Technical challenge C++ position – Forescout

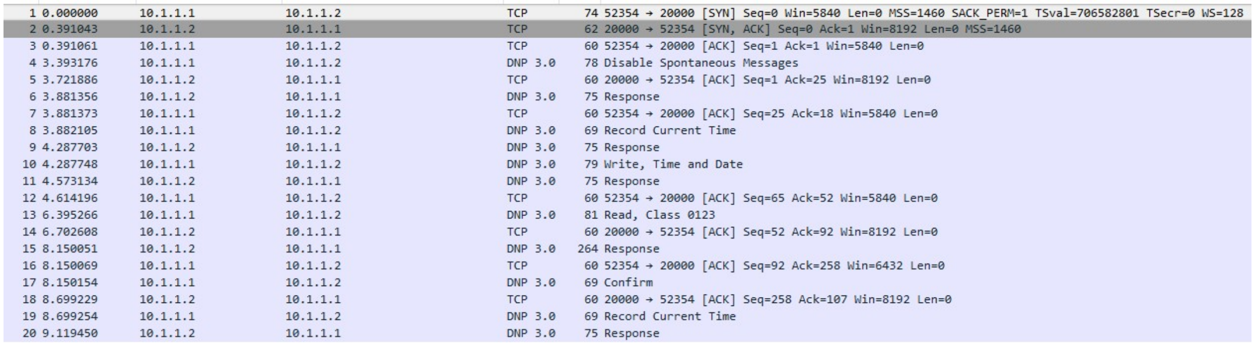
Let's pretend that you just started working at Forescout, and this challenge is the first new functionality that you are going to develop for the product. We expect that you interact with us as if you were already a Forescout employee, so please feel free to ask any questions or clarifications and be sure to keep us informed of your progress and major design choices. We are here to help!

Introduction

You already know that our product is a passive network monitoring solution, so its main purpose is to process network packets.

The product has two modes of processing packets:
Online mode where packets are forwarded by switches in the customer's network and processed immediately.
Offline mode where the packets are processed as fast as possible from a previously recorded PCAP file.

The product behavior needs to be the same for any given set of packets, no matter if they are processed in online or offline mode.



```
 1 0.000000   10.1.1.1   10.1.1.2   TCP       74 52354 → 20000 [SYN] Seq=0 Win=5840 Len=0 MSS=1460 SACK_PERM=1 TSval=706582801 TSecr=0 WS=128
 2 0.391043   10.1.1.2   10.1.1.1   TCP       62 20000 → 52354 [SYN, ACK] Seq=0 Ack=1 Win=8192 Len=0 MSS=1460
 3 0.391061   10.1.1.1   10.1.1.2   TCP       60 52354 → 20000 [ACK] Seq=1 Ack=1 Win=5840 Len=0
 4 3.393176   10.1.1.1   10.1.1.2   DNP 3.0   78 Disable Spontaneous Messages
 5 3.721886   10.1.1.2   10.1.1.1   TCP       60 20000 → 52354 [ACK] Seq=1 Ack=25 Win=8192 Len=0
 6 3.881356   10.1.1.2   10.1.1.1   DNP 3.0   75 Response
 7 3.881373   10.1.1.1   10.1.1.2   TCP       60 52354 → 20000 [ACK] Seq=25 Ack=18 Win=5840 Len=0
 8 3.882105   10.1.1.1   10.1.1.2   DNP 3.0   69 Record Current Time
 9 4.287703   10.1.1.2   10.1.1.1   DNP 3.0   75 Response
10 4.287748   10.1.1.1   10.1.1.2   DNP 3.0   79 Write, Time and Date
11 4.573134   10.1.1.2   10.1.1.1   DNP 3.0   75 Response
12 4.614196   10.1.1.1   10.1.1.2   TCP       60 52354 → 20000 [ACK] Seq=65 Ack=52 Win=5840 Len=0
13 6.395266   10.1.1.1   10.1.1.2   DNP 3.0   81 Read, Class 0123
14 6.702608   10.1.1.2   10.1.1.1   TCP       60 20000 → 52354 [ACK] Seq=52 Ack=92 Win=8192 Len=0
15 8.150051   10.1.1.2   10.1.1.1   DNP 3.0  264 Response
16 8.150069   10.1.1.1   10.1.1.2   TCP       60 52354 → 20000 [ACK] Seq=92 Ack=258 Win=6432 Len=0
17 8.150154   10.1.1.1   10.1.1.2   DNP 3.0   69 Confirm
18 8.699229   10.1.1.2   10.1.1.1   TCP       60 20000 → 52354 [ACK] Seq=258 Ack=107 Win=8192 Len=0
19 8.699254   10.1.1.1   10.1.1.2   DNP 3.0   69 Record Current Time
20 9.119450   10.1.1.2   10.1.1.1   DNP 3.0   75 Response
```

