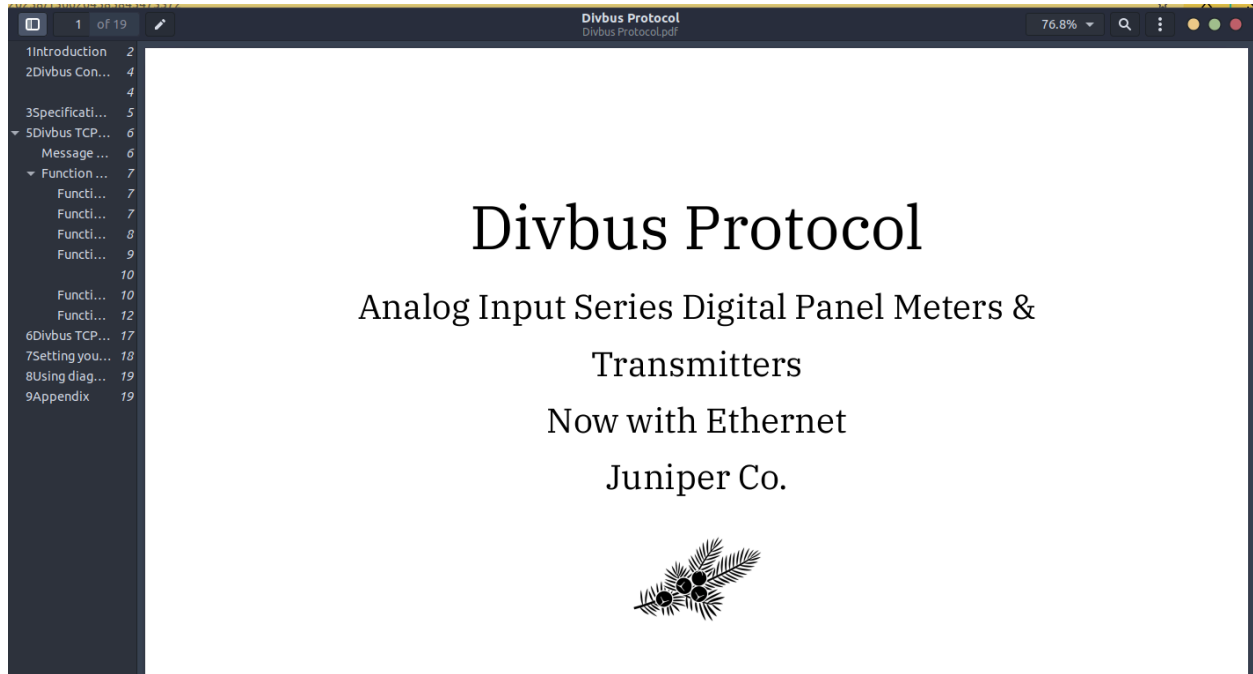


THERE WILL BE SIGNS WRITEUP

You are provided with a PCAP and a document

The document describes the made-up “Divbus Protocol” which is what is found in the PCAP

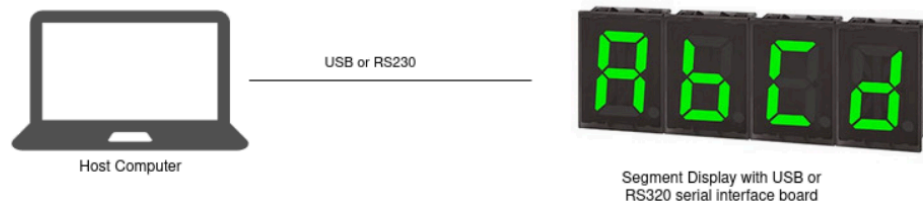


Divbus is basically just Modbus, here's a decent intro to Modbus:

https://www.csimn.com/CSI_pages/Modbus101.html

To understand this made-up protocol you probably need to understand Modbus, because the function codes used by Divbus are the same.

You were looking for specifically for the traffic controlling a 4 digit segment display.
Which is referenced here:



And here:

