Mobile Measurement of Path Transparency

Iain R. Learmonth    Gorry Fairhurst
{iain,gorry}@erg.abdn.ac.uk

University of Aberdeen
MAMI Project

2nd Workshop on Mobile Network Measurement (MNM’18)
June 25, 2018
Vienna, Austria

This project has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No 688421. The opinions expressed and arguments employed reflect only the authors’ view. The European Commission is not responsible for any use that may be made of that information.
Active Measurement of Path Transparency

Methodology:
1. Throw packets at the Internet
2. See what happens

Ideal: two-ended A/B testing
Scalable: one-ended A/B testing
Multiple sources: isolate on-path from near-target impairment

Figure: One-Ended vs. Two-Ended Testing
The original implementation supported by mPlane/RITE

Three distinct components:
- DNS List Resolver
- QoF Flow Meter
- Active Traffic Generator

Used hardcoded `sysctl(1)` and `iptables(1)` commands to cause packets to be emitted with various ECN-related flags

Source code:
https://github.com/britram/ecnspider

Figure: Original Architecture
**History**

*ecnspider* Results

- ECN negotiation was found to be successful for 56.17% of hosts connecting for IPv4, 65.41% for IPv6, from the Alexa top 1 million list [4]

- This continues a trend ETH started observing with *ecnspider* in 2013 [2]

**Figure**: ECN Support in the Alexa Top 1 Million
PATHspider 1.0

https://github.com/mami-project/pathspider/tree/1.0.1

- Architecture based closely on the original ecnspider
- Generalised to support more than just ECN
  - Added TCP Fast Open and DiffServ Codepoints
- Still performing A/B testing, but with more A/B tests
- Replaced QoF with a Python flowmeter implementation using python-libtrace
- Began to develop a generalised measurement methodology for path transparency testing
- Published at 2016 Applied Networking Research Workshop [3]
The plugin architecture was not as generalised as it could have been.

Plugin methods:

- config_zero
- config_one
- connect
- post_connect
- create_observer
- merge
**Built-In Flowmeter**

- PATHspider’s built in flow meter is extensible via the plugin architecture
- Using python-libtrace to dissect packets, any flow property imaginable can be reported back based on the raw packets:
  - ECN negotiation (IP/TCP headers)
  - Bleaching of bits, dropping of options
  - Checksum recalculations
PATHspider 1.0 Results

We presented some initial findings along with the publication of PATHspider 1.0 [3]:

Explicit Congestion Notification (ECN)

State of ECN server-side deployment, as measured from a Digital Ocean vantage point in Amsterdam on 13th June 2016:

<table>
<thead>
<tr>
<th></th>
<th>IPv4</th>
<th>IPv6</th>
<th>all</th>
</tr>
</thead>
<tbody>
<tr>
<td>No ECN connectivity issues</td>
<td>99.5%</td>
<td>99.9%</td>
<td>99.5%</td>
</tr>
<tr>
<td>ECN successfully negotiated</td>
<td>70.0%</td>
<td>82.8%</td>
<td>70.5%</td>
</tr>
</tbody>
</table>

ECN negotiation by Alexa rank bin:

DiffServ Code Points (DSCP)

Initial study: 10,006 of 96,978 (10.31%) of Alexa Top 100k websites had unexpected, non-zero DSCP values. More measurement was needed to better characterize these anomalies.

TCP Fast Open (TFO)

Initial study: 330 IPv4 and 32 IPv6 addresses of Alexa Top 1M are TFO-capable (of which 278 and 28 respectively are Google properties). DDoS prevention services, enterprise firewalls, and CPE tend to interfere with TFO. More measurement was necessary to analyze impairments.
PATHspider 2.0

- Architecture changed to add a flow combiner
- Generalised to support more than just A/B testing
  - Any permutation of any number of tests
- Replaced PATHspider’s HTTP code with cURL
- Added framework for packet forging based plugins using Scapy
- Completely rewritten (in Go) target list resolver
- Observer modules usable for standalone passive observation or analysis
- Source code: https://github.com/mami-project/pathspider/tree/2.0.0/
Plugin Types

- Synchronised (traditional ecnspider)
  - ECN, DSCP
- Desynchronised (traditional ecnspider, no configurator)
  - TFO, H2, TLS NPN/ALPN
- Forge (new in PATHspider 2.0!)
  - Evil Bit, UDP Zero Checksum, UDP Options
- Single (new, and fast)
  - Various TCP Options
Instead of writing client code, use the code that already exists

In the `pathspider.helpers` module:
- DNS (dnscrypt)
- HTTP/HTTPS (pycURL)
- TCP (Python socket)

For synchronised plugins, just use the helper

For desynchronised plugins, the helpers are customisable, e.g. cURL helpers accept arbitrary CURLOPTs
Synchronized Spider plugins use **built-in connection methods** along with **global system configuration** to change the behaviour of the connections.

