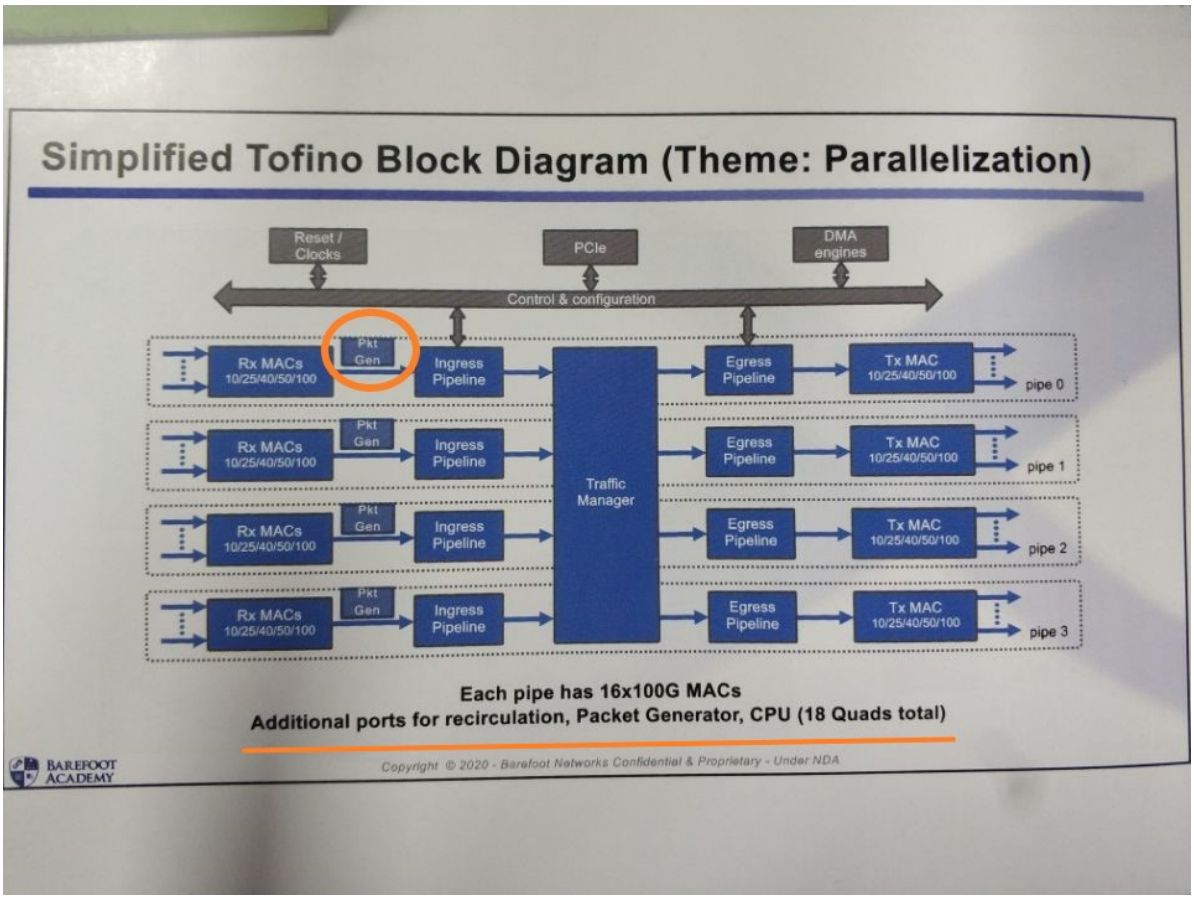
Tofino数据面数据包生成器——Pkt Gen

Pkt Gen

Tofino上实现了数据平面的数据包生成器Pkt Gen，理论上可以大大提高数据包的生成速率， 但是在Barefoot的P4讲义中对于Tofino上的数据平面的数据包生成器Pkt Gen只是一句话带过，所以针对Pkt Gen进行了进一步的调研。



不同的触发模式

```
// ------------------------------------------------------------------------
// PACKET GENERATION
// ------------------------------------------------------------------------
// Packet generator supports up to 8 applications and a total of 16KB packet
// payload. Each application is associated with one of the four trigger types:
// - One-time timer
// - Periodic timer
// - Port down
// - Packet recirculation
// For recirculated packets, the event fires when the first 32 bits of the
// recirculated packet matches the application match value and mask.
// A triggered event may generate programmable number of batches with
// programmable number of packets per batch.
```