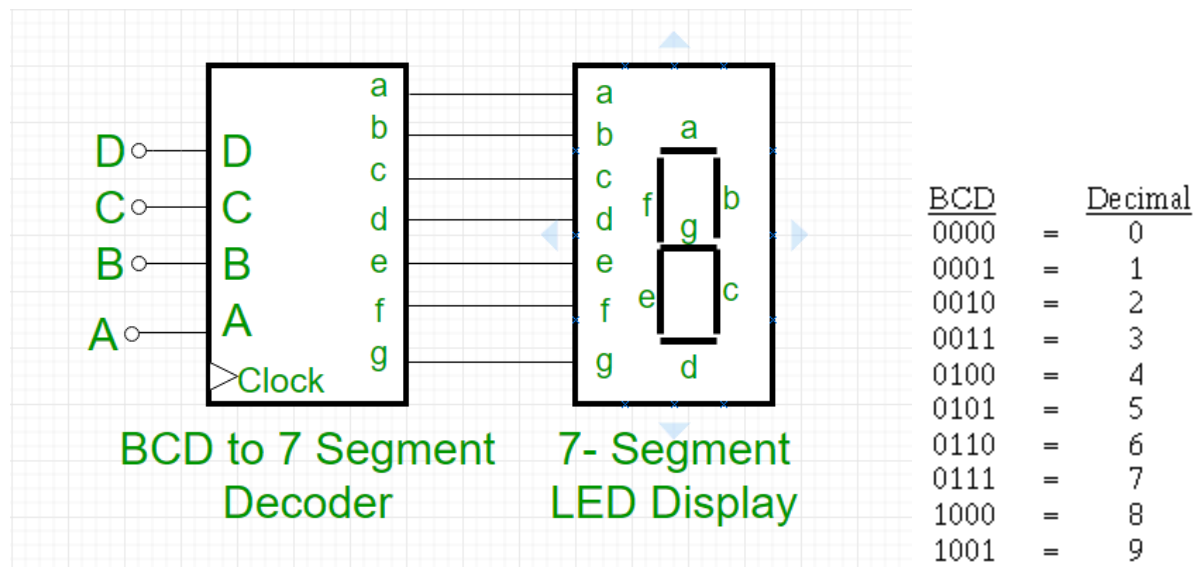
Function Code 01 - Connection Setup

Register		Name	Description
Dec	Hex		
110	00 74	Device Type	Bits 7:0 01 = DPM Meter 02 = Scale Meter 03 = Counter/timer Meter 05 = DPM transmitter 06 = Scale transmitter 07 = Counter/timer transmitter 08 - Segment Display

And most importantly here in the “Function Code 06: Write Single (Holding) Register” table:

		Scale Meter)	
96	00 60	Digit 1 Segment Display Data	Binary Coded Decimal (BCD)
97	00 61	Digit 2 Segment Display Data	Binary Coded Decimal (BCD)
98	00 62	Digit 3 Segment Display Data	Binary Coded Decimal (BCD)
99	00 63	Digit 4 Segment Display Data	Binary Coded Decimal (BCD)
100	00 64	Lockout 1	LCK_t

Digit N Segment Display Data - is just the digit being displayed on the physical segment display.
Binary Coded Decimal:



The format of these packets is also described in the PDF:

05	Request	1	MA	NB	FC	RA	RA	WW	WW	CL	CH			
	Response	1	MA	NB	FC	RA	RA	WW	WW	CL	CH			
06	Request	> 2.5	MA	NB	FC	RA	RA	DD	DD	CL	CH			
	Response	> 2.5	MA	NB	FC	RA	RA	DD	DD	CL	CH			
08	Request	1	MA	NB	FC	SF	SF	WW	WW	CL	CH			
	Response	1	MA	NB	FC	SF	SF	DD	DD	CL	CH			
10	Request	1	MA	NB	FC	NR	NR	RA	RA	NBD	DD	DD	CL	CH
	Response	1	MA	NB	FC	NR	NR	RA	RA	CL	CH			

MA	Meter Address
NB	Number of Bytes in Message Frame
FC	Function Code
DD	Data Bytes

Divbus uses ports 1201-1204, we can tell by looking at the protocol statistics:

