Analysis of Unidirectional IP Traffic to Darkspace with an Educational Data Kit

Tanja Zseby
Fraunhofer FOKUS
and CAIDA
tanja@caida.org

Alistair King
CAIDA
alistair@caida.org

Marina Fomenkov
CAIDA
marina@caida.org

kc claffy
CAIDA
kc@caida.org

ABSTRACT

This tutorial describes methods for analyzing unsolicited one-way Internet Protocol (IP) traffic destined to unassigned address space. We provide an educational dataset curated from data collected by the UCSD Network Telescope [6] and detailed step-by-step analysis instructions including the outcome of each of the steps. Our exercises target engineering or computer science undergraduate and graduate students familiar with the concept of IP addresses and port numbers, and IP header fields.

1. INTRODUCTION

Practical experience is essential for learning how to perform Internet data analysis and computer science students should have opportunity and be encouraged to work with the real Internet data. While traffic data from operational Internet links is difficult to obtain due to privacy constraints, we are able to collect samples of *darkspace* traffic that is more readily amenable to capture and analysis since it does not result from meaningful bidirectional communications.

A *darkspace* is a segment of globally routable Internet address space that has no active hosts. All traffic arriving to such IP darkspace is unsolicited and unidirectional, sometimes also called *Internet Background Radiation* (IBR). Observing and analyzing darkspace traffic can facilitate study of security-related Internet phenomena such as denial-of-service attacks from randomly spoofed sources, the automated spread of Internet worms and viruses, scanning of address space by attackers or malware looking for vulnerable targets, and various botnet activity. Darkspace traffic has also been used to analyze macroscopic Internet events unrelated to malware, such as country-level censorship of Internet communications, and natural disasters affecting reachability of significant regions of Internet infrastructure. The fact that darkspace traffic is less sensitive from a privacy standpoint, but still highly relevant to many Internet research questions, makes these data ideal for educating students on Internet data analysis methods, tools, and issues.

2. EDUCATIONAL DATA KIT

2.1 UCSD Network Telescope data

The UCSD Network Telescope is monitoring instrumentation that collects traffic destined to a large segment of dark (unassigned) address space (also known as an Internet darkspace, darknet, or blackhole). The UCSD Telescope's darkspace is a globally routed /8 network (approximately 1/256th of all IPv4 Internet addresses) that carries almost no legitimate traffic because most IP addresses in this prefix are not assigned to any hosts. After discarding traffic to the few hosts with assigned IP addresses, the remaining packets represent a continuous sample of anomalous unsolicited traffic [5]. (Active hosts also receive unsolicited traffic but we choose not to examine it for privacy reasons.)

We store traffic data captured by the UCSD Network Telescope in *pcap* format, each file containing all packets observed in one hour, each packet timestamped when the telescope receives it. The timestamps stored in the *pcap* files are in the *epoch time* format representing the number of seconds elapsed since January 1, 1970 midnight (UTC).

2.2 Raw Data: Patch Tuesday (PT) Dataset

To teach methods and tools for darkspace traffic analysis, we built this educational data kit around a specific activity relevant to global Internet security: "Patch Tuesday" (PT) [3]. Microsoft releases accumulated security patches on the second Tuesday of each month at 10:00 AM local time in Redmond, WA, which corresponds to either 17:00 UTC (when daylight saving time is in effect) or 18:00 UTC (otherwise). After Patch Tuesday, attackers sometimes use the released patch information to exploit vulnerabilities on unpatched machines, or they may check whether previously exploited security holes remain open. In general, launching new malware immediately after Patch Tuesday maximizes the potential duration of the malware's effectiveness before the next patch release. Our proposed exercises aim to investigate the impact of patching strategies on the observable characteristics of unsolicited traffic.

Filename	Description	size
example.pcap.gz	one hour long compressed raw *p*cap data from the PT data set	4.6 GB
example.flowtuple.cors.gz	compressed *F*lowTuple data generated from the example *p*cap file	12 MB
ucsd.[epoch_time].flowtuple.cors.gz	720 compressed hourly *F*lowTuple files generated for the whole PT dataset (30 days of April 2012, 24 files per day)	390 GB
example_ftlist.txt	a short list of three *F*lowTuple files from the PT dataset (to be used as an example)	279 B
apr2012_ftlist.txt	complete list of the 720 *F*lowTuple files generated for the whole PT dataset	65 kB
apr2012_pkt_count.txt	timestamp of the beginning of an hour and the number of packets per hour for the whole month April 2012	16 kB
apr2012_src_count.txt	timestamp of the beginning of an hour and the number of unique source IP addresses per hour for the whole month April 2012	14 kB
apr2012_src_types.txt	source types analysis data for the whole month April 2012	105 kB

Table 1: Data files comprising the Patch Tuesday educational data kit.

In April 2012, Patch Tuesday fell on April 10 at 17:00 UTC. The raw PT data set from which we curated the educational data kit consists of 720 *pcap* files of darkspace traffic captured by the UCSD Network Telescope throughout April 2012: 1 file per hour × 24 hours per day × 30 days. The pre- and post-release data establish a baseline for studying the effects of this PT update.

2.3 Data Anonymization

We removed the payload from the captured packets and zeroed out the first eight bits of the destination IP address to anonymize the address range of the UCSD Network Telescope darkspace. We also anonymized the source IP addresses in the data because some packets originated from victims of DDoS attacks (backscatter data); we hide the IP addresses of victim hosts to protect them from further malicious activity.

2.4 Data Kit Curation

Throughout this tutorial, we demonstrate processing steps on small subsets of the data, and provide computed results for the whole data set for further analysis.

Most of the exercises can be done using the aggregated data in the *FlowTuple* format[1] rather than raw data in the *pcap* format. Therefore, we illustrate the aggregation process on just one hour of the raw *pcap* data (*pt_example.pcap.gz*) and provide 720 aggregated *FlowTuple* files prepared for the whole PT data set.

Next, we show how to compute statistics of the darkspace traffic from the aggregated data using a short list of just three *FlowTuple* files in the file *example_ftlist.txt* as an example and again, provide the computed statistics for the whole PT data set.

Table 1 shows all the files comprising the educational data kit. The files with the suffix *.txt* are in ASCII format and can be viewed with any standard text editor or viewer. The *pcap* file *example.pcap.gz* and *FlowTuple* file *example_flowtuple.cors.gz* can be displayed using *tcpdump* and *cors2ascii*, respectively, described in Section 4.1.

[1]Described in Section 4.1, this format retains only a certain subset of fields from the captured packets, greatly reducing the data volume and the associated bandwidth, storage, and processing requirements.

3. WORK ENVIRONMENT AND TOOLS

Exercises described in this tutorial require three tools to be installed on a local host: *Corsaro* [9] (with the *libtrace* library); *Octave* [2] (we used v3.6.1); and *tcpdump* [4]. We recommend a Linux system with at least 4 GB memory, although we have tested the tools on other platforms (FreeBSD, OSX, etc.) as well.

3.1 Corsaro

CAIDA developed the *pcap*-trace processing software *Corsaro* in order to analyze darkspace traffic, although it would work on other types of passive traffic trace data. CAIDA has published a complete description of *Corsaro* features [9] and installation instructions [11]. To use the *FlowTuple* data provided, *Corsaro* should be configured using the following command:

```
./configure --with-flowtuple --with-smee
     --with-slash-eight=0
```

3.2 Octave Scripts

Octave is a high-level interpreted language, primarily intended for numerical computations and also providing extensive graphics capabilities. *Octave* normally works through its interactive command line interface, but can also be used to write non-interactive programs. Its language is quite similar to Matlab so that most programs are easily portable. This software is distributed under the terms of the GNU General Public License.

