# User Manual: Hadoop Network Inspection Tools

User Manual: Hadoop Network Inspection Tools 1

Packet Sniffer 1

Sniff Modes 1

Manually controlled 1

Volume based 1

Time Interval 1

From Cloudera Manager 2

Optimization Strategy: 2

Additional Resources 2

## Packet Sniffer

### Sniff Modes

The sniffer.py tool writes simple text files in CSV format. Avro output is achieved by the snaffer.py script. More control is provided by the snaffer.sh script. When packaged in a CSD (custom service descriptor) the tool can be deployed via Cloudera Manager to all nodes in a Hadoop cluster.

#### Manually controlled

Only the device has to be specified. The dump starts immediately and stops as soon as Ctrl-C is pressed.

$ python sniffer.py DEVICE

DEVICE:   
eth0, lo, or any other available device. Use ifconfig to get a list of available devices.

#### Volume based

For testing of the toolset and for individual scans in manual mode we restrict the number of

Collected packets to a maximum number. Using Ctrl-C allows to stop the procedure faster.

$ python snaffer.py DEVICE ZP

DEVICE:

(see above)

ZP:

Number of packets to collect.

#### Time Interval

We define a start time and a stop time for the tcpdump procedure. This allows a sequential start of the tool but a parallelized start of the data collection procedure.

Start and stop time are metadata for the packet bucket. Internally, each packet will have a time stamp for exact timings.

* We have to identify the appropriate time resolution (probably 1 s).
* We have to define an appropriate filter to lower the impact of the dump procedure on a host.

#### From Cloudera Manager

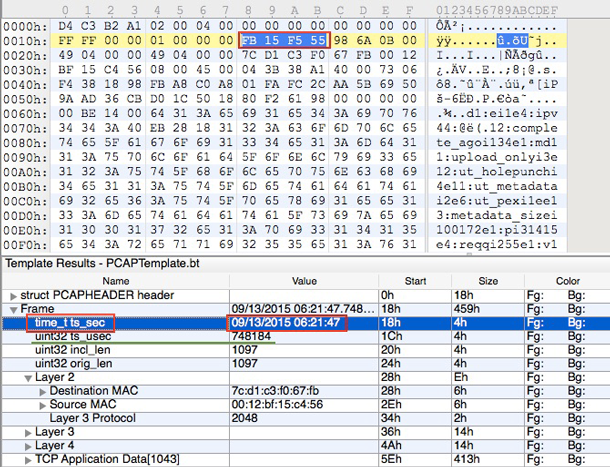
The snaffer service will be deployed on 1 or more cluster nodes, once per host. During deployment only local initialization per host is done by the bootstrap.sh script. After the deployment, an initialization of the HDFS folder and the Hive table is necessary. This is done by running the bootstrapHDFS.sh script. The CSD will provide wrappers around this scripts in the control.sh script.

## Optimization Strategy:

1. No local text dump
2. Packet content optionally
3. Ignore specific packet types
4. Ignore specific ports
5. Ignore specific IP-Adresses

## Recording the time stamp of a Packet

We can examine the given frame using a hex editor. The timestamp is recorded as a Hex value 0xFB 0x15 0xF5 0x55 followed by microseconds 0x98 0x6A 0x0B 0x00.



These **time\_t** **ts\_sec** hex values are represented in [Little-endian](https://en.wikipedia.org/wiki/Endianness) format (meaning that the most significant part of the data is stored last). We can then convert 0x55 0xF5 0x15 0xFF to a decimal 1442125311 number representing a number of seconds since Epoch time, and finally calculate a Human Readable Timestamp GMT/UTC: Sun, 13 Sep 2015 06:21:47. In the given example we used [010 Hex Editor](http://www.sweetscape.com/010editor/) with [a Binary PCAP Template](http://www.sweetscape.com/010editor/templates/files/PCAPTemplate.bt) that does all the necessary highlighting and calculations for us.

## Additional Resources

<https://www.elvidence.com.au/understanding-time-stamps-in-packet-capture-data-pcap-files/>

<http://www.binarytides.com/code-a-packet-sniffer-in-python-with-pcapy-extension/>

<https://www.solutionary.com/resource-center/blog/2012/12/packet-wrangling-101-building-your-own-packet-analyzer/>

<http://www.kroosec.com/2012/10/a-look-at-pcap-file-format.html>

<http://www.networksorcery.com/enp/protocol/tcp/option008.htm>

<http://owtrsnc.blogspot.de/2010/01/python-port-sniffer-with-pcapy-and.html>