Pkt Gen支持四种不同的触发模式：

- One-time timer
- Periodic timer
- Port down
- Packet recirculation

前两种是计时器触发，第三种是通过切换交换机端口的状态（up/down）触发生成数据包，第四种触发模式是数据包的循环触发。

不同的触发模式对应生成的数据包头部字段略有差异，但都是6个字节：

```
header pktgen_timer_header_t {
    @padding bit<3> _pad1;
    bit<2> pipe_id;                     // Pipe id
    bit<3> app_id;                      // Application id
    @padding bit<8> _pad2;

    bit<16> batch_id;                   // Start at 0 and increment to a
                                        // programmed number

    bit<16> packet_id;                  // Start at 0 and increment to a
                                        // programmed number
}

header pktgen_port_down_header_t {
    @padding bit<3> _pad1;
    bit<2> pipe_id;                     // Pipe id
    bit<3> app_id;                      // Application id
    @padding bit<15> _pad2;
    bit<9> port_num;                    // Port number

    bit<16> packet_id;                  // Start at 0 and increment to a
                                        // programmed number
}

header pktgen_recirc_header_t {
    @padding bit<3> _pad1;
    bit<2> pipe_id;                     // Pipe id
    bit<3> app_id;                      // Application id
    bit<24> key;                        // Key from the recirculated packet

    bit<16> packet_id;                  // Start at 0 and increment to a
                                        // programmed number
}
```

在计时器触发模式（其他模式类似）中，可以通过自定义batch_id和packet_id决定生成数据包的数量，数据包生成总数=batch_id * packet_id，理论上控制面一条发包指令，最多在数据面生成数据包数量为2^16 * 2^16,这是一个非常大的数字。

数据面如何生成数据包

首先通过控制面生成一个指定类型（可以实现自定义某些字段）的数据包，在Barefoot SDE中已经实现了以下类型的数据包生成函数：

- TCP
- TCPv6
- UDP
- SRv6

- GENEVE
- GRE
- GREv6
- VXLAN
- VXLANv6
- GRE ERSPAN
- IP
- IPv6
- ICMP
- ICMPv6
- ARP
- L2 Packet
- MPLS
- QINQ
- IGMP

然后将该数据包截去前6个字节（目的MAC地址）放入到缓存中，暂且称其为“模板”数据包。

```
logger.info("configure packet buffer")
pktgen_pkt_buffer_table.entry_add(
    target,
    [pktgen_pkt_buffer_table.make_key([gc.KeyTuple('pkt_buffer_offset', buff_offset),
                                       gc.KeyTuple('pkt_buffer_size', (pktlen - 6))])],
    [pktgen_pkt_buffer_table.make_data([gc.DataTuple('buffer', str(p)[6:])])])
```

为什么要截去前6个字节呢？因为Pkt Gen生成的数据包头部字段恰好是6个字节，数据面将这6个字节和缓存中的“模板”数据包拼接到一起，就生成了一个数据包。

疑问比较大的一点是，这前6个字节是如何生成的呢？因为我们发现控制面并没有做这件事，只是进行了校验的工作。通过后续的抓包分析，我们推断出这6个字节是数据面自动生成的。

校验时前6个字节的生成过程如下：

```
ipe_shift = 4
'%02x:00:%02x:%02x:%02x:%02x' % ((pipe_id << pipe_shift) | app_id,
                                 down_port >> 8,
                                 down_port & 0xFF,
                                 packet_id >> 8,
                                 packet_id & 0xFF)
```