Configuration functions are at the heart of a Synchronized Spider plugin.

Configuration functions may make calls to `sysctl` or `iptables` to make changes to the way that traffic is generated.

One function should be written for each of the configurations and PATHspider will ensure that the configurations are set before the corresponding traffic is generated. It is the responsibility of plugin authors to ensure that any configuration is reset by the next configuration function if that is required.
Synchronized Plugin

```python
class ECN(SynchronizedSpider, PluggableSpider):
    def config_no_ecn(self): # pylint: disable=no-self-use
        ""
        Disables ECN negotiation via sysctl.
        ""

        logger = logging.getLogger('ecn')
        subprocess.check_call(
            ['/sbin/sysctl', '-w', 'net.ipv4.tcpecn=2'],
            stdout=subprocess.DEVNULL,
            stderr=subprocess.DEVNULL)
        logger.debug("Configurator disabled ECN")

    def config_ecn(self): # pylint: disable=no-self-use
        ""
        Enables ECN negotiation via sysctl.
        ""

        logger = logging.getLogger('ecn')
        subprocess.check_call(
            ['/sbin/sysctl', '-w', 'net.ipv4.tcpecn=1'],
            stdout=subprocess.DEVNULL,
            stderr=subprocess.DEVNULL)
        logger.debug("Configurator enabled ECN")

configurations = [config_no_ecn, config_ecn]
```

Listing 1: Configuration Functions for the ECN Plugin
Desynchronized Plugin

- DesynchronizedSpider plugins modify the connection logic in order to change the behaviour of the connections. There is no global state synchronisation and so a DesynchronizedSpider can be more efficient than a SynchronizedSpider.
- Connection functions are at the heart of a DesynchronizedSpider plugin.
- These use a connection helper (or custom connection logic) to generate traffic towards with a target to get a reply from the target.
- One function should be written for each connection to be made, usually with at least two functions to provide a baseline followed by an experimental connection.
class H2(DesynchronizedSpider, PluggableSpider):

def conn_no_h2(self, job, config):
    # pylint: disable=unused-argument
    curlopts = {}  # pylint: disable=unused-variable
    curlinfos = [pycurl.INFO_HTTP_VERSION]
    if self.args.connect == "http":
        return connect_http(self.source, job, self.args.timeout, curlopts, curlinfos)
    if self.args.connect == "https":
        return connect_https(self.source, job, self.args.timeout, curlopts, curlinfos)
    else:
        raise RuntimeError("Unknown connection mode specified")

def conn_h2(self, job, config):
    # pylint: disable=unused-argument
    curlopts = {pycurl.HTTP_VERSION: pycurl.CURL_HTTP_VERSION_2_0}
    curlinfos = [pycurl.INFO_HTTP_VERSION]
    if self.args.connect == "http":
        return connect_http(self.source, job, self.args.timeout, curlopts, curlinfos)
    if self.args.connect == "https":
        return connect_https(self.source, job, self.args.timeout, curlopts, curlinfos)
    else:
        raise RuntimeError("Unknown connection mode specified")

connections = [conn_no_h2, conn_h2]

Listing 2: Connection Functions for the H2 Plugin
Forge Spider plugins use Scapy to send forged packets to targets.

The heart of a ForgeSpider is the `forge()` function.

This function takes two arguments, the job containing the target information and the sequence number.

This function will be called the number of times set in the packets metadata variable and `seq` will be set to the number of times the function has been called for this job.
Single Plugin

- SingleSpider uses the built-in connection helpers to make a single connection to the target which is optionally observed by Observer chains.
- This is the simplest model and only requires a `combine_flows()` function to generate conditions from the connection helper output and flow record output from the Observer.
Observer Modules

- While these used to be part of plugins in PATHspider 1.0, they are now independent and so can be reused across multiple plugins:
  - BasicChain, DNSChain, DSCPChain, ECNChain, EvilChain, ICMPChain, TCPChain, TFOChain
- These can also be used together, limiting each chain to just a single layer and letting the combiner produce conditions
- Chains can produce information to be consumed by other chains later in the list
- These can be used independently of a PATHspider measurement:

```bash
irl@z~$ pspdr observe tcp ecn
```

Listing 3: Running the PATHspider Observer independently
Target List Resolution

- Hellfire is a parallelised DNS resolver. It is written in Go and for the purpose of generating input lists to PATHspider, though may be useful for other applications.

