通过 WireShark 软件实现 ModbusTcp 报文捕捉与分析
基于 PC 控制 CMMP 伺服

姓名 曹鹏
Festo 技术支持
2020 年 7 月 31 日

关键词:
Client，Wireshark，Modbus/TCP，ModbusTcp，报文捕捉

摘要：
本文介绍如何使用 Wireshark 进行 ModbusTCP 报文捕捉与分析（PC 控制 CMMP 伺服时），文档主要内容包括 ModbusTCP 报文结构介绍，Wireshark 使用介绍，以及报文抓取案例分析等。

目标群体：
本文仅针对有一定自动化设备调试基础的工程师，需要对 ModbusTCP 通讯协议和 Wireshark 软件有一定了解。

声明：
本文档为技术工程师根据官方资料和测试结果编写，旨在指导用户快速上手使用 Festo 产品，如果发现描述与官方正式出版物冲突，请以正式出版物为准。
我们尽量罗列了实验室测试的软、硬件环境，但现场设备型号可能不同，软件/固件版本可能有差异，请务必在理解文档内容和确保安全的前提下执行测试。
我们会持续更正和更新文档内容，恕不另行通知。

目录

1 ModbusTCP 介绍

1.1 简介

Modbus 由 MODICON 公司于 1979 年开发，是一种工业现场总线协议标准。1996 年施耐德公司推出基于以太网 TCP/IP 的 Modbus 协议：ModbusTCP。
Modbus 协议是一项应用层报文传输协议，包括 ASCII、RTU、TCP 三种报文类型。
Modbus 标准协议物理层接口有 RS232、RS422、RS485 和以太网接口，采用 master/slave 方式通信。
IANA（Internet Assigned Numbers Authority，互联网编号分配管理机构）给 Modbus 协议赋予 TCP 端口号为 502，这是目前在仪表与自动化行业中唯一分配到的端口号。
Modbus 设备可分为主站(Poll)和从站(Slave)。一个网络中主站只有一个，从站可以有多个，主站向各从站发送请求帧，从站给予响应。在使用 TCP 通信时，主站为 Client 端，主动建立连接；从站为 Server 端，等待连接。

1.2 ModbusTCP 数据帧

ModbusTCP 的数据帧可分为两部分：MBAP+PDU。

1.2.1 报文头 MBAP

MBAP 为报文头，长度为 7 字节，组成如下：

事务处理标识	协议标识	长度	单元标识符
2 字节	2 字节	2 字节	1 字节

- **事务处理标识：**报文的序列号，一般每次通信之后就要加 1 以区别不同时间的通信数据报文。
- **协议标识：**00 00 表示 ModbusTCP 协议。
- **长度：**表示接下来的数据长度，单位为字节。
- **单元标识符：**可以理解为设备地址。

1.2.2 帧结构 PDU

PDU 由功能码+数据组成。功能码占用 1 个字节，数据长度不定，由具体功能决定。

1.2.3 功能码

Modbus 的操作对象有四种：线圈、离散输入、输入寄存器、保持寄存器。
线圈：PLC 的输出位，开关量，在 Modbus 中可读可写
离散量：PLC 的输入位，开关量，在 Modbus 中只读
输入寄存器：PLC 中由 I/O 系统提供的 16 位地址寄存器，在 Modbus 中只读
保持寄存器：PLC 中可由应用程序更改的 16 位地址寄存器，在 Modbus 中可读可写

根据控制对象不同，Modbus 的功能码主要有：
0x01：读线圈
0x05：写单个线圈
0x0F：写多个线圈
0x02：读离散量输入
0x04：读输入寄存器
0x03：读保持寄存器
0x06：写单个保持寄存器
0x10：写多个保持寄存器
0x17：读/写多个保持寄存器

1.2.4 应答格式

ModbusTCP 主站发送报文的格式如下：
字节 0：交易标识符 - 由服务器复制 - 通常为 0
字节 1：交易标识符 - 由服务器复制 - 通常为 0
字节 2：协议标识符= 0
字节 3：协议标识符= 0
字节 4：长度字段（高字节）= 0（因为所有消息均小于 256）
字节 5：长度字段（低位字节）=后面的字节数
字节 6：单元标识符（以前是“从站地址”）
字节 7：MODBUS 功能代码

字节 8：根据需要提供数据

ModbusTCP 从站响应报文格式同上，但从字节 7 开始有如下区别：

从站正常响应：请求功能码+响应数据
从站异常响应：异常功能码+异常码，其中异常功能码即将请求功能码的最高有效位置 1，异常码指示差错类型
如下为正常情况下应答示例：

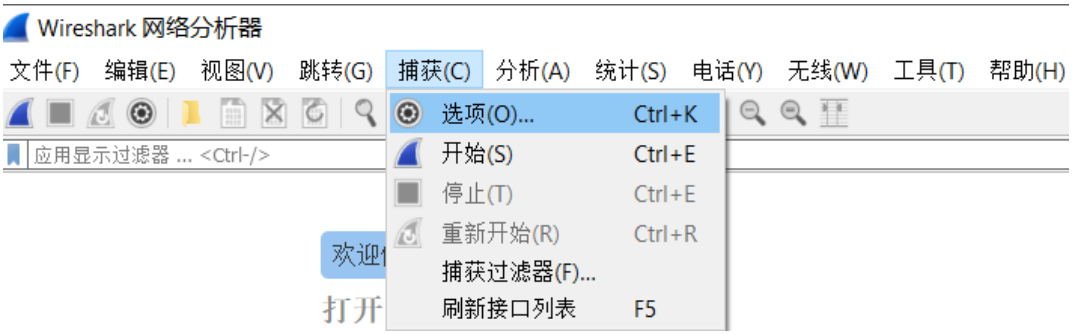
在从站 9 起始地址 4 处读取 1 个 byte 长度的数据	
请求	00 00 00 00 00 06 09 03 00 04 00 01
响应	00 00 00 00 00 05 09 03 02 00 05

2 Wireshark 基本用法

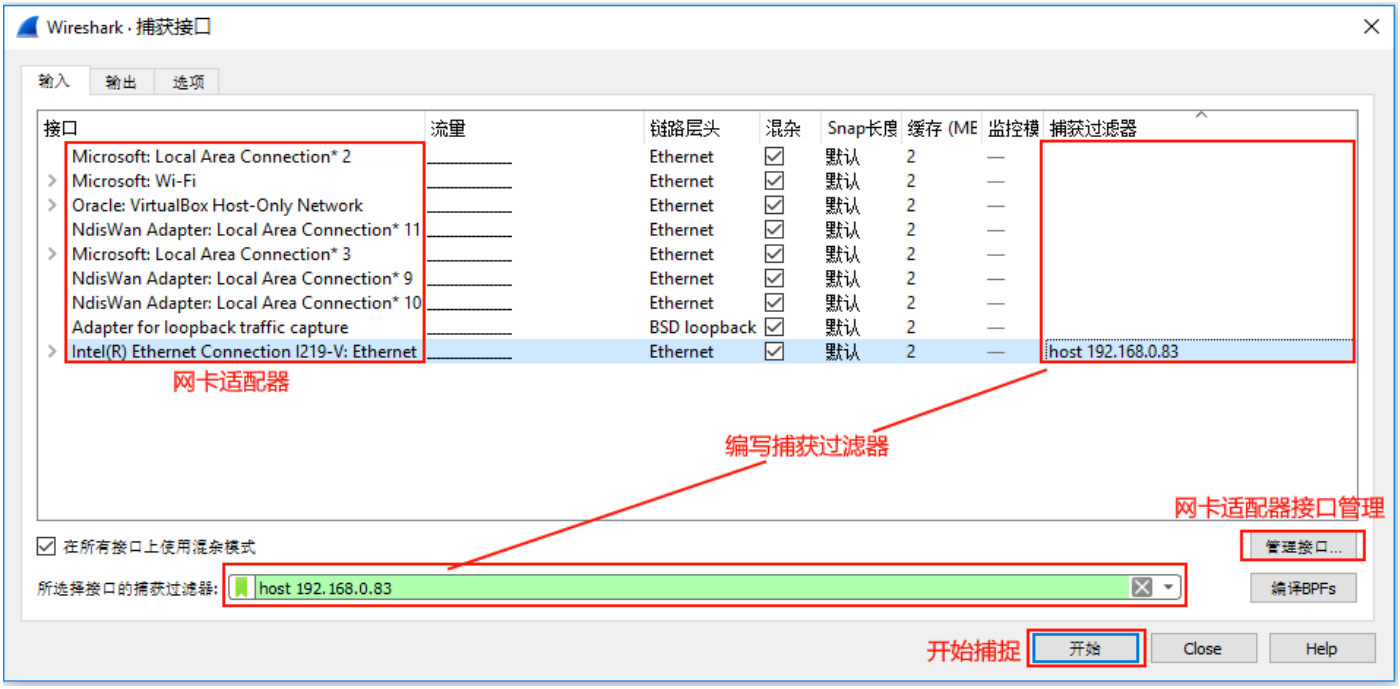
Wireshark（前称 Ethereal）是一个网络封包分析软件，可以截取各种网络封包，显示网络封包的详细信息；
Wireshark 使用 WinPCAP 作为接口，直接与网卡进行数据报文交换；
Wireshark 是开源软件，针对不同系统有相应软件版本。

2.1 选择网卡

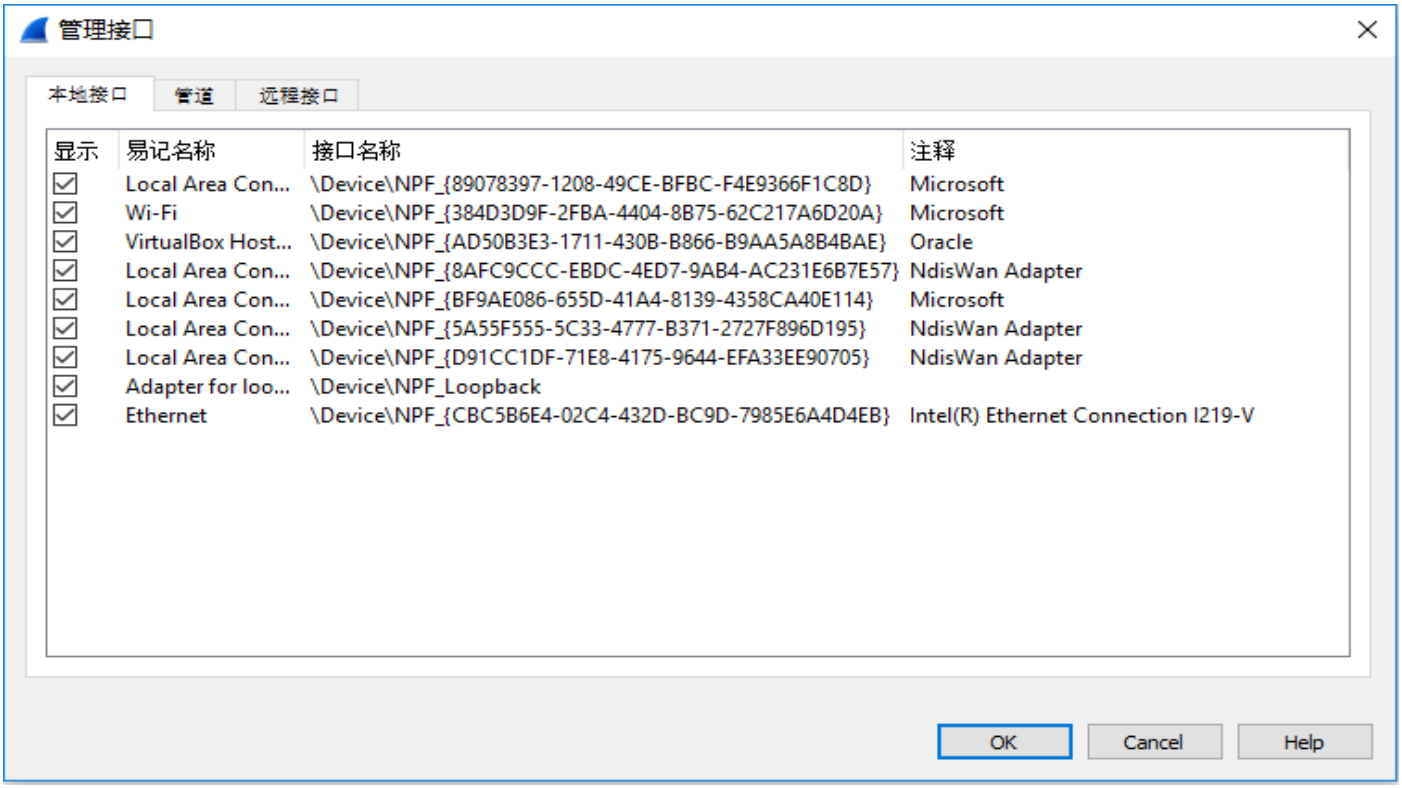
Wireshark 用于获取机器上的某一块网卡的网络包，当机器上有多块网卡的时候，请首先选择需要监控的那块网卡，如下图，点击“捕获->选项”出现下面对话框。



