Network Trace analysis using Python

Nevil Brownlee

U Auckland | *WAND*

NZNOG 2015 Tutorial
26 January 2015

Introduction

- Using Network Traces
  - There are lots of tools
    - tcpdump, wireshark, libtrace, python-libtrace, ...
  - Why use python?
    - to answer questions involving *big* traces
    - to produce reports, plot, etc that are specific to your site/network/user(s)
- Assumptions
  - You understand network protocols well
  - You've already tried using python

Python – Nevil's view

- python is deliberately simple-minded
  - It forces you to write many simple lines
  - Indenting as syntax (!)
    - at least for classes and function declarations
    - ; can separate multiple statements on same line
    - emacs has syntax-colouring, and commands to move blocks of lines in or out
- python has a huge collection of modules
  - We'll only look at a few of them
    - python-libtrace (of course)
    - numpy, scipy and mathplotlib

Python – Nevil's view (2)

- python has lots of built-in functions
  - You often need to use them for common operations, e.g. enumerate() to step through a python dictionary (i.e. hash)
- python *objects* have a big set of pre-defined functions
  - e.g. for comparison and iteration
  - you have to understand these, and use them!

Libtrace

- Web page
  http://wand.net.nz/trac/libtrace
- C library for analysing packet traces
- Reads and write compressed trace files directly (.gz or .bz2)
- URI specifies a 'trace',
  e.g. pcap:test.pcap.gz
- pcap:, pcapfile: or erf: for trace *files*
- live interfaces
  - linux int:, ring:, pcapint: BSD bpf:
  - documented in SupportedTraceFormats

Libtrace utilities

- tracesplit
  - Can collect new traces from an interface
    - tracesplit -c 10000 -m 1 -Zgzip -z5 pcap:eth5 \
      pcapfile:10kpackets.pcap.gz
    - reads packets from a pcap interface, writes 10kpackets to a single compressed pcap file
  - Can also split a trace file into smaller files
- traceanon
  - Anonymises IP addresses in packet headers
    - traceanon -sd -c"x yz" pcapfile:10kp-raw.pcap \
      pcapfile:10kp-anon.pcap
    - -c "key" uses cryptopan with key 'x yz'
    - -sd anonymises both source and destination addresses

python-libtrace (plt)

- Python module providing access to fields in packets via libtrace
- plt provides a clean, object-oriented view of packets
  - Network layers are subclasses of Packet class
- Includes pldns and natkit
  - python access to NLnetLabs ldns C library
  - natkit; a collection of 'useful' tools for network analysis, i.e.
    - get 2- and 4-byte integers from a ByteArray
    - TCP sequence number arithmetic
    - classes for building flow tables

Installing python-libtrace

- Libtrace
  - Check that you have libz and libbz2
  - Download latest libtrace from
    - research.wand.net.nz/software/libtrace.php
  - Follow instructions in INSTALL file
- ldns
  - Requires latest version of openssl
  - Download ldns C library from
    - www.nlnetlabs.nl/projects/ldns
- python
  - Requires python-dev
  - Can build for python 2 or 3 (I use python 2)

Installing python-libtrace (2)