All *Octave* instructions shown in our exercises are available as small scripts that one can start from the *Octave* command line. The option `-q` prevents *Octave* from displaying its startup message. Table 2 gives an overview of the scripts.

4. GENERAL DATA PRE-PROCESSING

Exercises in this Section teach the necessary data preprocessing steps (schematically shown in Figure 1): aggregating *pcap* data into *FlowTuple* format, computing

Scriptname	Description	Exercise
plot_pktcnt.oct	read in PT data, plot overall packet count for PT data	PT-1
calc_pkt_stats.oct	calculate statistics for the overall packet count for PT data	PT-1
plot_srccnt.oct	read in PT data, plot overall source count for PT data	PT-2
calc_src_stats.oct	calculate statistics for the overall source count for PT data	PT-2
proto_dist.oct	analyze protocol distribution, print top 10 protocols, plot protocol distribution for PT data	PT-3
dport_dist.oct	analyze TCP destination port distribution, print top 10 TCP destination ports, plot TCP destination port distribution for PT data	PT-4
source_types.oct	analyze source types	PT-5
spectrum.oct	calculate spectrum for time series of number of packets and spectrum for time series of number of sources	PT-6

Table 2: Octave scripts

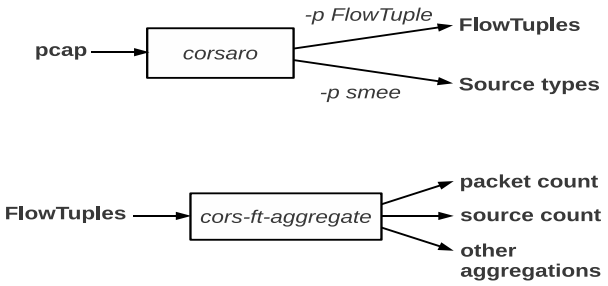

Figure 1: Corsaro Processing Steps

hourly counts of: packets, number of unique source IP addresses, and distributions of packets by source types.

4.1 Exercise G-1: From pcap to FlowTuple

Raw telescope data are stored in hourly files in *pcap* format. To get acquainted with the data, one can display the content of a *pcap* file as ASCII text characters using the *tcpdump* command with the `-r` option. Other useful options, rendering the output much faster, are: `-n` which prevents *tcpdump* from converting IP addresses to domain names, and `-t` which prevents printing a timestamp for each packet[2]. So, if the *pcap* file is compressed, the command:

```
$ zcat example.pcap.gz | tcpdump -t -nr -
```

will create the following output:

```
IP 150.26.54.148.2546 > 0.9.4.169.445: Flags [S], seq
    2112083767, win 65535, options [mss
    1380,nop,nop,sackOK], length 0
IP 74.40.194.245.55760 > 0.63.210.28.3544: UDP, length
    56
IP 153.242.252.91.80 > 0.72.61.72.22792: Flags [S.], seq
     2275567357, ack 2034702177, win 65332, options
    [mss 1460,nop,nop,sackOK], length 0
IP 141.13.71.196.4805 > 0.99.104.95.445: Flags [S], seq
    689364122, win 65535, options [mss
    1452,nop,nop,sackOK], length 0
...
```

[2]The tcpdump *man* page describes all available options.

Our first exercise is to aggregate raw *pcap* data into a *FlowTuple* format. The *FlowTuple* format retains only selected fields from captured packets instead of the whole packet, enabling a more efficient data storage, processing, and analysis. The *FlowTuple* consists of eight fields: source IP address; destination IP address; source port; destination port; protocol; TCP Flags; TTL; and IP length. For ICMP packets (protocol=1), the source and destination port fields represent the ICMP Type and Code fields[3].

Corsaro takes the *pcap* file as input and writes an aggregated *FlowTuple* file as output (cf. Figure 1). In the *FlowTuple* output file, the data is broken into intervals, each representing 60 seconds of data. Within an interval, each unique key (unique combination of the *FlowTuple* fields) observed in the raw *pcap* data is reported on a separate line in the following format:

```
<src_ip>|<dst_ip>|<src_port>|<dst_port>|<protocol>|
    <tcp_flags>|<ttl>|<ip_len>,<value>
```

The <value> that follows the *FlowTuple* fields is the number of packets in the interval whose header fields match this *FlowTuple* key.

In our first exercise, we process a file of *pcap* data *example.pcap.gz* into the *FlowTuple* file:

```
corsaro -o example.%P.cors.gz example.pcap.gz
```

The %P in the filename will be substituted with the plugin names included during the Corsaro installation. For example, if Corsaro is run with the *FlowTuple* plugin enabled (the default), Corsaro will create a *FlowTuple* output file by substituting the %P with the string `flowtuple`. In the example given above, Corsaro would create the output file *example.flowtuple.cors.gz*. By adding the extension *.gz* to the output file name, we ensure that *Corsaro* automatically compresses the output file using *gzip*, further reducing storage requirements.

Next, the *cors2ascii* command will display the *FlowTuple* output in a human-readable ASCII format:[4]

[3]See [10] for more details on the *FlowTuple* format.

[4]Flowtuple files are sorted into sub-categories, one of which is backscatter [12], listed first in this table.

```
$ cors2ascii example.flowtuple.cors.gz
# CORSARO_INTERVAL_START 0 1334098800
START flowtuple_backscatter 897162
167.215.128.118|0.20.22.88|3|1|1|33|0x00|86,1
167.214.141.110|0.209.4.107|3|1|1|35|0x00|56,2
167.214.141.110|0.185.166.52|3|1|1|36|0x00|56,2
59.83.224.172|0.176.14.199|3|3|1|36|0x00|86,1
59.83.224.172|0.176.14.199|3|3|1|36|0x00|144,1
194.19.32.187|0.252.229.185|3|3|1|36|0x00|86,1
194.19.32.187|0.252.229.185|3|3|1|36|0x00|144,1
...
```

Each line in this output corresponds to the format in Listing 4.1 and shows the eight *FlowTuple* fields separated by |. As explained in Section 2.3, the first octet of the destination IP address is set to 0. The last value on each line shows how many packets with this unique combination of fields were in the input *pcap* file.

The exercises in Sections 4.2 and 4.3 use aggregated *FlowTuple* files as input. To reduce bandwidth requirements for downloading the data kit, we provide 720 *FlowTuple* files *ucsd-nt.anon.[epoch_time].flowtuple.cors.gz* precomputed from the 720 original hourly *pcap* files of the UCSD Network Telescope Patch Tuesday data collected in April 2012[5]. The *[epoch_time]* field in each file name shows the starting time of the hour of data in this file in the *epoch* format.

4.2 Exercise G-2: Number of Packets per Hour

To compute hourly packet rates for the whole dataset, we need to sum the packet count values in each hourly *FlowTuple* file. We use the Corsaro FlowTuple aggregation tool, *cors-ft-aggregate*, as follows:

```
cors-ft-aggregate -i 3600 -v packet_cnt
     example_ftlist.txt > example_pkt_count.txt
```

The option `-i 3600` specifies an aggregation interval of one hour (3600 seconds). The option `-v packet_cnt` indicates that we want to aggregate the packet count value. To illustrate how the aggregation tool works, we input the file *example_ftlist.txt*, which lists 3 of the 720 *FlowTuple* files comprising the full the PT data set.