弹出下图界面

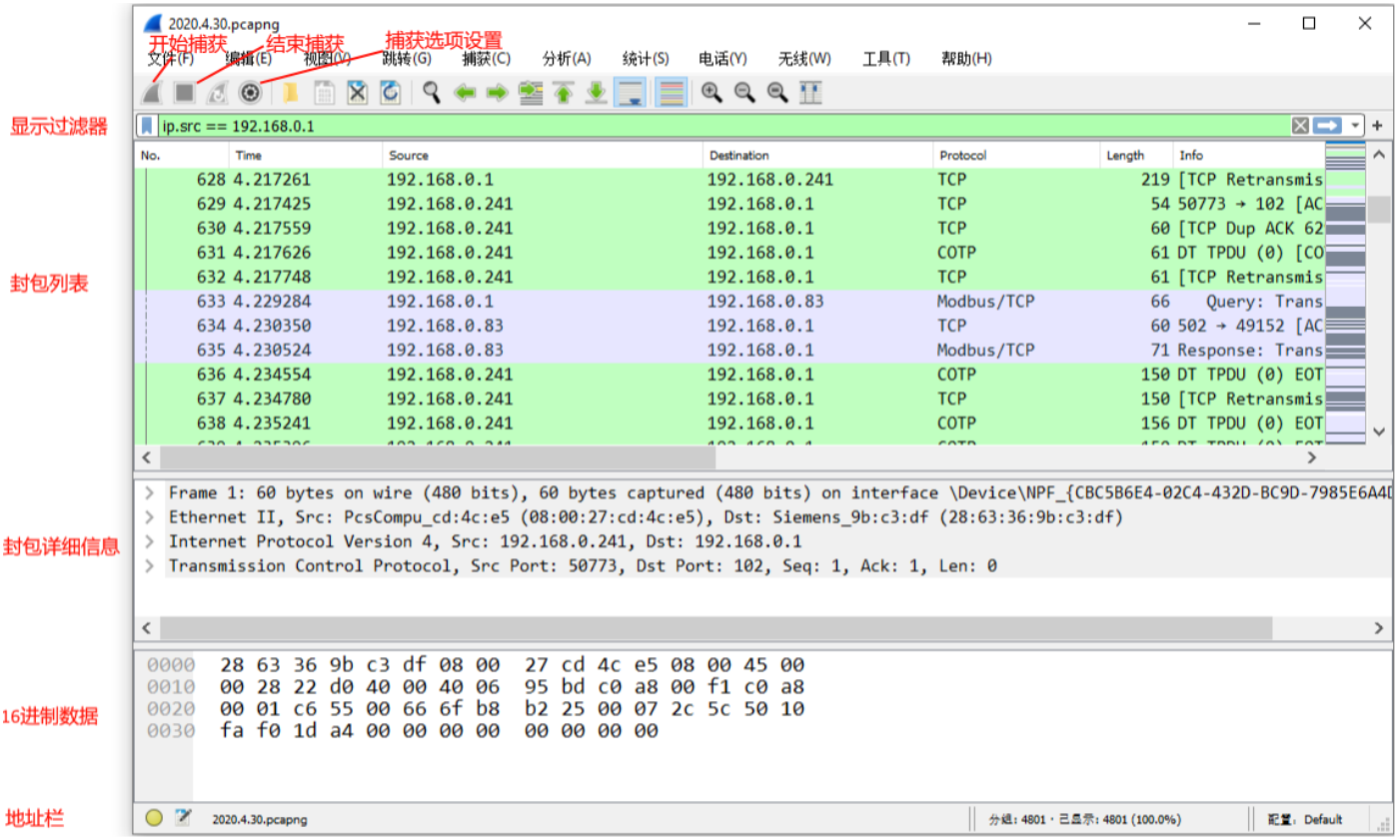


点击上图窗口中“管理接口”按钮，在弹出窗口中选择需要显示的网卡，可以减少不必要的选择项。



2.2 窗口介绍及使用技巧

窗口界面简单介绍，如下图，



Wireshark 窗口界面主要包含如下几个区域：

a. Display Filter(显示过滤器)：用于过滤封包列表所需显示的信息

b. Packet List Pane(封包列表)：显示捕获到的封包，有源地址和目标地址，端口号等

c. Packet Details Pane(封包详细信息)：显示封包中的字段

d. Dissector Pane(16 进制数据)

e. Miscellanous(地址栏，杂项)

2.2.1 过滤器

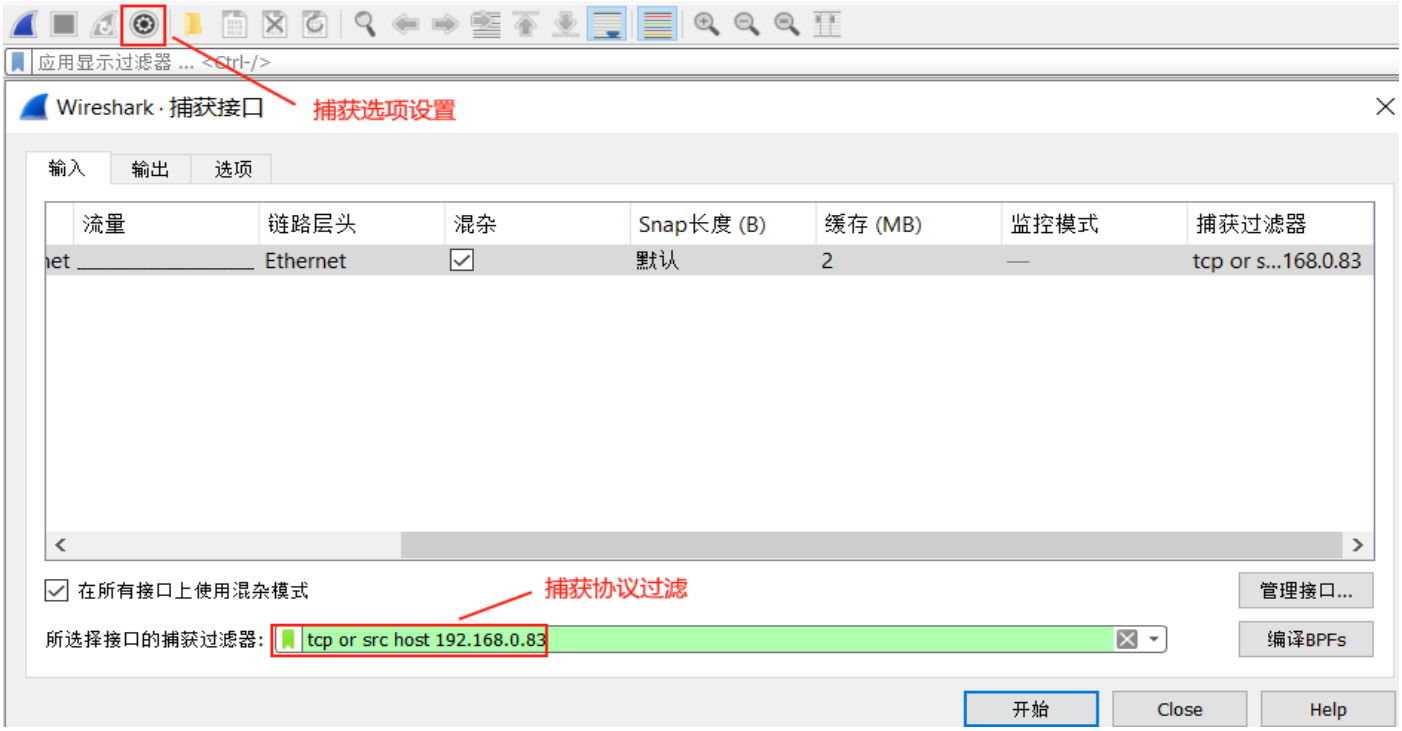
使用 Wireshark 时，将会得到大量的冗余信息，此时可考虑通过过滤器进行内容筛选。过滤器有两种，其过滤语法并不完全相同，具体如下。

2.2.1.1 捕获过滤器

捕捉过滤器在抓包前进行设置，决定抓取怎样的数据，以此减少被捕获的数据量；

捕捉过滤器仅支持协议过滤，区别于显示过滤器既支持协议过滤也支持内容过滤。

在“捕获->捕获过滤器”中设置，如下图：



协议过滤语法规则

语法：	Protocol	Direction	Host(s)	Value	Logical Operations	Other expression
例子：	tcp	dst	192.168.0.83	502	Or	tcp dst 192.168.0.83

字段详解：

Protocol（协议）：

可能值：ether, fddi, ip, arp, rarp, decnet, lat, sca, moprc, mopdl, tcp and udp.

如果没指明协议类型，则默认为捕捉所有支持的协议。

注：在 Wireshark 的 HELP-Manual Pages-Wireshark Filter 中可查询到支持的协议。

Direction（方向）：

可能值：src, dst, src and dst, src or dst

如果没指明方向，则默认使用 “src or dst” 作为关键字。

Host(s)：

可能值： net, port, host, portrange.

默认使用“host”关键字。

Value：

端口号，可选值。

Logical Operations（逻辑运算）：

可能值：not, and, or.

否(“not”)具有最高的优先级。或(“or”)和与(“and”)具有相同的优先级，运算时从左至右进行。

2.2.1.2 显示过滤器

对捕捉到的数据包依据协议或包的内容进行显示过滤,过滤有两种方式，协议过滤法和内容过滤法。

1）.协议过滤语法

语法:	Protocol	•	String1	•	String2	Comparison operator	Value	Logical Operations	Other expression
例子:	ip	•	src	•		==	192.168.0.24	or	Ip.dst 192.168.0.83

string1 和 string2 是可选的。

依据协议过滤时，可直接通过协议来进行过滤，也能依据协议的属性值进行过滤。

按协议进行过滤：snmp || dns || icmp 显示 SNMP 或 DNS 或 ICMP 封包。

按协议的属性值进行过滤：

ip.addr == 10.1.1.1 显示来源或目的 IP 地址为 10.1.1.1 的封包。

ip.src != 10.1.2.3 or ip.dst != 10.4.5.6 来源不为 10.1.2.3 或者目的不为 10.4.5.6 的封包。

ip.src == 10.230.0.0/16 显示来自 10.230 网段的封包。

tcp.port ==502 显示来源或目的 TCP 端口号为 502 的封包。

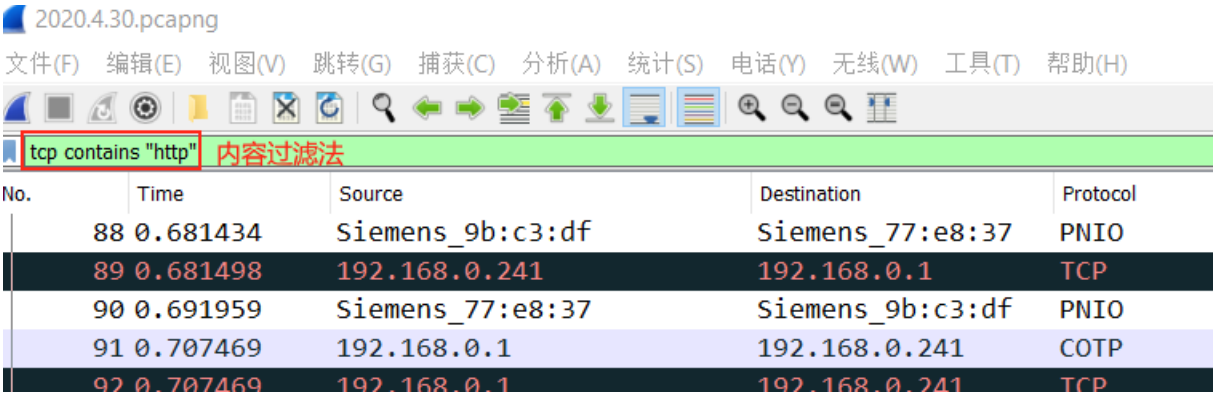
tcp.dstport == 502 目的 TCP 端口号为 502 封包。

http.request.method== "POST" 显示 post 请求方式的 http 封包。

http.host == "tracker.1ting.com" 显示请求的域名为 tracker.1ting.com 的 http 封包。

tcp.flags.syn == 0×02 显示包含 TCP SYN 标志的封包。

2）.内容过滤语法



a. 深度字符串匹配

contains ： Does the protocol, field or slice contain a value

示例:

tcp contains "http" 显示 payload 中包含"http"字符串的 tcp 封包。

b. 特定偏移处值的过滤

tcp[20:3] == 47:45:54　　/* 16 进制形式，tcp 头部一般是 20 字节，这个是对 payload 的前三个字节进行过滤 */

http.host[0:4] == "trac"

过滤中函数的使用（upper、lower）

upper(string-field) - converts a string field to uppercase

lower(string-field) - converts a string field to lowercase

示例:

