modbus/TCP

Com a grande utilização e facilidade à modificações do protocolo Modbus, foram incorporados novos meios de comunicações como o Ethernet que é utilizado pela variação do protocolo, denominada Modbus/TCP. Esta variação utiliza a pilha de protocolos TCP/IP para realizar as transações entre os dispositivos, que contribui para uma maior

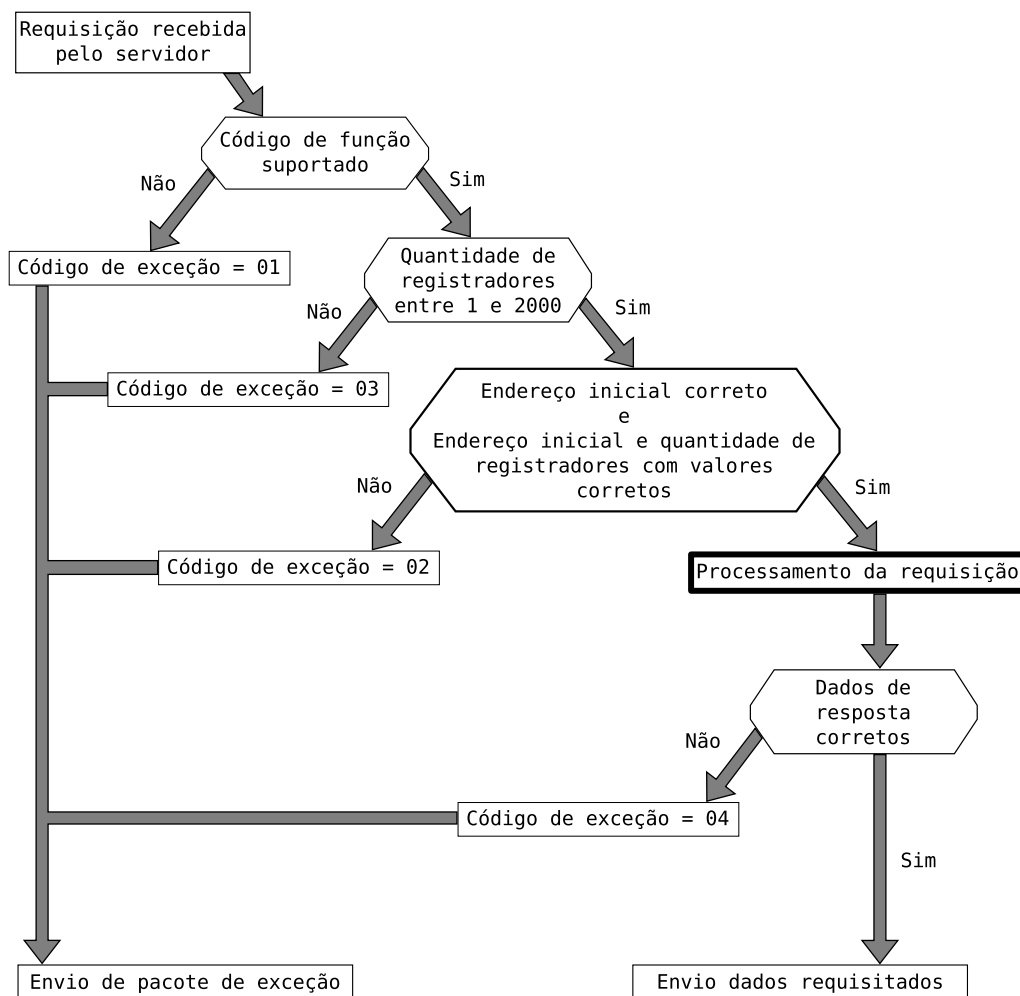


Figura 2.3: Fluxograma para pacotes Modbus com código de função igual a 1.

conectividade visto que permite interligar diferentes tipos de redes, compartilhando uma mesma infra-estrutura.

Os pacotes do Modbus/TCP consistem de uma modificação do pacote Modbus tradicional e do encapsulamento desse pacote em segmentos TCP, como se mostra na Figura 2.4. O Modbus/TCP contém somente dois campos do pacote Modbus tradicional (**Função** e **Dados**), sendo eliminados os campos de **Início**, de **Endereço** e de **Fim** que são desnecessários em comunicações sobre Ethernet, realizados por protocolos específicos. O campo de **Controle** também não está presente no Modbus/TCP porque a verificação de erros de transmissão é realizada através das camadas inferiores ao Modbus/TCP.

A camada do Modbus/TCP é composta por um cabeçalho para indicar parâmetros da transmissão, um campo para indicar o código de função utilizado e um campo contendo os dados que estão sendo transmitidos. Na Figura 2.5, apresentam-se esses campos existentes em pacotes Modbus/TCP, sendo o cabeçalho composto pelos quatro primeiros campos.

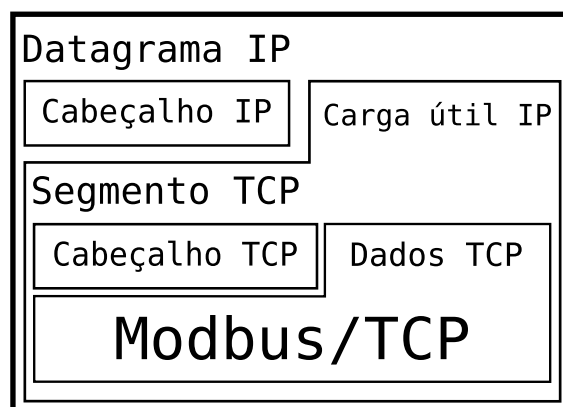


Figura 2.4: Pacote Modbus/TCP e encapsulamento dentro de um segmento TCP.

Identificador de Transação	Identificador de Protocolo	Tamanho	Identificador de Unidade	Função	Dados
2 Bytes	2 Bytes	2 Bytes	1 Byte	1 Byte	n Bytes

Figura 2.5: Conteúdo de um pacote Modbus/TCP.

Os quatro campos do cabeçalho Modbus/TCP podem ser definidos como:

- **Identificador de Transação:** este campo, com dois *bytes*, identifica a comunicação entre o servidor e o cliente Modbus. Cada requisição Modbus tem sua resposta com o mesmo identificador de transação. Para o Modbus/TCP, o número de requisições que os dispositivos respondem ou requisitam depende da capacidade dos mesmos. Este fato representa uma melhoria em relação ao Modbus tradicional que permite apenas uma única transação por vez.
- **Identificador de Protocolo:** este campo, com dois *bytes*, é utilizado para identificar o protocolo. O Modbus é especificado com valor 0 (zero).
- **Tamanho:** este campo, com dois *bytes*, contém a quantidade de *bytes* dos campos seguintes a ele (**Identificador de Unidade**, **Função** e **Dados**).
- **Identificador de Unidade:** este campo, com um *byte*, é utilizado para identificar os dispositivos Modbus que estejam interconectados na rede através de um *gateway*. Em geral, esses *gateways* são utilizados para interligar dispositivos que utilizam o barramento serial com os dispositivos da rede Ethernet.

Os outros dois campos do pacote Modbus/TCP, **Função** e **Dados**, são semelhantes aos campos existentes no Modbus tradicional. Eles contêm o código de função e os dados transmitidos, sendo o campo de dados determinado pelo código de função.

No próximo Capítulo, será apresentada a ferramenta desenvolvida, mostrando detalhes de implementação desse protocolo e da interface gráfica construída.

Capítulo 3

Implementação

Neste Capítulo, serão apresentados alguns detalhes da implementação da ferramenta desenvolvida. Inicialmente, será apresentado o *software* que foi utilizado como base do trabalho. Em seguida, será apresentada a arquitetura do *software* desenvolvido e detalhes da implementação do protocolo Modbus/TCP e da interface gráfica.

3.1 Descrição do Scapy

A base do nosso trabalho é um manipulador de pacotes. Algumas ferramentas com esta funcionalidade foram analisadas (Hping¹, Nemesis², PacGen³ e Scapy⁴) e dentre essas, optamos pela utilização do Scapy. Os motivos da escolha do Scapy como base de nossa implementação foram a disponibilidade do código-fonte do *software*, a variedade de protocolos implementados pela ferramenta bem como a facilidade para inclusão de novos protocolos e funcionalidades.

O Scapy é um *framework* desenvolvido em Python com muitas funcionalidades, dentre as quais a principal é a manipulação de pacotes. A ferramenta permite uma maior interatividade em relação a outras ferramentas existentes, permitindo a manipulação de pacotes de maneira flexível. Os pacotes podem ser construídos independentes da pilha de protocolos, não sendo necessário seguir a ordem existente na pilha.

O Scapy possui, ainda, outras funcionalidades como *sniffer* de rede, verificação de rotas, realização de ação maliciosa, dentre outras. A diferença do Scapy para outras ferramentas existentes é que ele não interpreta os dados recebidos, deixando essa responsabilidade para o usuário do *framework*.

A execução do Scapy possibilita a utilização do interpretador Python, permitindo o uso dos recursos dessa linguagem de programação. Desta forma, é possível combinar as funcionalidades da ferramenta como a manipulação e a captura de pacotes com o objetivo de realizar testes específicos.

¹Hping - Active Network Security Tool. <http://www.hping.org/>

²Nemesis packet injection tool suite. <http://nemesis.sourceforge.net/>

³Pacgen: Where do you want to send packets today. <http://pacgen.sourceforge.net/>

⁴Scapy. <http://www.secdev.org/projects/scapy/>

3.2 Arquitetura do IndPM

O **IndPM** foi desenvolvido com o objetivo de ser um manipulador de pacotes para protocolos industriais baseados em TCP/IP. A falta desse tipo de ferramenta foi a principal motivação para o desenvolvimento do trabalho. Um protótipo do **IndPM** foi a primeira extensão do Scapy desenvolvida para protocolos industriais [Kobayashi et al. 2007].

A ferramenta possui, atualmente, os seguintes módulos: Scapy, Módulo Modbus/TCP, Módulo DNP3 sobre TCP e a Interface Gráfica. Os demais módulos de protocolos industriais podem ser desenvolvidos de maneira semelhante ao Módulo Modbus/TCP, podendo ser incorporados ao **IndPM**. Um módulo para o protocolo DNP3 sobre TCP foi desenvolvido e está descrito no Apêndice A.

Na Figura 3.1, ilustra-se a arquitetura do **IndPM**, apresentando a interligação entre os módulos e a Interface Gráfica, bem como a dependência com o Scapy.

