

# PCAP stream processing

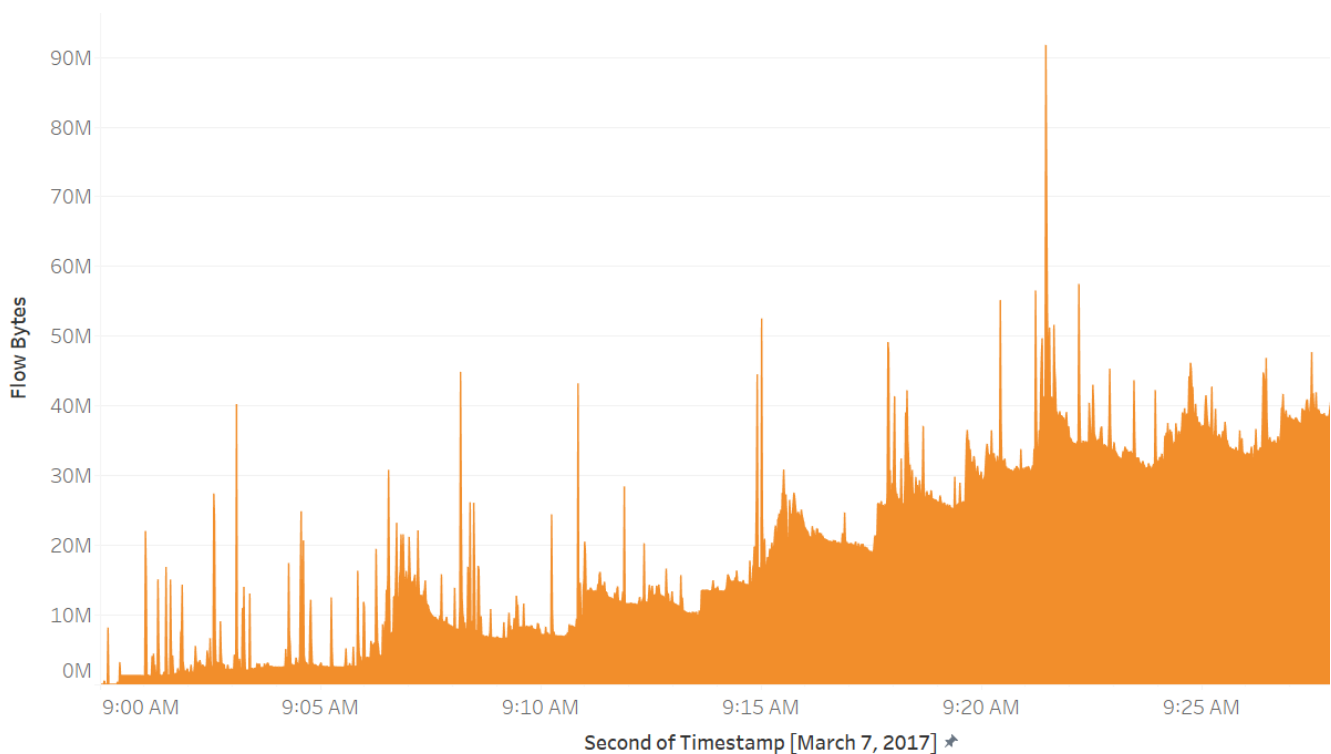
John Oberlin

Submission for DATA 603 Platforms for Big Data Processing, University of Maryland Baltimore County

In [this directory](#), one will find directories for the servers instance-1 and cluster-1-m, holding the essential programs to run the project, as well as an appendix directory that holds supplemental information, code, and visuals mentioned within this document. This document is written in Markdown and is best viewed in Chrome with the [Markdown Viewer](#) extension installed. Otherwise, see the PDF of the same name.

## Overview

This project is a proof of concept of real-time parsing and storage of enterprise network traffic. Processing network traffic flow is an unstructured data problem, in which 10's or 100's of megabytes could be easily passing a sensor per second (Figure 1). Exploring such technical infrastructure is important for real-time machine learning applications for detecting cyber exploits. However, PCAP is only one data source among several streaming sources that can enrich cyber operations analyses. A full implementation would include data streams via Kafka topics from sources such as log files and help desk tickets.



**Figure 1:** Packet bytes per second from an analysis of the CICDataset's Labelled Flows (Sharafaldin, Lashkari, and Ghorbani, 2018). The analysis calculated the average bytes per second of the full volume of the flow via the timestamp, flow duration, and flow bytes per second features in the flow-level traffic report. See [Monday-WorkingHours.pcap\\_ISCX\\_sample.csv](#), which was downloaded from [205.174.165.80/CICDataset/CIC-IDS-2017/Dataset/](#) (GeneratedLabelledFlows.zip).