The output file *example_pkt_count.txt* contains three lines for each one hour *FlowTuple* file listed in the input file. The first line shows the epoch time of the start of the one hour (3,600 s) interval. The second line shows the aggregated *FlowTuple* fields for this interval. All field values are equal to 0 because we aggregate over all of them; the only non-zero value (separated by a ‘,’) is the total packet count per interval. The third line is shows the end time of the interval in the epoch format.

```
$ cat example_pkt_count.txt
# CORSARO_INTERVAL_START 0 1334001600
0.0.0.0|0.0.0.0|0|0|0|0|0x00|0,143556017
# CORSARO_INTERVAL_END 0 1334005199
# CORSARO_INTERVAL_START 1 1334005200
0.0.0.0|0.0.0.0|0|0|0|0|0x00|0,135072630
# CORSARO_INTERVAL_END 1 1334008799
# CORSARO_INTERVAL_START 2 1334008800
0.0.0.0|0.0.0.0|0|0|0|0|0x00|0,152169670
# CORSARO_INTERVAL_END 2 1334012399
```

[5]The volume of the raw data is 3.8TB while the volume of the aggregated data is only 390GB

Finally, we invoke the perl script *cors-ft-timeseries.pl* (included with Corsaro) to combine the timestamp and packet count into one line:

```
cors-ft-timeseries.pl example_pkt_count.txt >
     example_pkt_count_ts.txt
```

The resulting output file *example_pkt_count_ts.txt* contains discrete time series of packet counts with one data point per hour:

```
$ cat example_pkt_count_ts.txt
1334001600,143556017
1334005200,135072630
1334008800,152169670
```

The first value on each line shows the start of an hour interval in epoch time format. The second value shows the number of packets observed in that hour. Since the input file *example_ftlist.txt* lists three hourly files, the output file *example_pkt_count_ts.txt* has three lines.

We pre-processed all 720 hourly *FlowTuple* files in the PT dataset by applying *cors-ft-aggregate* and *cors-ft-timeseries.pl* to all PT hourly files in *apr2012_ftlist.txt*. The resulting time series of the hourly packet rates for the whole month of April 2012 is available in the file *apr2012_pkt_count.txt*.

4.3 Exercise G-3: Number of Unique Source IP Addresses per Hour

To determine the overall number of unique source IP addresses seen per hour in the data, we again use the aggregation tool *cors-ft-aggregate*, but with different options. This time we want to aggregate all *FlowTuples* using only the source IP address as a key:

```
cors-ft-aggregate -i 3600 -v src_ip example_ftlist.txt >
      example_src_count.txt
```

Here the option `-i 3600` again specifies the desired aggregation interval of one hour. The option `-v src_ip` indicates that we want to aggregate over the *unique* source IP addresses. Again, we input a small subset of *FlowTuple* files listed in the file *example_ftlist.txt* to illustrate the aggregation process.

The format of the resulting file is similar to the output in the previous exercise. There are three lines of output for each input file. These lines show the interval starting and ending times, and the number of unique source IP addresses observed during that hour:

```
$ cat example_src_count.txt
# CORSARO_INTERVAL_START 0 1334001600
0.0.0.0|0.0.0.0|0|0|0|0|0x00|0,418961
# CORSARO_INTERVAL_END 0 1334005199
# CORSARO_INTERVAL_START 1 1334005200
0.0.0.0|0.0.0.0|0|0|0|0|0x00|0,429939
# CORSARO_INTERVAL_END 1 1334008799
# CORSARO_INTERVAL_START 2 1334008800
```

Type (column)	Proto	Dest. IPs	Ports	Packets
μTorrent (14)	any	any	any	μTorrent packets
Conficker-C (15)	any	any	any	Conficker-C
1 or 2 packets (16)	any	any	any	< 3 packets
TCP Probe (3)	TCP	one	one	all
TCP Vert. Scan (4)	TCP	one	multi	all
TCP 445 Horiz. Scan (20)	TCP	multi	445	all
TCP Horiz. Scan (5)	TCP	multi	one	all, except to dport 445
TCP Backscatter (17)	TCP	any	any	TCP-ACK, TCP-RST
UDP Probe (7)	UDP	one	one	all
UDP Vert. Scan (8)	UDP	one	multi	all
UDP Horiz. Scan (9)	UDP	multi	one	all
DNS Backscatter (18)	TCP/UDP	any	any	source port 53
TCP and UDP (13)	TCP/UDP	any	any	TCP *and* UDP
ICMP Backscatter (19)	ICMP	any	any	Time Exceeded, Dest.Unreach.
ICMP Only (11)	ICMP	any	any	all
TCP Unknown (6)	TCP	multi	multi	all remaining TCP
UDP Unknown (10)	UDP	multi	multi	all remaining UDP
Unclassified (2)	any	any	any	all remaining

Table 3: Source Types in descending order of classification

```
0.0.0.0|0.0.0.0|0|0|0|0|0x00|0,649596
# CORSARO_INTERVAL_END 2 1334012399
```

As in Section 4.2, we use the perl script *cors-ft-timeseries.pl* to combine the interval start time and the number of unique sources sending traffic to the UCSD Network Telescope during that hour onto one line:

```
cors-ft-timeseries.pl example_src_count.txt >
    example_src_count_ts.txt
```

The resulting *example_src_count.txt* file contains:

```
$ cat example_src_count.txt
1334001600,418961
1334005200,429939
1334008800,649596
```

The two values per line are the start of an hour interval in epoch format, and number of unique source IP addresses observed in that hour.

To obtain hourly counts of unique source IP addresses for the whole PT dataset, we again pre-processed all 720 hourly *FlowTuple* files using *cors-ft-aggregate*, *cors-ft-timeseries.pl* and the full list of PT hourly files in *apr2012_ftlist.txt*. The resulting time series of the number of unique source IP addresses per hour for April 2012 is available in the file *apr2012_src_count.txt*.

4.4 Exercise G-4: Calculating the Source Types

Another plugin provided by *Corsaro* to aggregate the raw *pcap* data, is *smee* (cf. Figure 1). This plugin is a *Corsaro* implementation of the *IATmon* tool [8] which classifies sources of observed darknet traffic into 18 mutually exclusive types based on protocol and temporal patterns across a configured time interval. All packets observed in this time frame are first aggregated according to their source IP address, and then each source IP address is assigned to one of the 18 source types based on what type and pattern of packets it generates. Table 3 lists attributes of the source types.

Again, we use a small subset of the compressed original data in file *example.pcap.gz* to illustrate the source type classification with *smee*:

```
corsaro -p smee -o example.%P.cors.gz example.pcap.gz
```

The `-p smee` option specifies the Corsaro plugin used that becomes part of the output file name, for example: *example.smee.cors.gz* (compressed using *gzip*).

Note: *Corsaro* can run with multiple plugins. In particular, one can generate both the *FlowTuple* aggregation (see Section 4.1) and the source type classification with a single command by including both plugins (‘`-p flowtuple -p smee`’) in the command line, producing two output files (one for each plugin).

The output file *example.smee-sum.cors.gz* produced above contains the source type analysis for the packets recorded in the one-hour long example *pcap* file *example.pcap.gz*: the number of source addresses per source type, the number of packets per source type, and some additional information.

