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*

Qualitative to Quantitative Data (Data 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., 0.1 seconds, 1 second, 10 seconds, and 1 minute) 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.

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

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

Note: 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
```

Format (3) pcap binary files can be obtained using tshark option "-w" do define output pacap file. To save disk space consider options "-s 40" or "-s 64".

>> Analyze the output file identifying the vales per line and per column. Note that the first column is the index of the sampling interval.

2. Load the sampled time sequences.

```
import numpy as np
import matplotlib.pyplot as plt
data=np.loadtxt(datFile,dtype=int)
plt.plot(data[:,0],plt.plot(data[:,1],'k')
plt.show()
plt.plot(data[:,0],plt.plot(data[:,3],'b')
plt.show()
```

>> Analyze the traffic behavior over time.

Observations

Sequential Windows

3. Based on your own data or on the provided data file (test_data.txt), and using as base the baseObsWindows.py script, divide data in sequential observation windows of different widths (e.g., 5 or 10 samples).

```
python baseObsWindows.py -i test_data.txt -m 1 -w 10
python baseObsWindows.py -i test_data.txt -m 1 -w 5
```

>> Analyze the data files created for each observation.

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

Note: The data file test_data.txt has in first column the sampling interval index, the second and third columns are the number of upload packets and bytes, and the fourth and fifth columns are the number of download packets and bytes. The sampling interval index should not be used as a data measurement.

Sliding Windows

4. Based on your own data or on the provided data file (test_data.txt), and using as base the baseObsWindows.py script, divide data in sliding observation windows of different widths (e.g., 5 or 10 samples) and using different sliding steps (e.g., 2 or 3 samples).

```
python baseObsWindows.py –i test_data.txt –m 2 –w 10 –s 3
python baseObsWindows.py –i test_data.txt –m 2 –w 10 –s 2
python baseObsWindows.py –i test_data.txt –m 2 –w 5 –s 3
python baseObsWindows.py –i test_data.txt –m 2 –w 5 –s 2
```

>> Analyze the data files created for each observation.

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

Multiple Sliding Windows

5. Based on your own data or on the provided data file (test_data.txt), and using as base the baseObsWindows.py script, divide data in multiple sliding observation windows of different widths (e.g., 5 and 10 samples) and using different sliding steps (e.g., 2 or 3 samples).

```
python baseObsWindows.py –i test_data.txt –m 3 –w 5 10 –s 3
python baseObsWindows.py –i test_data.txt –m 3 –w 5 10 –s 2
```

>> Analyze the data files created for each observation.

>> Test the observation window creation process with your own (longer) datasets using more realistic width and sliding values.

Extraction of Features

Time Independent Features (from observations)

6. Based on n your own data or on the provided data file (test_data.txt), and using as base the baseExtractFeatures.py script, extract data features from each observation window. Consider all available metrics (number of packets and bytes, download and upload) and the following features: minimum, maximum, mean, median, standard deviation, skewness, and percentiles 75%, 90%, 95%, and 98%.

>> Analyze the *_features.dat file created. Explain the number of columns and lines of that file.

>> Test the feature extraction process with your own (longer) datasets.

Note: Lines will be the classification object, and columns contain the features of each object.

Time Dependent Features (within an observation)

Activity and Silence Periods

7. For each observation window infer two additional data sequences: length of activity and silence periods. Consider an activity period as a set of consecutive samples greater than zero, and a silent period as a set of consecutive samples equal to zero.

Using as base the baseExtractFeatures.py script, extract the following features for each window: number, mean and standard deviation of activity and silent periods. Uncomment the following lines:

```
def extractFeatures(dirname,basename,nObs,allwidths):
    ...
    #faux2=extractStatsSilenceActivity(subdata)
    #features=np.hstack((features,faux2))
```

Note: When the number of activity (or silence) periods is zero, consider the statistical features as zero.

>> Analyze the *_features.dat file created. Explain the new number of columns and lines of that file.

>> Test the feature extraction process with your own (longer) datasets.

8 (optional). Repeat the previous task, but redefining the definition of activity and silence periods. An activity period should now be a sequence of samples greater than zero never interrupted by more than one sample (or two samples) equal to zero. A silence period should now be a sequence of samples equal to zero never interrupted by more than one sample (or two samples) greater than zero.

Periodicity Features

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

```
python baseObsWindows.py -i YouTube.txt -m 1 -w 120
python basePeriodicity.py -i YouTube_obs_m1 -w 120
```

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

Plot the scalograms from all observations.

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

>>Identify characteristics periodic components of the dataset.

10 (optional). Based on your own data or on the provided data file (YouTube.txt), and using as base the basePeriodicity.py script, infer the data periodogram using (i) the basic modulus-squared of the discrete FFT, and (ii) the Welch's method. Analyze the outputs and identify periodic components.

11. 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.