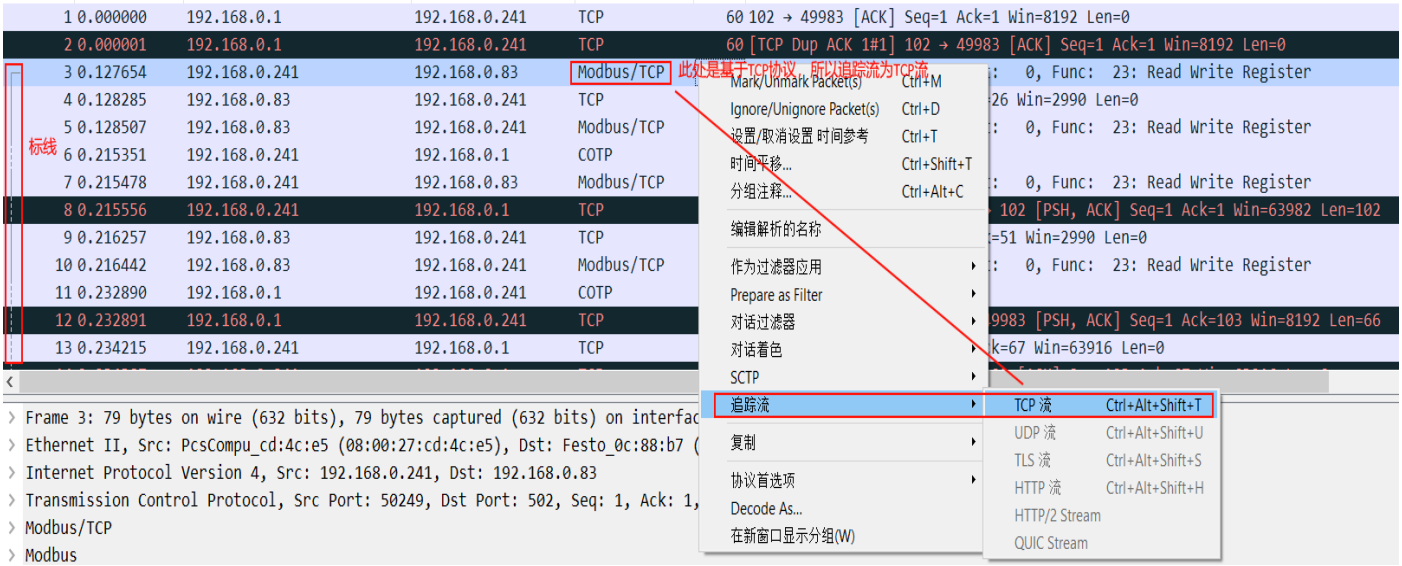
upper(http.request.uri) contains "ONLINE"

Wireshark 过滤支持比较运算符、逻辑运算符，内容过滤时还能使用位运算。

如果过滤器的语法是正确的，表达式的背景呈绿色。如果呈红色，说明表达式有误。

2.2.2　追踪流

点击每一行时，Wireshark 很智能的在记录前用标线表明了本次会话的记录范围（下图左侧标线范围）。
TCP 请求是“请求->响应”式的，需要查看对应请求的响应时，可以在下图中选取相应行号并点击，由于该数据包是通过 TCP 协议通讯，只能选择“追踪流（Follow TCP Stream）”。



下图是根据 TCP 流追踪到的报文

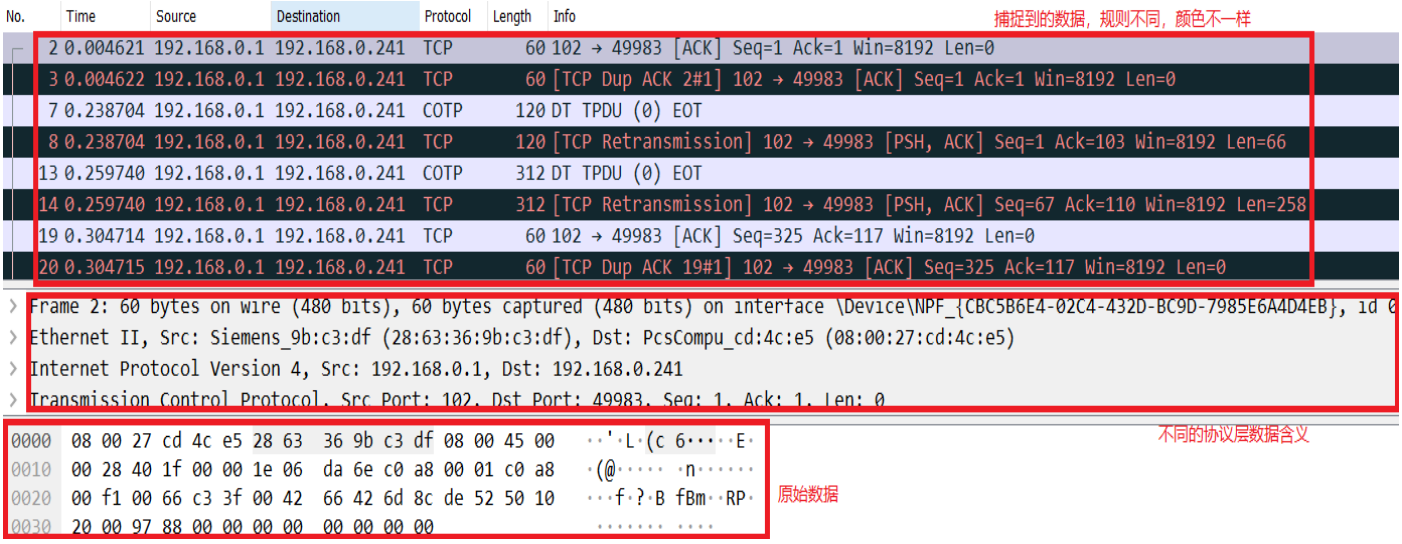
3	0.127654	192.168.0.241	192.168.0.83	Modbus/TCP	79　Query: Trans:　0; Unit:　0, Func:　23: Read Write Register
4	0.128285	192.168.0.83	192.168.0.241	TCP	60 502 → 50249 [ACK] Seq=1 Ack=26 Win=2990 Len=0
5	0.128507	192.168.0.83	192.168.0.241	Modbus/TCP	71 Response: Trans:　0; Unit:　0, Func:　23: Read Write Register
7	0.215478	192.168.0.241	192.168.0.83	Modbus/TCP	79　Query: Trans:　1; Unit:　0, Func:　23: Read Write Register
9	0.216257	192.168.0.83	192.168.0.241	TCP	60 502 → 50249 [ACK] Seq=18 Ack=51 Win=2990 Len=0
10	0.216442	192.168.0.83	192.168.0.241	Modbus/TCP	71 Response: Trans:　1; Unit:　0, Func:　23: Read Write Register
25	0.325564	192.168.0.241	192.168.0.83	Modbus/TCP	79　Query: Trans:　2; Unit:　0, Func:　23: Read Write Register
26	0.326263	192.168.0.83	192.168.0.241	TCP	60 502 → 50249 [ACK] Seq=35 Ack=76 Win=2990 Len=0
27	0.327451	192.168.0.83	192.168.0.241	Modbus/TCP	71 Response: Trans:　2; Unit:　0, Func:　23: Read Write Register
28	0.435402	192.168.0.241	192.168.0.83	Modbus/TCP	79　Query: Trans:　3; Unit:　0, Func:　23: Read Write Register
29	0.436290	192.168.0.83	192.168.0.241	TCP	60 502 → 50249 [ACK] Seq=52 Ack=101 Win=2990 Len=0
30	0.437501	192.168.0.83	192.168.0.241	Modbus/TCP	71 Response: Trans:　3; Unit:　0, Func:　23: Read Write Register
33	0.544230	192.168.0.241	192.168.0.83	Modbus/TCP	79　Query: Trans:　4; Unit:　0, Func:　23: Read Write Register

2.2.3 着色规则

在菜单“视图-着色规则”下查看，此处用于不同规则报文的显示颜色设置。

名称	过滤器
Bad TCP	tcp.analysis.flags && !tcp.analysis.window_update
HSRP State Change	hsrp.state != 8 && hsrp.state != 16
Spanning Tree Topology Change	stp.type == 0x80
OSPF State Change	ospf.msg != 1
ICMP errors	icmp.type eq 3 || icmp.type eq 4 || icmp.type eq 5 || icmp.type eq 11 || icmpv6.type eq 1 || icmpv6.type eq 2 || icmpv6.type eq 3 || icmpv6.type ec
ARP	arp
ICMP	icmp || icmpv6
TCP RST	tcp.flags.reset eq 1
SCTP ABORT	sctp.chunk_type eq ABORT
TTL low or unexpected	(! ip.dst == 224.0.0.0/4 && ip.ttl < 5 && !pim && !ospf) || (ip.dst == 224.0.0.0/24 && ip.dst != 224.0.0.251 && ip.ttl != 1 && !(vrrp || carp))
Checksum Errors	eth.fcs.status=="Bad" || ip.checksum.status=="Bad" || tcp.checksum.status=="Bad" || udp.checksum.status=="Bad" || sctp.checksum.status=="Bad
SMB	smb || nbss || nbns || netbios
HTTP	http || tcp.port == 80 || http2
DCERPC	dcerpc
Routing	hsrp || eigrp || ospf || bgp || cdp || vrrp || carp || gvrp || igmp || ismp
TCP SYN/FIN	tcp.flags & 0x02 || tcp.flags.fin == 1
TCP	tcp
UDP	udp

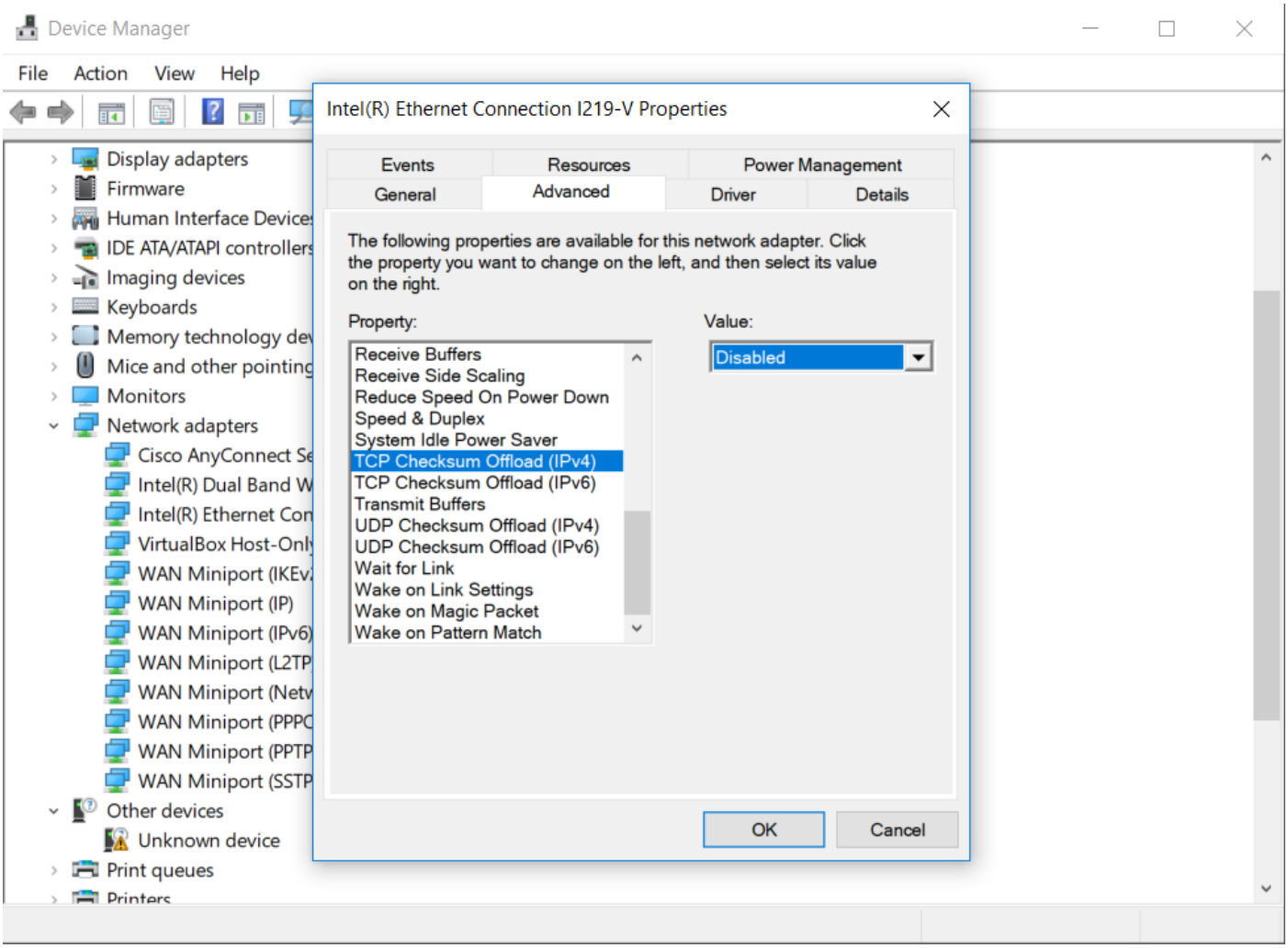
2.3 捕获结果



如果捕获的结果错误包过多， Wireshark 抓包时显示 TCP 校验和错误（Checksum incorrect），但应用层的反应显示报文的 TCP 校验和没问题，很可能网卡配置->高级->Rx Checksum Offload/Tx Checksum Offload 这两处设置是 Enable，将其整成 Disable 即可，见下图，

代价是网络性能降低。一般由操作系统的 TCP/IP 协议栈完成 TCP/UDP/IP 校验和的计算工作，这两处设置成 Enable 之后，协议栈不再进行校验和的计算，而是由网卡自己完成。

如果在前述位置没有发现 Rx Checksum Offload/Tx Checksum Offload 项，有两种可能，一种是网卡本身不支持这种功能，另一种是网卡驱动未提供配置项，后一种情形居多。