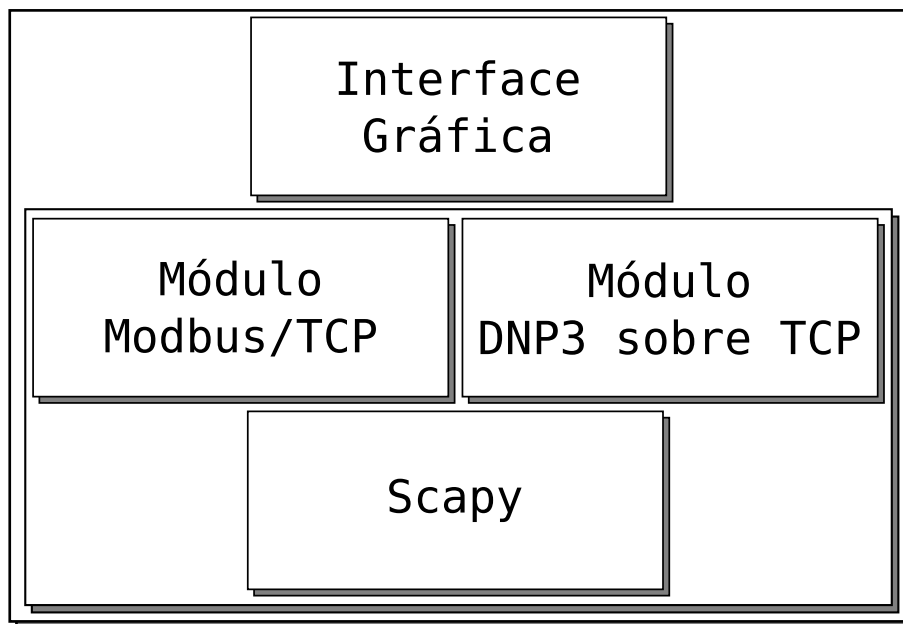


Figura 3.1: Arquitetura do **IndPM**.

As implementações dos protocolos foram desenvolvidas em Python para que pudessem ser utilizadas em conjunto com o Scapy. Os módulos dos protocolos foram desenvolvidos de forma independente e podem ser executados sem uso da Interface Gráfica, que foi desenvolvida em GTK (*GIMP ToolKit*)⁵. A interface gráfica desenvolvida serve para visualizar e executar a manipulação de pacotes utilizando os módulos dos protocolos.

Nas subseções a seguir, alguns detalhes da implementação desta arquitetura são apresentados.

⁵GTK+ project. <http://www.gtk.org/>

3.2.1 Módulo Modbus/TCP

O protocolo Modbus/TCP foi implementado com base na especificação apresentada na Seção 2.1. Foi desenvolvido um módulo que é executado em conjunto com o Scapy, capaz de manipular pacotes do protocolo Modbus/TCP de acordo com os parâmetros que o usuário especifica. Com este módulo, foi possível fazer análises de dispositivos que implementam o Modbus/TCP [Kobayashi et al. 2007].

Este módulo foi desenvolvido em Python com os conceitos de orientação a objetos, onde algumas classes criadas herdam de classes existentes no Scapy. As principais classes que o **IndPM** herda do Scapy são: **StrField** e **Packet**. A primeira foi usada para criação de classes que definem alguns campos do Modbus/TCP, já a segunda foi utilizada para construir os pacotes do protocolo.

Para cada código de função do Modbus foi definido uma classe que corresponde a um pacote Modbus/TCP, visto que os campos existentes na área de dados do pacote dependem do código de função utilizado. Todas as classes que definem um código de função do Modbus herdam de uma classe, denominada **ModbusPDU**. Esta classe determina os campos existentes em qualquer pacote do protocolo Modbus/TCP, são eles: **Identificador de Transação**, **Identificador de Protocolo**, **Tamanho**, **Identificador de Unidade** e **Função**. O campo **Dados** é o campo que varia de acordo com o código de função desejado, não sendo incluído na classe **ModbusPDU**.

Um exemplo da utilização do Módulo Modbus/TCP é apresentado na Figura 3.2, onde são manipulados pacotes com código de exceção igual a 1. Para este código de função, foi definida uma classe denominada **ReadCoilRequest** que quando instanciada, cria um objeto representando um pacote Modbus/TCP contendo o respectivo código de função. Desta maneira é possível manipular o conteúdo do campo **Dados**, definindo dentro do atributo **data**, os valores para os dois campos existentes nesse código de função (*address* e *count*).

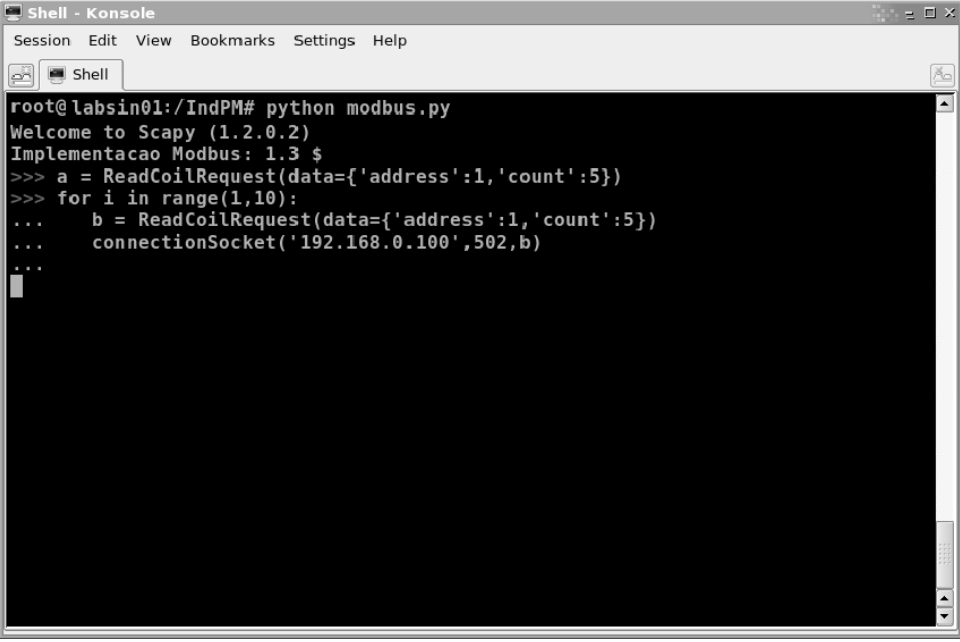
Neste exemplo, na Figura 3.2, é apresentado também a vantagem do Scapy em utilizar a linguagem de programação Python para criação dos testes. Neste exemplo foi criado, dentro de um laço, nove pacotes com código de função igual a 1 e enviados ao dispositivo alvo. Como o Módulo Modbus/TCP desenvolvido é utilizado em conjunto com o Scapy, ele também permite essa utilização dos recursos do Python para a criação de testes.

3.2.2 Interface Gráfica

A Interface Gráfica foi desenvolvida para facilitar a manipulação de pacotes através da utilização dos módulos que implementam os protocolos. Com a Interface Gráfica, o usuário pode modificar os valores para cada campo existente nos pacotes do Modbus/TCP presentes como botões na interface. Este procedimento elimina a necessidade do usuário conhecer alguns detalhes das especificações dos protocolos e dos módulos desenvolvidos.

A interface foi desenvolvida através do Glade⁶, que é um *software* usado para auxiliar o desenvolvimento de interfaces gráficas utilizando GTK. O Glade possui vários objetos prontos como botões, janelas, elementos de agrupamento dentre outros que facilitam o

⁶Glade - a User Interface Designer for GTK+ and GNOME. <http://glade.gnome.org/>



```
root@labsin01:/IndPM# python modbus.py
Welcome to Scapy (1.2.0.2)
Implementacao Modbus: 1.3 $
>>> a = ReadCoilRequest(data={'address':1,'count':5})
>>> for i in range(1,10):
...     b = ReadCoilRequest(data={'address':1,'count':5})
...     connectionSocket('192.168.0.100',502,b)
... 
```

Figura 3.2: Exemplo de utilização do Módulo Modbus/TCP do **IndPM**.

desenvolvimento de interfaces gráficas. Além disso, gera o código fonte da interface num arquivo XML (*eXtensible Markup Language*) que pode ser interpretado por várias linguagens de programação, dentre elas o Python que utilizamos para implementação dos outros módulos.

O módulo da Interface Gráfica é composto de dois arquivos: o arquivo em XML gerado pelo Glade e o arquivo em Python contendo toda a programação da interface. O arquivo XML contém as definições e os eventos de botões, de textos, de janelas, dentre outros componentes presentes na interface. Já o arquivo em Python é responsável por interligar os módulos dos protocolos com a interface, possuindo todas as definições de eventos dos componentes existentes na mesma.

A interface gráfica do **IndPM** possui, atualmente, duas interfaces para o usuário, uma para o protocolo Modbus/TCP e outra para o protocolo DNP3 sobre TCP (ver Apêndice A). A interface para o Modbus/TCP é mostrada na Figura 3.3. Ela contém botões do tipo *SpinButton* representando os possíveis valores de cada campo do protocolo de acordo com a especificação do Modbus/TCP.

Alguns botões estão agrupados com o objetivo de aumentar a facilidade de uso da ferramenta. Como pode ser visto na Figura 3.3, existem alguns botões representando o cabeçalho de um pacote Modbus/TCP. Além do cabeçalho, existe outro grupo para definir o conteúdo do pacote Modbus/TCP. Este grupo é composto pelo campo que contém o código de função e os dados do pacote. Dependendo do valor do campo de código de função, o campo de dados é modificado sendo visualizados os campos existentes para o código de função selecionado.

Existe ainda na interface do Modbus/TCP do **IndPM**, uma área de texto para indicar

The screenshot shows the 'IndPM.py' application window with the 'Modbus' tab selected. The interface is divided into several sections:

- Modbus DNP3**: A sub-tab header.
- Cabeçalho Modbus**: A section with four input fields: 'Identificador de Transação' (0), 'Identificador de Protocolo' (0), 'Tamanho' (0), and 'Identificador de Unidade' (0).
- Configuração TCP/IP**: A section with 'IP' (192.168.0.100) and 'TCP' (502) fields, and an 'Enviar' button.
- Modbus Request**: A section with three input fields: 'Código de Função' (1), 'Endereço' (1), and 'Quantidade de Registradores' (1).
- Pacote em Hexadecimal**: A text area displaying the hexadecimal representation of the packet: '00 00 00 00 00 06 00 01 00 01 00 01'.
- Estado da Conexão**: A text area displaying the connection status: 'Received 1 packets, got 1 answers, remaining 0 packets'.
- Radio Buttons**: At the bottom, there are three radio buttons: 'Requisição Modbus' (selected), 'Resposta Modbus', and 'Exceção Modbus'.

Figura 3.3: Tela da interface da manipulação de pacotes do protocolo Modbus/TCP.

o pacote Modbus/TCP que está sendo criado. Esta área mostra o pacote no formato hexadecimal. Há também outra área de texto para indicar o estado da conexão, mostrando a saída gerada pelo Scapy. Caso ocorra erros de conexão é possível identificá-los nesta área.

A Interface Gráfica proporciona maior facilidade de uso para o usuário do **IndPM**, sendo necessário apenas a manipulação dos campos apresentados pela interface. Isto abstrai o conhecimento prévio dos protocolos industriais e da implementação deles.

No Capítulo seguinte, detalhes de testes realizados com o **IndPM** são apresentados.

Capítulo 4

Testes Realizados

Neste Capítulo, serão apresentados resultados de testes realizados com a utilização da ferramenta **IndPM**. Inicialmente, serão apresentados os ambientes utilizados para a realização desses testes. Em seguida, será apresentada a metodologia utilizada nos testes e os resultados obtidos. Todos os testes apresentados neste Capítulo foram realizados no Laboratório de Segurança da Informação, LabSIN, do Departamento de Engenharia de Computação e Automação da Universidade Federal do Rio Grande do Norte.

4.1 Ambientes de Testes

Em nossos testes, foram utilizados dois dispositivos diferentes como servidor Modbus/TCP: um simulador e um módulo de comunicação Modbus/TCP conectado a um CLP. Nestas duas situações, foram montados ambientes de testes semelhantes, interligando o manipulador de pacotes e os servidores Modbus/TCP através de um *switch*. A Figura 4.1 mostra os dois ambientes de testes. O **IndPM** foi utilizado para manipular e enviar pacotes para o módulo de comunicação Modbus/TCP do CLP e para o simulador.