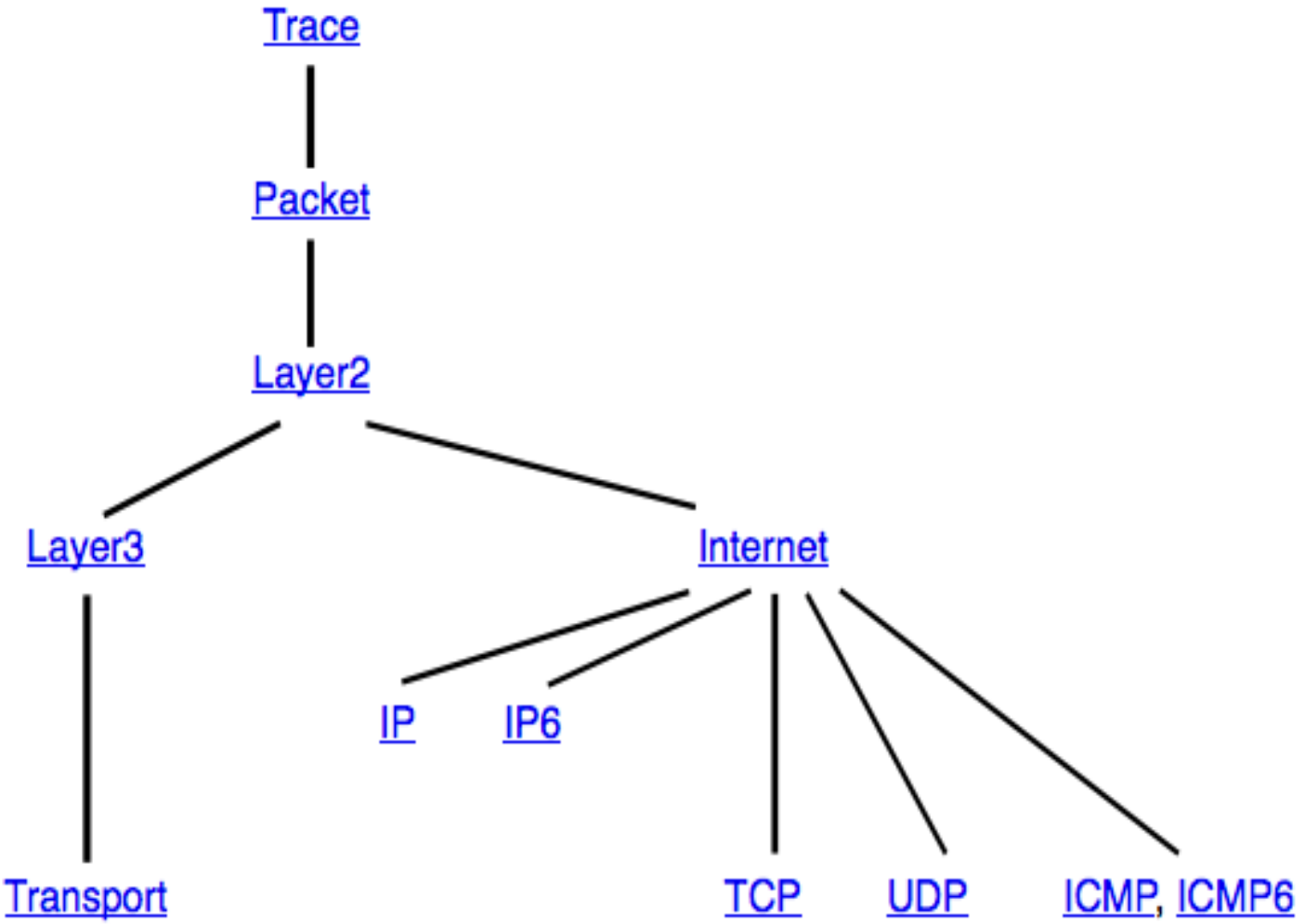
- python-libtrace
  - Download latest python-libtrace (plt) from
    - www.cs.auckland.ac.nz/~nevil
  - Follow instructions in INSTALL file
    - tar zxf python-libtrace-x.y.tgz (currently 1.4)
    - cd python-libtrace-1.4
    - make install-py2 # for python 2 # or py3
  - Install will run tests, don't panic if some fail
    - Tests compare output of test programs on your system with output on my development system
    - Please send bug reports to me so that I can improve the testing!
      - Nevil Brownlee <n.brownlee@auckland.ac.nz>

python-libtrace documentation

- html documentation included in the distribution tarball, along with some simple example programs
  - In python-libtrace-1.4/doc
  - Also on web at
    - www.cs.auckland.ac.nz/~nevil/python-libtrace
  - A page for each part or subclass within plt

plt overview

- plt provides a class hierarchy for a Packet



Tutorial plt programs

- This tutorial provides a set of programs, intended to show how to use python-libtrace (plt)
- My example traces have been anonymised using traceanon
  - 10,000 packets from a network edge, snap length 80 (i.e. only first 80 bytes)
  - smaller anonymised DNS traces
- As we work through them, I'll explain how they work, and the python and plt features they use ...