To save students the effort of running the source type analysis for all *pcap* files in the PT dataset, we generated the file *apr2012_src_types.txt* which contains the time series of the number of unique source IPs per hour for each source type. Each line of the file accounts for 1 hour of data and contains 22 values. The first value is the hour start time in the epoch format while columns 2-19 contain the number of source IPs observed during this hour and attributed to a given source type. The number in parentheses in Table 3 shows which position in the line of file *apr2012_src_types.txt* contains the data for this source type. For example, the source type ‘TCP vertical scan’ in the Table is marked with ‘(4)’. In the following output we see that during the first hour of the collected data we observed 997 unique IP addresses

classified as the source type 'TCP vertical scan' (column 4). The last two values in each line are: the date (in YYYYmmDDHHMM format) (position 21) and the sum of the counts of source IPs for all types for this hour (position 22, the last value in each line).

```
$ cat apr2012_src_types.txt
1333238400, 2345, 14557, 997, 34411, 8970, 86019, 2842,
     3580, 119332, 9586, 0, 13289, 13807, 11969,
     1016813, 314, 61, 0, 71428, 201204010000, 1410320
1333242000, 2183, 14843, 852, 34768, 9592, 95560, 2301,
     4722, 127874, 9294, 0, 13858, 13678, 12222,
     1002042, 298, 58, 0, 73760, 201204010100, 1417905
1333245600, 2299, 14618, 774, 34729, 8874, 91928, 1856,
     4879, 114732, 8830, 0, 13615, 14153, 12286, 930489,
      244, 55, 0, 75834, 201204010200, 1330195 $
```

5. ANALYZING THE PT EFFECTS

The objective of the following exercises is to analyze whether one can discern any unusual characteristics of unwanted darkspace traffic during or after the April 2012 Patch Tuesday. First, we consider the overall packet count, the number of sources that contribute to the unwanted traffic, and the distributions of protocols and destination ports observed in the captured packets. Next, we look into the number of packets and the number of sources per source type to find out if we observe any unusual behavior attributable to specific source types. Finally, we analyze the temporal behavior of the two time series formed by the packet count per hour and the number of sources per hour.

In this section we apply *Octave* scripts to the prepared files *apr2012_pkt_count.txt*, *apr2012_src_count.txt* and *apr2012_src_types.txt* for the data analysis and visualization of the results.

5.1 Exercise PT-1: Analyzing the Number of Packets

In order to check for unusual patterns in the overall amount of traffic, we analyze a discrete time series formed by the number of packets per hour observed by our darkspace monitor. In Section 4.2 we showed how to extract these hourly numbers of packets from the aggregated *FlowTuple* files. We performed the necessary calculations for the whole PT darkspace data set of 720 hourly files collected in April 2012 and use the prepared file *apr2012_pkt_count.txt* for the analysis described in this section.

We use the *Octave* software to plot and analyze packet rates over time. We configure the *Octave* environment to enable printing of numbers in a field wider than 10 characters long and also change the default font to Helvetica and font size to 20:

```
> format long
> set(gca, 'fontname', 'Helvetica', 'fontsize', 20);
```

Then we load the file *apr2012_pkt_count.txt* into a matrix apr2012_pkt using the csvread *Octave* function with the filename as an argument:

```
> apr2012_pkt=csvread('apr2012_pkt_count.txt');
```

This command creates the matrix apr2012_pkt in *Octave* and reads the comma separated values from the file *apr2012_pkt_count.txt* into the matrix. Typing the name of the matrix will display its contents:

```
> apr2012_pkt
apr2012_pkt =
   1333238400   158193818
   1333242000   158154053
   1333245600   137226309
   1333249200   105671679
   1333252800   107655723
   1333256400   122484762
   1333260000   113172621
   1333263600   130804718
   1333267200   133450608
   ...
```

One can also use indices to access different parts of the matrix. For example, typing apr2012_pkt(2,1) will print the matrix element in the second row and the first column. The operator ':' is used to select all elements of a row or column. We can access the entire first column containing the timestamps as apr2012_pkt(:,1) while the second column containing the number of packets per hour can be accessed by apr2012_pkt(:,2). For instance, the following command will calculate the total number of packets observed in April 2012 by summing up the values in the second column of the matrix:

```
> apr2012_pktcnt=sum(apr2012_pkt(:,2))
apr2012_pktcnt = 108687558356
```

Next, we plot the number of packets per hour vs. time. To make it easier to see when Patch Tuesday took place, we want to display the timestamps, given in the epoch format, as the actual dates and hours in a human readable form. Thus, we apply the function datenum to convert the epoch time into the *datenum* format, which represents the time as the number of days starting from Jan 1, 0000.

```
datenum (year, month, day, hour, minute, second)
```

To convert the epoch time to the *datenum* format, we set the values for year, month, day, hour and minute to Jan 1, 1970 midnight (when epoch time starts) and then add the epoch time as number of seconds. For the first element of the matrix apr2012_pkt (row=1, column=1) the corresponding *datenum* value is 734960:

```
> datenum(1970,1,1,0,0, apr2012_pkt(1,1));
ans = 734960
```

To plot the data over time, we convert the epoch times in the entire first column apr2012_pkt(:,1) into the *datenum* format and then plot the number of packets stored in the second column apr2012_pkt(:,2):

```
> stem(datenum(1970,1,1,0,0,apr2012_pkt(:,1)),
     apr2012_pkt(:,2)/10^6, 'marker', 'none')
```

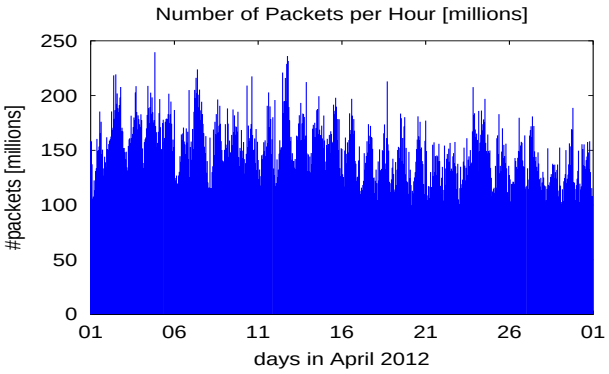

Figure 2: Number of packets per hour vs. time

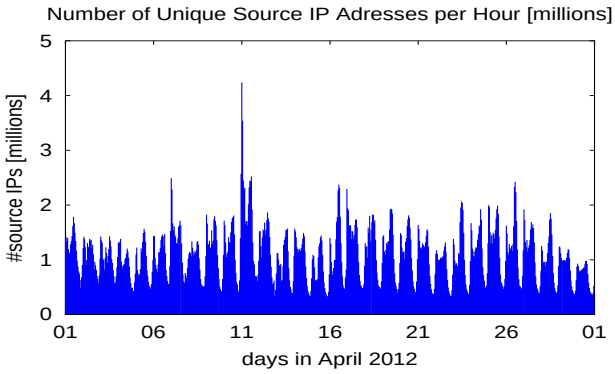

Figure 3: Number of unique source IP addresses per hour

```
> datetick('x', 'dd', 'keepticks');
> xlabel('days in April 2012')
> ylabel('#packets [millions]')
> title('Number of Packets per Hour [millions]')
```

This command `stem` creates a stem plot, which represents each value by a vertical line. The `datetick` function changes tick labels on the x-axis from the *datenum* into a date format specifying day of the month (with year, month, day, hour and minutes or a combination of thereof). We name the axes using the `xlabel` and `ylabel` commands and give the graph a title. To plot the number of packets in millions, we divide the values in the second column by 10^6. Figure 2 shows the resulting plot, revealing no unusually high volume of packets on or around Patch Tuesday.