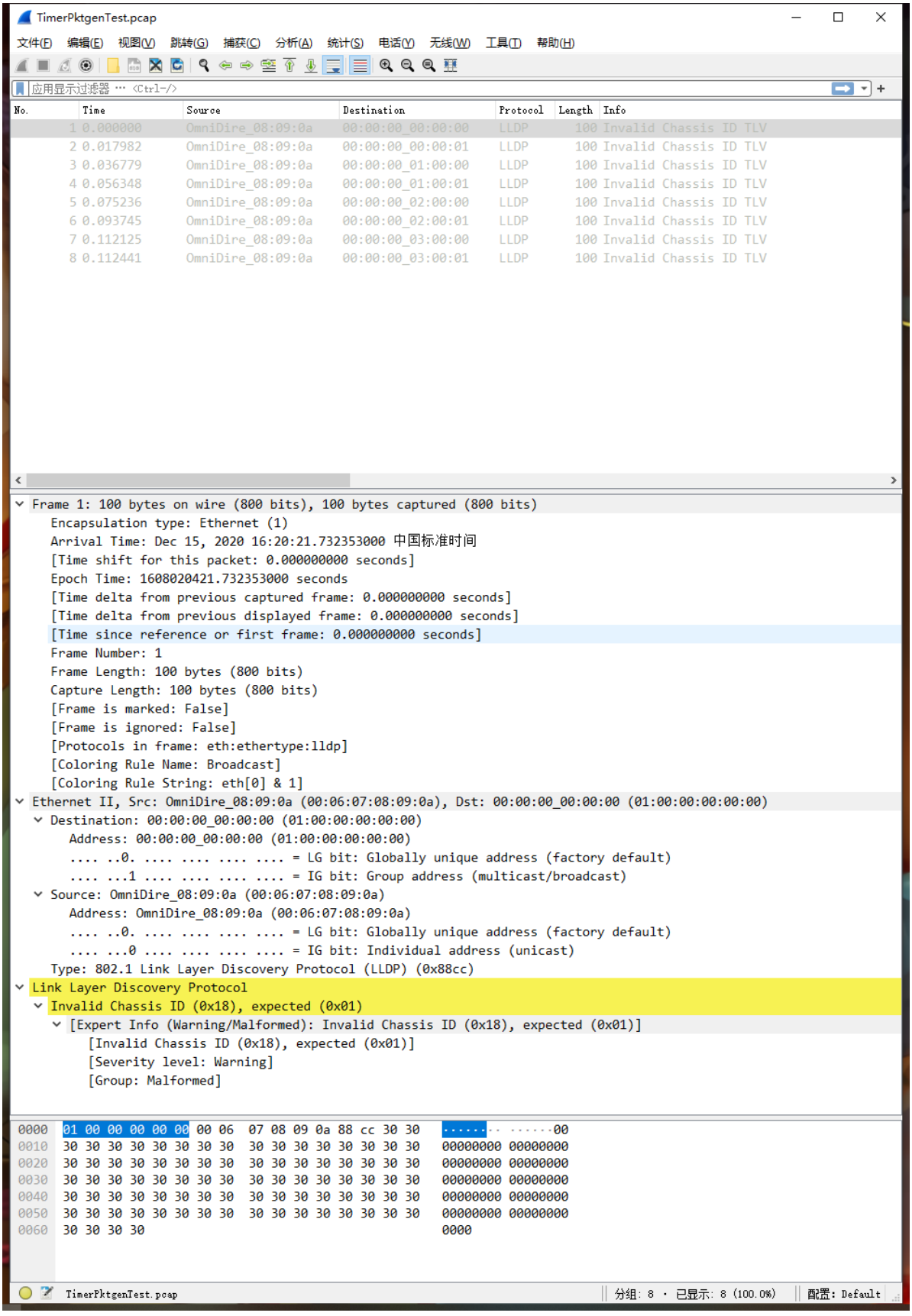
运行程序，抓包分析

tna_pktgen给出了两种触发模式的测试程序：

- TimerPktgenTest（One - time timer）
- PortDownPktgenTest（Port down）

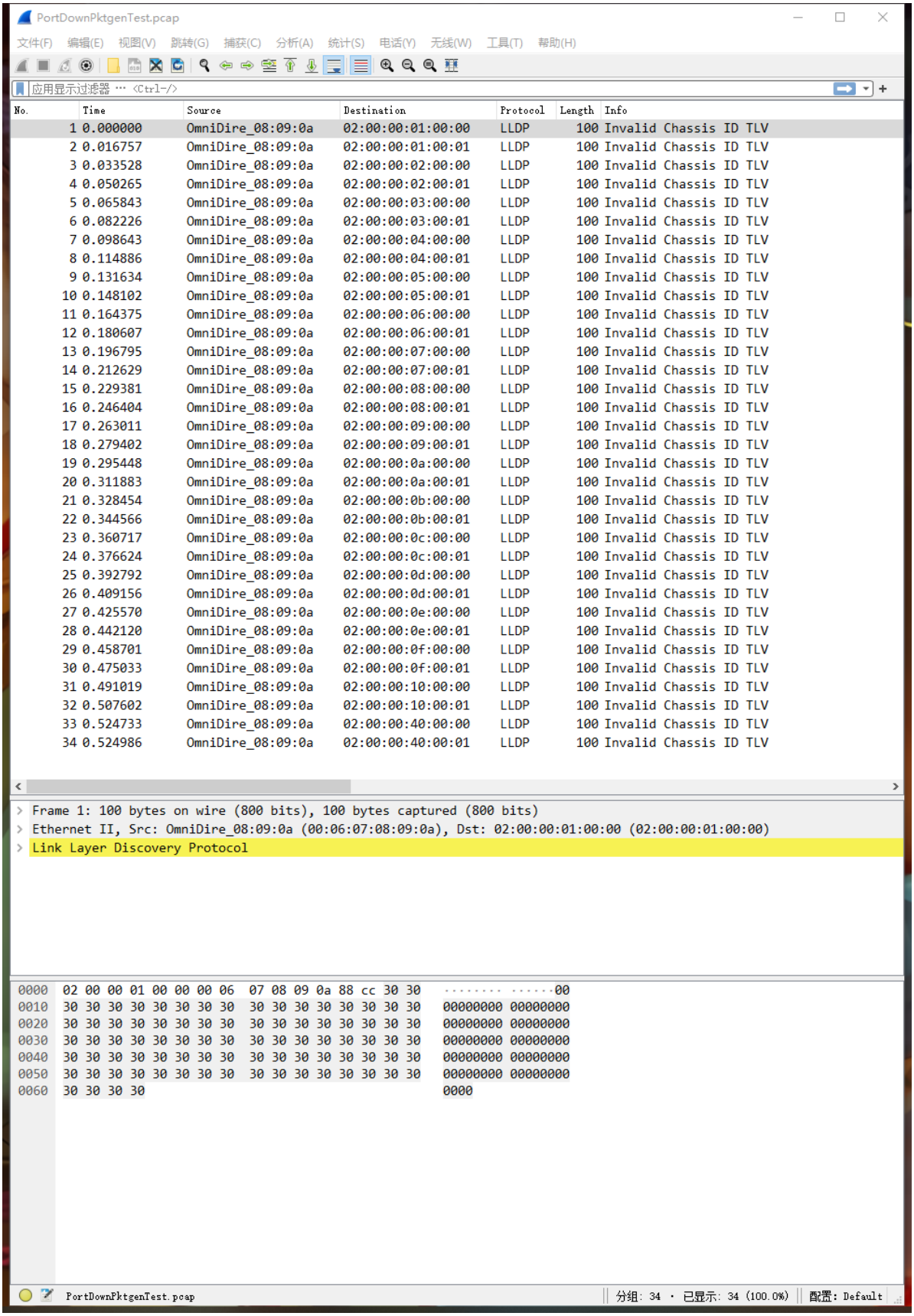
在PTF测试平台上运行测试（“模板为eth数据包”），抓取数据包：

在TimerPktgenTest测试中，自定义batch_count(4)和packet_count(2)，因此生成8个数据包。可以看到，数据包大小都是一致的，只是在目的MAC（即前6个字节）上有区别。进一步地，我们就可以发现目的MAC的规律，恰好是与batch_count（0-3）、packet_count（0-1）紧密相关。