2.4 数据包的大致结构

此处以主站发送报文为例，

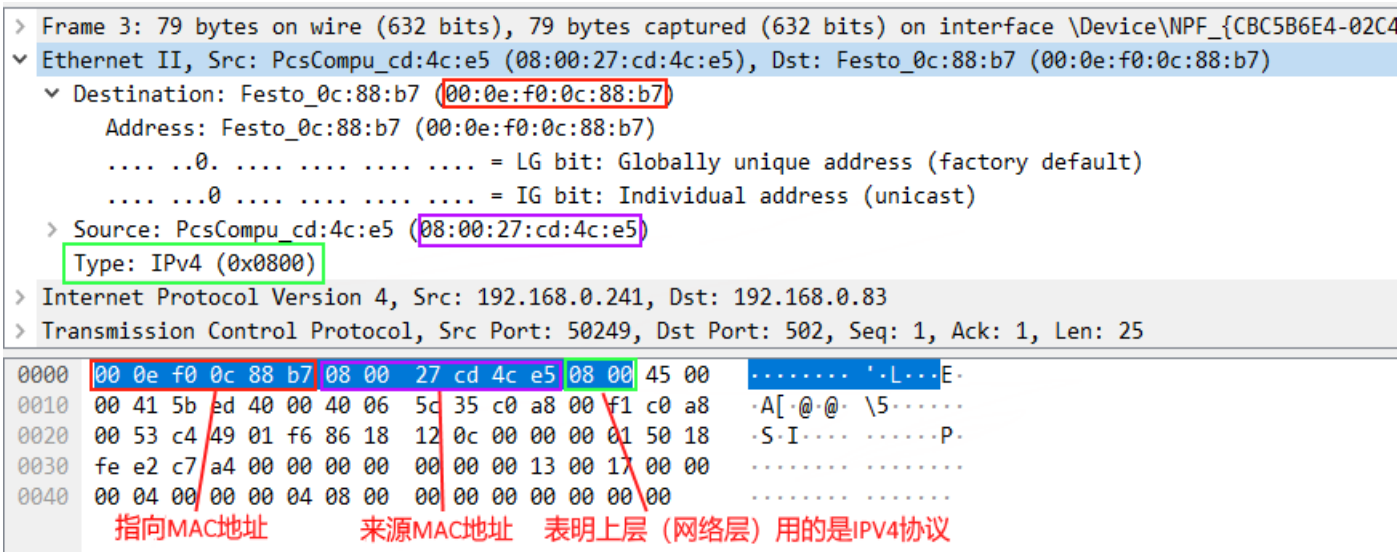
第一行：数据包整体概述

No.	Time	Source	Destination	Protocol	Length	Info
816	4.903118	192.168.0.241	192.168.0.1	COTP	61	DT TPDU (0) [COTP fragment, 0 bytes]
817	4.903346	192.168.0.241	192.168.0.1	TCP	61	[TCP Retransmission] 50773 → 102 [PSH, ACK] Seq=5261 Ack=9772 Win=63776 Len=7
818	4.903524	192.168.0.83	192.168.0.1	TCP	60	502 → 49152 [ACK] Seq=349 Ack=418 Win=2990 Len=0
819	4.904392	192.168.0.83	192.168.0.1	Modbus/TCP	66	Response: Trans: 25; Unit: 255, Func: 16: Write Multiple Registers
820	4.905116	Siemens_9b:c3:df	Siemens_77:e8:37	PNIO	60	RTC1, ID:0x8000, Len: 40, Cycle:16384 (Valid,Primary,Ok,Run)
821	4.921548	Siemens_77:e8:37	Siemens_9b:c3:df	PNIO	60	RTC1, ID:0x8000, Len: 40, Cycle:49152 (Valid,Primary,Ok,Run)
822	4.923150	192.168.0.1	192.168.0.83	Modbus/TCP	66	Query: Trans: 26; Unit: 255, Func: 3: Read Holding Registers
823	4.923474	192.168.0.83	192.168.0.1	TCP	60	502 → 49152 [ACK] Seq=361 Ack=430 Win=2990 Len=0
824	4.924425	192.168.0.83	192.168.0.1	Modbus/TCP	71	Response: Trans: 26; Unit: 255, Func: 3: Read Holding Registers
825	4.939329	192.168.0.1	192.168.0.83	Modbus/TCP	75	Query: Trans: 27; Unit: 255, Func: 16: Write Multiple Registers
826	4.939644	192.168.0.83	192.168.0.1	TCP	60	502 → 49152 [ACK] Seq=378 Ack=451 Win=2990 Len=0

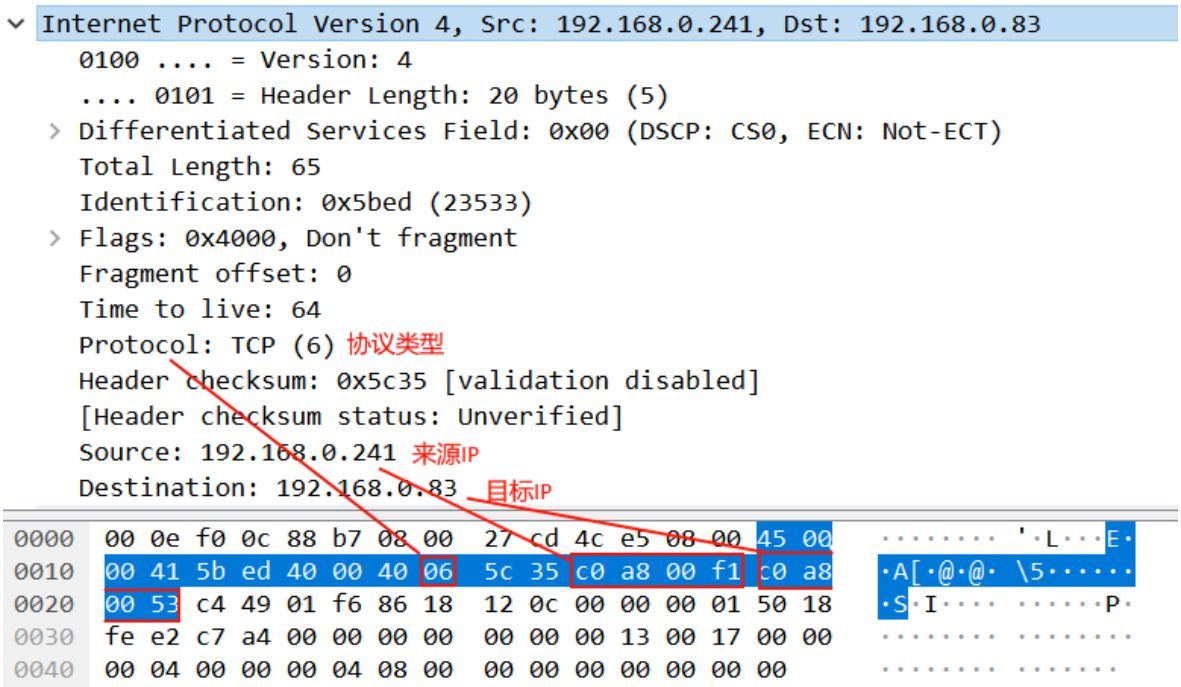
```
> Frame 819: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface \Device\NPF_{CBC5B6E4-02C4-432D-BC9D-7985E6A4D4EB}, id 0 数据包整体概述
> Ethernet II, Src: Festo_0c:88:b7 (00:0e:f0:0c:88:b7), Dst: Siemens_9b:c3:df (28:63:36:9b:c3:df) 链路层,双方的MAC地址
> Internet Protocol Version 4, Src: 192.168.0.83, Dst: 192.168.0.1 网络层，双方的IP
> Transmission Control Protocol, Src Port: 502, Dst Port: 49152, Seq: 349, Ack: 418, Len: 12 传输层，双方的端口
> Modbus/TCP ModbusTCP数据帧
> Modbus ModbusTCP帧结构PDU
```

```
0000  28 63 36 9b c3 df 00 0e  f0 0c 88 b7 08 00 45 00   (c6····· ·····E·
0010  00 34 17 88 00 00 40 06  e1 97 c0 a8 00 53 c0 a8   ·4····@· ·····S··
0020  00 01 01 f6 c0 00 00 00  01 5d 00 03 a4 7c 50 10   ········ ·]···|P·
0030  0b ae bb 6e 00 00 00 19  00 00 00 06 ff 10 00 00   ···n···· ········
0040  00 04                                             ··
```

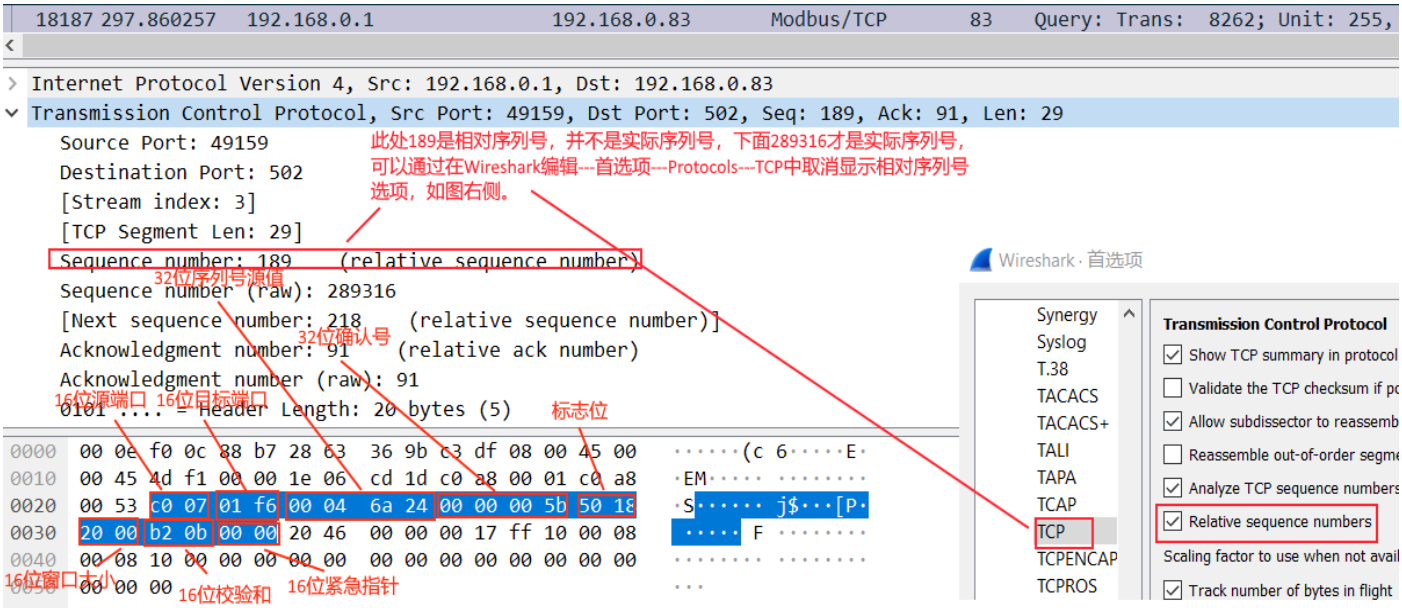
第二行：链路层详细信息，主要的是双方的 MAC 地址报文头 MBAP



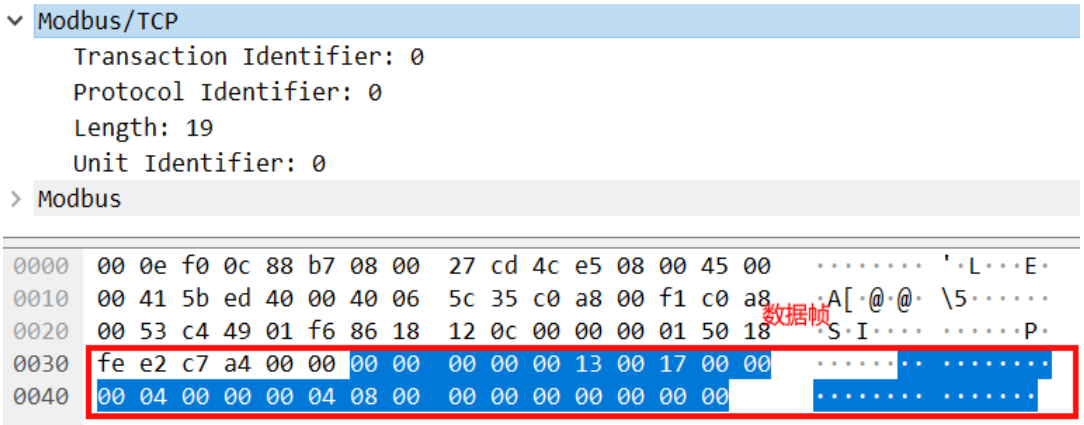
第三行：网络层详细信息，主要的是双方的 IP 地址



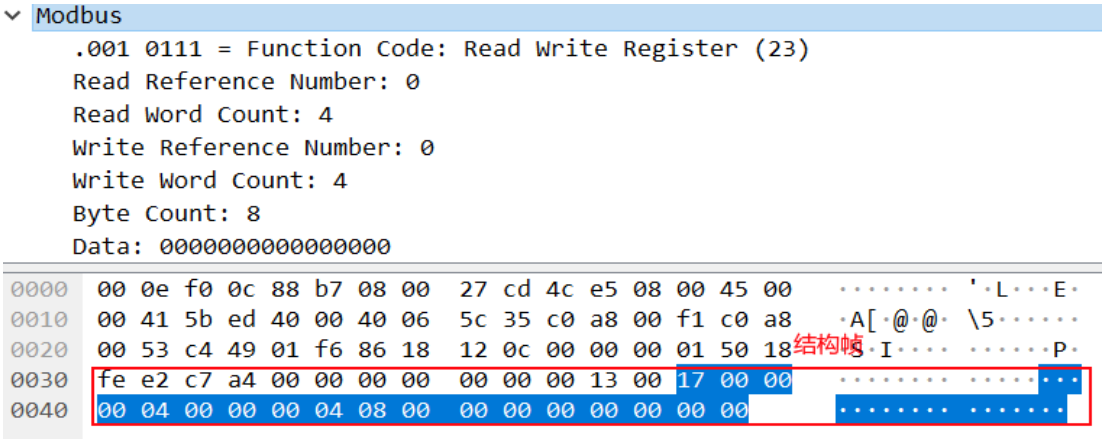
第四行：传输层的详细信息，主要的是双方的端口号，下图中 49159 是来源电脑端口号，502 是目标端口号，TCP 会话的每一端都包含一个 32 位（bit）的序列号， 当主机开启一个 TCP 会话时，初始序列号是随机的，可能是 0 和 4,294,967,295 之间的任意值，而 Wireshark 默认显示的都是相对序列号/确认号，而不是实际序列号/确认号，相对序列号/确认号是和 TCP 会话的初始序列号相关联的，因为比起真实序列号/确认号，跟踪更小的相对序列号/确认号会相对容易一些，下图中 189 就是相对序列号。