Next, we check the maximum and minimum hourly packet counts in the PT dataset using the functions `max()` and `min()`, correspondingly (shown below). These functions return the maximum (minimum) value of the dataset and the index (that is, the row of the matrix) where the returned value is. So, if the maximum were in the third row of the dataset we would get the index=3. Using the index we can then display the whole row (with a timestamp and a packet count) and therefore find out when the maximum (or minimum) was observed:

```
> [max_v max_i]=max(apr2012_pkt(:,2))
max_v = 239613469
max_i = 93
> apr2012_pkt(max_i,:)
ans =
   1333569600   239613469
> datestr(datenum(1970,1,1,0,0,1333569600), 'yyyy-mm-dd
    HH:MM')
ans = 2012-04-04 20:00
```

```
> [min_v min_i]=min(apr2012_pkt(:,2))
min_v = 99396882
min_i = 461
> apr2012_pkt(min_i,:)
ans =
   1334894400    99396882
> datestr(datenum(1970,1,1,0,0,1334894400), 'yyyy-mm-dd
    HH:MM')
ans = 2012-04-20 04:00
```

The maximum packet count per hour is 239,613,469 observed in row 93 of the matrix `apr2012_pkt` at epoch time `1333569600`, and the minimum hourly packet count of 99,396,882 is observed in row 461 at epoch time `1334894400`. The `datestr` function translates epoch times into a more readable format, revealing that the maximum hourly packet count in April occurred on April 4, 2012 at 20:00 while the minimum hourly packet count was April 20, 2012 at 04:00. Both dates appear unrelated to Patch Tuesday (April 10).

Finally, we find the mean value of 150,954,942.16 packets per hour in April 2012 using the following:

```
> mean=mean(apr2012_pkt(:,2))
ans = 150954942.161111
```

5.2 Exercise PT-2: Number of Unique Source IP Addresses

The goal of this exercise is to discover unusual patterns in the number of active sources related to the Patch Tuesday. We plot the number of sources per hour vs. time using the same *Octave* functions as in Section 5.1:

```
> apr2012_src=csvread('apr2012_src_count.txt');
> stem(datenum(1970,1,1,0,0,apr2012_src(:,1)),
    apr2012_src(:,2)/10^6, 'marker', 'none')
> datetick('x', 'dd', 'keepticks');
> xlabel('days in April 2012')
> ylabel('#source IPs [millions]')
> title('Number of Unique Source IP Adresses per Hour
    [millions]')
```

Figure 3 shows the resulting stem plot and reveals two interesting features: a periodical temporal pattern and an unusually high number of unique sources on April 11, 2012. We will further investigate the apparent periodicity in the hourly counts of unique sources in Section 5.6. To analyze the peak activity, we calculate the maximum, minimum and mean values of hourly counts

of unique sources and time of their occurrence using the same *Octave* instructions as in section 5.1:

```
> [max_v max_i]=max(apr2012_src(:,2))
max_v = 4240359
max_i = 241
> apr2012_src(max_i,:)
ans =
   1334102400  4240359
> datestr(datenum(1970,1,1,0,0,1334102400), 'yyyy-mm-dd
    HH:MM')
ans = 2012-04-11 00:00
> [min_v min_i]=min(apr2012_src(:,2))
min_v = 329835
min_i = 358
> apr2012_pkt(min_i,:)
ans =
   1334523600  329835
> datestr(datenum(1970,1,1,0,0,1334523600), 'yyyy-mm-dd
    HH:MM')
ans = 2012-04-15 21:00
> mean(apr2012_src(:,2))
ans = 1101808.702778
```

The maximum hourly count of active sources, which exceeded the mean value by almost a factor of 4, occurred on April 11, 2012 at midnight – the first night after Path Tuesday (April 10). We analyze which types of sources became active in Section 5.5.

5.3 Exercise PT-3: Analyzing Protocols

In this exercise we analyze which protocols are used to send packets to the darkspace and, in particular, whether there are any differences in protocol usage before and after Patch Tuesday. The protocol specified in the IP packet header matches one of the fields in the *FlowTuple* format. Thus, we use the pre-computed *FlowTuple* files to generate the overall distribution of protocols for the entire month of April 2012, as well as the protocol distributions for the hour when Microsoft releases patches (Patch Tuesday 17:00) and for one hour of the subsequent day (Exploit Wednesday 00:00).

The file *apr2012_ftlist.txt* contains a list of all *FlowTuple* files for April 2012. The file *PT_ftlist.txt* contains the name of the *FlowTuple* file for April 10, 2012 17:00 (patch release hour) while the file *EW_ftlist.txt* contains the name of the *FlowTuple* file for April 11, 2012 00:00. For each of those files, we aggregate the packet counts per protocol per hour using the Corsaro aggregation tool *cors-ft-aggregate*:

```
// aggregate according to protocol number
$ cors-ft-aggregate -i 3600 -f protocol -v packet_cnt
    apr2012_ftlist.txt > apr2012_proto_dist.txt
$ cors-ft-aggregate -i 3600 -f protocol -v packet_cnt
    PT_ftlist > PT_proto_dist.txt
$ cors-ft-aggregate -i 3600 -f protocol -v packet_cnt
    EW_ftlist > EW_proto_dist.txt

// get the relevant columns: protocol, packet count
$ sed -e '/^\#/d' -e 's/|/,/g' apr2012_proto_dist.txt |
    cut -f5,9 -d, > apr2012_proto_dist_e.txt
$ sed -e '/^\#/d' -e 's/|/,/g' PT_proto_dist.txt | cut
    -f5,9 -d, > PT_proto_dist_e.txt
$ sed -e '/^\#/d' -e 's/|/,/g' EW_proto_dist.txt | cut
    -f5,9 -d, > EW_proto_dist_e.txt
```

Option `-f` specifies that we want to aggregate the protocol field and option `-v` specifies that we want to report the aggregated value of the packet count. The tool *cors-ft-aggregate* then sums up all packet counts that have the same protocol number in the FlowTuple interval. The corresponding output files are: *apr2012_proto_dist.txt* for the whole month of data, *PT_proto_dist.txt* for the patch release hour, and *EW_proto_dist.txt* for the first hour of Wednesday following the release. We do some postprocessing with the standard Unix utility `sed` to output only the protocol number and the associated number of packets as comma separated values.