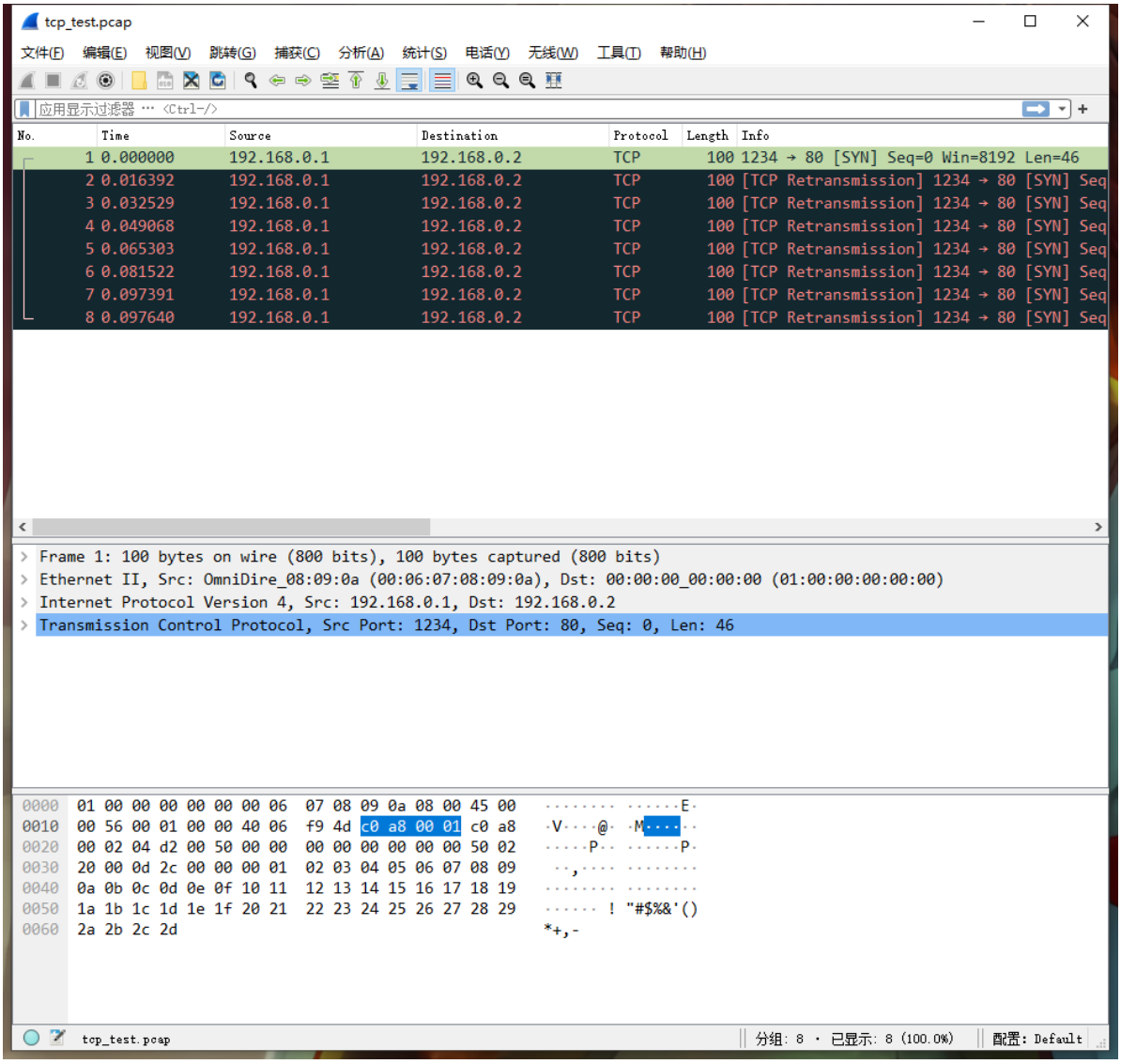


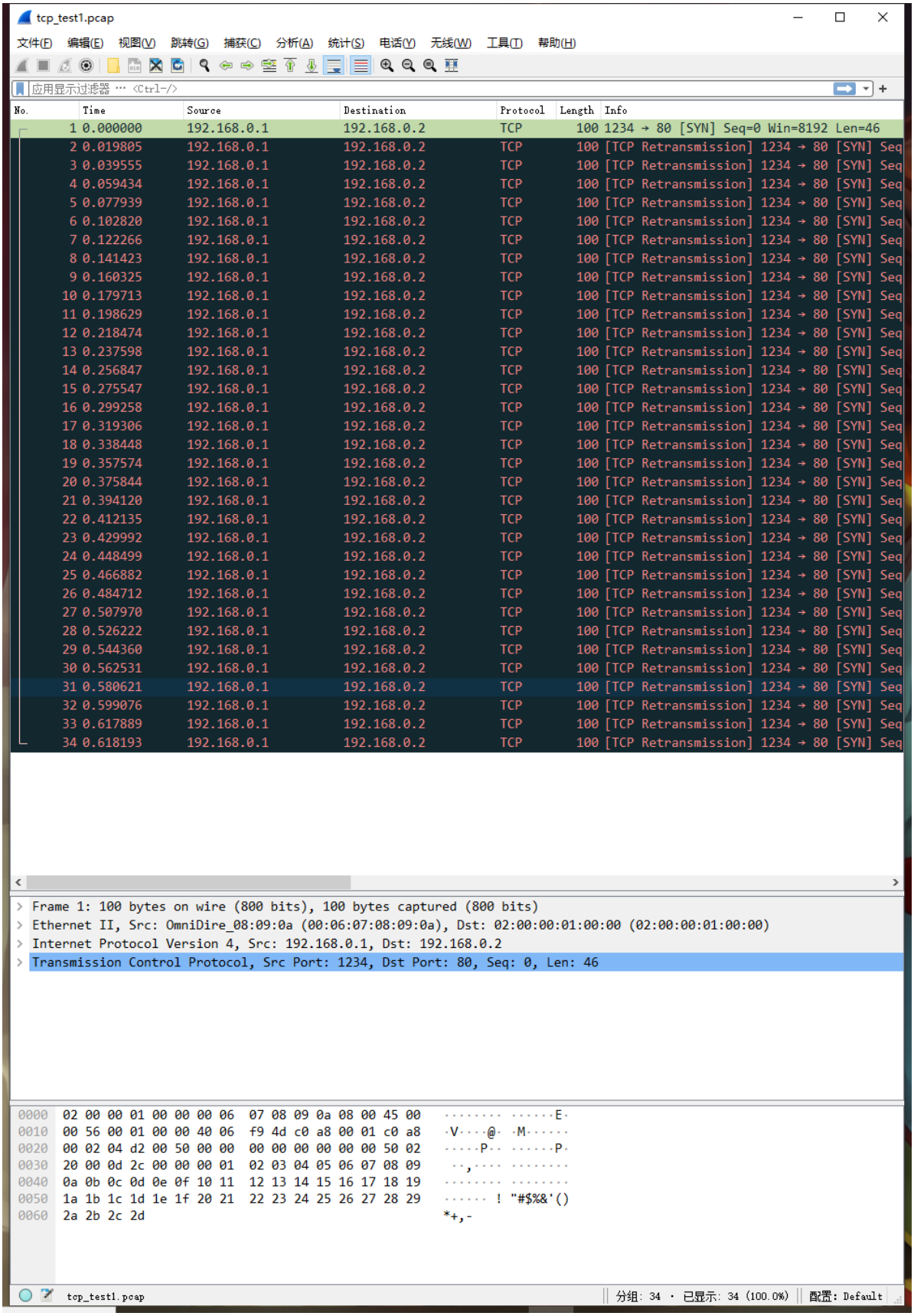
在PortDownPktgenTest测试中，扫描可用的端口数量，自定义packet_count(2)/端口，扫描到可用端口数量为17（port1-16，port64），因此生成34个数据包。

```
enable pktgen port
configure pktgen application
configure packet buffer
enable pktgen
Clear port down
Take port 1 down
()
Take port 2 down
()
Take port 3 down
()
Take port 4 down
()
Take port 5 down
()
Take port 6 down
()
Take port 7 down
()
Take port 8 down
()
Take port 9 down
()
Take port 10 down
()
Take port 11 down
()
Take port 12 down
()
Take port 13 down
()
Take port 14 down
()
Take port 15 down
()
Take port 16 down
()
Take port 64 down
()
Bring port 1 up
Bring port 2 up
Bring port 3 up
Bring port 4 up
Bring port 5 up
Bring port 6 up
Bring port 7 up
Bring port 8 up
Bring port 9 up
Bring port 10 up
Bring port 11 up
Bring port 12 up
Bring port 13 up
Bring port 14 up
Bring port 15 up
Bring port 16 up
Bring port 64 up
disable pktgen
disable port for pktgen
ok

----------------------------------------------------------------------
Ran 1 test in 1.743s

OK
➜  bf-sde-9.1.1 sudo ./run_p4_tests.sh -p tna_pktgen -t ~/bf-sde-9.1.1/pkgsrc/p4-examples/p4_16_programs/tna_pktgen -s test.TimerPktgenTest
```