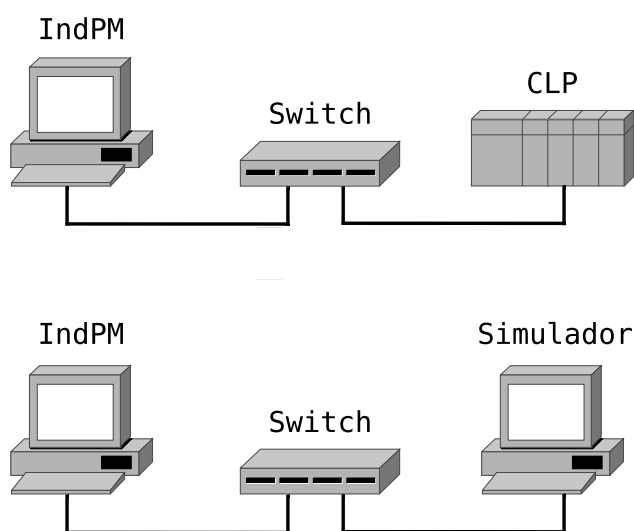


Figura 4.1: Ambientes de testes utilizados.

O servidor Modbus/TCP utilizado como simulador foi o **TCPSlaveTest**, uma das ferramentas do **jamod**¹. O **jamod** é uma biblioteca desenvolvida em Java para o protocolo Modbus, que permite o desenvolvimento de aplicações utilizando o protocolo. Nos testes realizados, o **TCPSlaveTest** foi utilizado em sua configuração padrão.

O CLP utilizado é comumente encontrado em ambientes de automação, com módulo de comunicação Modbus/TCP específico para este modelo de CLP. O módulo de comunicação Modbus/TCP atua como servidor, recebendo e respondendo às requisições realizadas pelo **IndPM**.

4.2 Metodologia e Resultados

Os testes foram realizados manipulando pacotes com todas as funções do Modbus implementadas pelo **IndPM**. Estes pacotes foram enviados para os dois dispositivos e suas respectivas respostas foram analisadas. Essas análises foram separadas em duas categorias, testes de conformidades e teste de segurança, que são apresentadas nas subseções seguintes.

4.2.1 Testes de Conformidades

Cada função possui uma resposta padrão descrita na especificação do protocolo Modbus [Modbus-IDA 2006a]. Para os dois casos, o módulo e o simulador, algumas respostas são diferentes entre si, e em outros casos, as respostas são diferentes da especificação do protocolo. Os resultados que são apresentados referem-se às respostas em discordância com a especificação do protocolo.

Um dos testes foi realizado usando pacotes com código de função igual a 1. Esta função é utilizada para leitura dos estados das saídas booleanas do dispositivo. Um pacote Modbus com este código de função possui duas informações no campo de dados: uma para representar o endereço inicial a ser lido e outra informando a quantidade de saídas booleanas que deverão ser lidas. Cada uma destas informações tem uma escala de valores e tratamento de exceção indicados pela especificação [Modbus-IDA 2006a].

Os pacotes contendo o código de função igual a 1 foram respondidos pelo módulo de comunicação do CLP, em discordância com a especificação do Modbus, em dois casos. Um dos casos, foi o envio de um pacote contendo a quantidade de saídas booleanas a serem lidas igual a 0 e no outro caso quando esta quantidade foi um valor maior que 2000. De acordo com a especificação, estes pacotes deveriam ser respondidos com uma exceção (pacote contendo código de função igual a 129 e código de exceção igual a 3). Para o caso do pacote contendo a quantidade de saídas booleanas igual a 0, o módulo Modbus/TCP respondeu normalmente a este tipo de requisição. No outro caso, o módulo Modbus/TCP retornou uma exceção com o campo de código de exceção errado, com o valor 2 ao invés de 3.

No teste com o simulador de Modbus/TCP, estes mesmos pacotes foram enviados e as respostas recebidas também foram diferentes da especificação do protocolo. Nos

¹Java Modbus Library. <http://sourceforge.net/projects/jamod>

dois casos, o simulador retornou exceções com o campo do código de exceção igual a 2, quando deveria ser igual a 3. Nas Figuras 4.2 e 4.3, são apresentados diagramas que resumem estes testes.

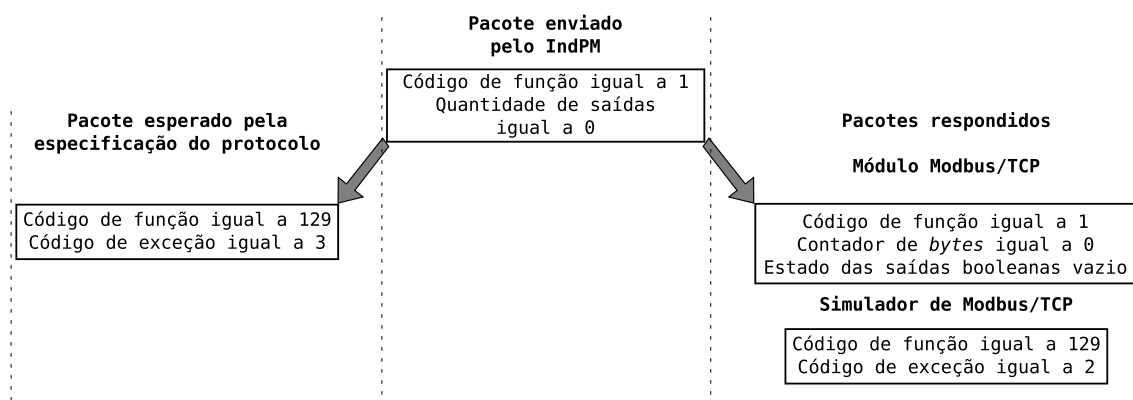


Figura 4.2: Teste realizado com pacote contendo o código de função igual a 1 e quantidade de registradores igual a 0.

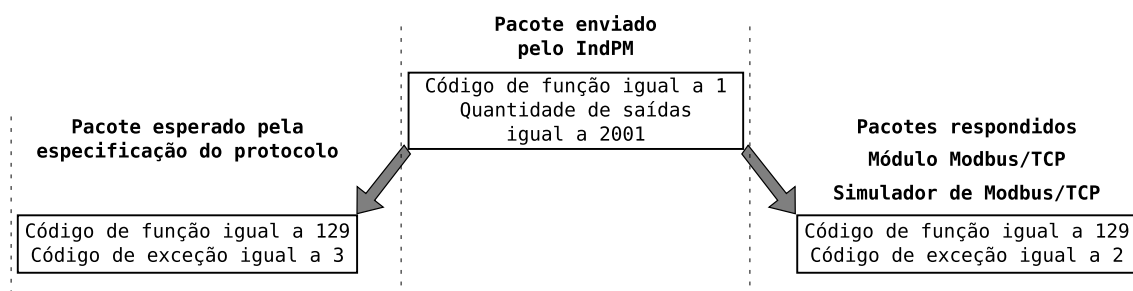


Figura 4.3: Teste realizado com pacote contendo o código de função igual a 1 e quantidade de registradores maior que 2000.

Além de pacotes com código de função igual a 1, existem outros casos em que o módulo Modbus/TCP deveria responder determinadas requisições com uma exceção mas responde normalmente. O simulador de Modbus/TCP, em alguns desses casos, respondeu com código de exceção errado ou não implementava a função solicitada.

Os pacotes contendo códigos de função iguais a 2, a 3 e a 4, com a quantidade de registradores a serem lidos iguais a 0, deveriam ser respondidos com um pacote de exceção e com código de exceção igual a 3. Para estes casos, o módulo Modbus/TCP respondeu normalmente com um pacote contendo os estados dos registradores vazio. Nesses casos, o simulador de Modbus/TCP respondeu com código de exceção errado, 2 ao invés de 3. Os diagramas que ilustram esses testes estão apresentados nas Figuras 4.4, 4.5 e 4.6.

Os testes mostram que, além dos pacotes com códigos de função de leitura, alguns pacotes contendo códigos de função de escrita, 15 e 16, também apresentaram discordância com a especificação. Os pacotes, contendo código de função igual a 15 e a quantidade de

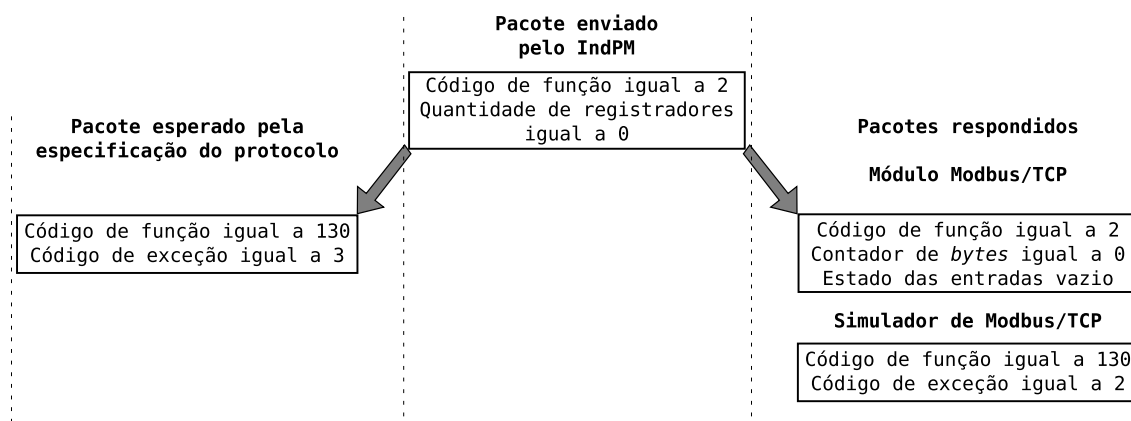


Figura 4.4: Teste realizado com pacote contendo o código de função igual a 2 e quantidade de registradores igual a 0.

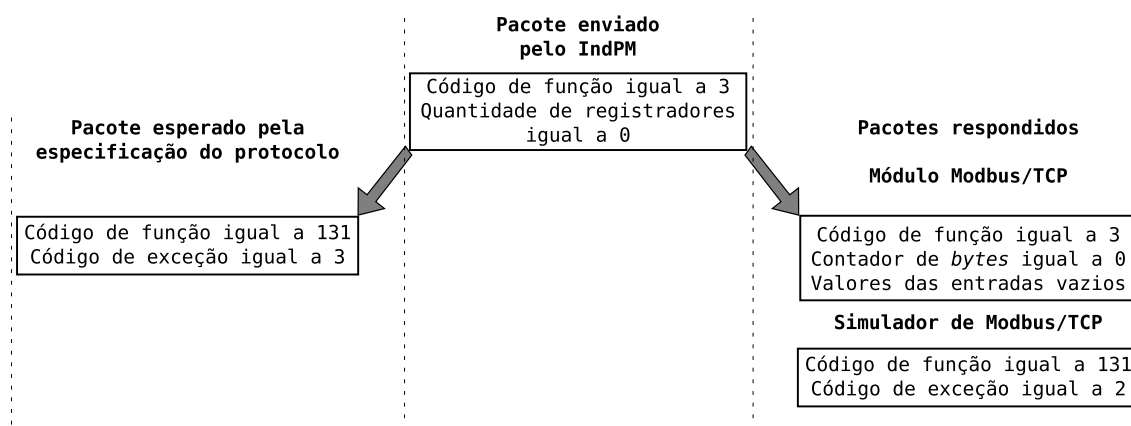


Figura 4.5: Teste realizado com pacote contendo o código de função igual a 3 e quantidade de registradores igual a 0.

saídas que se deseja modificar o estado iguais a 0, foram respondidos normalmente pelo módulo Modbus/TCP quando deveriam ser respondidos com uma exceção com código igual a 3. Nesse caso, o simulador de Modbus/TCP respondeu com código de exceção errado, da mesma maneira que os códigos de função de leitura. Na Figura 4.7, está representado este teste.