Next, we input the results into *Octave* (using the `csvread` function described in Section 5.1) and compute packet counts per month and for the two special hours.

```
> proto_all=csvread('apr2012_proto_dist_e.txt');
> proto_PT=csvread('PT_proto_dist_e.txt');
> proto_EW=csvread('EW_proto_dist_e.txt');

>apr2012_pktcnt=sum(proto_all(:,2))
apr2012_pktcnt = 108687558356

> EW_pktcnt=sum(proto_EW(:,2))
EW_pktcnt = 141186833
> PT_pktcnt=sum(proto_PT(:,2))
PT_pktcnt = 150272386
```

Finally, we plot the different protocol distributions using the `subplot` function to create three subgraphs of the percentage of packets associated with each protocol (Figure 4):

```
> subplot(3,1,1)
> stem(proto_all(:,1),
    proto_all(:,2)*100/apr2012_pktcnt, 'marker','x',
    'markersize',4)
> xlim([0,256])
> title('Distribution of Protocols')
> legend('Whole Month April 2012')
> axis('labely')

> subplot(3,1,2)
> stem(proto_PT(:,1), proto_PT(:,2)*100/PT_pktcnt,
    'marker','x', 'markersize',4)
> xlim([0,256])
> ylabel('packets/total packets [%]')
> legend('Patch Tuesday 17:00')
> axis('labely')

> subplot(3,1,3)
> stem(proto_EW(:,1), proto_EW(:,2)*100/EW_pktcnt,
    'marker','x', 'markersize',4)
> xlim([0,256])
> xlabel('Protocol Number')
> legend('Exploit Wednesday 00:00')

% limit number of ticks in graph
> numticks=5
> L=get(gca,'YLim')
> set(gca, 'YTick', linspace(L(1),L(2),numticks));
```

The last three lines in the script limit the number of ticks on the y-axis to 5 for better readability.

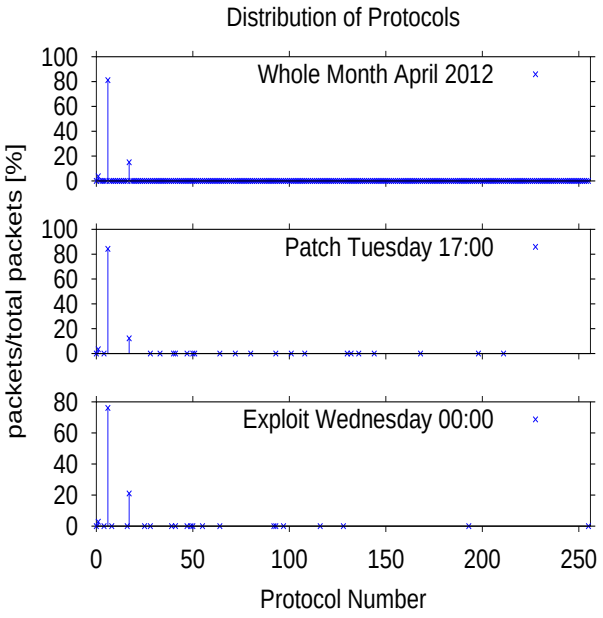

Figure 4: Distribution of Protocol Numbers. All three panels show that the most frequently used protocols are: 6 (TCP), 17 (UDP) and 1 (ICMP). In the top panel, the x-axis looks like a thick line because in a whole month of data, most protocol numbers are observed at least once and the corresponding cross symbols overlap. In contrast, the other two panels each shows only one hour of data and the distributions of observed protocols are sparse.

As expected, the three most common protocol numbers stand out: there are maxima in the plotted distributions at protocol numbers 6 (TCP), 17 (UDP), and 1 (ICMP). Yet the distributions are different: during the patch hour on patch Tuesday we observe 84% TCP, 12% UDP and 3% ICMP traffic while for the hour on Exploit Wednesday midnight the fraction of UDP traffic almost doubles: 76% TCP, 21% UDP and 3% ICMP traffic.

5.4 Exercise PT-4: Analyzing Destination Port Numbers

We compare TCP and UDP destination port numbers in packets collected over the month, and during the two specific hours on Patch Tuesday and Exploit Wednesday.

We use the same commands as in Section 5.3, although this time we aggregate the data by looking at two different fields: protocol number and destination port. That means that each row in the result file aggregates all packets with the same protocol number and destination port. The commands to perform analysis for the TCP packets are shown below. These same steps apply to analyzing UDP packets, but filtering with *grep* should be for protocol number 17 (UDP) instead of 6 (TCP).

```
// aggregate based on protocol and destnation port
$ cors-ft-aggregate -i 3600 -f protocol -f dst_port -v
     packet_cnt apr2012_ftlist >
     apr2012_proto_dport_dist.txt
$ cors-ft-aggregate -i 3600 -f protocol -f dst_port -v
     packet_cnt EW_ftlist > EW_dport_dist.txt
$ cors-ft-aggregate -i 3600 -f protocol -f dst_port -v
     packet_cnt PT_ftlist > PT_dport_dist.txt

// get columns for port,proto, packetcount
$ sed -e '/^\#/d' -e 's/|/,/g'
     apr2012_proto_dport_dist.txt | cut -f4,5,9 -d, >
     apr2012_proto_dport_dist_e.txt
$ sed -e '/^\#/d' -e 's/|/,/g' EW_dport_dist.txt | cut
     -f4,5,9 -d, > EW_dport_dist_e.txt
$ sed -e '/^\#/d' -e 's/|/,/g' PT_dport_dist.txt | cut
     -f4,5,9 -d, > PT_dport_dist_e.txt

// get only the TCP data
$ grep ',6,' apr2012_proto_dport_dist_e.txt >
     apr2012_TCP_dport_dist.txt
$ grep ',6,' PT_dport_dist_e.txt > PT_TCP_dport_dist.txt
$ grep ',6,' EW_dport_dist_e.txt > EW_TCP_dport_dist.txt
```

We now process and plot this data in *Octave*:

```
% read in data for TCP ports
> tcp_dport_all=csvread('apr2012_TCP_dport_dist.txt');
> tcp_dport_PT=csvread('PT_TCP_dport_dist.txt');
> tcp_dport_EW=csvread('EW_TCP_dport_dist.txt');

% get total number of tcp packets for the whole month
> tcp_pktcnt=sum(tcp_dport_all(:,3))
> tcp_pktcnt = 88227584302

% get total number of tcp packets for Exploit Wednesday
     00:00
> EW_tcp_pktcnt=sum(tcp_dport_EW(:,3))
> EW_tcp_pktcnt = 107461474

% get total number of tcp packets for Patch Tuesday
     17:00
> PT_tcp_pktcnt=sum(tcp_dport_PT(:,3))
> PT_tcp_pktcnt = 126622374

% plot data

> subplot(3,1,1);
> stem(tcp_dport_all(:,1),
     tcp_dport_all(:,3)*100/tcp_pktcnt, 'marker','x',
     'markersize',1);
> xlim([0,500]); % limits x-axis to first 500 ports
> set(gca,'TickDir','out')
> title('Distribution of TCP Destination Ports');
> legend('Whole Month April 2012','location','north');
> axis('labely');

> subplot(3,1,2)
> stem(tcp_dport_PT(:,1),
     tcp_dport_PT(:,3)*100/PT_tcp_pktcnt, 'marker','x',
     'markersize',1)
> xlim([0,500])
> set(gca,'TickDir','out')
> ylabel('packets/total packets [%]')
> legend('Patch Tuesday 17:00','location','north')
> axis('labely')
> numticks=5
> L=get(gca,'YLim')
> set(gca, 'YTick', linspace(L(1),L(2),numticks));

> subplot(3,1,3)
```

