
# Laziness is a virtue

## Telemetry with Python, Pexpect, EasySNMP, and python-rrdtool

www.viewqwest.com

Tan Shao Yi

# What my team deals with

**We use DWDM and SDH to transport telecoms traffic over the Causeway, Second Link and MIC1/MIC2 submarine cable systems**

- OTU4/2, 100GE, 10GE, STM64, STM1/4/16 protocols and various Ethernet bit rates

**Challenges:**

- Optical transport equipment are quite unlike the L2/L3 equipment most network engineers are familiar with
- Optical transport engineers are in short supply [in Singapore]

# Challenges with optical transport network equipment

- **Proprietary NMSes**
- **Limited machine-machine telemetry interfaces**
  - 1984: Bellcore's cross-vendor, cross-technology Transaction Language 1 (TL1) provides full FCAPS
  - Somewhere in between: SNMP, proprietary CLI
  - More recent: REST APIs
- **Various optical transport network vendors organise/model output differently** 😡😡😡
  - Don't expect even a simple SNMP GET output to be organised in the same way

# SNIPS: Systems & Network Integrated Polling System

- Developed during my days in Pacnet to monitor Unix and Windows systems and L3 network equipment.
  - Now adapted to monitor optical transport network equipment
- Originally in Perl, Expect, Net::SNMP, RRDtool and PHP
- Internals re-adapted in Python, Pexpect, EasySNMP and python-rrdtool
  - https://pypi.org/project/pexpect/
  - https://pypi.org/project/easysnmp/
  - https://pypi.org/project/python-rrdtool/
- Provides a single view of the network, harmonising telemetry from various equipment types and brands

# What we monitor?

Optical transport network equipment from

- BTI Systems (now Juniper Networks)
- Tellabs, Coriant (now Infinera)
- ADTRAN (since phased-out)

L2/3 network equipment from

- Cisco Systems
- Juniper Networks

Some Unix-based services

# Collecting telemetry

```
import argparse
import collections
import os
import pathlib
import pexpect
import random
import re
import shutil
import sys
import threading
import time

def read_hosts(hosts):
   # Gather list of hosts that you need to monitor

def login_to_equipment(host, session):
   # Different equipment login differently

def logout_from_equipment(host, session):
   # Different equipment logout differently

def collect_alarms(host, commands):
   # Main code to collect alarms. Separated so we can thread
```

# Collecting telemetry

```
# Program begins here.

parser=argparse.ArgumentParser(description='Logs into the
transmissions equipment to constantly gather alarms')
parser.add_argument('-d', dest='debug', action='store_true',
help='debug mode')

args=parser.parse_args()
debug=args.debug

# hosts collection that contains login credentials
hosts=collections.defaultdict(dict)
hosts=read_hosts(hosts)

# commands collection that contains the commands to be executed
commands=collections.defaultdict(dict)
commands=read_commands(commands)

threads={}
```

```
# Start collector threads
for hostname in hosts:
   if hostname.startswith('A') or hostname.startswith('B') or
hostname.startswith('C'):
      threads[hostname]=threading.Thread(target=collect_alarms,
                        args=(hosts[hostname],
                        commands[hostname]), daemon=True)
      threads[hostname].start()
      # Sleep a while to prevent a barrage of login requests
      time.sleep(random.randint(1, SLEEP_BETWEEN_LOGINS+1))

# Let's keep the threads alive
while True:
    for hostname in threads:
        if threads[hostname].is_alive()==False:
           threads[hostname]=threading.Thread(
                              target=collect_alarms,
                              args=(hosts[hostname],
                              commands[hostname]), daemon=True)
            threads[hostname].start()
            time.sleep(random.randint(1, SLEEP_BETWEEN_LOGINS+1))

    time.sleep(SLEEP_THREADS_KEEPALIVE_CHECK)

exit(0)
```