Para os pacotes com código de função igual a 16, o módulo do Modbus/TCP respondeu normalmente aos pacotes contendo a quantidade de registradores, no qual se especifica a modificação dos seus estados, superior ao limite definido pelo protocolo. Os testes foram realizados com quantidade de registradores igual a 121. Nestes casos, a especificação define o limite de registradores a serem modificados, com código de função 16, entre 0 e 120. Pacotes com valores fora desse limite deveriam ser respondidos com uma exceção e código de exceção igual a 3. Os pacotes com código de função igual a 16 não foram respondidos pelo simulador de Modbus/TCP. O diagrama que ilustra esse teste está

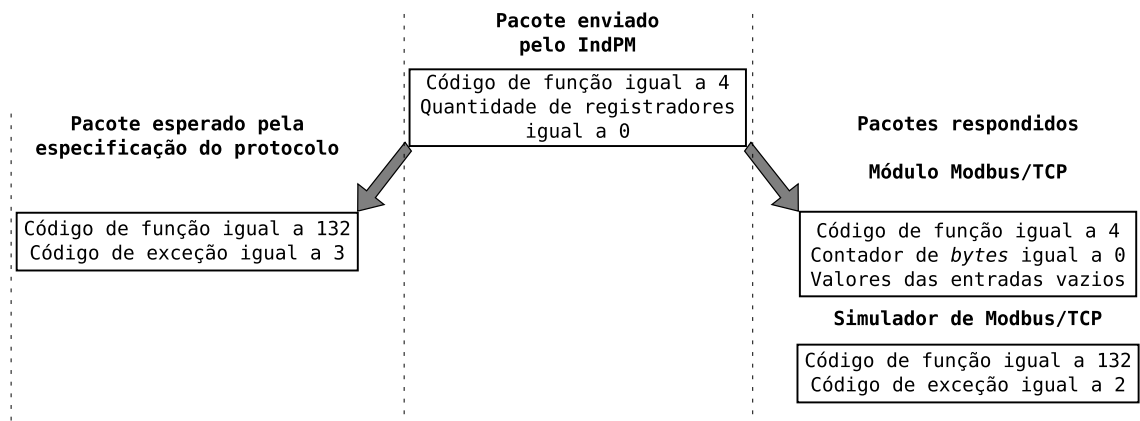


Figura 4.6: Teste realizado com pacote contendo o código de função igual a 4 e quantidade de registradores igual a 0.

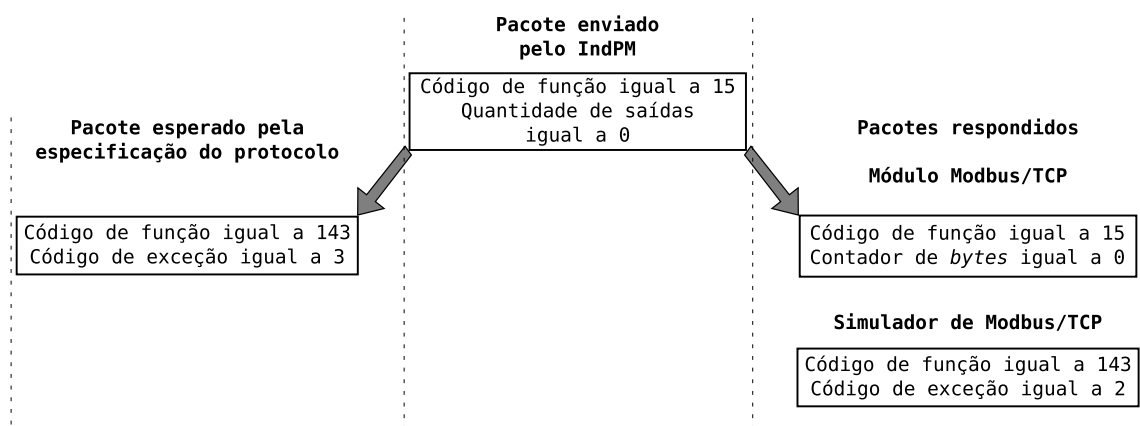


Figura 4.7: Teste realizado com pacote contendo o código de função igual a 15 e quantidade de registradores igual a 0.

representado na Figura 4.8.

Outros pacotes que foram respondidos incorretamente pelo módulo Modbus/TCP foram os pacotes contendo códigos de função iguais a 23. Estes pacotes são utilizados para ler e modificar os valores dos registradores numa única transação. Este tipo de pacote não foi respondido pelo simulador de Modbus/TCP. Para pacotes contendo o código de função 23, o módulo do CLP respondeu normalmente a requisições contendo a quantidade de registradores a serem lidos ou modificados iguais a 0, quando deveria responder estas requisições com exceção contendo o código de exceção igual a 3. Nas Figuras 4.9 e 4.10, são apresentados esses testes.

Os dispositivos testados, módulo Modbus/TCP e simulador de Modbus/TCP, apresentaram outras situações além das apresentadas nas Figuras 4.2 e 4.3, em que esses dispositivos responderam com código de exceção diferente do esperado. Essas situações ocorreram para os pacotes contendo códigos de função iguais a 2, a 3 e a 4, contendo a

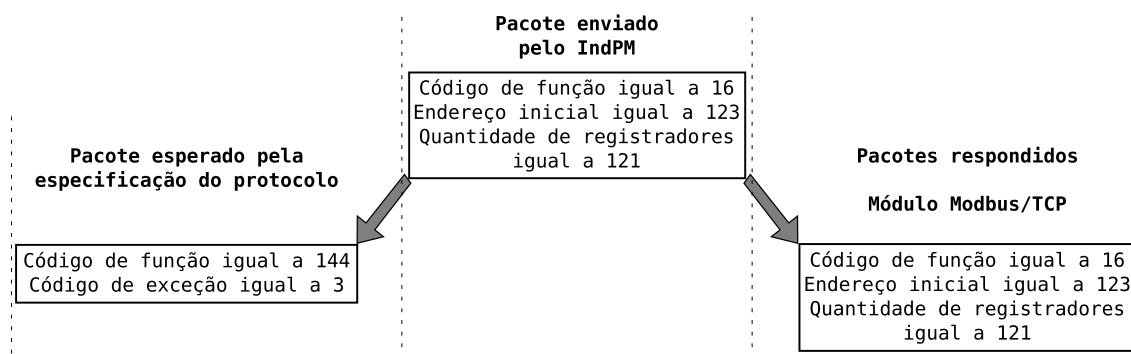


Figura 4.8: Teste realizado com pacote contendo o código de função igual a 16 e quantidade de registradores maior que 120.

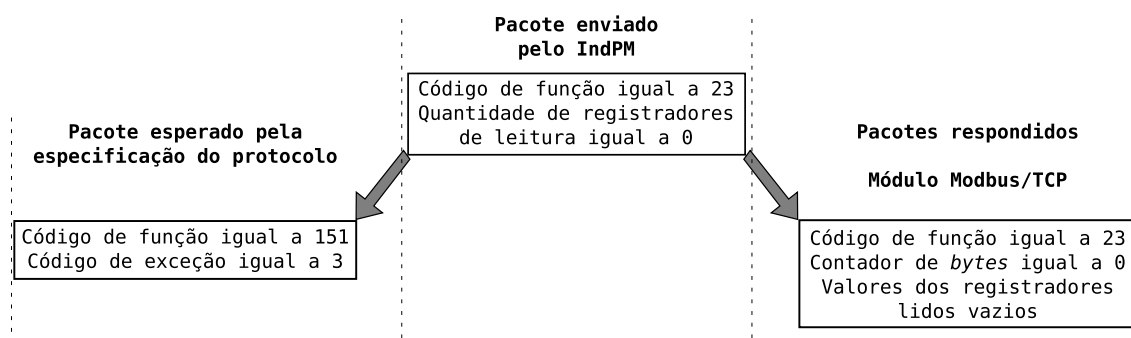


Figura 4.9: Teste realizado com pacote contendo o código de função igual a 23 e quantidade de registradores leitura igual a 0.

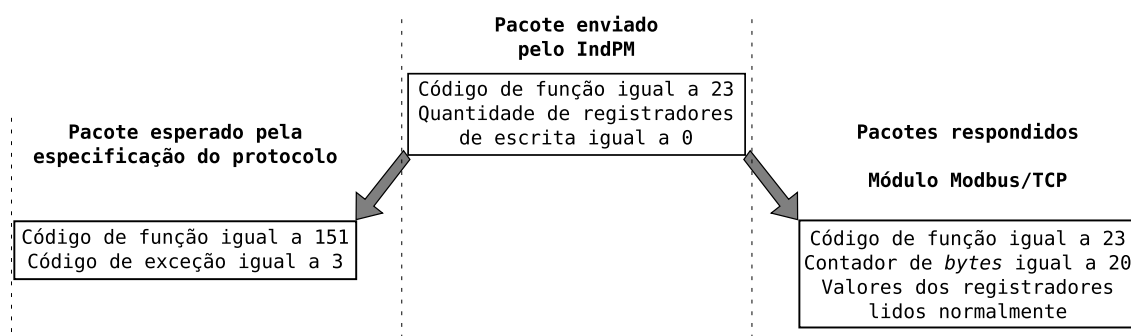


Figura 4.10: Teste realizado com pacote contendo o código de função igual a 23 e quantidade de registradores de escrita igual a 0.

quantidade de registradores a serem lidos fora do limite definido pela especificação.

No caso dos pacotes com código de função igual a 2, o limite especificado pelo protocolo para a quantidade máxima de registradores a serem lidos é de 2000 registradores. Os testes realizados com **IndPM** solicitando a leitura de 2001 registradores foram respondi-

dos incorretamente pelos dispositivos que, ao invés de responderem com um pacote de exceção contendo código de exceção igual a 3, responderam com um pacote de exceção com código de exceção igual a 2. Esse teste está representado na Figura 4.11.

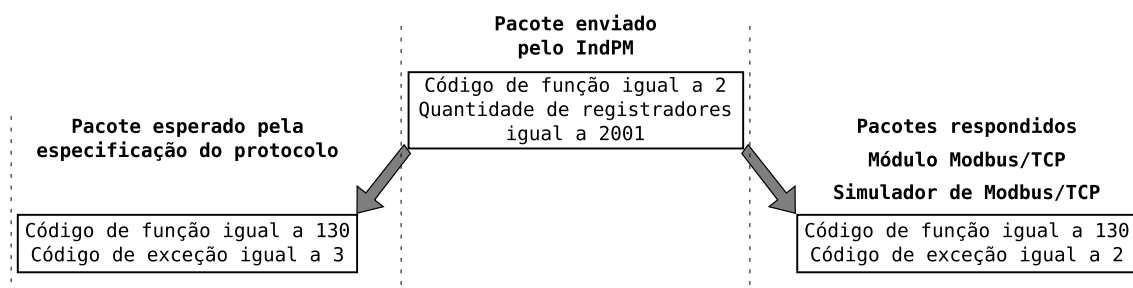


Figura 4.11: Teste realizado com pacote contendo o código de função igual a 2 e quantidade de registradores maior que 2000.