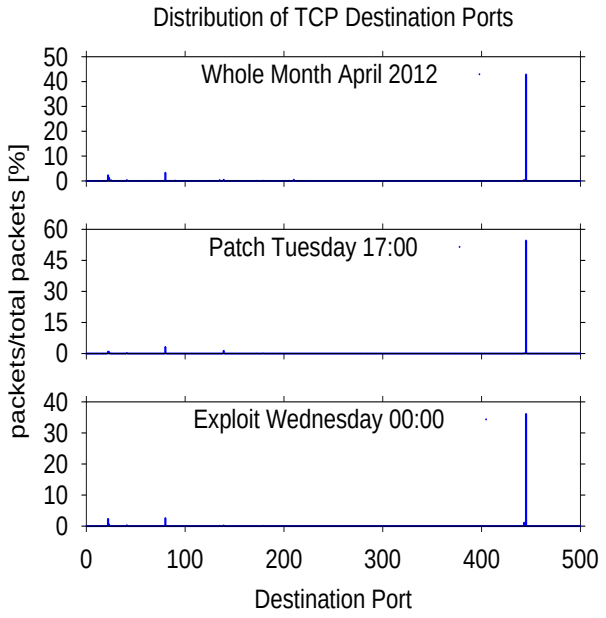

Figure 5: Distribution of Destination Ports. There is one prominent peak at the same destination port value in all three panels, and a few smaller peaks at varying positions.

```
> stem(tcp_dport_EW(:,1),
     tcp_dport_EW(:,3)*100/EW_tcp_pktcnt, 'marker','x',
     'markersize',1)
> xlim([0,500])
> set(gca,'TickDir','out')
> label('Destination Port')
> legend('Exploit Wednesday 00:00','location','north')
> numticks=5
> L=get(gca,'YLim')
> set(gca, 'YTick', linspace(L(1),L(2),numticks));
```

Figure 5 shows the resulting graph. The x-axis is limited to show only the first 500 destination ports using the `xlim([0,500])` *Octave* command. One can zoom in to see less common ports using the `ylim[0,limit]` command, e.g., `ylim([0,5])` will show only x-values that have y-axis values below 5% of the total packet count.

Each panel exhibits several peaks. To determine the exact port numbers at which those peaks occur, i.e., the most frequent TCP destination ports, we first sort the rows in the matrix according to the third column (number of packets) in descending order and then look at the top five port numbers in the sorted data:

```
> tcp_dport_all_sorted=sortrows(tcp_dport_all,-3);
> tcp_dport_PT_sorted=sortrows(tcp_dport_PT,-3);
> p_dport_EW_sorted=sortrows(tcp_dport_EW,-3);

> tcp_dport_all_sorted (1:5,:)
ans =
       445          6   37785558992
      1433          6    3651108893
      3389          6    3450635362
        80          6    2888150337
        22          6    1988126660

> tcp_dport_PT_sorted (1:5,:)
ans =
       445          6   69100424
      3389          6    4144063
        80          6    4000496
       139          6    1688069
      8080          6    1575357

> tcp_dport_EW_sorted (1:5,:)
ans =
       445          6   38787557
      3306          6   11750422
      3389          6    3534629
        80          6    2744440
      8080          6    2505147
```

The destination port 445 received the largest number of TCP packets for the whole month of April as well as during the individual hours on Patch Tuesday and Exploit Wednesday. TCP scans to port 445 are common, in large part due to scanning activity of Conficker-infected hosts [7].

5.5 Exercise PT-5: Analyzing the Source Types

Since we observed an unusually high hourly number of sources at midnight following Patch Tuesday, we want to analyze what types of darknet traffic sources became active. Pre-computed results of the source type analysis for the whole month of April are stored in the file *apr2012_src_types.txt*. Again, we use *Octave* and read the whole file into one matrix. We can look at the number of sources for different source types by varying the column that we select for the y-axis in the plots. In the following example, we plot the data in the fifth column, which contains the number of sources per hour of the type 'TCP horizontal scans' (see Table 3 in Section 4.4 to map columns to source types).

```
> src_types=csvread('apr2012_src_types.txt');
> stem(src_types(:,1), src_types(:,5), 'marker', 'none')
```

To find out which source types were unusually active during the first hour of the Exploit Wednesday 00:00, we compare the number of sources of each type during that hour with this type's mean hourly count. To do this analysis, first we need to check where in the matrix `src_types` we can find the data for the hour of Exploit Wednesday 00:00. We know that this hour coresponds to an epoch time of 1334102400. The *Octave* command `find` takes the epoch time as input and returns the corresponding index (the row number):

```
> index=find(src_types(:,1)==1334102400)
ans=241
```

The EW hour data is stored in the 241st row.

We then use a `for` loop in *Octave* to calculate the mean and ratio for each of the columns (each source type). We store the mean value for each source type i into a vector `mean_v(i)` and the ratio of the number of sources during the EW hour to the mean number of

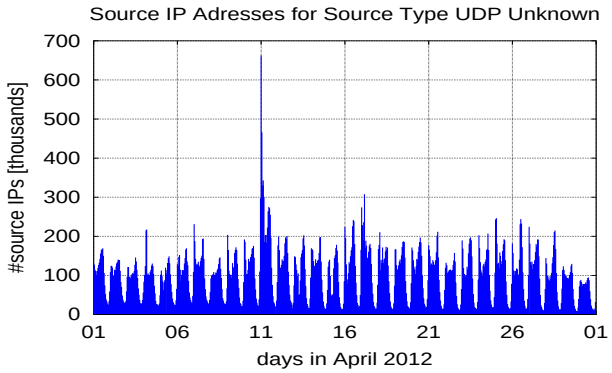

Figure 6: Unique Source IP Addresses per Hour for Source Type UDP Unknown

sources in the vector `ew_mean_ratio(i)`.

```
> for i = 1:columns(src_types)
> mean_v(i)=mean(src_types(:,i));
> ew_mean_ratio(i)= src_types(index,i)/mean_v(i);
> endfor
```

When displaying the values stored in `ew_mean_ratio`, we find the highest values in columns 10 (6.26) and 16 (4.58). Column 10 contains the source type 'UDP unknown' and column 16 the source type '1 or 2 packets' (cf. Table 3). So, these two source types contribute significantly to the increase in the overall number of sources. Figure 6 produced by the following *Octave* code illustrates this behavior.

```
> stem(src_types(:,1), src_types(:,10), 'marker',
    'none')
> stem(datenum(1970,1,1,0,0,src_types(:,1)),
    src_types(:,10)/10^3, 'marker', 'none')
> datetick('x', 'dd', 'keepticks');
> xlabel('days in April 2012')
> ylabel('#source IPs [thousands]')
> title('Source IP Adresses for Source Type UDP
    Unknown')
> grid on
```

5.6 Exercise PT-6 (Advanced): Analyzing Temporal Patterns

This exercise is advanced since it requires basic knowledge of signal processing.

Time series of hourly counts of packets (Figure 2) and of unique source addresses (Figure 3) both seem to exhibit periodic variations, the temporal pattern being more noticeable for the latter. In this Section, we show how to analyze the temporal behavior of these darknet traffic characteristics by using the frequency spectrum of the time series signal.

The frequency spectrum represents the time series signal as a superposition of multiple sinusoidal signals. Each sinusoidal signal is defined by its amplitude, frequency, and phase shift. The Fourier transform is a mathematical procedure used to calculate the frequency spectrum and convert the signal from the time domain into the frequency domain.