The datasets used in this project are PCAP files of capture data of generated network activity. [The Intrusion Detection Evaluation Dataset](#) (CICIDS2017) is hosted by the University of New Brunswick: Canadian Institute for Cybersecurity. The dataset consists of full packet payloads in pcap format, along with labeled network flows and extracted datasets for machine learning (sets of packet exchanges between hosts). However, for this particular project, the interest is in the original raw packet activity within the passage of time. The set has a full week's worth of traffic. One day is labeled as benign, and other

days include cyber attacks. Two of the files, a benign day and a DDoS attack day was downloaded to instance-1, the simulated switch server with the `wget` command.

```
[{ "file": "Monday-WorkingHours.pcap",
  "url": "http://205.174.165.80/CICDataset/CIC-IDS-2017/Dataset/PCAPs/Monday-WorkingHours.pcap",
  "GB": "10",
  "exploit": "benign"},
{ "file": "Wednesday-WorkingHours.pcap",
  "url": "http://205.174.165.80/CICDataset/CIC-IDS-2017/Dataset/PCAPs/Wednesday-WorkingHours.pcap",
  "GB": "12",
  "exploit": "DDoS attack"}]
```

## Architecture overview

Without access to an enterprise network switch, this project simulates the traffic by replaying the CICDataset on a cloud server, which listens to the traffic and sends unstructured packet lines to a Spark cluster for processing. The master node then stores the lines transformed as key-value pairs (Figure 2).

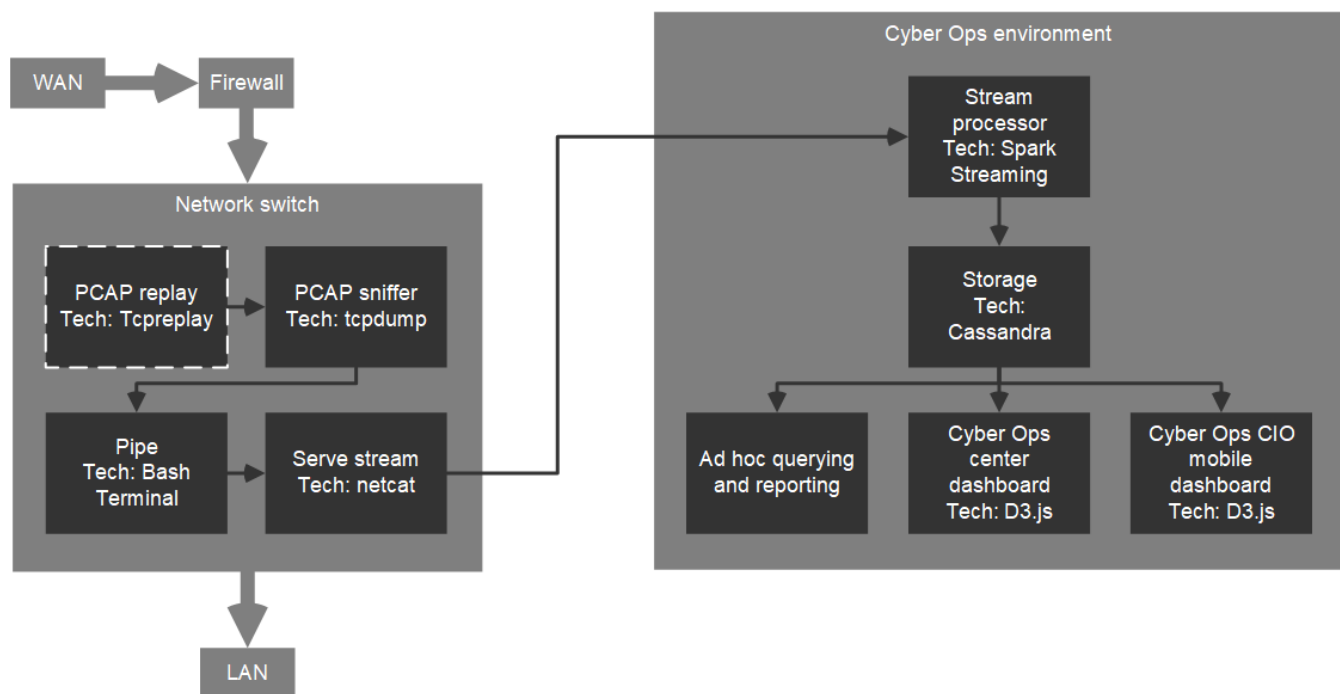


Figure 2

## Network switch

The simulated network switch, named instance-1, replays the PCAP files with the command line tool `tcpreplay`. "The basic operation of `tcpreplay` is to resend all packets from the input file(s) at the speed at which they were recorded, or a specified data rate, up to as fast as the hardware is capable" (`tcpreplay` man page, Debian). Another common networking tool, `tcpdump` captures the replayed packets. (Note that the tool also captures actual instance-1 SSH session traffic, which could be filtered out.) The standard output is piped to yet another networking tool, `netcat`. Netcat serves the data on a specified port for consumption of the Spark cluster. It may be noted that `netcat` is a quick solution and that a queuing or messaging service ought to be implemented. A service like Kafka provides scalability and high-availability message queues on clusters, with a level of security. The following configuration includes the installation of Kafka on instance-1; however it is not implemented in the current project version.

## Configuration

- Infrastructure: [Google Cloud Platform Compute Engine](#)

- Name: instance-1
- Operating system: Debian
- vCPU: 1\*
- Memory: 3.75 GB
- Disk 30GB

\*tcpreplay's CPU usage (97%) probably calls for increasing the number of cores (Figure 3).