Esta mesma situação ocorreu com pacotes contendo os códigos de função iguais a 3 e a 4, só que nesses casos o limite definido para a quantidade de registradores a serem lidos é de 125. Os testes foram realizados definindo pacotes com essa quantidade igual a 126. Esses testes de manipulação incorreta das exceções por parte dos dispositivos estão ilustrados nas Figuras 4.12 e 4.13.

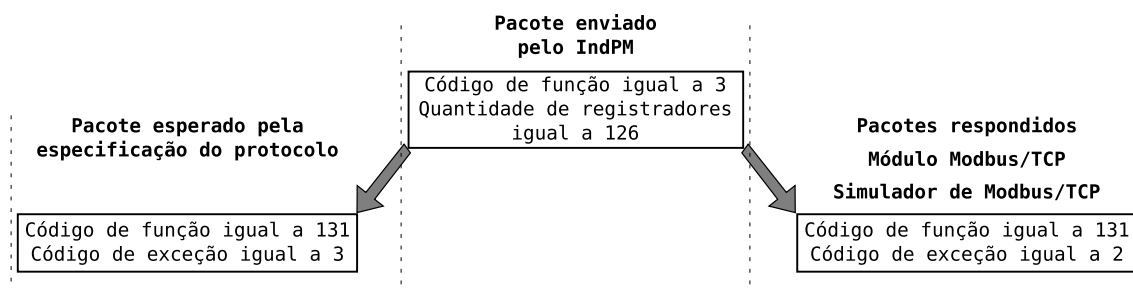


Figura 4.12: Teste realizado com pacote contendo o código de função igual a 3 e quantidade de registradores maior que 125.

Os testes realizados mostram, ainda, que os dispositivos testados responderam com exceção a requisições que deveriam ser respondidas normalmente de acordo com o protocolo. Nas Figuras 4.14 e 4.15, são apresentados diagramas que mostram os pacotes com códigos de função de leitura (3 e 4) que foram respondidos com exceção, quando deveriam ser respondidos normalmente. Esses pacotes foram manipulados para que a soma do endereço inicial dos registradores a serem lidos mais a quantidade desses registradores fosse um valor maior que 10.000. Nesses casos, o módulo Modbus/TCP e o simulador de Modbus/TCP responderam com uma exceção com código de exceção igual a 2. Estes tipos de pacotes deveriam ser respondidos com os valores dos registradores que foram requisitados.

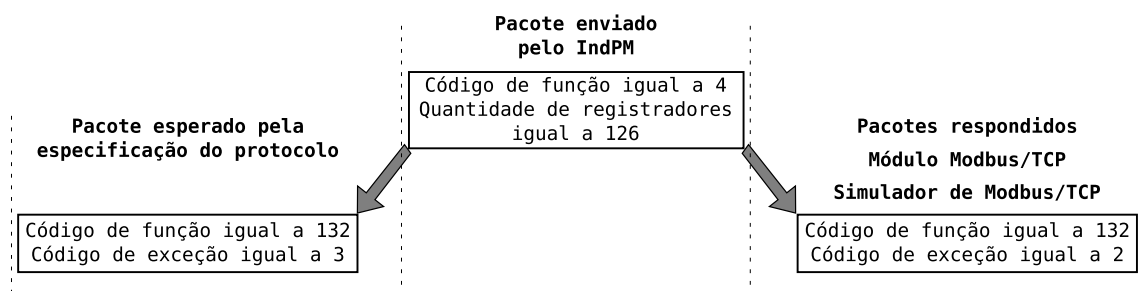


Figura 4.13: Teste realizado com pacote contendo o código de função igual a 4 e quantidade de registradores maior que 125.

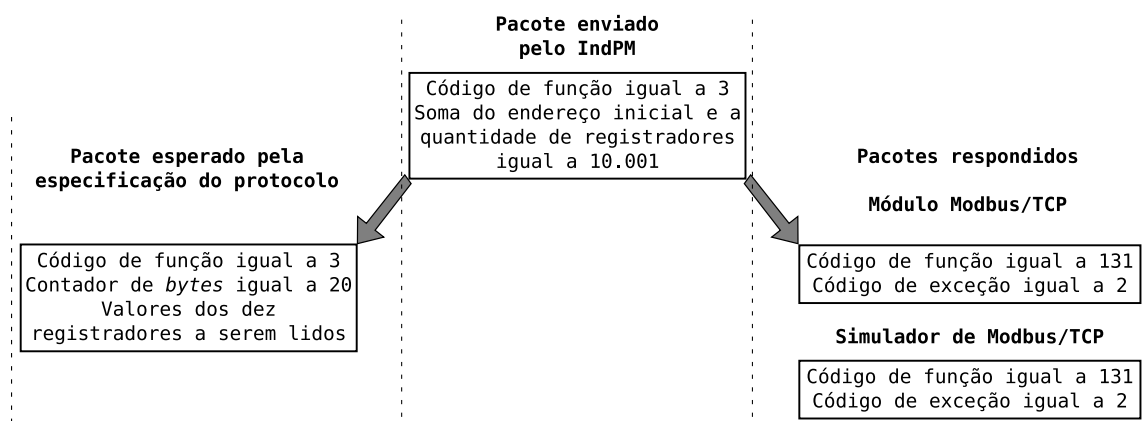


Figura 4.14: Teste realizado com pacote contendo o código de função igual a 3 e a soma do endereço inicial e a quantidade de registradores maior que 10.000.

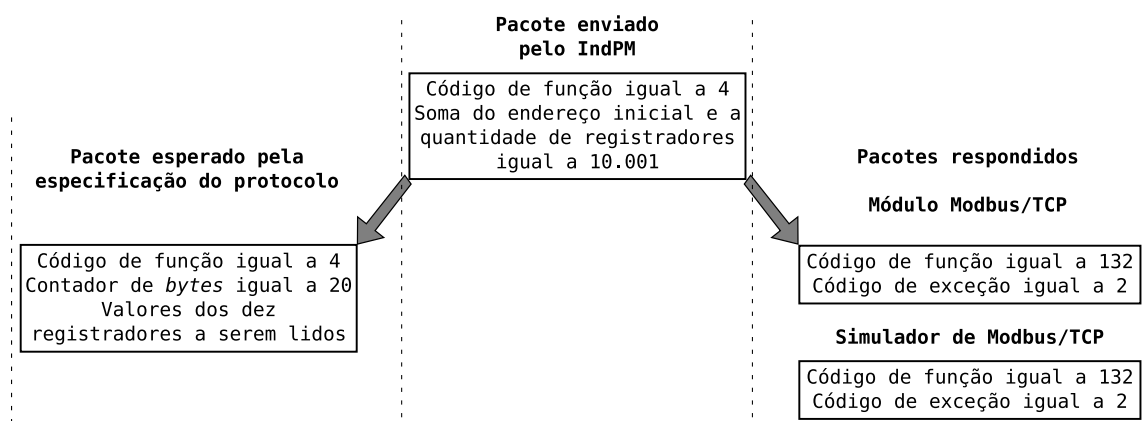


Figura 4.15: Teste realizado com pacote contendo o código de função igual a 4 e a soma do endereço inicial e a quantidade de registradores maior que 10.000.

Alguns pacotes com códigos de função de escrita também apresentaram discordância com a especificação. Nas Figuras 4.16, 4.17 e 4.18, são apresentados os pacotes com códigos de função iguais a 6, a 15 e a 16, respectivamente, que apresentaram essas discordâncias. O pacote contendo código de função igual a 6 e com o endereço inicial dos registradores a serem modificados igual a 10.001, foi respondido com um pacote de exceção contendo código de exceção igual a 2 tanto pelo módulo Modbus/TCP quanto pelo simulador de Modbus/TCP. Estes pacotes deveriam ser respondidos com uma resposta contendo o mesmo conteúdo da requisição.

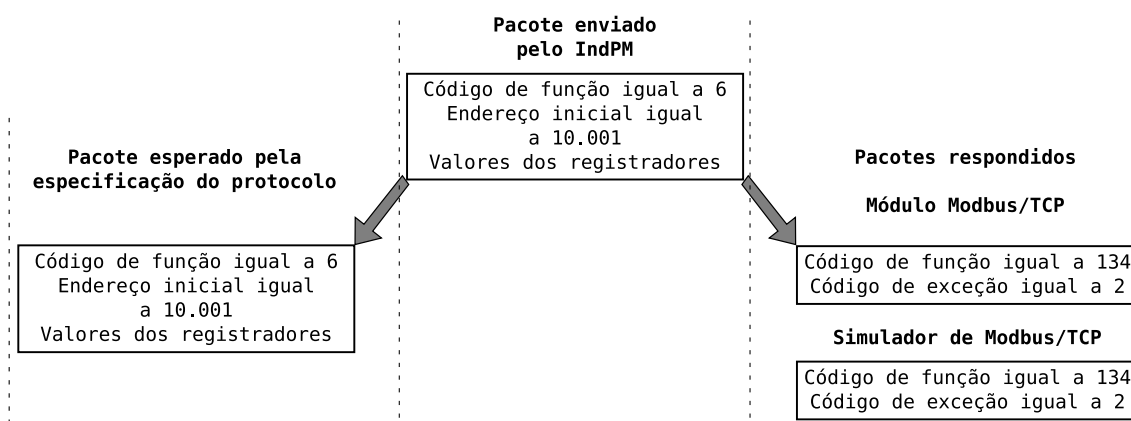


Figura 4.16: Teste realizado com pacote contendo o código de função igual a 6 e o endereço inicial maior que 10.000.

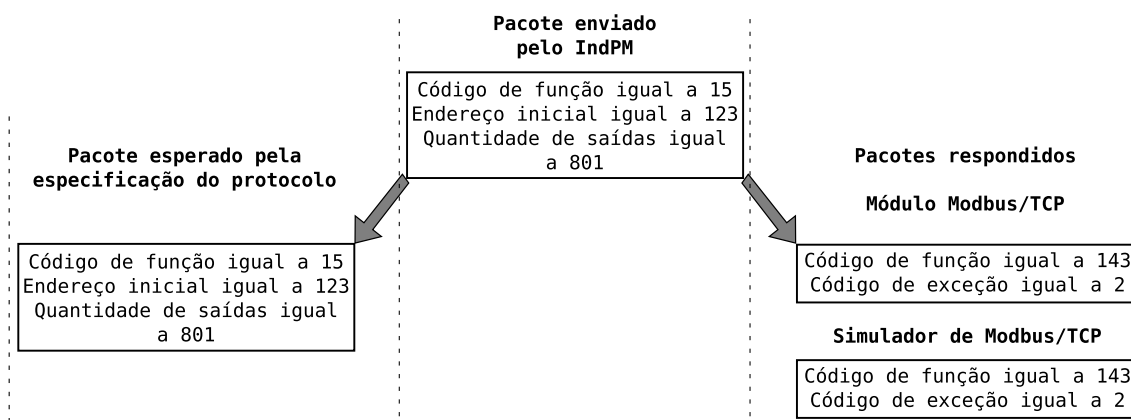


Figura 4.17: Teste realizado com pacote contendo o código de função igual a 15 e a quantidade de saídas maior que 800.