第五行：ModbusTCP 数据帧（按住 ctrl 单击快速跳转链接）

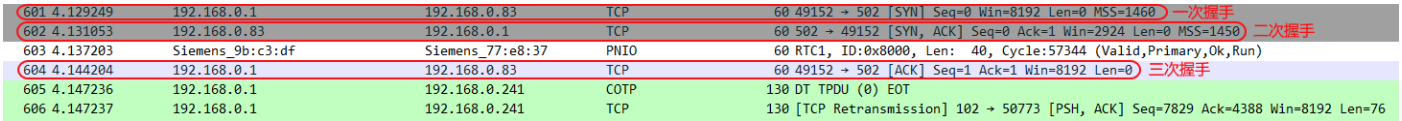


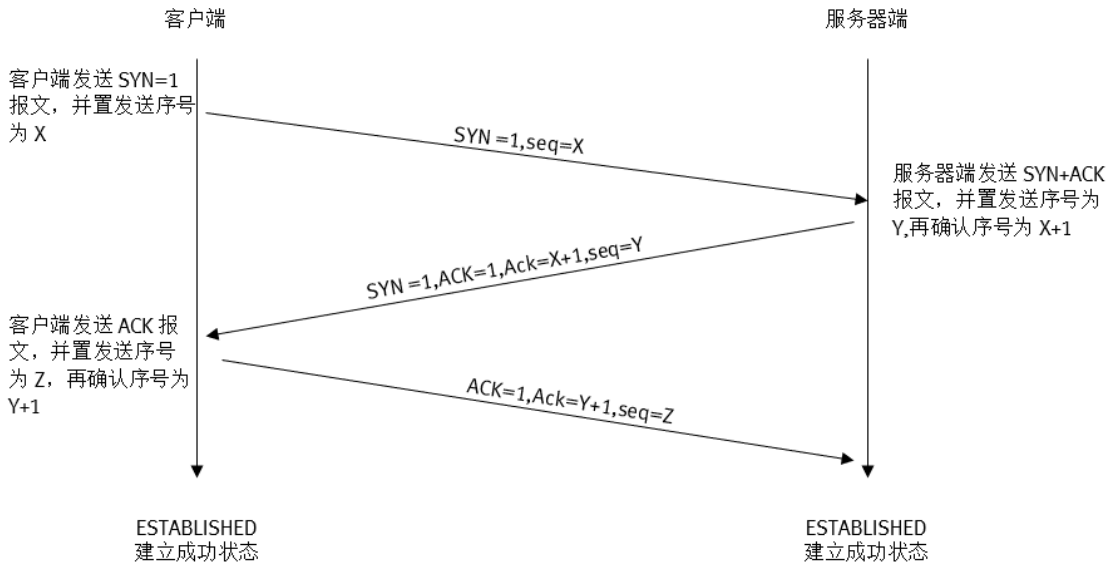
第六行：ModbusTCP 帧结构 PDU（按住 ctrl 单击快速跳转链接）

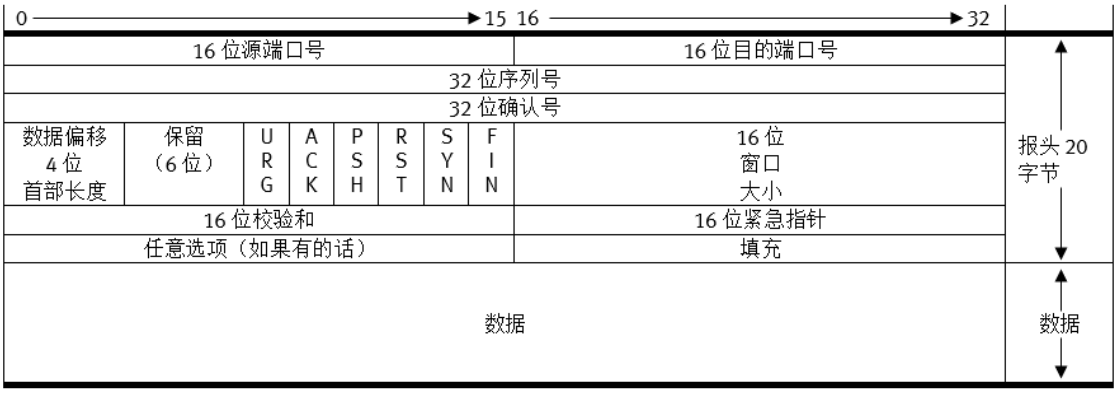


2.5 三次握手

ModbusTCP 通讯连接遵从 TCP 连接标准，初始连接时需要通过三次握手建立连接。
一次握手，客户端->服务器：发送标识为 SYN=1、随机产生的客户端序号 seq（发送序号）
二次握手，服务器->客户端：发送标识为 SYN=1、ACK=1、第一步产生的客户端序号 seq+1（Ack 确认序号）、随机产生的服务端序号 seq
三次握手，客户端->服务器：发送标识 ACK=1,第二步产生的服务端序号 seq+1（确认序号）,随机产生客户端序列号 seq
图中 ACK 为确认标志位，Ack 为确认序列号







标志位对应的功能：

URG： 紧急指针（ urgent pointer）有效。

ACK： 确认序号有效。

PSH： 从站应尽快响应主站。

RST： 重建连接。

SYN： 同步序号用来发起一个连接。

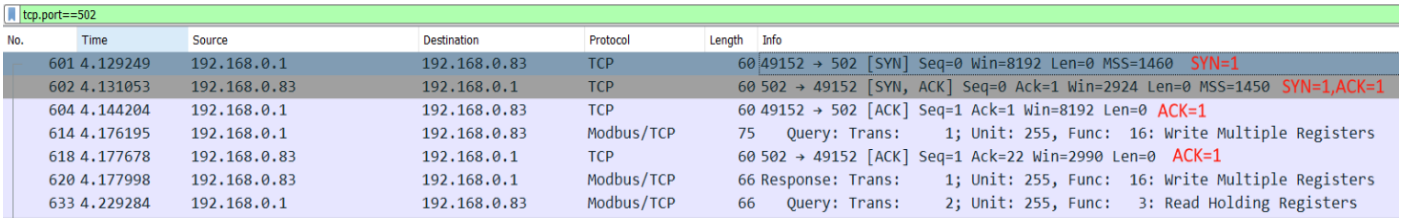
FIN： 发端完成发送任务。

窗口大小：用于流量控制。

检验和：检验和覆盖了整个的 TCP 报文段： TCP 首部和 TCP 数据，与 udp 相似需要计算伪首部 TCP 数据包结构及在 Wireshark 中的位置。

