Técnicas de Perceção de Redes
Network Awareness

Data Processing (Feature Extraction)

Python references: *SciPy* – SciPy.org, http://www.scipy.org/
NumPy – NumPy Routines, https://docs.scipy.org/doc/numpy/reference/routines.html
matplotlib.pyplot - http://matplotlib.org/api/pyplot_api.html

Extract Features from Packet Captures

Sampling

1. Based on your own data or on the provided data files, and using as base the basePktSampling.py script, convert your qualitative data to quantitative data using several sampling intervals (e.g., 0.01 seconds, 0.1 seconds, 1 second, and 10 seconds) for upload and download packet and byte counters.

Assume three possible input formats:

(1) ASCII line "*timestamp* IP packet from *srcIP* to *dstIP* (*protocol:srcPort:dstPort*) *lengthIP*";

(2) ASCII line "*timestamp* *srcIP* *dstIP* *lengthIP* ";

(3) Binary pcap format. pcap

The script may be invoked as (where 192.168.91.0/25 is the local network):

```
python basePktSampling.py -i test.pcap -f 3 -d 0.1 -c 192.168.91.0/25 -s 0.0.0.0/0
```

Note: Option -f defines the format of the input file. Format 1 is the default format of provided capture python script; format 2 can be obtained from tshark ASCII stdout output using format options

```
-T fields -e frame.time_relative -e ip.src -e ip.dst -e ip.len
```
?

and format 3 is a standard pcap binary file.

pcap files can be obtained using tshark option "-w" do define the output file. To save disk space consider options "-s 40" or "-s 64" to limit the stored file size (all packets are recorded, but only the first 40/64 bytes are stored).

Option -d defines the sampling interval.

Option -c defines the client network, and the option -s defines the server side network. This allows to filter traffic and discriminate the upload and download packets

>> Analyze the output file identifying the vales per line and per column.

2. Load the sampled time sequences.

```
import numpy as np
import matplotlib.pyplot as plt
 datFile='outFile.txt'
data=np.loadtxt(datFile,dtype=int)
plt.plot(data[:,1],'k')
plt.show()
plt.plot(data[:,3],'b')
plt.show()
```

>> Analyze the traffic behavior over time.

Note: datFile is the name of the output of the previous sampling procedure. The output data file can be obtained using the option -o or with output redirecting:

```
python basePktSampling.py -i test.pcap -f 3 -d 1 -c 192.168.91.0/25 -s 0.0.0.0/0 -o outFile.txt
python basePktSampling.py -i test.pcap -f 3 -d 1 -c 192.168.91.0/25 -s 0.0.0.0/0 > outFile.txt
```

Feature Extraction

3. Using as base the basePktFeaturesExt.py script, extract the mean, median and standard deviation of all raw metrics obtained by the sampling process. Use sequential observation windows with an width of 10 sampling intervals.

```
python basePktFeaturesExt.py -i outFile.txt -m 1 -w 10
```
Maior janela, menor tamanho do array

>> Analyze the features files created for each run.

>> How many features were created for each observation? Explain the resulting number. 12

>> How many observation were made? Explain the resulting number. Tamanho da captura vs tamanho da janela: maior janela => menor array de features

>> Identify the respective columns. Media, mediana, desvio padrao, media(silencio),mediana(silencio), desvio padrao(silencio)

>> Test the observation window creation process with your own (longer) datasets using more realistic width values (e.g., 60, 120, 300).

Note: Option -m 1 defines the usage of sequential observation windows. Option -w defines the width of the window.

4. Using as base the basePktFeaturesExt.py script, extract the mean, median and standard deviation of all raw metrics obtained by the sampling process. Use sliding observation windows with an width of 10 sampling intervals, and slide of 5 sampling intervals.

```
    python basePktFeaturesExt.py -i outFile.txt -m 2 -w 10 -s 5
```

>> Analyze the features files created for each run.

>> How many features were created for each observation? Explain the resulting number.

>> How many observation were made? Explain the resulting number.

>> Identify the respective columns.