Para os pacotes com código de função igual a 15, as respostas foram obtidas incorretamente quando a quantidade de saídas a serem modificados tinha valor maior que 800. Esses pacotes deveriam ser respondidos normalmente. Nesses casos, as requisições foram

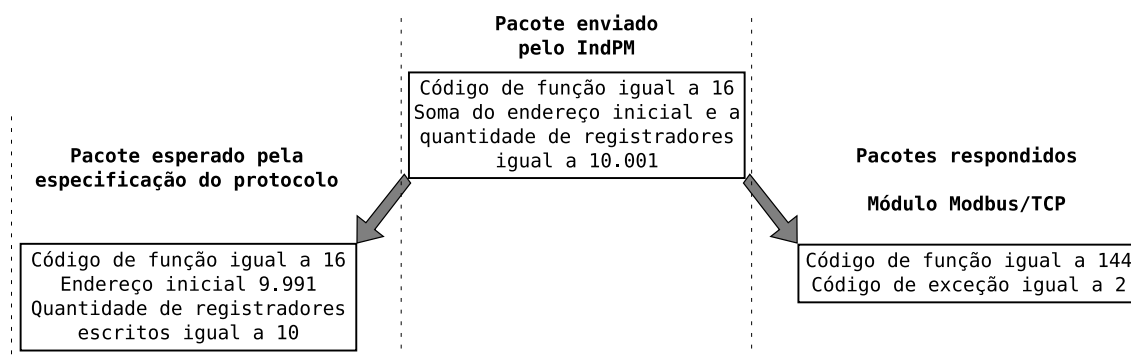


Figura 4.18: Teste realizado com pacote contendo o código de função igual a 16 e a soma do endereço inicial e a quantidade de registradores maior que 10.000.

respondidas, pelo módulo Modbus/TCP e pelo simulador de Modbus/TCP, com exceções contendo código de exceção igual a 2.

Os pacotes, com código de função igual a 16, foram manipulados para que a soma, do endereço inicial dos registradores a serem modificados juntamente com a quantidade desses registradores, tivessem um valor maior que 10.000. Esses pacotes são respondidos com uma exceção pelo módulo Modbus/TCP, já o simulador de Modbus/TCP não responde a essas requisições. Nesses casos, esses pacotes deveriam ser respondidos normalmente com o mesmo conteúdo dessas requisições.

Os testes realizados com pacotes contendo código de função igual a 22 mostram que o módulo Modbus/TCP também responde incorretamente a essas requisições. Este código de função é utilizado para modificar os valores dos registradores através da combinação de operações lógicas. Os testes que apresentam discordâncias do protocolo são os pacotes que possuem endereços a serem modificados maiores que 10.000. As requisições enviadas nos testes possuíam valores iguais a 10.001 nesse campo e deveriam ser respondidas normalmente com um pacote de mesmo conteúdo. Nesse caso, o módulo Modbus/TCP responde com uma exceção com código de exceção igual a 2 e o simulador de Modbus/TCP não responde a pacotes contendo este código de função. Na Figura 4.19, apresenta-se o diagrama correspondente desse teste.

Na Figuras 4.20 e 4.21, apresentam-se, ainda, os testes realizados com pacotes contendo códigos de função iguais a 23. Para esses pacotes, o módulo Modbus/TCP respondeu com exceções a pacotes onde foram manipulados os campos que definem a leitura e a escrita. Quando a soma dos campos, que definem a leitura ou a escrita, tem um valor maior que 10.000, o módulo Modbus/TCP responde a essas requisições com uma exceção que tem código de exceção igual a 2. O simulador de Modbus/TCP não responde a estas requisições. Estas requisições deveriam ser respondidas normalmente com os valores solicitados.

Os testes de conformidades mostraram também que pacotes com códigos de função, que deveriam ser respondidos somente em dispositivos que utilizam canais de comunicação serial, foram respondidos normalmente pelo módulo Modbus/TCP. O simulador de Modbus/TCP testado não respondeu a esses tipos de pacotes.

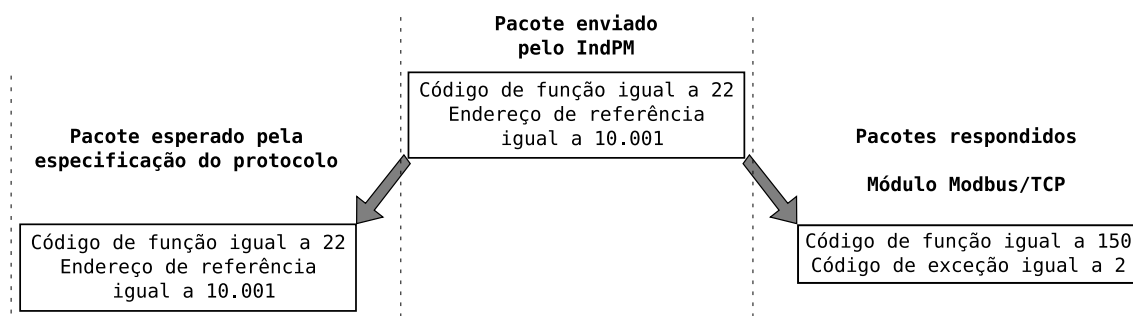


Figura 4.19: Teste realizado com pacote contendo o código de função igual a 22 e o endereço inicial maior que 10.000.

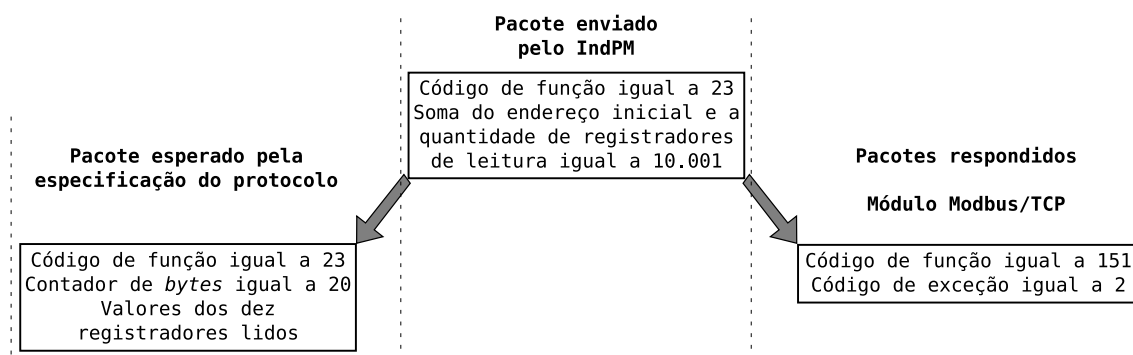


Figura 4.20: Teste realizado com pacote contendo o código de função igual a 23 e a soma do endereço inicial e a quantidade de registradores de leitura maior que 10.000.

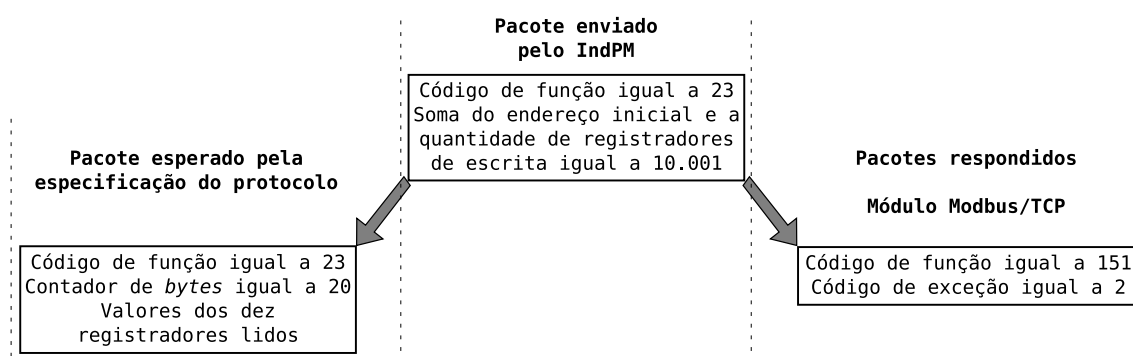


Figura 4.21: Teste realizado com pacote contendo o código de função igual a 23 e a soma do endereço inicial e a quantidade de registradores de escrita maior que 10.000.

Os testes apresentam que dois códigos de função, 7 e 8, reservados para comunicação serial foram respondidos normalmente. Os pacotes contendo código de função igual a 7 deveriam ser respondidos, por dispositivos que utilizam o Modbus/TCP, com uma exceção contendo o código de exceção igual a 1. Este código de exceção é utilizado para indicar

que o código de função utilizado não é permitido. Na Figura 4.22, é apresentado esse teste.

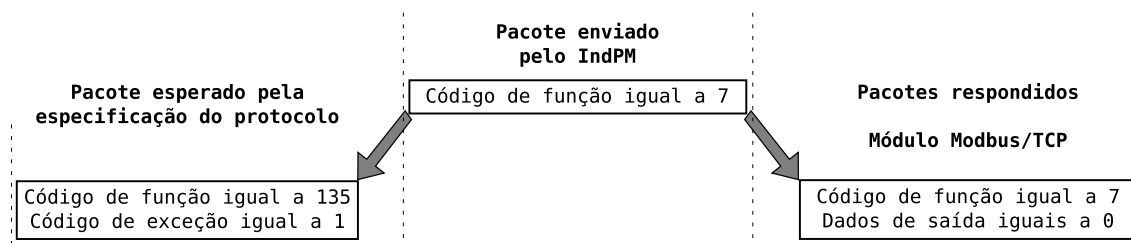


Figura 4.22: Teste realizado com pacote contendo o código de função igual a 7.

Os pacotes, contendo código de função igual a 8, possuem um campo que define uma sub-função. Este campo possui valores especificados pelo protocolo, que definem a funcionalidade do pacote. Nos testes realizados, os pacotes contendo nesse campo valores definidos, e também os não-definidos, deveriam ser respondidos com uma exceção. No caso dos valores definidos pela especificação, os pacotes deveriam ser respondidos com código de exceção igual a 1 pois é um código de função desenvolvido para comunicações seriais. Além do mais, quando se tem um valor do campo de sub-função fora da faixa de valores definidos pelo protocolo, o dispositivo também deveria responder com uma exceção com código de exceção igual a 1. O módulo Modbus/TCP respondeu normalmente a estas requisições, nos dois casos, com pacotes de mesmo conteúdo. Esses testes são apresentados nos diagramas das Figuras 4.23 e 4.24.

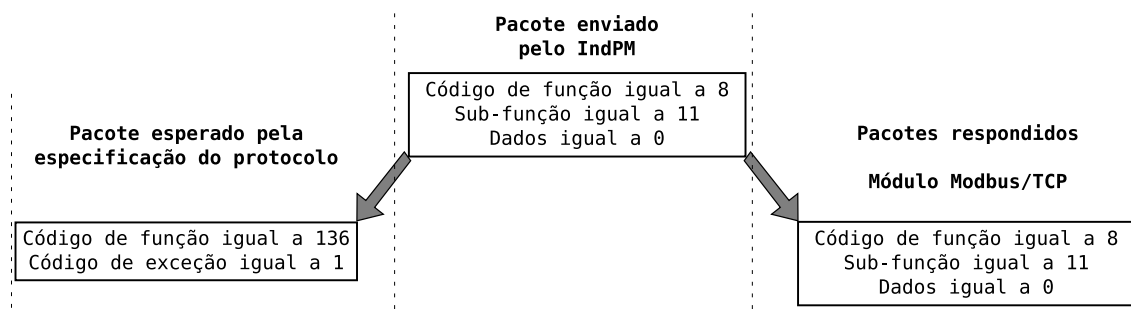


Figura 4.23: Teste realizado com pacote contendo o código de função igual a 8 e sub-função igual a 1.