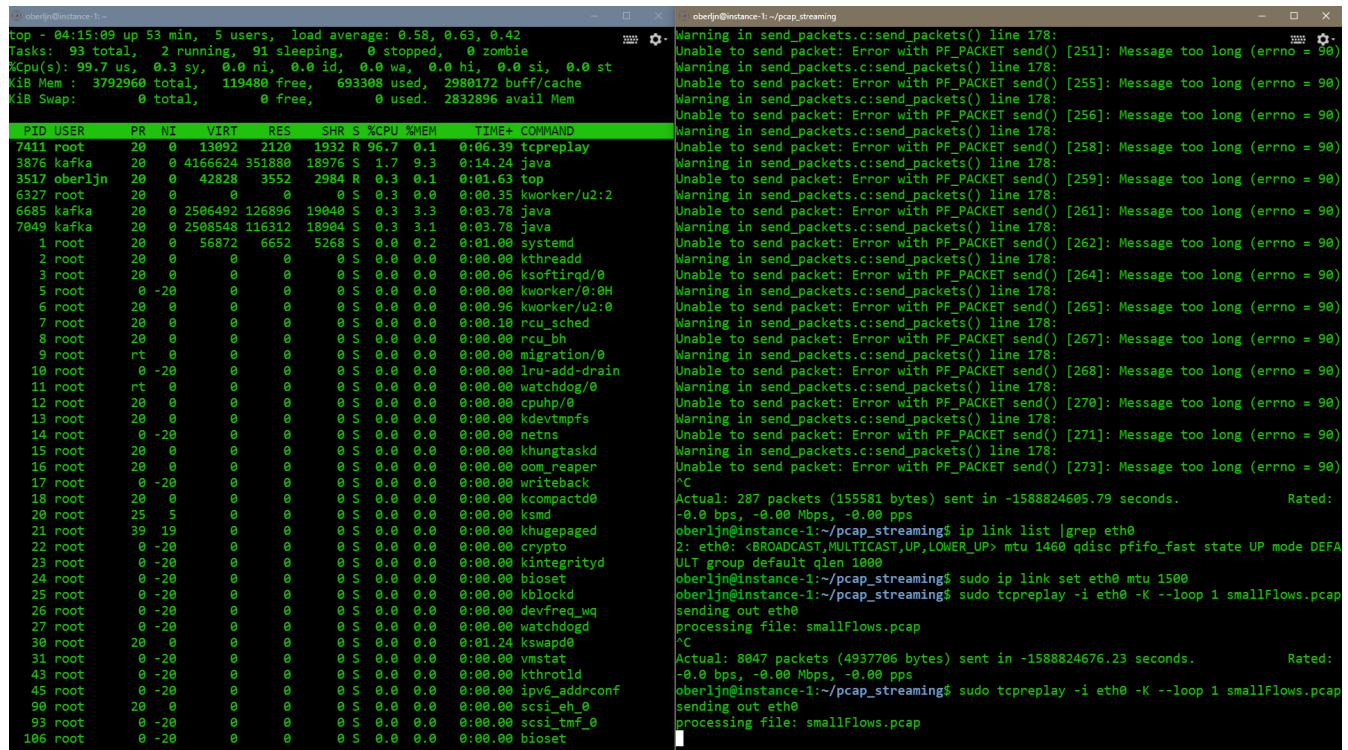


Figure 3 The left session running the "top" command shows the CPU usage of tcpreplay. It is undetermined what results from tcpreplay maxing out the CPU, such as dropped or delayed packets.

## Resize the disk

Storing the two CICDataset PCAP files requires at least 25GB available. The disk must be resized and filesystem extended (Resizing the file system, Google Cloud).

Instance's disk page -> Edit -> input new size -> Save

Check disks with

```
sudo df -h
sudo lsblk
```

Make sure growpart from cloud-guest-utils is installed:

```
sudo apt-get install cloud-guest-utils
```

resize with device ID and partition number.

```
sudo growpart /dev/sda 1
```

Extend the file system in order to use the additional space.

```
sudo resize2fs /dev/sda1
```

## Install tcpdump, tcpreplay, and Java JDK

Java is required for Kafka.

```
sudo apt-get update
sudo apt-get upgrade

sudo apt-get install tcpreplay
sudo apt-get install tcpdump
sudo apt install default-jdk
```

## Install and configure Kafka

Make Kafka user and give sudo privileges.

```
sudo useradd kafka -m
sudo passwd kafka
sudo adduser kafka sudo
```

Log into user and download and extract the Kafka binaries. Check that the binary version exists at [downloads.apache.org/kafka](https://downloads.apache.org/kafka).

```
su -l kafka
mkdir ~/Downloads
curl "https://downloads.apache.org/kafka/2.5.0/kafka_2.13-2.5.0.tgz" -o ~/Downloads/kafka.tgz
mkdir ~/kafka
cd ~/kafka
tar -xvzf ~/Downloads/kafka.tgz --strip 1
```

Configure the Kafka server by editing its properties.

```
vim ~/kafka/config/server.properties
```

Add to the end of the file:

```
delete.topic.enable = true
```

Create the systemd unit files for zookeeper and Kafka. Upload [zookeeper.service](#) and [kafka.service](#) to the instance. Copy to system directory.

```
sudo cp ../oberljn/zookeeper.service /etc/systemd/system/zookeeper.service
sudo cp ../oberljn/kafka.service /etc/systemd/system/kafka.service
```

This daemon reload may be necessary.

```
sudo systemctl daemon-reload
```

Start servers.

```
sudo systemctl start zookeeper
sudo systemctl start kafka
```