p01_read_test.py

- Create a trace, start it
- Read its packets, count them
- Note:
  - import plt to use python-libtrace
  - Specify the trace URI
  - start() the trace
    - must do this before trying to use it
  - iterate through the traces Packets
    - python iterator loop using 'in'
  - close the trace (function with no parameters)
  - print the count; printf-style, format using '%'

p02_count_ethertypes.py

- Count the number of packets for each ethertype in a trace
- Note:
  - ; separating two statements on same line
  - python dictionary for ethertypes seen
  - dictionary keys must be Strings (immutable)
  - value of dictionary items is just an integer
  - print dictionary in sorted() order
    - no parameters → increasing order of item values
  - tuple of objects to print (et, ethertypes[et])

p03_count_protos.py

- Count the number of packets for each IP or IP6 protocol in a trace
- Note:
  - nested if statements
  - python dictionary for ethertypes seen
  - trace contains IP and IP6 packets
  - print dictionary in sorted() order
    - key= expects a function parameter, protocols.get is a function that gets the value for each key
    - reverse=True for descending order ('T' for python true)

p04_transit_ip_pkts.py

- Count packets that are transiting a 'home' network
- Notes:
  - Using IPprefix methods, imported by plt
  - from_s() to make an IPprefix for 'home'
  - ignore IP6 packets in this example
  - have to set src_ and dst_prefix length to 32
  - home.is_prefix(a) tests whether home is a prefix of a, i.e. a lies within home
  - print src_ and dst_prefixes for each new 'foreign' packet

p05_tcp_fin_vs_reset.py

- Count the FIN and RESET flags in the trace's TCP packets
- Notes:
  - pkt.tcp gets a TCP object from a packet, it returns False if it wasn't TCP

p06_http_fin_vs_reset.py

- Separate http FIN and RESTS counts from total FIN and RESET counts
- Notes:
  - Same as p05, but tests for http first

p07_tcp_port_counts.py

- Looks for ports with highest byte counts
- Notes:
  - class port_counts to hold information for each port
    - class functions (methods) have self as first parameter
    - instance variables are prefixed with self.
    - __init()__ creates a class object
    - __str()__ prints the object
  - sorted()'s key is an anonymous function
    - here k is lambda's only parameter

p08_ports_fin_vs_reset.py

- Count FIN and RESET flags for each TCP port
- Notes:
  - Combination of p07 and p05

p09_pldns_demo.py

- Demonstration of pldns – working with DNS packets
- Notes:
  - must import pldns – it's not part of plt
  - reads 1kp-dns-anon file, 1k full DNS records
  - pldns.ldns() makes a pldns object from a packet's UDP payload
    - ldns expects a complete packet!
    - pldns has functions that return (python) lists of LdnsRR objects
    - an LdnsRR object has attributes that return information about a DNS RR

p10_pldns_count_dnssec.py

- Count DNS records that contain DNSSEC RRs
- Notes:
  - tuple for RR types (integers)
  - gets authority list of RRs for each packet
  - searches it for an RR in the tuple

p11_dns_find_our_servers.py

- Counts the nameservers in our 'home' network
- Notes:
  - combination of p04 and p10
  - uses pldns to look for DNS request src_dests, i.e. incoming requests from other networks
  - counts are high for our site nameservers, but there are lots of unanswered requests to other hosts !?

p12_dns_server_users.py

- Count users (i.e. requesting hosts) of a nameserver
- Notes:
  - h = str() # Need to tell python we want a string

p13_dns_response_lengths.py

- Plot distribution of DNS response lengths
- Notes:
  - gets lengths of DNS response packets in a python list
  - converts that to a numpy array
  - uses numpy and scipy modules to print statistics of the distribution
  - uses mathplotlib module to plot it

p14_ipflow_demo.py

- Demonstrates natkit's IPflow class
- Notes:
  - build IPflow object from packet
  - trap exceptions from non-ip packet
  - fwd_key is a ByteArray containing
    - version, protocol, src/dst ports, src/dst addresses
  - rev_key has same data, but src and dst fields are swapped

p15_find_biggest_flows.py

- Builds a flow table, then prints data about the largest few flows
- Notes:
  - class flow holds the flow's IPflow, plus fwd/rev byte counts
  - 'fwd' direction is that of flow's first packet
  - direction matching algorithm is from RFC 2722 (1999)

p16_biggest_home_flows.py

- Uses 'FlowHome' objects in a flow table
- We only need one lookup for each packet
- Notes:
  - class flow has
    - separate functions count_in and count_out
    - sort_key function returns flow's total bytes
    - is_inward() is True if only one of src/dst addresses is in one of our home networks
    - sorted's lambda function uses flow.sort_key()

Summary

- It's fun to work with plt, pldns, natkit
- These example are just a beginning - what would you find useful in your own network?

- Thanks for the feedback on the NZNOG list!
- Please email comments, suggestions, bug reports, etc to
  - n.brownlee@auckland.ac.nz