后续，我又尝试修改“模板”数据包为TCP的SYN包，同样抓取到了预想数目的数据包：


tcp_test.pcap
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)
应用显示过滤器 … <Ctrl-/>
No. Time Source Destination Protocol Length Info
1 0.000000 192.168.0.1 192.168.0.2 TCP 100 1234 → 80 [SYN] Seq=0 Win=8192 Len=46
2 0.016392 192.168.0.1 192.168.0.2 TCP 100 [TCP Retransmission] 1234 → 80 [SYN] Seq
3 0.032529 192.168.0.1 192.168.0.2 TCP 100 [TCP Retransmission] 1234 → 80 [SYN] Seq
4 0.049068 192.168.0.1 192.168.0.2 TCP 100 [TCP Retransmission] 1234 → 80 [SYN] Seq
5 0.065303 192.168.0.1 192.168.0.2 TCP 100 [TCP Retransmission] 1234 → 80 [SYN] Seq
6 0.081522 192.168.0.1 192.168.0.2 TCP 100 [TCP Retransmission] 1234 → 80 [SYN] Seq
7 0.097391 192.168.0.1 192.168.0.2 TCP 100 [TCP Retransmission] 1234 → 80 [SYN] Seq
8 0.097640 192.168.0.1 192.168.0.2 TCP 100 [TCP Retransmission] 1234 → 80 [SYN] Seq
Frame 1: 100 bytes on wire (800 bits), 100 bytes captured (800 bits)
Ethernet II, Src: OmniDire_08:09:0a (00:06:07:08:09:0a), Dst: 00:00:00_00:00:00 (01:00:00:00:00:00)
Internet Protocol Version 4, Src: 192.168.0.1, Dst: 192.168.0.2
Transmission Control Protocol, Src Port: 1234, Dst Port: 80, Seq: 0, Len: 46
0000 01 00 00 00 00 00 00 06 07 08 09 0a 08 00 45 00
0010 00 56 00 01 00 00 40 06 f9 4d c0 a8 00 01 c0 a8
0020 00 02 04 d2 00 50 00 00 00 00 00 00 00 00 50 02
0030 20 00 0d 2c 00 00 00 01 02 03 04 05 06 07 08 09
0040 0a 0b 0c 0d 0e 0f 10 11 12 13 14 15 16 17 18 19
0050 1a 1b 1c 1d 1e 1f 20 21 22 23 24 25 26 27 28 29
0060 2a 2b 2c 2d
tcp_test.pcap
分组: 8 · 已显示: 8 (100.0%)
配置: Default