- Can use many sources for inputs:
  - Alexa Top 1 Million Global Sites
  - Cisco Umbrella 1 Million
  - Citizen Lab Test Lists
  - OpenDNS Public Domain Lists
  - Comma-Separated Values Files
  - Plain Text Domain Lists
Target List Resolution
Using *hellfire*

```
irl@z:~$ hellfire
Usage:

hellfire --topsites [--file <filename>] [--output <individual | array | oneeach>] [--type <host | ns | mx>] [--canid <canid address>]
hellfire --cisco [--file <filename>] [--output <individual | array | oneeach>] [--type <host | ns | mx>] [--canid <canid address>]
hellfire --citizenlab [--country <cc>] [--file <filename>] [--output <individual | array | oneeach>] [--type <host | ns | mx>] [--canid <canid address>]
hellfire --.opendns [--list <name>] [--file <filename>] [--output <individual | array | oneeach>] [--type <host | ns | mx>] [--canid <canid address>]
hellfire --csv [--file <filename>] [--output <individual | array | oneeach>] [--type <host | ns | mx>] [--canid <canid address>]
hellfire --txt [--file <filename>] [--output <individual | array | oneeach>] [--type <host | ns | mx>] [--canid <canid address>]
```

**Listing 4:** hellfire’s Usage Help

```
irl@z~$ hellfire --cisco
```

**Listing 5:** Start Resolving the Cisco Umbrella List
Packet Forging

- PATHspider uses the Scapy library for Python for packet forging
- This is the most flexible method of creating new measurement plugins for PATHspider
Make a Packet

- Scapy packets are constructed layer by layer
- While you can specify raw bytes, Scapy provides a number of useful classes for common protocols, which makes things a lot easier
Make a Packet

Scapy must be launched with sudo as we will need to use “raw” sockets to emit forged packets.

```
irl@z:~$ sudo scapy3

Welcome to Scapy
Version 2.4.0
https://github.com/secdev/scapy
Have fun!
We are in France, we say Skappee.
OK? Merci.
— Sebastien Chabal

using IPython 5.5.0
```

Listing 6: Launching Scapy
Make a Packet
IPv4 Header - Create and Dissect

```
>>> IP()
<IP |>
>>> i = IP()
>>> i.summary()
'127.0.0.1 > 127.0.0.1 hopopt'
>>> i.display()
###[ IP ]###
    version= 4
    ihl= None
    tos= 0x0
    len= None
    id= 1
    flags= 0
    frag= 0
    ttl= 64
    proto= hopopt
    chksum= None
    src= 127.0.0.1
    dst= 127.0.0.1
\options\```

Listing 7: Creating and Dissecting an IPv4 Header
Make a Packet
IPv4 Header - Customize

```python
>>> i = IP(src="192.0.2.1", dst="198.51.100.1", ttl=10)
>>> i.summary()
'192.0.2.1 > 198.51.100.1 hopopt'
```

Listing 8: Customizing an IPv4 Header
Listing 9: Create a PDF Export of a Dissection of the IP Header

Figure: PDF Export of IP Header Dissection
Listing 10: Creating and Dissecting a TCP Header

```python
>>> TCP()
<TCP |>
>>> t = TCP()
>>> t.summary()
'TCP ftp_data > http S'
>>> t.display()
###[ TCP ]###
  sport= ftp_data
dport= http
  seq= 0
  ack= 0
  dataofs= None
  reserved= 0
  flags= S
  window= 8192
  chksum= None
  urgp= 0
  options= []
```
Make a Packet

TCP Header: Customizing

```python
>>> t = TCP(dport=443)
>>> t
<TCP dport=https |>
>>> t.summary()
'TCP ftp_data > https S'
>>> t.display()
###[ TCP ]###
  sport= ftp_data
dport= https
  seq= 0
  ack= 0
dataofs= None
  reserved= 0
  flags= S
window= 8192
chksum= None
urgptr= 0
options= []
```

Listing 11: Customizing a TCP Header
The `/` operator is used to join layers together.

Scapy will automatically set fields, such as the IP Protocol field, when you do this.

When dissecting, Scapy will automatically choose the dissector to use based on fields such as the IP Protocol field.

```python
>>> p = i / t
>>> p.summary()
'IP / TCP 192.0.2.1:ftp_data > 198.51.100.1:https S'
>>> p.display()
[... output snipped ...]
```

Listing 12: Sticking the IP and TCP Headers Together
```
>>> wrpcap("/tmp/scapy.pcap", [p])
```

Listing 13: Exporting a PCAP File from Scapy

Figure: Dissection of the packet created in Scapy, in Wireshark
Send a Packet

- The `sr1()` function sends a single packet, and returns a single packet if a reply is received.
- Start Wireshark capturing before executing the `sr1()` function.