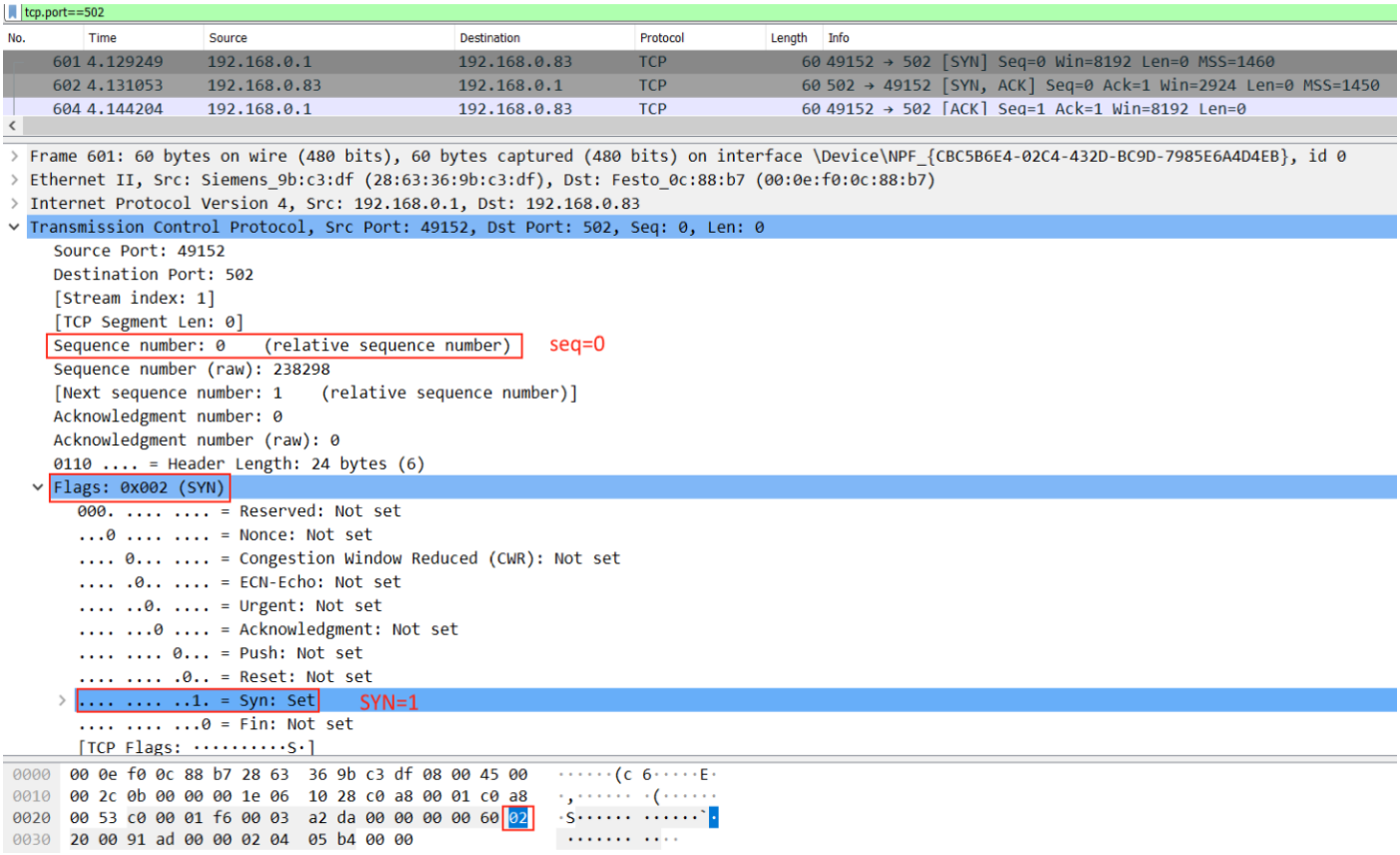
报文标志位含义

其中[SYN]意为 SYN 位为 1（如果没有，则表示为 0）。同理如果[]中有 ACK，表示 ACK 位为 1



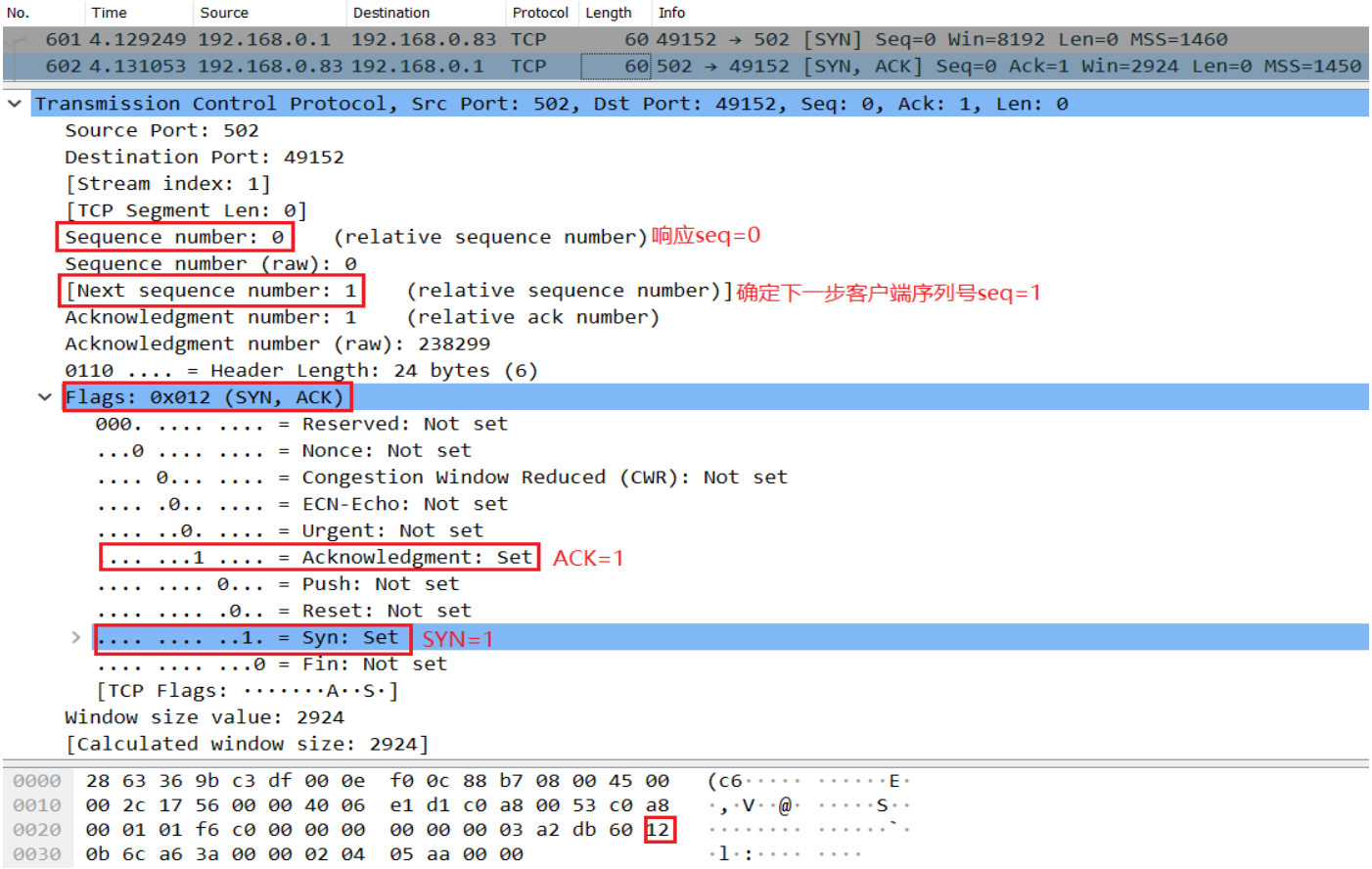
2.5.1 第一次握手

客户端→服务器端，客户端发送第一个包，其中 SYN 标志位为 1，ACK=0，发送序列号 seq=0。客户端进入 SYN 发送状态，等待服务器确认。



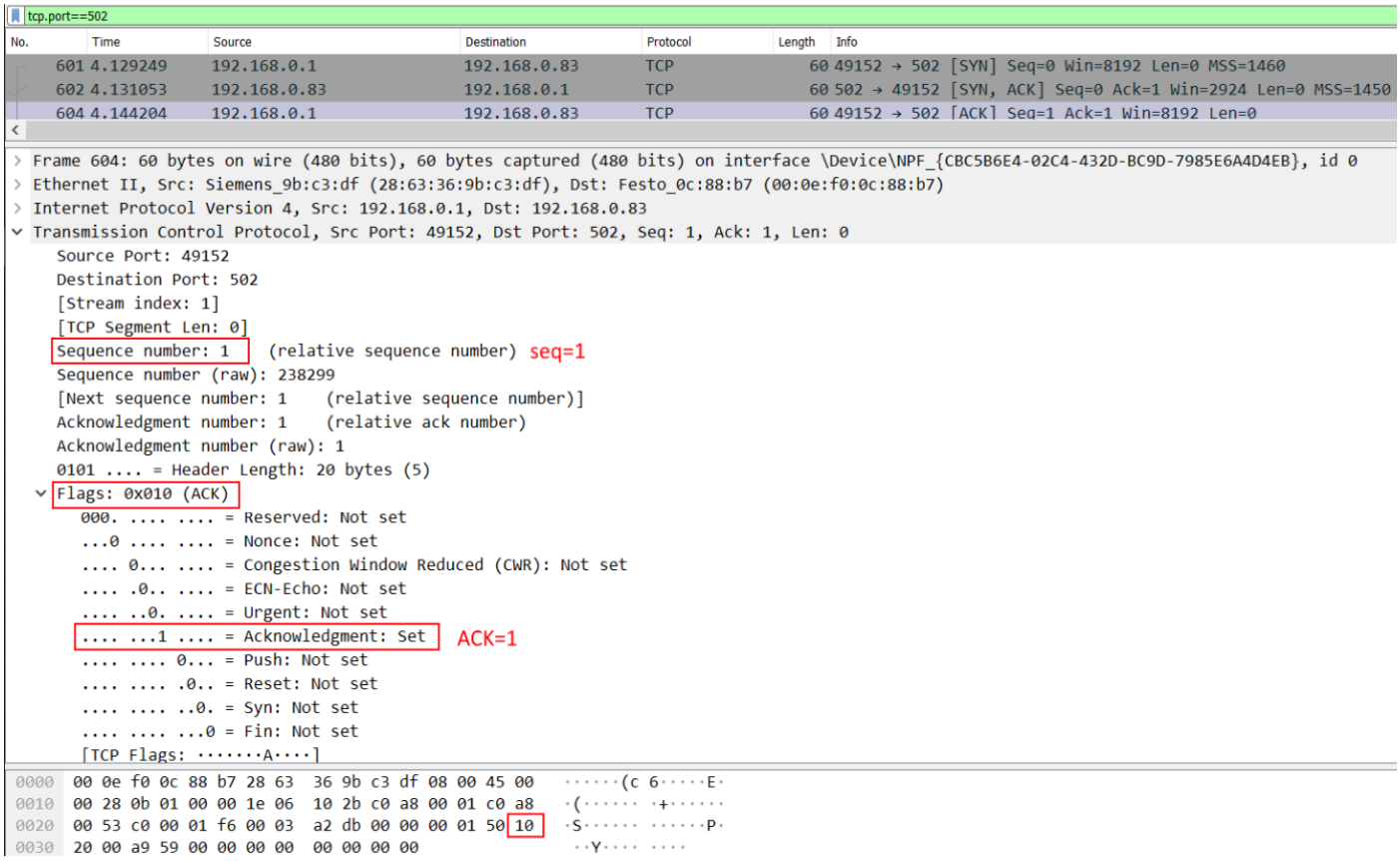
2.5.2 第二次握手

服务器端→客户端，服务器收到这个包后发送第二个包，其中包 SYN、ACK 标志位为 1，发送顺序号 seq=0，接收顺序号 Ack=0+1（此处 0 为一次握手 seq=0），此时服务器进入 SYN 接收状态。



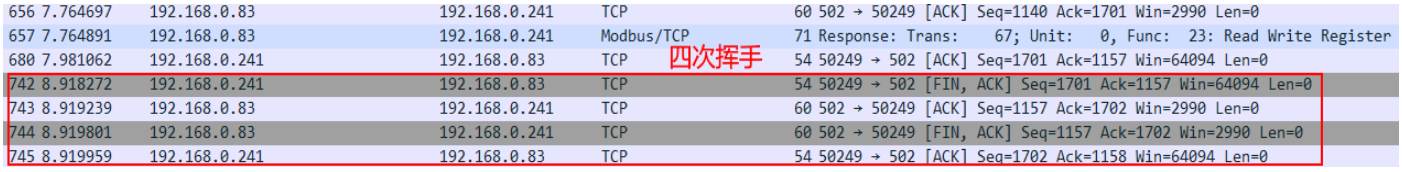
2.5.3 第三次握手

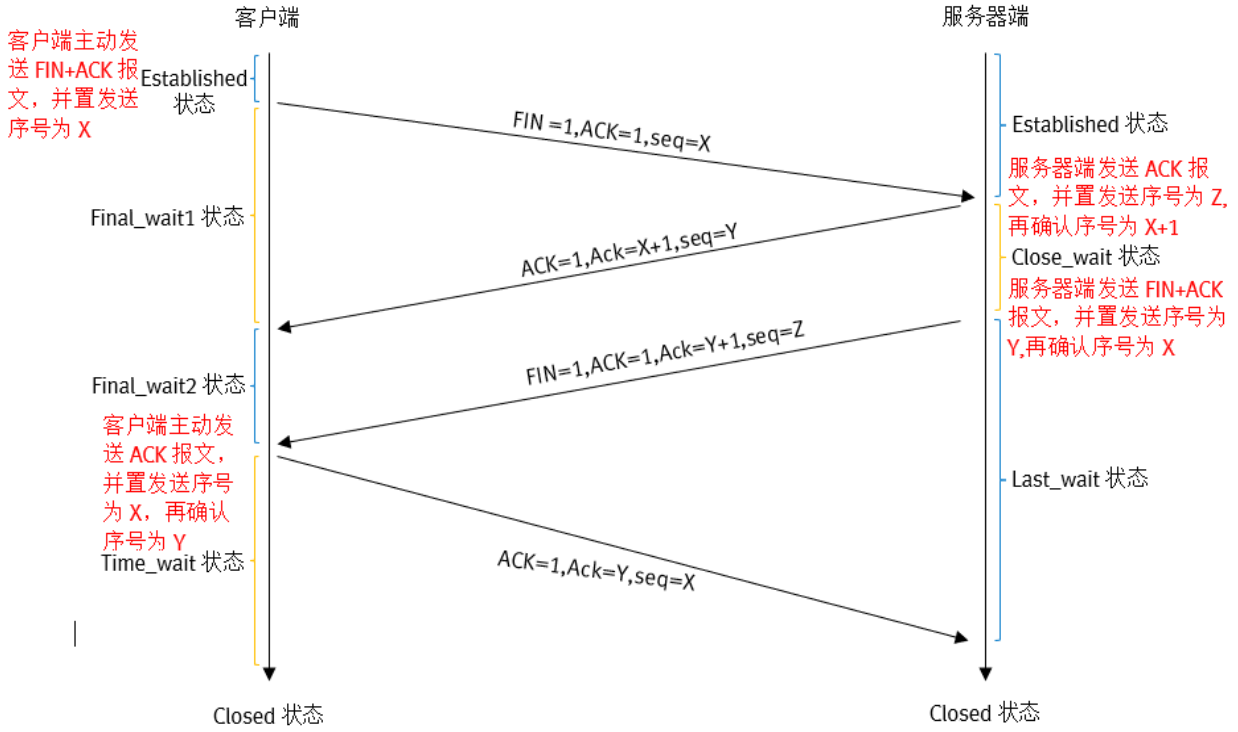
客户端→服务器，客户端收到服务器传来的包后，向服务器发送第三个包，SYN=0，ACK=1，接收顺序号 Ack =0+1（此处 0 为二次握手 seq=0），发送顺序号 seq=1。此包发送完毕，客户端和服务器进入 ESTABLISHED 建立成功状态，完成三次握手。



2.6 四次挥手

假设 Client 端发起中断连接请求，也就是发送 FIN 报文。Server 端接到 FIN 报文后，意思是说"Client 端没有数据要发送了"，但是如果 Client 端还有数据没有发送完成，则 Server 端不必急着关闭 Socket，可以继续发送数据。所以 Server 端先发送 ACK，“告诉 Client 端，你的请求我收到了，但是我还没准备好，请你继续等我的消息”。这个时候 Client 端就进入 FIN_WAIT 状态，继续等待 Server 端的 FIN 报文。当 Server 端确定数据已发送完成，则向 Client 端发送 FIN 报文，“告诉 Client 端，这边数据发完了，准备好关闭连接了”。Client 端收到 FIN 报文后，"就知道可以关闭连接了，但是他还是不相信网络，怕 Server 端不知道要关闭，所以发送 ACK 后进入 TIME_WAIT 状态，如果 Server 端没有收到 ACK，Server 端超时后重传 FIN+ACK 报文段，则 Client 端可以重传,Server 端收到 ACK 后，“就知道可以断开连接了”。Client 端等待了 2MSL（特定时间单位，即 TIME_WAIT 时间）后依然没有收到回复，则证明 Server 端已正常关闭， Client 端也可以关闭连接了， TCP 连接就这样关闭了。





四次数据包
第一次挥手：主动关闭方发送第一个包，其中 FIN 标志位为 1，发送顺序号 seq 为 X。
第二次挥手：被动关闭方收到 FIN 包后发送第二个包，其中发送顺序号 seq 为 Y，接收顺序号 Ack 为 X+1。
第三次挥手：被动关闭方再发送第三个包，其中 FIN 标志位为 1，发送顺序号 seq 为 Z，接收顺序号 Ack 为 Y+1。
第四次挥手：主动关闭方发送第四个包，其中发送顺序号为 X，接收顺序号为 Y。至此，完成四次挥手。