tcp_test1.pcap
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)
应用显示过滤器 … <Ctrl-/>
No. Time Source Destination Protocol Length Info
1 0.000000 192.168.0.1 192.168.0.2 TCP 100 1234 → 80 [SYN] Seq=0 Win=8192 Len=46
2 0.019805 192.168.0.1 192.168.0.2 TCP 100 [TCP Retransmission] 1234 → 80 [SYN] Seq
3 0.039555 192.168.0.1 192.168.0.2 TCP 100 [TCP Retransmission] 1234 → 80 [SYN] Seq
4 0.059434 192.168.0.1 192.168.0.2 TCP 100 [TCP Retransmission] 1234 → 80 [SYN] Seq
5 0.077939 192.168.0.1 192.168.0.2 TCP 100 [TCP Retransmission] 1234 → 80 [SYN] Seq
6 0.102820 192.168.0.1 192.168.0.2 TCP 100 [TCP Retransmission] 1234 → 80 [SYN] Seq
7 0.122266 192.168.0.1 192.168.0.2 TCP 100 [TCP Retransmission] 1234 → 80 [SYN] Seq
8 0.141423 192.168.0.1 192.168.0.2 TCP 100 [TCP Retransmission] 1234 → 80 [SYN] Seq
9 0.160325 192.168.0.1 192.168.0.2 TCP 100 [TCP Retransmission] 1234 → 80 [SYN] Seq
10 0.179713 192.168.0.1 192.168.0.2 TCP 100 [TCP Retransmission] 1234 → 80 [SYN] Seq
11 0.198629 192.168.0.1 192.168.0.2 TCP 100 [TCP Retransmission] 1234 → 80 [SYN] Seq
12 0.218474 192.168.0.1 192.168.0.2 TCP 100 [TCP Retransmission] 1234 → 80 [SYN] Seq
13 0.237598 192.168.0.1 192.168.0.2 TCP 100 [TCP Retransmission] 1234 → 80 [SYN] Seq
14 0.256847 192.168.0.1 192.168.0.2 TCP 100 [TCP Retransmission] 1234 → 80 [SYN] Seq
15 0.275547 192.168.0.1 192.168.0.2 TCP 100 [TCP Retransmission] 1234 → 80 [SYN] Seq
16 0.299258 192.168.0.1 192.168.0.2 TCP 100 [TCP Retransmission] 1234 → 80 [SYN] Seq
17 0.319306 192.168.0.1 192.168.0.2 TCP 100 [TCP Retransmission] 1234 → 80 [SYN] Seq
18 0.338448 192.168.0.1 192.168.0.2 TCP 100 [TCP Retransmission] 1234 → 80 [SYN] Seq
19 0.357574 192.168.0.1 192.168.0.2 TCP 100 [TCP Retransmission] 1234 → 80 [SYN] Seq
20 0.375844 192.168.0.1 192.168.0.2 TCP 100 [TCP Retransmission] 1234 → 80 [SYN] Seq
21 0.394120 192.168.0.1 192.168.0.2 TCP 100 [TCP Retransmission] 1234 → 80 [SYN] Seq
22 0.412135 192.168.0.1 192.168.0.2 TCP 100 [TCP Retransmission] 1234 → 80 [SYN] Seq
23 0.429992 192.168.0.1 192.168.0.2 TCP 100 [TCP Retransmission] 1234 → 80 [SYN] Seq
24 0.448499 192.168.0.1 192.168.0.2 TCP 100 [TCP Retransmission] 1234 → 80 [SYN] Seq
25 0.466882 192.168.0.1 192.168.0.2 TCP 100 [TCP Retransmission] 1234 → 80 [SYN] Seq
26 0.484712 192.168.0.1 192.168.0.2 TCP 100 [TCP Retransmission] 1234 → 80 [SYN] Seq
27 0.507970 192.168.0.1 192.168.0.2 TCP 100 [TCP Retransmission] 1234 → 80 [SYN] Seq
28 0.526222 192.168.0.1 192.168.0.2 TCP 100 [TCP Retransmission] 1234 → 80 [SYN] Seq
29 0.544360 192.168.0.1 192.168.0.2 TCP 100 [TCP Retransmission] 1234 → 80 [SYN] Seq
30 0.562531 192.168.0.1 192.168.0.2 TCP 100 [TCP Retransmission] 1234 → 80 [SYN] Seq
31 0.580621 192.168.0.1 192.168.0.2 TCP 100 [TCP Retransmission] 1234 → 80 [SYN] Seq
32 0.599076 192.168.0.1 192.168.0.2 TCP 100 [TCP Retransmission] 1234 → 80 [SYN] Seq
33 0.617889 192.168.0.1 192.168.0.2 TCP 100 [TCP Retransmission] 1234 → 80 [SYN] Seq
34 0.618193 192.168.0.1 192.168.0.2 TCP 100 [TCP Retransmission] 1234 → 80 [SYN] Seq
Frame 1: 100 bytes on wire (800 bits), 100 bytes captured (800 bits)
Ethernet II, Src: OmniDire_08:09:0a (00:06:07:08:09:0a), Dst: 02:00:00:01:00:00 (02:00:00:01:00:00)
Internet Protocol Version 4, Src: 192.168.0.1, Dst: 192.168.0.2
Transmission Control Protocol, Src Port: 1234, Dst Port: 80, Seq: 0, Len: 46
0000 02 00 00 01 00 00 00 06 07 08 09 0a 08 00 45 00
0010 00 56 00 01 00 00 40 06 f9 4d c0 a8 00 01 c0 a8
0020 00 02 04 d2 00 50 00 00 00 00 00 00 00 00 50 02
0030 20 00 0d 2c 00 00 00 01 02 03 04 05 06 07 08 09
0040 0a 0b 0c 0d 0e 0f 10 11 12 13 14 15 16 17 18 19
0050 1a 1b 1c 1d 1e 1f 20 21 22 23 24 25 26 27 28 29
0060 2a 2b 2c 2d
tcp_test1.pcap
分组: 34 · 已显示: 34 (100.0%)
配置: Default