Wireshark - Conversations - chall.pcap

Ethernet - 1439		IPv4 - 279		IPv6		TCP - 2629		UDP - 122											
Address A	Port A	Address B	Port B	Packets	Bytes	Packets A → B	Bytes A → B	Packets B → A	Bytes B → A	Rel Start	Duration	Bits/s A → B	Bits/s B → A						
10.10.25.102	49806	10.10.25.119	1204	10	558	6	333	4	225	0.106620	0.0002	—	—						
10.10.25.102	52199	10.10.25.3	1204	10	558	6	333	4	225	1.098145	0.0004	—	—						
10.10.25.102	51344	10.10.25.79	1204	10	563	6	338	4	225	2.058168	0.0002	—	—						
10.10.25.102	55773	10.10.25.3	1204	10	557	6	332	4	225	2.104509	0.0003	—	—						
10.10.25.102	51371	10.10.25.63	1204	10	558	6	333	4	225	3.056245	0.0003	—	—						
10.10.25.102	61040	10.10.25.124	1204	10	558	6	333	4	225	4.070133	0.0003	—	—						
10.10.25.102	56290	10.10.25.232	1204	10	557	6	332	4	225	4.078592	0.0004	—	—						
10.10.25.102	56154	10.10.25.160	1204	10	558	6	333	4	225	5.055916	0.0004	—	—						
10.10.25.102	60341	10.10.25.63	1204	10	565	6	340	4	225	5.069801	0.0004	—	—						
10.10.25.102	65100	10.10.25.3	1204	10	563	6	338	4	225	5.115163	0.0002	—	—						
10.10.25.102	54464	10.10.25.190	1204	10	557	6	332	4	225	6.069950	0.0006	—	—						
10.10.25.102	57369	10.10.25.172	1204	10	558	6	333	4	225	7.084482	0.0005	—	—						
10.10.25.102	54137	10.10.25.124	1204	10	557	6	332	4	225	8.082937	0.0003	—	—						
10.10.25.102	52757	10.10.25.232	1204	10	559	6	334	4	225	8.088461	0.0003	—	—						
10.10.25.102	56055	10.10.25.160	1204	10	565	6	340	4	225	9.064524	0.0002	—	—						
10.10.25.102	52626	10.10.25.160	1204	10	558	6	333	4	225	10.067257	0.0006	—	—						
10.10.25.102	58057	10.10.25.124	1204	10	573	6	333	4	240	12.099314	0.0004	—	—						
10.10.25.102	53581	10.10.25.232	1204	10	558	6	333	4	225	13.108132	0.0003	—	—						
10.10.25.102	50266	10.10.25.79	1204	10	558	6	333	4	225	15.090089	0.0005	—	—						
10.10.25.102	54140	10.10.25.63	1204	10	558	6	333	4	225	16.110650	0.0007	—	—						
10.10.25.102	61111	10.10.25.133	1204	10	559	6	334	4	225	16.130113	0.0002	—	—						
10.10.25.102	52022	10.10.25.190	1204	10	557	6	332	4	225	17.128970	0.0004	—	—						
10.10.25.102	52482	10.10.25.172	1204	10	558	6	333	4	225	19.126799	0.0005	—	—						
10.10.25.102	57650	10.10.25.160	1204	10	565	6	333	4	232	20.094840	0.0002	—	—						
10.10.25.102	62662	10.10.25.190	1204	10	557	6	332	4	225	20.213748	0.0004	—	—						
10.10.25.102	54617	10.10.25.172	1204	10	573	6	333	4	240	22.143971	0.0003	—	—						

☐ Name resolution ☐ Limit to display filter ☐ Absolute start time

? Help

Copy

Follow Stream...

Graph...

Close

Conversation Types

Now we write a script that gets traffic on ports 1201-1204 that writes to one of the segment display registers (ie. `[96, 97, 98, 99]` described above):

We keep store the data written to each data separately:

```
from scapy.all import *
import struct

controller_ip = "10.10.25.102"
node_ip = "10.10.25.119"
divbus_ports = [1201, 1202, 1203, 1204]

noToSegmentHex = {
    0x3f: 0, 0x6: 1, 0x5b: 2, 0x4f: 3, 0x66: 4, 0x6d: 5, 0x7d: 6, 0x07: 7,
    0x7f: 8, 0x6f: 9
}

reg_address = [96, 97, 98, 99]

r = rdpcap("chall.pcap")
n = len(r)
x = ""
flag = ""
meters_display_registers = {}
for i in range(n):
    pkt = r[i]
    if Raw in pkt and pkt.haslayer(TCP) and (pkt[IP].src in controller_ip or
    pkt[IP].dst in node_ip):
        # and (pkt[TCP].sport in divbus_ports or pkt[TCP].dport in
    divbus_ports):
```

```

payload = pkt[TCP].load

#print(payload)
ma = struct.unpack_from('<B', payload, 0)[0]
nb = struct.unpack_from('<B', payload, 1)[0]
fc = struct.unpack_from('<B', payload, 2)[0]
ra = struct.unpack_from('<H', payload, 3)[0]
#print(fc)
if fc == 6 and ra in reg_address:
    data = struct.unpack_from('<H', payload, 5)[0]
    if data in noToSegmentHex:
        # add the str of these 2 outputs convert to int then char
        d = str(noToSegmentHex[data])
        x += d
        if len(x) == 3:
            if not ma in meters_display_registers:
                meters_display_registers[ma] = {}
            if not ra in meters_display_registers[ma]:
                meters_display_registers[ma][ra] = []

            c0 = chr(int(x))
            meters_display_registers[ma][ra] += [c0]
            x = ""

for k, v in meters_display_registers.items():
    print(f'Meter Address: {k}')
    for k0, v0 in v.items():
        print(f'Register Address: {k0}')
        print(''.join(v0))

```

```
crazy@es-base: ~/Desktop/PCAP-Generator
crazy@es-base:~/Desktop/PCAP-Generator$ python3 solve_sign_proto.py
Meter Address: 112
Register Address: 98
flag{scr0ll1ng_t3xt_g0es_z000000000m_8427e
Register Address: 97
dÅg{ÇnÃxÃeÃz000
0000 _842
Register Address: 96
dfbk{q_w,ll0wd]s8yn`i+lo_}p000)0000lc7842iy
Register Address: 99
flag{scr0ll1ng_t3xt_g0es_z000000000m_8427e}
crazy@es-base:~/Desktop/PCAP-Generator$
```