Check that Kafka is running.

```
sudo journalctl -u kafka
```

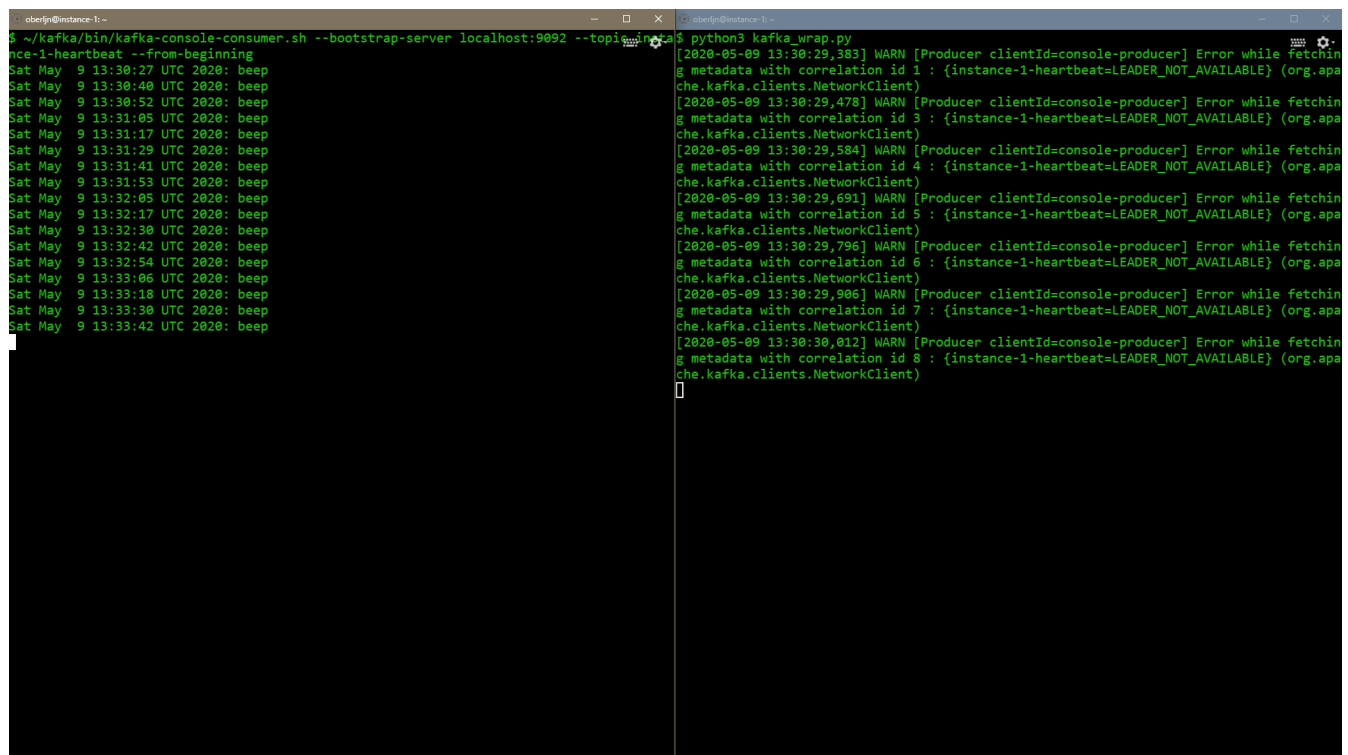
Enable Kafka on boot.

```
sudo systemctl enable kafka
```

Make a heartbeat topic that will be part of the cyber operation's monitoring of its infrastructure. Such a topic might include timestamped CPU and memory usage stats via a tool like sysstat.

```
~/kafka/bin/kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 1 --partitions 1 --topic instan
```

This Python script [kafka\\_heartbeat.py](#) is a wrapper around the Kafka publish command that sends a stats message every N seconds (Figure 4).



```
oberlin@instance-1: ~$ ~/kafka/bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic instance-1-heartbeat --from-beginning
Sat May 9 13:30:27 UTC 2020: beep
Sat May 9 13:30:40 UTC 2020: beep
Sat May 9 13:30:52 UTC 2020: beep
Sat May 9 13:31:05 UTC 2020: beep
Sat May 9 13:31:17 UTC 2020: beep
Sat May 9 13:31:29 UTC 2020: beep
Sat May 9 13:31:41 UTC 2020: beep
Sat May 9 13:31:53 UTC 2020: beep
Sat May 9 13:32:05 UTC 2020: beep
Sat May 9 13:32:17 UTC 2020: beep
Sat May 9 13:32:30 UTC 2020: beep
Sat May 9 13:32:42 UTC 2020: beep
Sat May 9 13:32:54 UTC 2020: beep
Sat May 9 13:33:06 UTC 2020: beep
Sat May 9 13:33:18 UTC 2020: beep
Sat May 9 13:33:30 UTC 2020: beep
Sat May 9 13:33:42 UTC 2020: beep

oberlin@instance-1: ~$ python3 kafka_wrap.py
[2020-05-09 13:30:29,383] WARN [Producer clientId=console-producer] Error while fetching metadata with correlation id 1 : {instance-1-heartbeat=LEADER_NOT_AVAILABLE} (org.apache.kafka.clients.NetworkClient)
[2020-05-09 13:30:29,478] WARN [Producer clientId=console-producer] Error while fetching metadata with correlation id 3 : {instance-1-heartbeat=LEADER_NOT_AVAILABLE} (org.apache.kafka.clients.NetworkClient)
[2020-05-09 13:30:29,584] WARN [Producer clientId=console-producer] Error while fetching metadata with correlation id 4 : {instance-1-heartbeat=LEADER_NOT_AVAILABLE} (org.apache.kafka.clients.NetworkClient)
[2020-05-09 13:30:29,691] WARN [Producer clientId=console-producer] Error while fetching metadata with correlation id 5 : {instance-1-heartbeat=LEADER_NOT_AVAILABLE} (org.apache.kafka.clients.NetworkClient)
[2020-05-09 13:30:29,796] WARN [Producer clientId=console-producer] Error while fetching metadata with correlation id 6 : {instance-1-heartbeat=LEADER_NOT_AVAILABLE} (org.apache.kafka.clients.NetworkClient)
[2020-05-09 13:30:29,906] WARN [Producer clientId=console-producer] Error while fetching metadata with correlation id 7 : {instance-1-heartbeat=LEADER_NOT_AVAILABLE} (org.apache.kafka.clients.NetworkClient)
[2020-05-09 13:30:30,012] WARN [Producer clientId=console-producer] Error while fetching metadata with correlation id 8 : {instance-1-heartbeat=LEADER_NOT_AVAILABLE} (org.apache.kafka.clients.NetworkClient)
```