整个挥手过程发送数据包在一定的时间周期内没有收到相应的 ACK，等待一定的时间，超时之后就认为这个数据包丢失，就会重新发送。这个等待时间被称为 RTO。

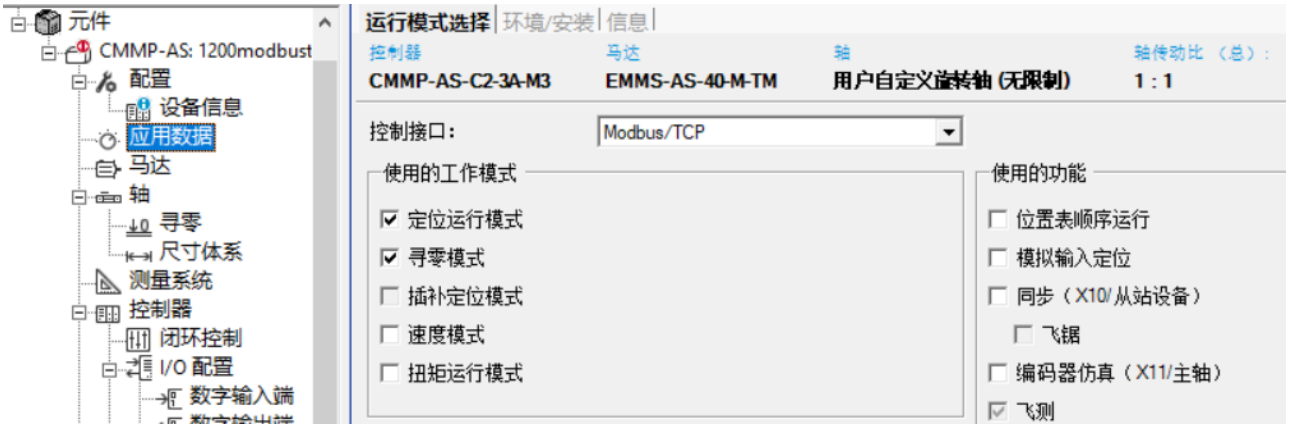
3 操作过程

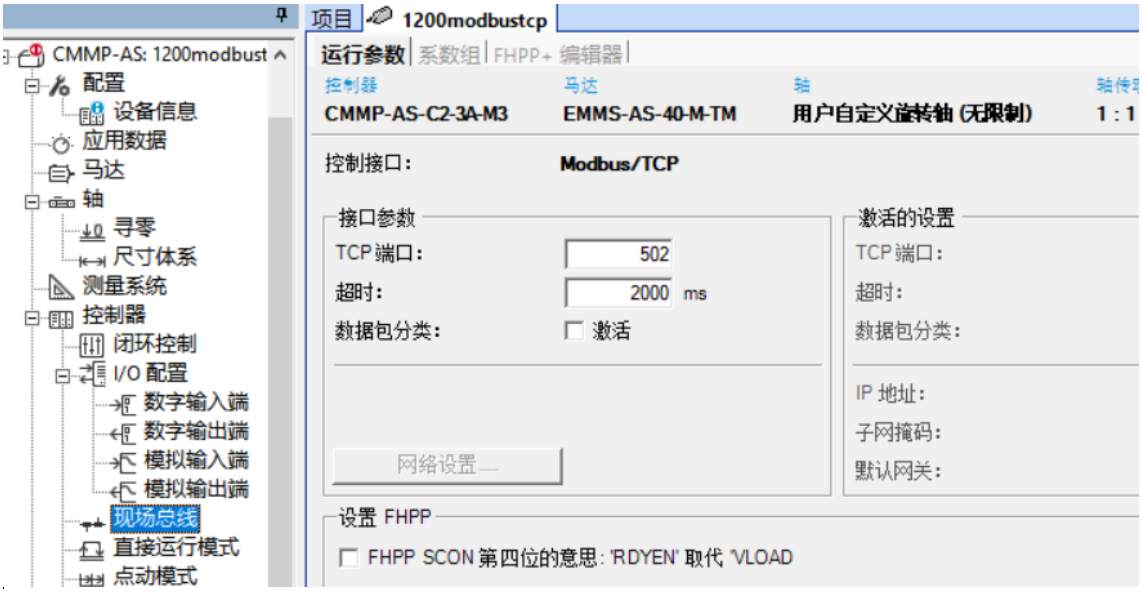
设备软硬件型号及版本如下：
西门子网管型交换机 XF208，固件版本 V5.2.4
伺服控制器 CMMP-AS-C2-3A-M3，固件版本 4.0.1501.2.4
CMMP 控制软件：Modbus TCP Client V1.0

本文准备通过 CMMP 控制软件 Modbus TCP Client 走 ModbusTCP 控制 CMMP 伺服控制器，中间通过西门子交换机 XF208 进行端口映射，最终在电脑侧通过 WireShark 实现报文捕捉。

3.1 CMMP 设置

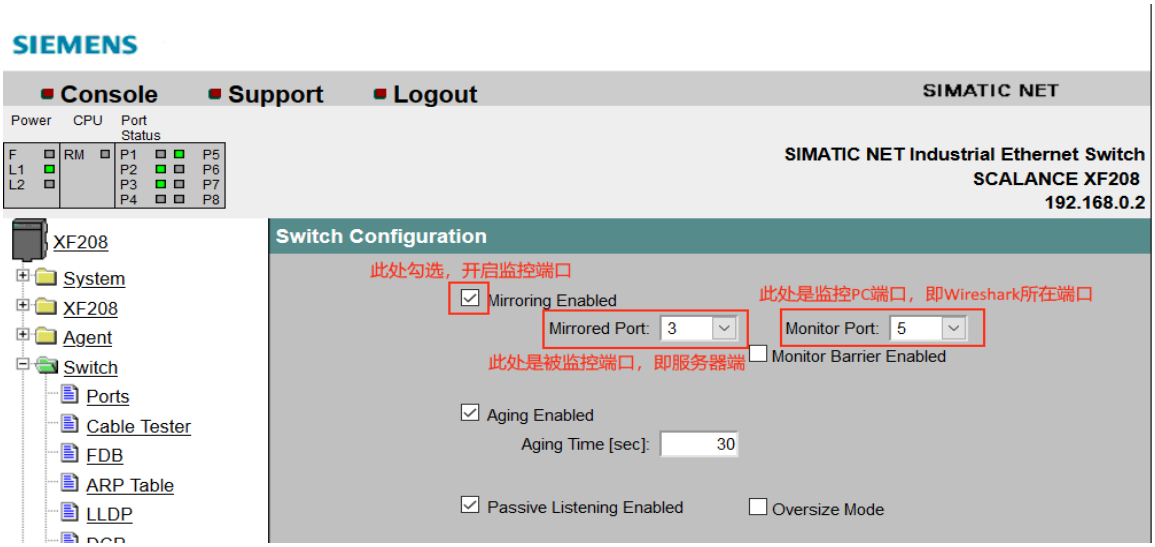
驱动器上 X18 口作为 ModbusTCP 通讯网口，DIN1-DIN8 拨到 OFF；在应用数据中将控制接口更改为 Modbus/TCP，在现场总线中设置 TCP 端口以及超时时间。





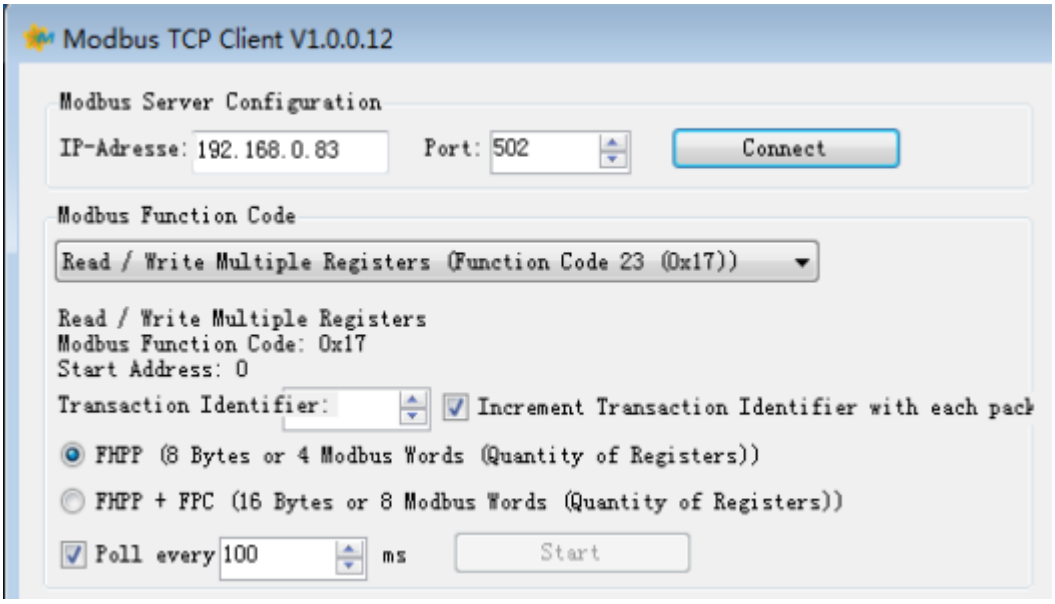
3.2 XF208 设置

交换机需要有端口映射功能，如下在西门子 XF208 上设置监控端口，如下图：



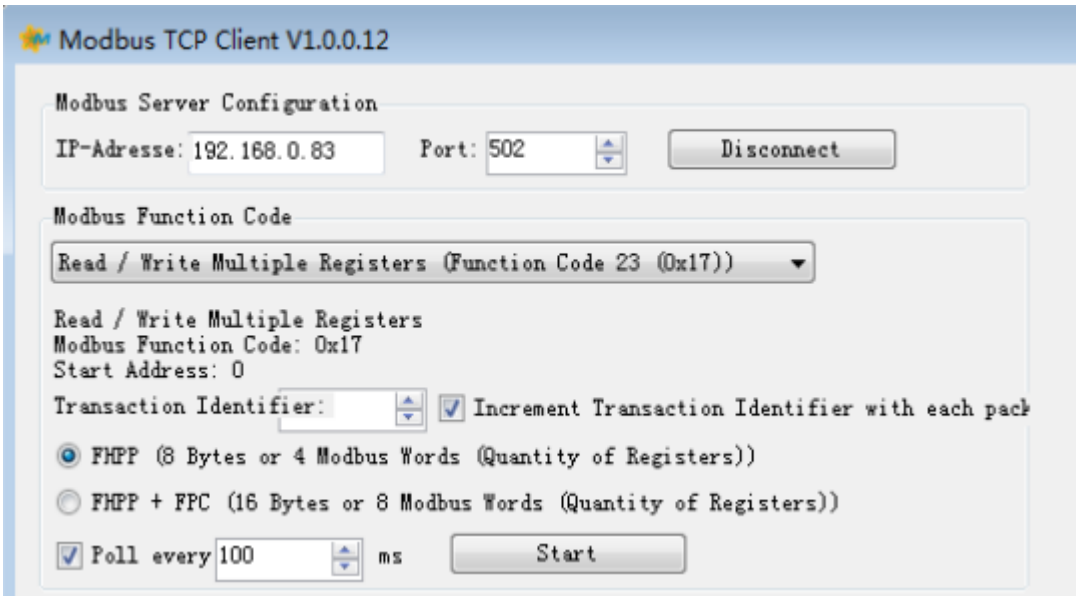
3.3 Modbus Client 设置

打开 Modbus TCP Client，参数如下图设置，IP 地址使用 CMMP-AS 的 X18 口地址，此处为 192.168.0.83，功能码使用 23，使用 FHPP 标准字节通讯。