>> Test the observation window creation process with your own (longer) datasets using more realistic width values (e.g., widths: 60, 120, 300, slides: 10, 20, 30).

Note: Option -m 2 defines the usage of sliding observation windows. Option -w defines the width of the window, and option -s the sliding value.

5. Using as base the basePktFeaturesExt.py script, extract the mean, median and standard deviation of all raw metrics obtained by the sampling process. Use sliding observation windows with width of 10 and 20 sampling intervals, and slide of 5 sampling intervals.

```
    python basePktFeaturesExt.py -i outFile.txt -m 3 -w 10 20 -s 5
```

>> Analyze the features files created for each run.

>> How many features were created for each observation? Explain the resulting number.

>> How many observation were made? Explain the resulting number.

>> Identify the respective columns.

>> Test the observation window creation process with your own (longer) datasets using more realistic width values (e.g., widths: 60, 120, 300, slides: 10, 20, 30).

Note: Option -m 3 defines the usage of multiple window width and sliding observation windows. Option -w defines the widths of the window (space separated list), and option -s the sliding value.

6. Additional time independent statistical/values may be added as features. Examples are: maximum, minimum, percentiles (75%,90%,95%,98%), etc...

>> Add additional time independent features.

>> How many features were created for each observation? Explain the resulting number.adicionado percentis, 28 features?

>> How many observation were made? Explain the resulting number.

>> Identify the respective columns.

>> Test the observation window creation process with your own (longer) datasets using more realistic width values (e.g., widths: 60, 120, 300, slides: 10, 20, 30).

Note: Option -m 3 defines the usage of multiple window width and sliding observation windows. Option -w defines the widths of the window (space separated list), and option -s the sliding value.

7. Additional time dependent statistical/values may be added as features. Examples are: count/sum, average and standard deviation of the silence periods, average and standard deviation of the activity periods, etc...

>> Using the prototype function `extractStatsAdv(…)`, add additional time dependent features.

>> How many features were created for each observation? Explain the resulting number.

>> How many observation were made? Explain the resulting number.

>> Identify the respective columns.

Extract Features from Flow Data

8. Using a flow NFStream datafile, which can be obtained from a packet capture file with:

```
python baseNFStream.py -r test.pcap -w flowsNF.dat
```

em vez de .dat é .csv

And as base the baseFlowFeaturesExt.py script, filter the flows initiated by a specific host, and extract the following metrics/features within sequential observation windows (e.g., 1 second width) to characterize the client behavior: number of flows, average (and standard deviation) uploaded/downloaded bytes, and the average/standard deviation of the interval between flow starts:

```
python baseFlowFeaturesExt.py -i flowsNF.dat -c 192.168.17.150
```

>> Add different filters (flow differentiation). Examples are: (i) client IP to remote TCP port (443, 80 , 22) or (ii) client IP do specific remote networks/services.

>> Experiment different observation widths.

>> Extract additional features.

(Optional) Periodicity Features

9. Based on your own packet data or on the provided data file (YouTube.txt), and using as base the basePeriodicity.py script, with the upload or download activity data, infer the scalogram (with a CWT using the FFT algorithm, and using the Morlet wavelet). For 120 seconds sequential observation windows:

```
python basePktSampling.py -i YouTube.pcap -f 3 -d 1 -c 192.168.91.0/25 -s 0.0.0.0/0 > YouTube.txt
python basePeriodicity.py -i YouTube.txt
```

Plot the scalograms from all observations.

```
import numpy as np
import matplotlib.pyplot as plt
scalograms=np.loadtxt("YouTube_period_features_w120")
plt.plot(scalograms)
plt.show()
```

>>Analyze the outputs (file *_per_features.dat).

>>Identify characteristics periodic components of the dataset.

10. Consider as potential data features (per observation window):

- All power values,
- Location (frequency/scale) of the first *n* maximums.
- Location (frequency/scale) of the first *n* maximums and relative power values (local maximum value divided by overall maximum).

Note: when using multiple sliding observation windows, periodicity analysis may be applied only to the “largest” window.