As discordâncias entre a especificação e a implementação do protocolo nos dispositivos podem ocasionar problemas na interpretação dos dados. Este fato pode levar a tomada de decisões errôneas.

4.2.2 Teste de Segurança

Através do *software* de manipulação de pacotes Modbus também foi possível encontrar um grave problema no módulo Modbus/TCP do CLP testado. Os pacotes contendo

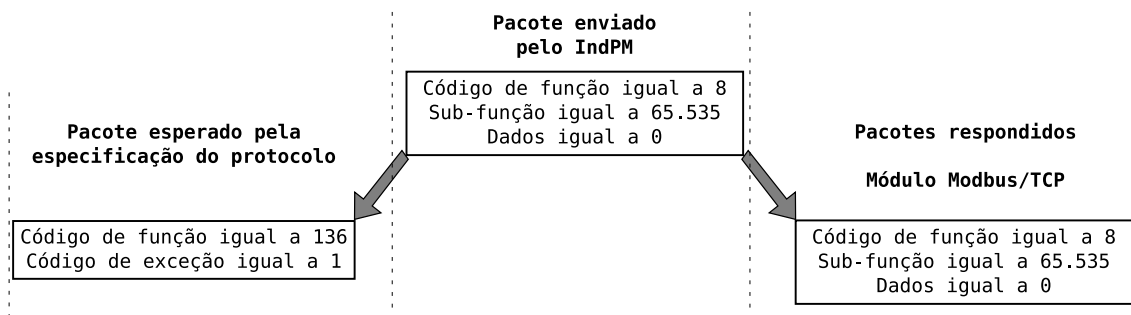


Figura 4.24: Teste realizado com pacote contendo o código de função igual a 8 e sub-função igual a 65.535.

valores específicos, dentro do campo **Tamanho** no cabeçalho, fazem com que o módulo Modbus/TCP não responda mais a requisições posteriores. A execução desta ação pode ser considerada um ataque de negação de serviço (DoS - *Denial of Service*). O módulo de comunicação do CLP deixa de fornecer o serviço Modbus/TCP quando submetido a esta situação, voltando a operar normalmente somente após a sua reinicialização.

Nas Figuras 4.25 e 4.26, os resultados dos testes que ocasionam o DoS no módulo Modbus/TCP são apresentados. Na Figura 4.25, é apresentado o tráfego entre o módulo Modbus/TCP e o **IndPM**, capturado utilizando o Wireshark². O valor que ocasiona o problema, em destaque na figura, não é apresentado por questões de segurança. Na Figura 4.26, mostra-se o momento em que ocorre a perda de comunicação do CLP, quando não responde mais a requisições de *ping* através do módulo Modbus/TCP.

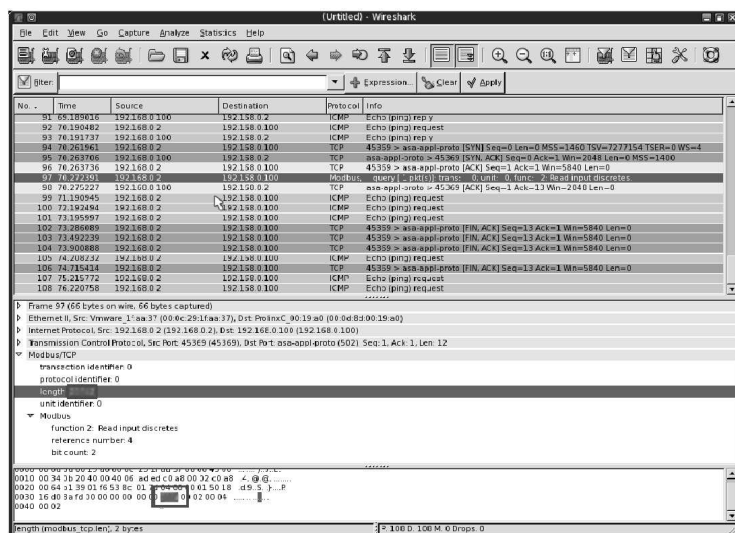
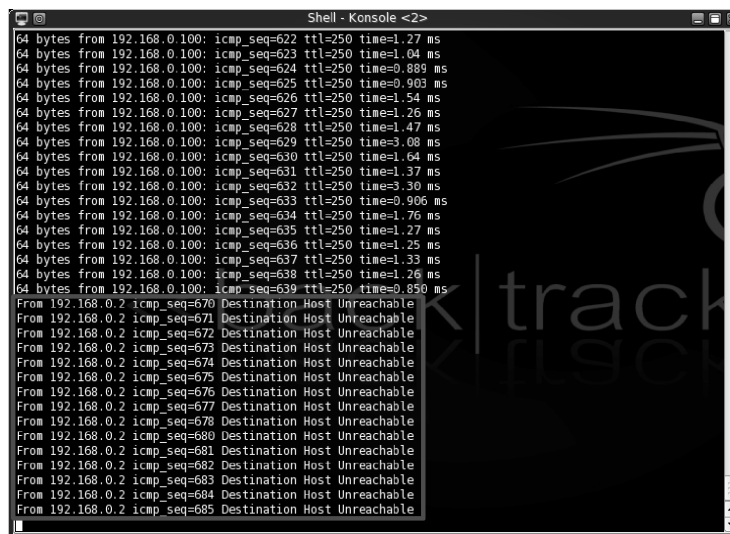


Figura 4.25: Tráfego entre o módulo do CLP e o **IndPM** capturado com Wireshark. O valor em destaque não é mostrado por motivos de segurança.

²Wireshark: Go deep. <http://www.wireshark.org/>



```
Shell - Konsole <2>
64 bytes from 192.168.0.100: icmp_seq=622 ttl=250 time=1.27 ms
64 bytes from 192.168.0.100: icmp_seq=623 ttl=250 time=1.04 ms
64 bytes from 192.168.0.100: icmp_seq=624 ttl=250 time=0.889 ms
64 bytes from 192.168.0.100: icmp_seq=625 ttl=250 time=0.903 ms
64 bytes from 192.168.0.100: icmp_seq=626 ttl=250 time=1.54 ms
64 bytes from 192.168.0.100: icmp_seq=627 ttl=250 time=1.26 ms
64 bytes from 192.168.0.100: icmp_seq=628 ttl=250 time=1.47 ms
64 bytes from 192.168.0.100: icmp_seq=629 ttl=250 time=3.08 ms
64 bytes from 192.168.0.100: icmp_seq=630 ttl=250 time=1.64 ms
64 bytes from 192.168.0.100: icmp_seq=631 ttl=250 time=1.37 ms
64 bytes from 192.168.0.100: icmp_seq=632 ttl=250 time=0.50 ms
64 bytes from 192.168.0.100: icmp_seq=633 ttl=250 time=0.906 ms
64 bytes from 192.168.0.100: icmp_seq=634 ttl=250 time=1.76 ms
64 bytes from 192.168.0.100: icmp_seq=635 ttl=250 time=1.27 ms
64 bytes from 192.168.0.100: icmp_seq=636 ttl=250 time=1.25 ms
64 bytes from 192.168.0.100: icmp_seq=637 ttl=250 time=1.33 ms
64 bytes from 192.168.0.100: icmp_seq=638 ttl=250 time=1.26 ms
64 bytes from 192.168.0.100: icmp_seq=639 ttl=250 time=0.850 ms
From 192.168.0.2 icmp_seq=670 Destination Host Unreachable
From 192.168.0.2 icmp_seq=671 Destination Host Unreachable
From 192.168.0.2 icmp_seq=672 Destination Host Unreachable
From 192.168.0.2 icmp_seq=673 Destination Host Unreachable
From 192.168.0.2 icmp_seq=674 Destination Host Unreachable
From 192.168.0.2 icmp_seq=675 Destination Host Unreachable
From 192.168.0.2 icmp_seq=676 Destination Host Unreachable
From 192.168.0.2 icmp_seq=677 Destination Host Unreachable
From 192.168.0.2 icmp_seq=678 Destination Host Unreachable
From 192.168.0.2 icmp_seq=680 Destination Host Unreachable
From 192.168.0.2 icmp_seq=681 Destination Host Unreachable
From 192.168.0.2 icmp_seq=682 Destination Host Unreachable
From 192.168.0.2 icmp_seq=683 Destination Host Unreachable
From 192.168.0.2 icmp_seq=684 Destination Host Unreachable
From 192.168.0.2 icmp_seq=685 Destination Host Unreachable
```

Figura 4.26: Monitoramento do módulo Modbus/TCP utilizando *ping*. Observar que o módulo deixa de responder ao comando no quadrado em destaque na figura.

No próximo Capítulo, são apresentadas as conclusões deste trabalho e são feitas algumas propostas de trabalhos futuros.

Capítulo 5

Conclusões

Os protocolos de comunicação baseados em TCP/IP utilizados nas redes de automação, em geral, são simples e desenvolvidos com pouca preocupação em relação aos aspectos de segurança. A análise dos protocolos utilizados pelos dispositivos pode mostrar falhas existentes, como foi apresentado. Esta análise pode auxiliar na definição dos mecanismos de segurança que se deve utilizar para cada dispositivo.

O **IndPM** é uma ferramenta que possibilita a análise de protocolos baseados em TCP/IP, utilizados pelos dispositivos em redes industriais. Essa ferramenta realiza testes de segurança e de conformidade de protocolos, de forma intrusiva na rede, através da manipulação e envio de pacotes.

Com a manipulação de pacotes utilizando o **IndPM**, foram obtidos resultados referentes à discordância entre a especificação do protocolo e o que foi efetivamente implementado pelos dispositivos. Essas respostas em discordância com a especificação do protocolo podem provocar uma ação errada no controle dos outros dispositivos da rede.

O **IndPM** permitiu também encontrar uma condição de negação de serviço. Este tipo de ataque necessita de atenção em dispositivos industriais, visto que são dispositivos importantes na comunicação dos sistemas de controle. Para o CLP testado faz-se necessário a utilização de ferramentas de segurança para detectar e impedir o recebimento indevido de pacotes que possam ocasionar a negação de serviço.

O trabalho desenvolvido mostra que a ferramenta **IndPM** pode ser utilizada num processo de certificação de segurança e de conformidades de protocolos industriais baseados em TCP/IP.

A validação do módulo de manipulação para pacotes DNP3 sobre TCP é uma proposta para trabalho futuro, além de testes em diferentes dispositivos tanto para o módulo DNP3 sobre TCP quanto o módulo Modbus/TCP. A implementação de outros protocolos baseados em TCP/IP, como o IEC 60870-5-104, também é proposta de continuidade deste trabalho.