Consider the PCAP trace above. In online mode it takes roughly 9 seconds to process these 19 packets but if it is processed in offline mode it would only take a couple of milliseconds, yet the behavior in offline mode needs to be as if it took 9 seconds as well.

Challenge description

We ask you to implement an interface that allows an arbitrary piece of C++ code to be executed periodically. Let's call this a 'PeriodicTask'.

The interface needs to allow a developer to

- Add a PeriodicTask with a given interval, in seconds
- Remove a PeriodicTask, so that it is no longer executed periodically
- Change the interval of an already-existing PeriodicTask

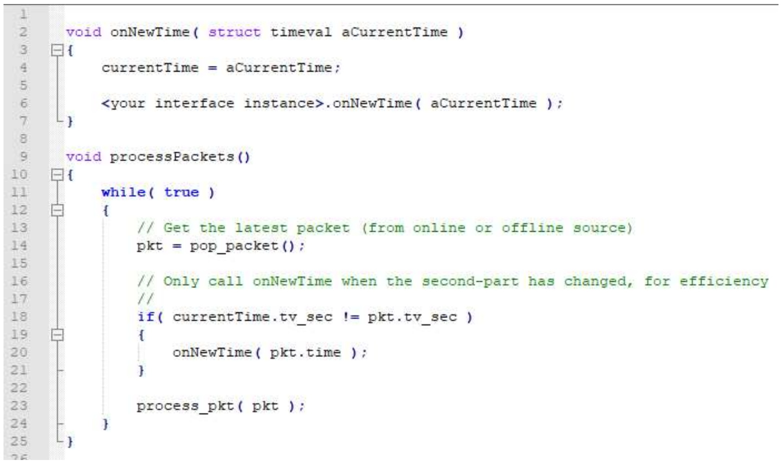
The following requirements apply:

- Adding and removing must be possible *at any time, from anywhere in the code*
- The interface must be thread-safe
- The logic must use an externally provided time and not rely on system time (see below)

In addition, your solution must

- compile using **g++-4.7** or **g++-5.4** under Ubuntu
- use modern features of the C++ language
- be unit tested (we use **Boost.Test**, feel free to use something else)
- include a **CMakeLists.txt** to build your solution
- follow a consistent style

The logic must use an externally provided time
To guarantee the same results in online and offline mode the product internally keeps track of the current time, based on the timestamp of the latest network packet.



```
void onNewTime( struct timeval aCurrentTime )
{
    currentTime = aCurrentTime;

    <your interface instance>.onNewTime( aCurrentTime );
}

void processPackets()
{
    while( true )
    {
        // Get the latest packet (from online or offline source)
        pkt = pop_packet();

        // Only call onNewTime when the second-part has changed, for efficiency
        //
        if( currentTime.tv_sec != pkt.tv_sec )
        {
            onNewTime( pkt.time );
        }

        process_pkt( pkt );
    }
}
```

Roughly speaking the code follows the logic above. *processPackets* is executed in each thread that process network traffic. An ‘onNewTime’ function is called whenever it processes a packet whose time in seconds in different than the previous time in seconds.

Your interface should expose the same onNewTime function and use it to drive the logic of calling the correct PeriodicTasks.