Figure 4

## Simulating the traffic, capturing packets, and serving the stream

It was found that, to use Kafka, tcpdump is piped to netcat, which pipes to the Kafka producer script for the topic. For example, after making a topic "instance-1-pcap", in an instance-1 session, sniff network traffic with tcpdump and pipe to port 4444 with netcat.

```
sudo tcpdump -i eth0 -nn --dont-verify-checksums | netcat localhost 4444
```

Here, switch `-i` to point tcpdump to the ethernet interface `eth0`, `-nn` to disable IP and port name resolution, and `v` to increase the verbosity of the output.

In another session, listen on port 4444 with netcat and pipe lines to Kafka.

```
netcat -l -p 4444 | ~/kafka/bin/kafka-console-producer.sh --broker-list localhost:9092 --topic instance-1-pcap > /dev
```

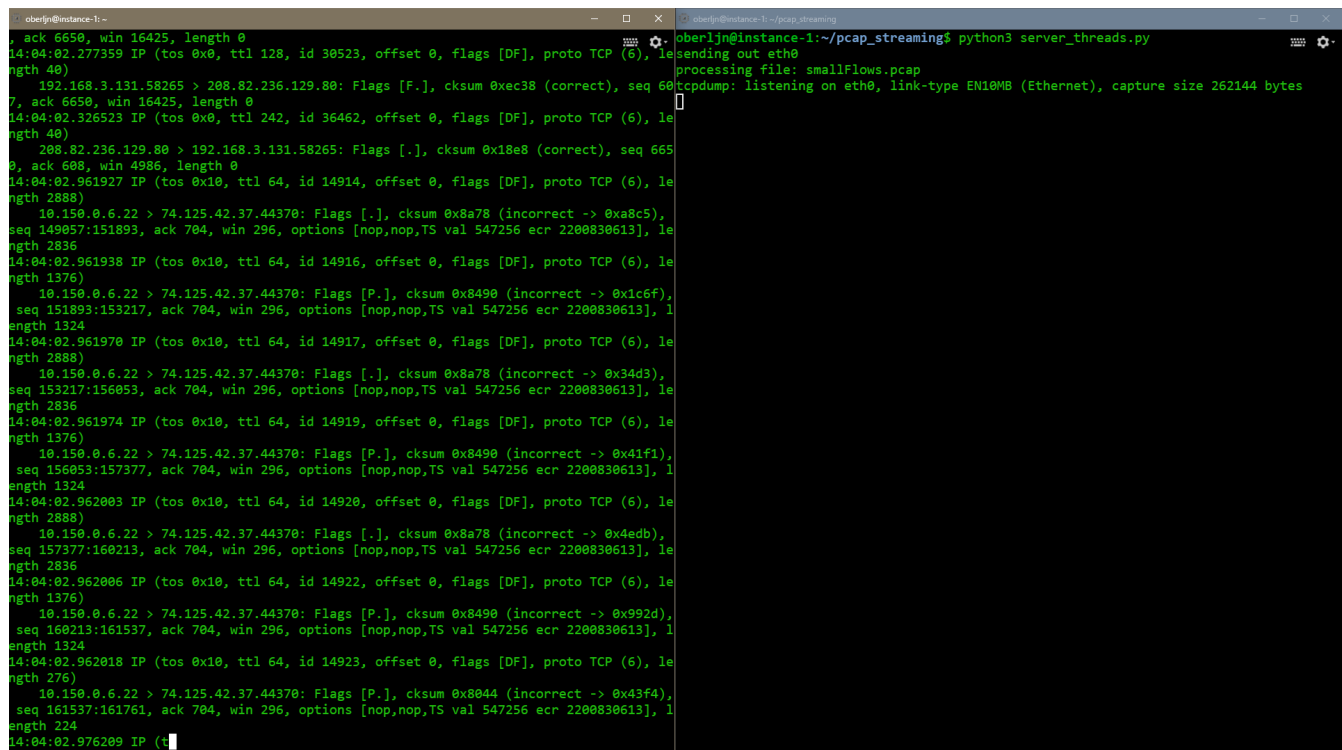
However, due to errors when trying to consume the Kafka topic from the Spark cluster, and a lack of time for troubleshooting, this project opts to use only netcat to serve the data, for illustrative purpose.