The signal in our time series signal is discrete as it contains one packet count (or source count) value per hour. For transforming the discrete finite time signal we use the Discrete Fourier Transform (DFT). For a time series of N data points $x_0 ... x_n ... x_{N-1}$, the DFT calculates a set of N complex numbers $X_0 ... X_k ... X_{N-1}$:

$$X_k = \sum_{k=0}^{N-1} x_n e^{-i2\pi k \frac{n}{N}}$$

where each complex number X_k represents a sinusoidal signal that contributes to the time series. Our time series has one data point per hour and contains data from 30 days. In total we have $N = 24 \times 30 = 720$ data points, which means $N = 720$ complex numbers in the frequency domain.

A fast way to calculate the DFT is the Fast Fourier Transform (FFT) algorithm. *Octave* provides the function `fft(x)` for calculating the FFT from a discrete time signal with values stored in the vector x:

```
> N=length(apr2012_pkt(:,2))
> pkt_fft=fft(apr2012_pkt(:,2));
> pkt_amp=abs(pkt_fft);
```

The packet counts per hour are stored in the second column of the matrix `apr2012_pkt`; the resulting complex spectrum values are in `pkt_fft`. The *Octave* function `abs()` yields the amplitudes (the absolute values) of the complex numbers calculated by the FFT.

We plot the amplitudes of the calculated spectrum with the index k on the x-axis and the vertical line showing the signal amplitude at the corresponding k:

```
> k=(0:N-1);
> stem(k(2:(N/2)+1),pkt_amp(2:(N/2)+1)/10^6, 'marker',
    'none')
> xlabel('k')
> ylabel('Amplitude [millions of packets]')
> title('Amplitude Spectrum for Number of Packets
    [millions]')
```

We only need to plot the first $N/2$ coefficients because the spectrum repeats itself. Also, since the first value X_0 at $k = 0$ represents the signal offset, we exclude it from the plot of the frequency spectrum in order to make the other frequencies more visible. Note that *Octave* indexing always starts at 1, whereas k in the DFT formula starts at 0. So, to plot the data from $k = 1$ to $k = N/2$ we need to use the indices $ind = k + 1 = 2$ to $ind = k + 1 = (N/2) + 1$.

Figure 7 shows the resulting plot. Each k corresponds to the number of cycles for the sinusoidal signal within the whole duration of the analyzed time series (720 hours). Therefore, the period of the k-th signal is $p_k = \frac{720\ hours}{k\ cycles}$ and the corresponding frequency is $f_k = \frac{1}{p_k} = \frac{k\ cycles}{720\ hours}$.

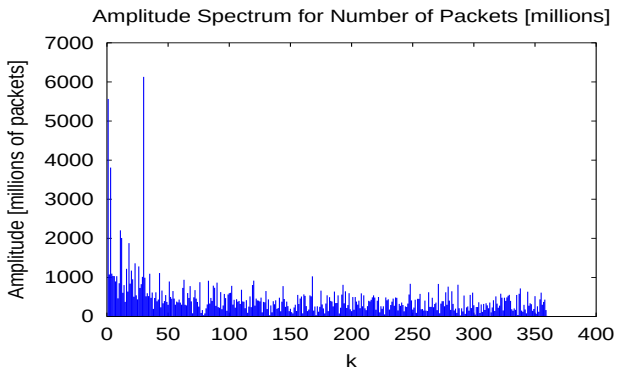

Figure 7: Spectrum for number of packets per hour

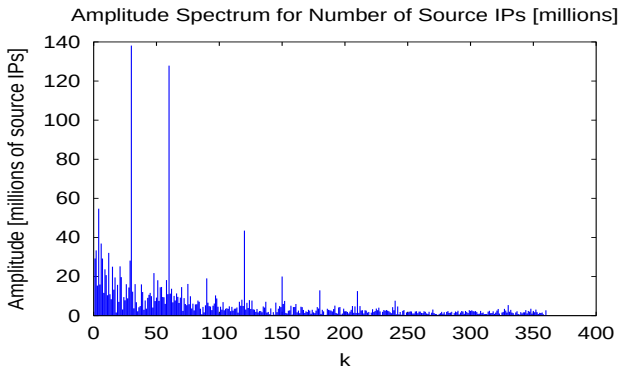

Figure 8: Spectrum for number of source IPs per hour

In the diagram we see a high amplitude for the sinusoidal signal with $k = 30$. The period for this signal is $p_{30} = \frac{720\ h}{30\ cycles} = 24\ hours$, which corresponds to a diurnal pattern in the data. Diurnal patterns are common in Internet traffic data since many hosts are turned off at night.

We perform the same steps to calculate and plot the spectrum of hourly counts of unique source addresses (Figure 8):

```
> src_fft=fft(apr2012_src(:,2));
> src_amp=abs(src_fft);
> stem(k(2:(N/2)+1),src_amp(2:(N/2)+1)/10^6, 'marker',
    'none')
> xlabel('k')
> ylabel('Amplitude [millions of source IPs]')
> title('Amplitude Spectrum for Number of Source IPs
    [millions]')
```

k and N values remain the same. Again, we see a strong spectrum amplitude at $k = 30$ that corresponds to a 24-hour pattern. We also observe a peak at $k = 60$, which indicates a 12-hour pattern. One behavior that could generate this traffic pattern is using 12-hour time format for some internal functions.

6. CONCLUSION

We have described an educational data kit curated from darkspace traffic observed by the UCSD Network Telescope in April 2012, and presented a set of exercises developed for this data. After completing the exercises, the students will be familiar with the raw format used to store the captured traffic packets (*pcap*), various aggregations of the raw data, and basic statistics commonly used for traffic characterization. They will acquire hands-on experience in using specialized traffic analysis software *Corsaro* and a scripting high-level computational language *Octave*. The students should be able to apply this knowledge to analyze other packet trace data available in (*pcap*) format, including bi-directional data. One can find additional relevant reading in CAIDA's online papers directory [1] by filtering for keyword "network telescope".

Acknowledgments

The work was supported by U.S. NSF grants II-EN-1059439 and CNS-1228994, DHS S&T Cyber Security Division (DHS S&T/CSD) Cooperative Agreement FA8750-12-2-0326 (PREDICT project) and by Fraunhofer FOKUS. This material represents the position of the authors and not of the sponsoring agencies.

7. REFERENCES

[1] CAIDA research papers. http://www.caida.org/publications/papers/.
[2] GNU Octave. www.gnu.org/software/octave/.
[3] Patch Tuesday. http://en.wikipedia.org/wiki/Patch_Tuesday.
[4] tcpdump. http://www.tcpdump.org/.
[5] UCSD Network Telescope Global Attack Traffic. http://www.caida.org/data/realtime/telescope/.
[6] The UCSD Network Telescope, 2001.
[7] Emile Aben. Conficker/Conflicker/Downadup as seen from the UCSD Network Telescope. Technical report, CAIDA, February 2009.
[8] Nevil Brownlee. iatmon. http://www.caida.org/tools/measurement/iatmon/.
[9] Alistair King. Corsaro. http://www.caida.org/tools/corsaro/, October 2012.
[10] Alistair King. Corsaro file formats. http://www.caida.org/tools/corsaro/docs/formats.html, October 2012.
[11] Alistair King. Corsaro quick-start guide. http://caida.org/tools/corsaro/docs/quickstart.html, October 2012.
[12] David Moore, Colleen Shannon, Douglas J Brown, Geoffrey M Voelker, and Stefan Savage. Inferring Internet denial-of-service activity. *ACM Trans. Comput. Syst.*, 24(2):115139, May 2006. ACM ID: 1132027.