```python
>>> p=IP ( dst="139.133.210.32")/TCP()
>>> a=sr1(p)

Begin emission:
. Finished sending 1 packets.

* Received 2 packets, got 1 answers, remaining 0 packets

>>> a
<IP version=4 ihl=5 tos=0x0 len=44 id=0 flags=DF frag=0 ttl=47 proto=tcp chksum=0xa98d src=139.133.210.32 dst=172.22.152.130 options=[] |<TCP sport=http dport=ftp_data seq=3081101820 ack=1 dataofs=6 reserved=0 flags=SA window=29200 chksum=0xe9c7 urgptr =0 options=[('MSS', 1452)] |<Padding load=':v' |

>>> a.summary()
'IP / TCP 139.133.210.32:http > 172.22.152.130:ftp_data SA / Padding'
```

Listing 14: Create and Send an IP/TCP Packet
The evil bit is a fictional IPv4 packet header field proposed in RFC 3514 [1], a humorous April Fools’ Day RFC from 2003 authored by Steve Bellovin. The RFC recommended that the last remaining unused bit, the "Reserved Bit," in the IPv4 packet header be used to indicate whether a packet had been sent with malicious intent, thus making computer security engineering an easy problem — simply ignore any messages with the evil bit set and trust the rest.

– Wikipedia
The flags in the IP header are just an attribute you can modify:

```python
>>> i = IP()
>>> i.flags = 'evil'
```

Listing 15: Setting the Evil Bit on an IPv4 Header with Scapy
PATHspider Plugins

ForgeSpider

```python
class EvilBit(ForgeSpider, PluggableSpider):

    name = "evilbit"
    description = "Evil bit connectivity testing"
    version = '0.0.0'
    chains = [BasicChain, TCPChain, EvilChain]
    connect_supported = ['tcpsyn']
    packets = 2

    def forge(self, job, seq):
        ...
```

Listing 16: Outline for Evil Bit plugin using ForgeSpider
def forge(self, job, seq):
    sport = 0
    while sport < 1024:
        sport = int(RandShort())
    l4 = (TCP(sport=sport, dport=job['dp']))
    if '::' in job['dp']:
        ip = IPv6(src=self.source[1], dst=job['ip'])
    else:
        ip = IP(src=self.source[0], dst=job['ip'])
    if seq == 1:
        ip.flags = 'evil'
    return ip/l4

Listing 17: Creating Packets With and Without the Evil Bit
MONROE’s objective is to design and operate the first European transnational open platform for independent, large-scale monitoring and assessment of performance of MBB networks in heterogeneous environments.

https://www.monroe-project.eu/
PATHspider on MONROE

- PATHspider is not particularly lightweight
- Return to a split model:
  - Active traffic generation and packet capture
  - Offline analysis using PATHspider
Observations from PATHspider

- Path (e.g. source and destination IP address)
- Condition (e.g. ecn.negotiation.succeeded)
with PATHspider, we’ve seen:
  - measurements from a single machine
  - in a single run

How to compare measurements...
  - from multiple vantage points
  - across longer time scales?

Answer: centralise analysis in an observatory

https://observatory.mami-project.eu/
Active Measurement of Path Transparency

Methodology:
1. Throw packets at the Internet
2. See what happens

Ideal: two-ended A/B testing
Scalable: one-ended A/B testing
Multiple sources: isolate on-path from near-target impairment

Figure: One-Ended vs. Two-Ended Testing
Learn more about how to use PATHspider!

SIGCOMM Tutorial on

Repeatability and Comparability in Measurement (RCM)
on August 20, 2018, 2pm-5:45pm, Budapest,

• Part I: Introduction and Topology Measurement
  • Welcome and Introduction (Brian Trammell, ETH Zurich)
  • Tracebox: Topology Measurement and Impairment Discovery (Korian Edeline, U. Liege)

• Part II: Path Transparency and Data Collection
  • PATHspider: A Tool for Controlled Hybrid Measurement (Iain R. Learmonth, U. Aberdeen)
  • Observatories: Collection, Preservation, Metadata and Provenance for Active Measurement (Brian Trammell, ETH Zurich)
  • The Path Transparency Observatory (Brian Trammell, ETH Zurich)

See https://conferences.sigcomm.org/sigcomm/2018/tutorial-rcm.html
References