Along with the tcpdump and netcat commands above, tcpreplay is ran on a third session:

```
sudo tcpreplay -i eth0 -K --loop 1 smallFlows.pcap
```

The three commands are wrapped in [server\\_threads.py](#), which also includes the configuration of the instance's maximum transmission unit. The MTU is the size of the largest protocol data unit that is allowed to be transmitted. One can check the data flow on another session with the following. Note, this command must run after running `server_threads.py`, specifically after `netcat -lk -p 4444`.

```
netcat localhost 4444
```



**Figure 5** Left session runs `serve_threads.py`, which consists of `tcpreplay`, `tcpdump`, and `netcat`. Right session listens with `netcat` and shows the streaming packet capture. This same stream will be consumed by the Spark cluster.

[^ top](#)

## Processing cluster

The Spark cluster processes the PCAP stream of lines consumed from the network switch, instance-1. Its primary transformation is to extract elements from the unstructured stream of lines. The cluster could also be used to perform Natural Language Processing on the packet, in order to classify traffic as an exploit.

## Configuration

Enable the Cloud Dataproc API and create a cluster for Spark with the GCP [Dataproc](#). A master node and two worker nodes. Each consists of four CPUs, 15 GB memory. The workers include a YARN NodeManager and an HDFS DataNode.

- cluster-1-m
- cluster-1-w-0
- cluster-1-w-1

## Installs

Install pip and pyspark.

```
sudo apt-get install python3-pip
pip3 install pyspark
```

Repeat Kafka install and configuration steps as in instance-1 configuration, for eventual Kafka implementation.

## Check connection to and view stream from instance-1

Run the following on cluster-1-m *after* running server\_threads.py on instance-1:

```
netcat 10.150.0.6 4444
```

```

berlin@instance-1:~/pcap_streaming$
^Ctcpdump: Unable to write output: Interrupted system call
Exception ignored in: <module 'threading' from '/usr/lib/p
python3.5/threading.py'>
Traceback (most recent call last):
  File "/usr/lib/python3.5/threading.py", line 1288, in _s
uttdown
    t.join()
  File "/usr/lib/python3.5/threading.py", line 1054, in jo
in
    self._wait_for_tstate_lock()
  File "/usr/lib/python3.5/threading.py", line 1070, in _w
ait_for_tstate_lock
    elif lock.acquire(block, timeout):
KeyboardInterrupt
berlin@instance-1:~/pcap_streaming$
Actual: 356 packets (183910 bytes) sent in -1589064304.46
seconds.      Rated: -0.0 bps, -0.00 Mbps, -0.00
pps
ip addr show
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue stat
e UNKNOWN group default qlen 1
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc
pfifo_fast state UP group default qlen 1000
    link/ether 42:01:0a:96:00:06 brd ff:ff:ff:ff:ff:ff
    inet 10.150.0.6/32 brd 10.150.0.6 scope global eth0
        valid_lft forever preferred_lft forever
    inet6 fe80::4001:aff:fe96:6/64 scope link
        valid_lft forever preferred_lft forever
berlin@instance-1:~/pcap_streaming$ python3 server_threads.py
sending out eth0
processing file: smallFlows.pcap
tcpdump: listening on eth0, link-type EN10MB (Ethernet), capture size 262144 by
tes
]

Expires: Tue, 25 Jan 2011 19:23:34 GMT
22:51:25.536363 IP (tos 0x0, ttl 242, id 28507, offset 0, flags [DF], proto
TCP (6), length 1500)
    208.82.236.129.80 > 192.168.3.131.58265: Flags [P.], cksum 0xcd01 (corr
ect), seq 1461:2921, ack 607, win 4986, length 1460: HTTP
22:51:25.536375 IP (tos 0x0, ttl 242, id 28509, offset 0, flags [DF], proto
TCP (6), length 1500)
    208.82.236.129.80 > 192.168.3.131.58265: Flags [P.], cksum 0x2fca (corr
ect), seq 2921:4381, ack 607, win 4986, length 1460: HTTP
22:51:25.536416 IP (tos 0x0, ttl 128, id 30515, offset 0, flags [DF], proto
TCP (6), length 40)
    192.168.3.131.58265 > 208.82.236.129.80: Flags [.], cksum 0xf516 (corre
ct), seq 607, ack 4381, win 16425, length 0
22:51:25.587196 IP (tos 0x0, ttl 242, id 32341, offset 0, flags [DF], proto
TCP (6), length 1500)
    208.82.236.129.80 > 192.168.3.131.58265: Flags [.], cksum 0xac6a (corre
ct), seq 4381:5841, ack 607, win 4986, length 1460: HTTP
22:51:25.587967 IP (tos 0x0, ttl 242, id 32342, offset 0, flags [DF], proto
TCP (6), length 848)
    208.82.236.129.80 > 192.168.3.131.58265: Flags [P.], cksum 0x97be (corr
ect), seq 5841:6649, ack 607, win 4986, length 808: HTTP
22:51:25.587972 IP (tos 0x0, ttl 242, id 32343, offset 0, flags [DF], proto
TCP (6), length 40)
    208.82.236.129.80 > 192.168.3.131.58265: Flags [F.], cksum 0x18e9 (corr
ect), seq 6649, ack 607, win 4986, length 0
22:51:25.588008 IP (tos 0x0, ttl 128, id 30522, offset 0, flags [DF], proto
TCP (6), length 40)
    192.168.3.131.58265 > 208.82.236.129.80: Flags [.], cksum 0xec39 (corre
ct), seq 607, ack 6650, win 16425, length 0
22:51:25.588716 IP (tos 0x0, ttl 128, id 30523, offset 0, flags [DF], proto
TCP (6), length 40)
    192.168.3.131.58265 > 208.82.236.129.80: Flags [F.], cksum 0xec38 (corr
ect), seq 607, ack 6650, win 16425, length 0
22:51:25.637879 IP (tos 0x0, ttl 242, id 36462, offset 0, flags [DF], proto
TCP (6), length 40)
    208.82.236.129.80 > 192.168.3.131.58265: Flags [.], cksum 0x18e8 (corre
ct), seq 6650, ack 608, win