Referências Bibliográficas

- Bailey, David & Edwin Wright (2003), *Practical SCADA for Industry*, 1ª edição, Newnes.
- Byres, Eric J., Dan Hoffman & Nate Kube (2006), ‘On Shaky Ground - A Study of Security Vulnerabilities in Control Protocols’, *5th American Nuclear Society International Topical Meeting on Nuclear Plant Instrumentation, Controls, and Human Machine Interface Technology*. .
- Clarke, Gordon, Deon Reynders & Edwin Wright (2004), *Practical Modern SCADA Protocols: DNP3, 60870.5 and Related Systems*, 1ª edição, Newnes.
- Grzelak, Daniel (2008), ‘SCADA Penetration Testing - Hacking Modbus Enabled Devices’, *RUXCON* .
- Kobayashi, Tiago Hiroshi, Aguinaldo Batista Bezerra Junior, Agostinho Medeiros Brito Junior & Paulo Sérgio Motta Pires (2007), ‘Using a Packet Manipulation Tool for Security Analysis of Industrial Network Protocols’, *IEEE Conference on Emerging Technologies and Factory Automation, 2007. ETFA*. pp. 744–747.
- Lobashov, Maxim & Thilo Sauter (2006), ‘Vertical Communication from the Enterprise Level to the Factory Floor - Integrating Fieldbus and IP-based Networks’, *IEEE Conference on Emerging Technologies and Factory Automation, 2006. ETFA '06*. pp. 1214–1221.
- Mander, Todd, Farhad Nabhami, Lin Wang & Richard Cheung (2007), ‘Data Object Based Security for DNP3 Over TCP/IP for Increased Utility Commercial Aspects Security’, *IEEE Power Engineering Society General Meeting 2007*. pp. 1–8.
- Modbus-IDA (2006a), *Modbus Application Protocol Specification V1.1b*.
URL: <http://www.Modbus-IDA.org>, acessado em julho de 2009
- Modbus-IDA (2006b), *MODBUS Messaging on TCP/IP Implementation Guide V1.0b*.
URL: <http://www.Modbus-IDA.org>, acessado em julho de 2009
- Pires, Paulo Sérgio Motta & Luiz Affonso H Guedes Oliveira (2006), ‘Security Aspects of SCADA and Corporate Network Interconnection: An Overview’, *International Conference on Dependability of Computer Systems, 2006. DepCos-RELCOMEX '06*. pp. 127–134.

- Sauter, Thilo (2005), ‘Integration Aspects in Automation - a Technology Survey’, *10th IEEE Conference on Emerging Technologies and Factory Automation, 2005. ETFA 2005.* **2**, 255–263.
- Shifflet, Nancy Gill (2004), ‘A Control System Specific Threat and Vulnerability Taxonomy’, *ISA Automation West. AUTOWEST 2004.* .
- Stevens, W. Richard (2001), *TCP/IP Illustrated, Volume 1: The Protocols*, 1ª edição, Addison-Wesley.
- Tanenbaum, Andrew S. (2003), *Redes de Computadores*, 4ª edição, Editora Campus.

Apêndice A

DNP3 sobre TCP

A.1 DNP3

O protocolo DNP3 é utilizado com o mesmo objetivo do Modbus: fazer a comunicação entre os dispositivos de redes industriais. Cita-se na literatura, um uso maior desse protocolo em companhias elétricas [Clarke et al. 2004]. O DNP3 é um protocolo proprietário e foi desenvolvido para operar somente em canais de comunicação serial mas, atualmente, também pode ser utilizado sobre Ethernet. Para utilizar o DNP3 num canal de comunicação Ethernet, não é necessário fazer qualquer modificação do pacote que é utilizado em meios de comunicação serial. O pacote, com todas as suas camadas, é encapsulado dentro de um segmento TCP. Devido a este encapsulamento, é comum se utilizar a denominação de DNP3 sobre TCP (“DNP3 over TCP”).

Um pacote DNP3 é composto de 3 camadas: enlace, transporte e aplicação (*Data Link Layer, Transport Functions e Application Layer*). Os pacotes do DNP3 over TCP contém o mesmo conteúdo de um pacote DNP3 serial, sendo o pacote encapsulado em um segmento TCP. Na Figura A.1, mostra-se como este encapsulamento é realizado.

A camada de enlace do protocolo DNP3 é responsável pela ligação com a camada física, pelo controle e endereçamento dos pacotes e pela ligação com a camada superior de transporte. O quadro do DNP3 (pacote da camada de enlace) é formado por dois campos: um cabeçalho e os dados. O cabeçalho possui campos para indicar as responsabilidades do quadro. Um campo informa o início do quadro, outro campo indica o tamanho do quadro, um campo para informar algumas opções de controle, outros dois campos para indicar os endereços de origem e destino do quadro e por fim um campo para checar a integridade do frame, através do CRC. O CRC também é calculado para cada 16 *bytes* do campo de dados de um pacote DNP3 [Clarke et al. 2004].

A camada de transporte do protocolo DNP3 é considerada como uma pseudo-camada de transporte e possui a função de interligar as camadas de enlace e de aplicação do protocolo, bem como prover alguns serviços de controle de conexão. As mensagens da pseudo-camada de transporte contém apenas um cabeçalho e os dados transmitidos. O cabeçalho é simples, contendo apenas um *byte*, que tem um *bit* para representar se existe fragmentação, um *bit* para representar se é o primeiro fragmento e os outros 6 *bits* para representar o número de sequência da mensagem [Clarke et al. 2004].

A camada de aplicação do protocolo DNP3 foi desenvolvida com o intuito de satisfazer necessidades dos sistemas de automação distribuídos e sistemas SCADA. A camada

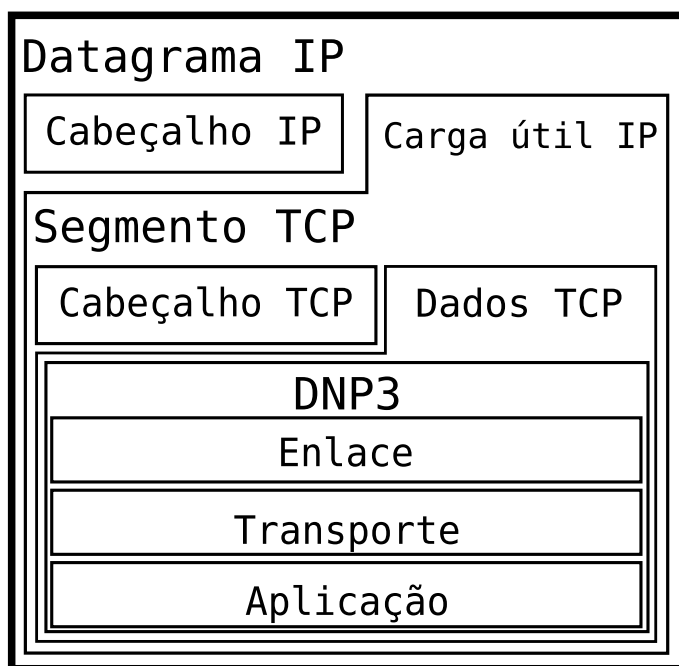


Figura A.1: Encapsulamento do DNP3 sobre TCP.

de aplicação é responsável pela comunicação com a camada do usuário, abstraindo todas as outras camadas inferiores à camada de aplicação. Portanto, o usuário necessita apenas do conhecimento da camada de aplicação para o desenvolvimento dos sistemas de controle que se utilizam deste protocolo. A camada dos usuários faz uso da camada de aplicação para enviar e receber mensagens entre os dispositivos que utilizam DNP3.

Os pacotes da camada de aplicação são constituídos por três campos: cabeçalho do pacote, cabeçalho dos dados e dados. O cabeçalho contém o propósito de mensagem e as informações de controle da camada de aplicação (APCI - *Application Control Protocol Information*). O cabeçalho de dados identifica os dados enviados pelo dispositivo no campo de dados. E por fim, o campo de dados contém a mensagem enviada para os dispositivos que utiliza o DNP3 [Clarke et al. 2004].

A.2 Módulo DNP3 sobre TCP do IndPM

O módulo que implementa o protocolo DNP3 sobre TCP, assim como o Módulo Modbus/TCP do **IndPM**, fornece interatividade para o usuário, possibilitando a modificação do conteúdo dos campos existentes num pacote DNP3. Como mostrado na Seção A.1, o DNP3 sobre TCP implementa três camadas, as quais foram implementadas em diferentes classes neste módulo.

Diferente do Modbus/TCP, os pacotes do protocolo DNP3 não sofrem variação do conteúdo, apenas dos valores dos campos existentes nas três camadas do protocolo. Estas camadas estão representadas por classes que herdam da classe **Packet** do Scapy. Assim

como no módulo do Modbus/TCP, desenvolvemos algumas classes que herdam os atributos da classe **StrField**, definindo alguns campos necessários para o DNP3.

A interface do **IndPM** referente ao DNP3 sobre TCP pode ser observada na Figura A.2. Esta interface contém os campos existentes num pacote DNP3 sobre TCP. Os campos são representados por botões do tipo SpinButton possuindo somente os valores especificados pelo protocolo DNP3. Os botões estão agrupados de acordo com as camadas do protocolo DNP3: enlace, transporte e aplicação. Existe outro grupo de botões para indicar a configuração TCP/IP, tendo botões para definir o IP e a porta TCP de destino.

The screenshot shows the 'IndPM.py' window with the 'DNP3' tab selected. The interface is organized into several sections:

- Camada de Enlace (Link Layer):** Contains fields for 'Início' (1380), 'Tamanho' (5), 'Controle' (0), 'Destino' (0), 'Origem' (0), and a 'Função' section with sub-fields 'DIR', 'PRM', 'FCB/ RES', 'FCV/ DFC', and 'Função' (all set to 0). A 'CRC' field is also set to 0.
- Camada de Transporte (Transport Layer):** Contains a 'Cabeçalho' field set to 0, and 'FIN', 'FIR', and 'Sequência' fields (all set to 0).
- Camada de Aplicação (Application Layer):**
 - Cabeçalho de Requisição (Request Header):** Includes 'Controle de Aplicação' (0) and 'Código de Função' (0). Below are 'FIR', 'FIN', 'Con', and 'Sequência' fields (all set to 0).
 - Cabeçalho de Objeto (Object Header):** Includes 'Objeto' (0), 'Qualificador' (0), and 'Variação' (0). Below are 'Grupo do Objeto' and 'Variação do Objeto' (both set to 0), and 'R', 'Índice', and 'Código Qualificador' fields (all set to 0).
- Configuração TCP/IP:** Located at the bottom, it includes an 'IP' field (0.0.0.0) and a 'TCP' field (0), with an 'Enviar' button.

Figura A.2: Tela da interface da manipulação de pacotes do protocolo DNP3 sobre TCP.

Esta interface já está incorporada ao **IndPM** faltando apenas validá-la com testes em dispositivos que se comunicam com o protocolo DNP3 sobre TCP.

Apêndice B

Publicações

Com este trabalho, foram obtidas duas publicações em congressos internacionais na área. São elas:

- *Using a Packet Manipulation Tool for Security Analysis of Industrial Network Protocols*, publicado em Setembro de 2007, no ETFA2007 - 12th IEEE Conference on Emerging Technology and Factory Automation, pp. 744-747, Patras - Grécia;
- *Analysis of Malicious Traffic in Modbus/TCP Communications*, publicado em Outubro de 2008, no CRITIS'08 -3rd Internacional Workshop on Critical Information Infrastructure Security, Frascati - Itália.