# Collecting telemetry

```
def login_to_equipment(host, session):
   tries=0
   while tries<RETRIES:
      tries+=1
      try:
         if host['hostname'].startswith('A'):
            session=pexpect.spawn('%s %s %s' % (TELNETCMD,
                    host['hostname'], TYPEAPORT),
                    timeout=TIMEOUT)
            session.expect('PROMPT> ')

            session.send('…')

          elif host['hostname'].startswith('B'):
            session=pexpect.spawn('%s %s %s' % (TELNETCMD,
                    host['hostname'], TYPEBPORT),
                    timeout=TIMEOUT)
            session.expect('Escape character is')

            session.sendline('ACT-USER::'+host['userid']+':'
            +str(host['tl1commandcounter'])
            +'::PASSWORD='+host['password']+';')
            session.expect('M  '+str(host['tl1commandcounter'])
            +' COMPLD')

            session.send('…')
```

```
          elif host['hostname'].startswith('C'):
            session=pexpect.spawn('%s -l %s %s' % (SSHCMD,
                    host['userid'], host['hostname']),
                    timeout=TIMEOUT)

            session.expect('\[ ne \]')
            session.expect('> ')

            session.send('…')

         return session
      except pexpect.EOF:
         print('Exception: EOF during login_to_equipment')
      except pexpect.TIMEOUT:
         print('Exception: TIMEOUT during login_to_equipment')
      except Exception as e:
         print('Exception: str(e)+' during login_to_equipment')
      if tries<RETRIES:
         time.sleep(SLEEP)
   return None
```

# Telemetry using EasySNMP

```
try:
   session=easysnmp.Session(hostname=host['hostname'], community=host['snmppassword'], version=2, timeout=SNMPTIMEOUT, retries=RETRIES)
   # 32-bit OIDs
   items=session.walk('.1.3.6.1.2.1.2.2')
   # 64-bit OIDs
   items+=session.walk('.1.3.6.1.2.1.31.1.1.1')

   …
   except easysnmp.EasySNMPConnectionError:
      print('Exception: '+host['hostname']+' encountered SNMP Connection Error')
   except easysnmp.EasySNMPTimeoutError:
      print('Exception: '+host['hostname']+' encountered SNMP Timeout Error')
   except easysnmp.EasySNMPUnknownObjectIDError:
      print('Exception: '+host['hostname']+' encountered SNMP Unknown Object Error')
   except easysnmp.EasySNMPNoSuchObjectError:
      print('Exception: '+host['hostname']+' encountered SNMP No Such Object Error')
   except easysnmp.EasySNMPNoSuchInstanceError:
      print('Exception: '+host['hostname']+' encountered SNMP No Such Instance Error')
   except Exception as e:
      print('Exception: '+host['hostname']+' encountered '+str(e))
```

# Graphing with python-rrdtool

```
try:
  rrdtool.create( dfile,
     '--step', '300',
     'DS:traffic_in:COUNTER:600:0:'+str(rrdmax),
     'DS:traffic_out:COUNTER:600:0:'+str(rrdmax),
     'RRA:LAST:0.5:1:600',
     'RRA:LAST:0.5:6:700',
     'RRA:LAST:0.5:24:775',
     'RRA:LAST:0.5:1440:3985',
     'RRA:AVERAGE:0.5:1:600',
     'RRA:AVERAGE:0.5:6:700',
     'RRA:AVERAGE:0.5:24:775',
     'RRA:AVERAGE:0.5:1440:3985',
     'RRA:MAX:0.5:1:600',
     'RRA:MAX:0.5:6:700',
     'RRA:MAX:0.5:24:775',
     'RRA:MAX:0.5:1440:3985')

  except rrdtool.ProgrammingError:
     print('Exception: '+hostname+' '+interface+' encountered
           Programming Error')
  except rrdtool.OperationalError:
     print('Exception: '+hostname+' '+interface+' encountered
           Operational Error')
  except Exception as e:
     print('Exception: '+hostname+' '+interface+' encountered
          '+str(e))
```

```
  rrdtool.update(dfile,
              '--daemon', RRDCACHEDSOCK,
              'N:'+str(results[hostname][interface]['bytesreceived'])
             +':'+str(results[hostname][interface]['bytessent']))

  except rrdtool.ProgrammingError:
     print('Exception: '+hostname+' '+interface+' encountered
           Programming Error')
  except rrdtool.OperationalError:
     print('Exception: '+hostname+' '+interface+' encountered
           Operational Error')
  except Exception as e:
     print('Exception: '+hostname+' '+interface+' encountered
          '+str(e))
```

# Takeaways

- Systematic naming convention for devices
- Thread where possible for scalability
  - Consider using Celery (http://www.celeryproject.org) if we need more advanced task/job queuing?
- Choose the most efficient mechanism
  - REST may be sexy but sometimes CLI (or some other mechanism) can be less CPU-intensive and/or quicker
  - Your code should be flexible enough to handle different mechanisms


Thank You
Singapore
20 Bendemeer Road, #01-09, Singapore 339914
tansy@viewqwest.com