```

Figure 6 Left session on instance-1 runs server\_threads, whihc includes the tcpreplay, tcpdump, and netcat on port 4444. Right session on cluster-1-m connects to instance-1 with netcat and dumps the PCAP stream.

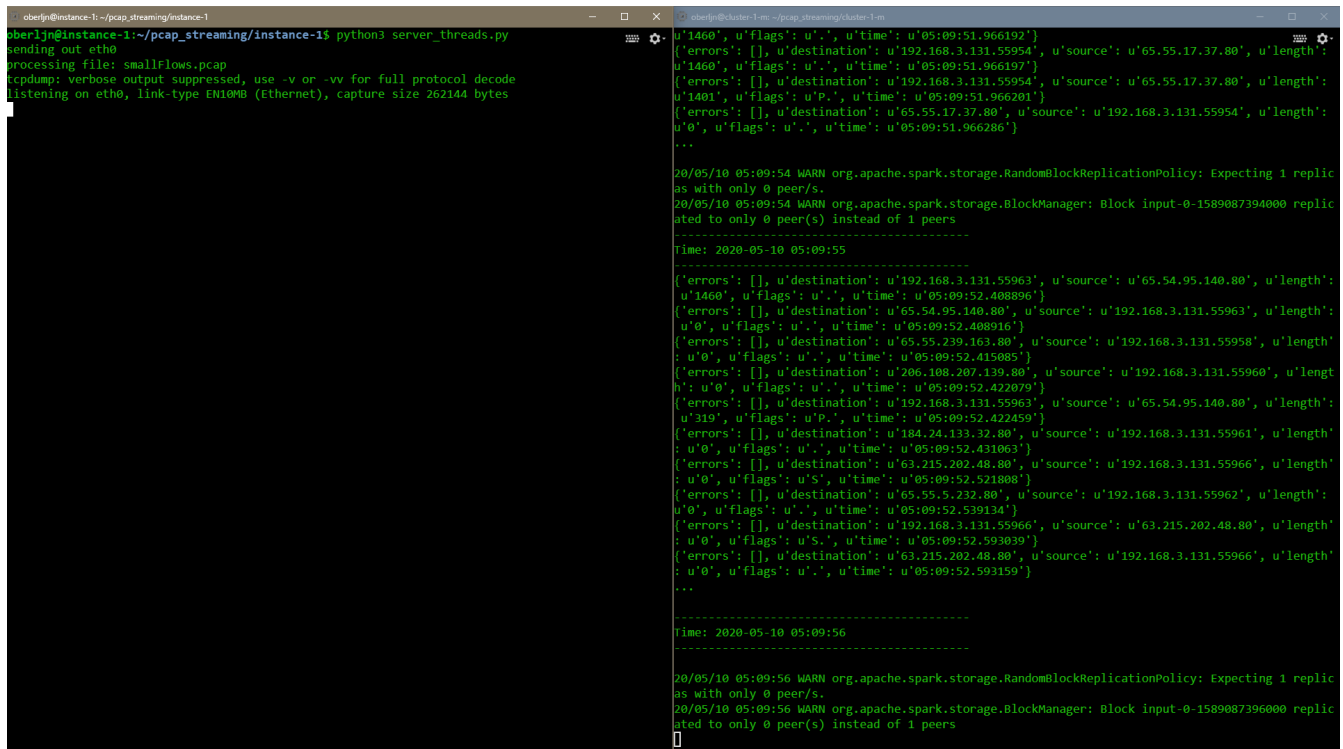
## PCAP parsing

A Spark streaming program consumes the stream with the socket receiver. It uses regular expressions to parse key packet data, such as timestamp, source IP, destination IP, protocol, packet size, etc. An example of raw PCAP data can be seen here, [http\\_request\\_dump.txt](#), which is a single HTTP request to [www.example.com](#).

For reference, the first attempt with Kafka was made with [spark\\_kafka\\_FAIL.py](#), which requires more time for troubleshooting. [spark\\_process.py](#) was written using the Spark socket receiver instead of the Kafka utility.