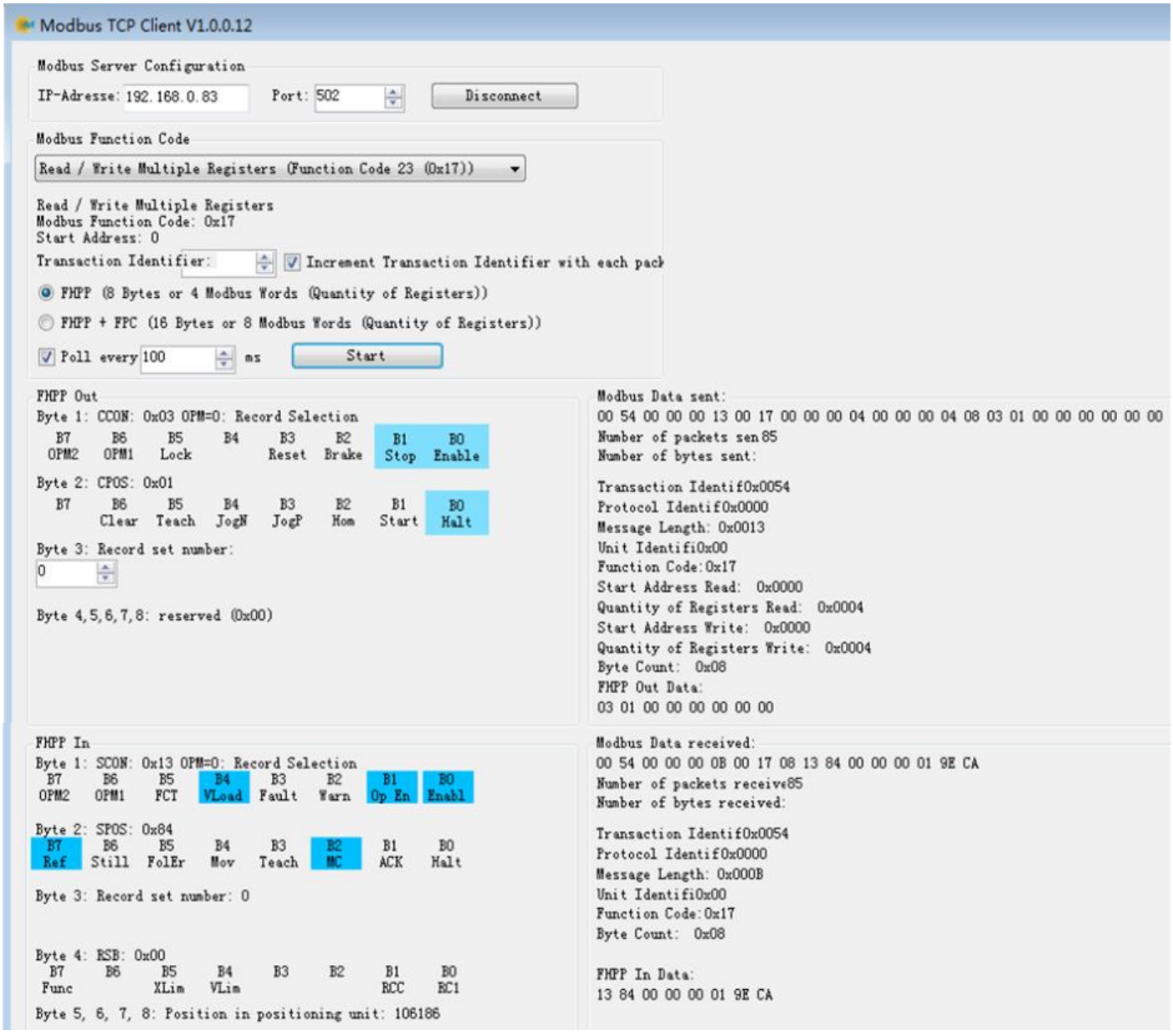


3.4 进行报文抓取

点击 Connect，此时如果能够建立连接，Connect 图标会变成 Disconnect，如下图，



CMMP 伺服正常供电后，点击 Start 开始进行控制，点击 Enable,Stop，Halt，此时驱动器使能成功，如下图，



使用 Wireshark 抓取报文，如下图，抓取的 8 个控制字（03 01 00 00 00 00 00 00），将其 16 进制准换成 2 进制，得到第 1 个控制字为 00000011，和图片中 OpEn，Enabl 状态对应，得到第 2 个控制字为 00000001，和上图中 Halt 状态对应，以此类推，可以得到其它报文定义。

```
    942 18.515272    192.168.0.82          192.168.0.83          Modbus/TCP      79    Query: Trans:    84; Unit:   0, Func:  23: Read Writ
<
> Frame 942: 79 bytes on wire (632 bits), 79 bytes captured (632 bits) on interface \Device\NPF_{CBC5B6E4-02C4-432D-BC9D-7985E6A4D4EB}, id 0
> Ethernet II, Src: PcsCompu_cd:4c:e5 (08:00:27:cd:4c:e5), Dst: Festo_0c:88:b7 (00:0e:f0:0c:88:b7)
> Internet Protocol Version 4, Src: 192.168.0.82, Dst: 192.168.0.83
> Transmission Control Protocol, Src Port: 49341, Dst Port: 502, Seq: 2101, Ack: 1429, Len: 25
v Modbus/TCP
    Transaction Identifier: 84
    Protocol Identifier: 0
    Length: 19
    Unit Identifier: 0
v Modbus
    .001 0111 = Function Code: Read Write Register (23)
    Read Reference Number: 0
    Read Word Count: 4
    Write Reference Number: 0
    Write Word Count: 4
    Byte Count: 8
    Data: 0301000000000000

0000  00 0e f0 0c 88 b7 08 00  27 cd 4c e5 08 00 45 00   ........ '.L...E.
0010  00 41 39 d5 40 00 40 06  7e ec c0 a8 00 52 c0 a8   .A9.@.@. ~....R..
0020  00 53 c0 bd 01 f6 83 85  0c b7 00 00 05 95 50 18   .S...... ......P.
0030  f9 4e d2 60 00 00 00 54  00 00 00 13 00 17 00 00   .N.`...T ........
0040  00 04 00 00 00 04 08 03  01 00 00 00 00 00 00 00   ........ ........
                           控制字
```

同理，如下图，抓取的 8 个状态字（13 84 00 00 00 00 01 9E CA），将其 16 进制准换成 2 进制，得到第 1 个状态字为 00010011，与图中 VLoad，OpEn，Enabl 状态对应，得到第 2 个状态字为 10000100，与图片中 Ref，MC 状态对应，第 5-8 个状态字为 00 01 9E CA，将其 16 进制准换成 10 进制，得到的值为 106186，与图中相同。

```
    946 18.516781    192.168.0.83          192.168.0.82          Modbus/TCP      71 Response: Trans:    84; Unit:   0, Func:  23: Read Write Register
<
> Frame 946: 71 bytes on wire (568 bits), 71 bytes captured (568 bits) on interface \Device\NPF_{CBC5B6E4-02C4-432D-BC9D-7985E6A4D4EB}, id 0
> Ethernet II, Src: Festo_0c:88:b7 (00:0e:f0:0c:88:b7), Dst: PcsCompu_cd:4c:e5 (08:00:27:cd:4c:e5)
> Internet Protocol Version 4, Src: 192.168.0.83, Dst: 192.168.0.82
> Transmission Control Protocol, Src Port: 502, Dst Port: 49341, Seq: 1429, Ack: 2126, Len: 17
v Modbus/TCP
    Transaction Identifier: 84
    Protocol Identifier: 0
    Length: 11
    Unit Identifier: 0
v Modbus
    .001 0111 = Function Code: Read Write Register (23)
    [Request Frame: 942]
    [Time from request: 0.001509000 seconds]
    Byte Count: 8
    Data: 1384000000019eca

0000  08 00 27 cd 4c e5 00 0e  f0 0c 88 b7 08 00 45 00   ..'.L... ......E.
0010  00 39 01 10 00 00 40 06  f7 b9 c0 a8 00 53 c0 a8   .9....@. .....S..
0020  00 52 01 f6 c0 bd 00 00  05 95 83 85 0c d0 50 10   .R...... ......P.
0030  0b ae 71 59 00 00 00 54  00 00 00 0b 00 17 08 13   ..qY...T ........
0040  84 00 00 00 01 9e ca                               .......
      状态字
```

4 其他

4.1 FHPP 报文展开

文档附件中包含报文分析 EXCEL 表格，必要时可借助其进一步分析伺服 CMMP 控制时的 ModbusTcp 报文。

抓到的控制字报文，截取属于 ModbusTCP 部分，如原始数据 005400000013001700000004000000040803010000000000000，点击查询，便可得到结果如下，

查询原始数据 005400000013001700000004000000040803010000000000000　1.输入截取到的ModbusTCP报文

报文拆分	事物处理标识符	协议标识符	长度	单元标识符	功能码	状态字数量	读状态字数量（word格式）	写保持寄存器起始地址	写控制字数量（word格式）	控制字字节数	数据1	数据2	数据3	数据4	数据5	数据6	数据7	数据8
	54	0	13	0	17	0	4	0	4	8	3	1	0	0	0	0	0	0
定义	ModbusTCP通讯协议度验证正确				读/写多个保持寄存器													

查询　2.点击查询

抓到的状态字报文，截取属于 FHPP 报文部分，如原始数据 00540000000b0017081384000000019eca，点击查询，便可得到结果如下，

响应原始数据 00540000000b0017081384000000019eca 1.输入截取到的ModbusTCP报文

报文拆分	事物处理标识符	协议标识符	长度	单元标识符	功能码	状态字数量	数据1	数据2	数据3	数据4	数据5	数据6	数据7	数据8
	0054	0000	000b	00	17	08	13	84	00	00	00	01	9e	ca
定义		ModbusTCP 通讯协议端口502	长度验证正确		读输入寄存器									

响应 2.点击响应

备注：表格只供参考，测试数量有限，不排除有不足。

4.2 内容扩展（西门子 PLC 控制 CMMP）

如下图，当使用西门子 PLC 对 CMMP 进行控制时，抓取报文，报文按照 3 次握手结构建立连接，但挥手结构异于标准结构，西门子直接使用 RST 指令直接结束通讯（RST 指令由西门子 ModbusTCP 标准功能块内部指定使用），此处断开连接是根据 TCP 提供了异常终止连接的方法，即给对方发送一个复位报文段，一旦发送了复位报文段，发送端所有排队等待发送的数据都将被丢弃。

No.	Time	Source	Destination	Protocol	Length	Info
3539	14.939349	192.168.0.1	192.168.0.83	Modbus/TCP	66	Query: Trans: 408; Unit: 255, Func: 3: Read Holding Registers
3542	14.955636	192.168.0.1	192.168.0.83	Modbus/TCP	75	Query: Trans: 409; Unit: 255, Func: 16: Write Multiple Registers
3547	14.986376	192.168.0.1	192.168.0.83	Modbus/TCP	66	Query: Trans: 410; Unit: 255, Func: 3: Read Holding Registers
3560	15.007522	192.168.0.1	192.168.0.83	TCP	60	49152 → 502 [ACK] Seq=6766 Ack=5946 Win=8182 Len=0
3571	15.051660	192.168.0.1	192.168.0.83	Modbus/TCP	75	Query: Trans: 411; Unit: 255, Func: 16: Write Multiple Registers
3588	15.114381	192.168.0.1	192.168.0.83	TCP	60	49152 → 502 [ACK] Seq=6787 Ack=5958 Win=8192 Len=0
3589	15.116363	192.168.0.1	192.168.0.83	Modbus/TCP	66	Query: Trans: 412; Unit: 255, Func: 3: Read Holding Registers
3608	15.207335	192.168.0.1	192.168.0.83	Modbus/TCP	75	Query: Trans: 413; Unit: 255, Func: 16: Write Multiple Registers
3641	15.290391	192.168.0.1	192.168.0.83	Modbus/TCP	66	Query: Trans: 414; Unit: 255, Func: 3: Read Holding Registers
3644	15.311407	192.168.0.1	192.168.0.83	TCP	60	49152 → 502 [ACK] Seq=6832 Ack=6004 Win=8175 Len=0
3667	15.355339	192.168.0.1	192.168.0.83	TCP	60	49152 → 502 [RST, ACK] Seq=6832 Ack=6004 Win=8192 Len=0

```
Transmission Control Protocol, Src Port: 49152, Dst Port: 502, Seq: 6832, Ack: 6004, Len: 0
    Source Port: 49152
    Destination Port: 502
    [Stream index: 1]
    [TCP Segment Len: 0]
    Sequence number: 6832    (relative sequence number)
    Sequence number (raw): 245130
    [Next sequence number: 6832    (relative sequence number)]
    Acknowledgment number: 6004    (relative ack number)
    Acknowledgment number (raw): 6004
    0101 .... = Header Length: 20 bytes (5)
  > Flags: 0x014 (RST, ACK)
    Window size value: 8192
    [Calculated window size: 8192]
    [Window size scaling factor: -2 (no window scaling used)]
    Checksum: 0x7733 [unverified]
    [Checksum Status: Unverified]
    Urgent pointer: 0
  > [Timestamps]
```

```
0000  00 0e f0 0c 88 b7 28 63  36 9b c3 df 08 00 45 00   ......(c 6.....E.
0010  00 28 0e 20 00 00 1e 06  0d 0c c0 a8 00 01 c0 a8   .(. .... ........
0020  00 53 c0 00 01 f6 00 03  bd 8a 00 00 17 74 50 14   .S...... .....tP.
0030  20 00 77 33 00 00 00 00  00 00 00 00               .w3..... ....
```

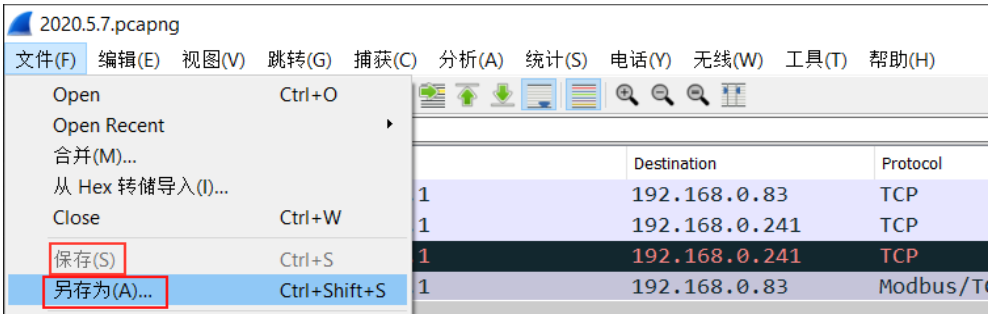
4.3 模拟通讯中断

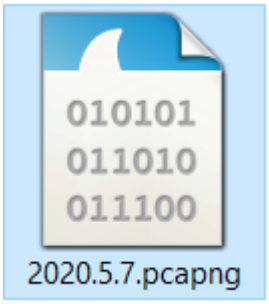
模拟客户端通讯突然中断，此处通过拔取网线，实现物理中断，使用 Wireshark 抓取报文，可以发现服务器会重复发送报文 PSH 指令。服务器把 PSH 置 1，并立即创建一个报文段发往客户端，客户端收到 PSH=1 的报文段，会尽快响应服务器（不再等到整个接收缓存都填满），下图中连续发送两次 PSH 指令而得不到响应，基本可以判断通讯发生过中断。

999	26.733976	192.168.0.254	192.168.0.83	TCP	129	[TCP Retransmission] 49906 → 502 [PSH, ACK] Seq=4451 Ack=3027 Win=64417 Len=75
1000	26.734157	192.168.0.254	192.168.0.83	TCP	129	[TCP Retransmission] 49906 → 502 [PSH, ACK] Seq=4451 Ack=3027 Win=64417 Len=75

5 报文导入导出

报文抓取结束后，可以通过“另存为”保存当前抓取的报文，备份或供其它用途；当需要重新分析报文时，直接双击该导出的文档便可重新打开。如下图，保存格式为.pcapng，如果需要导入报文，直接双击文件打开。


2020.5.7.pcapng
文件(F)
编辑(E)
视图(V)
跳转(G)
捕获(C)
分析(A)
统计(S)
电话(Y)
无线(W)
工具(T)
帮助(H)
Open Ctrl+O
Open Recent
合并(M)...
从 Hex 转储导入(I)...
Close Ctrl+W
保存(S) Ctrl+S
另存为(A)... Ctrl+Shift+S
Destination
Protocol
1 192.168.0.83 TCP
1 192.168.0.241 TCP
1 192.168.0.241 TCP
1 192.168.0.83 Modbus/T


010101
011010
011100
2020.5.7.pcapng