After the following Spark stream starts up, run `python3 server_threads.py` on instance-1 or test with `netcat -l 4444` and type some text. (Figure 7)

```
/home/oberljn/.local/lib/python3.5/site-packages/pyspark/bin/spark-submit spark_process.py 10.150.0.6 4444
```



```
oberljn@instance-1:~/pcap_streaming/instance-1$ python3 server_threads.py
Sending out eth0
Processing file: smallFlows.pcap
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
Listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes

...
20/05/10 05:09:54 WARN org.apache.spark.storage.RandomBlockReplicationPolicy: Expecting 1 replic
as with only 0 peer/s.
20/05/10 05:09:54 WARN org.apache.spark.storage.BlockManager: Block input-0-1589087394000 replic
ated to only 0 peer(s) instead of 1 peers
-----
Time: 2020-05-10 05:09:55
-----
{'errors': [], 'destination': 'u'192.168.3.131.55963', 'source': 'u'65.54.95.140.80', 'length':
u'1460', 'flags': 'u'', 'time': 'u'05:09:52.408896'}
{'errors': [], 'destination': 'u'65.54.95.140.80', 'source': 'u'192.168.3.131.55963', 'length':
u'0', 'flags': 'u'', 'time': 'u'05:09:52.408896'}
{'errors': [], 'destination': 'u'65.55.239.163.80', 'source': 'u'192.168.3.131.55958', 'length':
u'0', 'flags': 'u'', 'time': 'u'05:09:52.415005'}
{'errors': [], 'destination': 'u'206.108.207.139.80', 'source': 'u'192.168.3.131.55960', 'length':
u'0', 'flags': 'u'', 'time': 'u'05:09:52.422079'}
{'errors': [], 'destination': 'u'192.168.3.131.55963', 'source': 'u'65.54.95.140.80', 'length':
u'319', 'flags': 'u'P.', 'time': 'u'05:09:52.422459'}
{'errors': [], 'destination': 'u'184.24.133.32.80', 'source': 'u'192.168.3.131.55961', 'length':
u'0', 'flags': 'u'', 'time': 'u'05:09:52.431063'}
{'errors': [], 'destination': 'u'63.215.202.48.80', 'source': 'u'192.168.3.131.55966', 'length':
u'0', 'flags': 'u'S.', 'time': 'u'05:09:52.521808'}
{'errors': [], 'destination': 'u'65.55.5.232.80', 'source': 'u'192.168.3.131.55962', 'length':
u'0', 'flags': 'u'', 'time': 'u'05:09:52.539134'}
{'errors': [], 'destination': 'u'192.168.3.131.55966', 'source': 'u'63.215.202.48.80', 'length':
u'0', 'flags': 'u'S.', 'time': 'u'05:09:52.593039'}
{'errors': [], 'destination': 'u'63.215.202.48.80', 'source': 'u'192.168.3.131.55966', 'length':
u'0', 'flags': 'u'', 'time': 'u'05:09:52.593159'}
...
-----
Time: 2020-05-10 05:09:56
-----
20/05/10 05:09:56 WARN org.apache.spark.storage.RandomBlockReplicationPolicy: Expecting 1 replic
as with only 0 peer/s.
20/05/10 05:09:56 WARN org.apache.spark.storage.BlockManager: Block input-0-1589087396000 replic
ated to only 0 peer(s) instead of 1 peers
```

Figure 7 Left session, instance-1 runs serves tcpdump on port 4444. Right session, cluster-1-m processed the packets into data elements: source IP, destination IP, packet length (size), and timestamp. The code also includes an error capture.

[^ top](#)

## Storage

For further development, processed PCAP data will be stored in a Cassandra database on Hadoop. Cassandra has been chosen due to the need for high availability in an environment like cyber ops. Cassandra has multiple master nodes that can continue to run the DB if one goes down Whereas MongoDB has one master node in a cluster that, if it goes down, is not replaced until after 10 to 30 seconds During the replacement process, the cluster cannot take input. Additionally, Cassandra provides ad-hoc reporting via query language CQL, which would be familiar to SQL users. However, MongoDB's JSON document open schema may be ideal considering the dictionary data structure used in the Spark processing.

## Configuration

Cassandra (Google Click to Deploy) <https://console.cloud.google.com/marketplace/details/click-to-deploy-images/cassandra>

2 CPUs and 8 MB memory for testing purposes.

Data disk size: 50 GB

- ☒ Allow TCP port 7000-7001 traffic between VMs in this group
- ☒ Allow TCP port 7199 traffic between VMs in this group

Add to spark-submit command.

```
spark-submit \
  --packages anguenot/pyspark-cassandra:<version> \
  --conf spark.cassandra.connection.host=your,cassandra,node,names
```

[^ top](#)



## Sources

---

- Iman Sharafaldin, Arash Habibi Lashkari, and Ali A. Ghorbani. Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization. 4th International Conference on Information Systems Security and Privacy (ICISSP), Portugal, January 2018
- Resizing the file system and partitions on a zonal persistent disk. Google Cloud. url: [https://cloud.google.com/compute/docs/disks/add-persistent-disk?hl=en\\_US&ga=2.94629659.-684521909.1584918365#resize\\_partitions](https://cloud.google.com/compute/docs/disks/add-persistent-disk?hl=en_US&ga=2.94629659.-684521909.1584918365#resize_partitions)
- Structured Streaming Programming Guide. Apache Spark. url: <https://spark.apache.org/docs/latest/structured-streaming-programming-guide.html>
- tcpreplay man page. Debian. url: <https://manpages.debian.org/unstable/tcpreplay/tcpreplay.1.en.html>

[^ top](#)