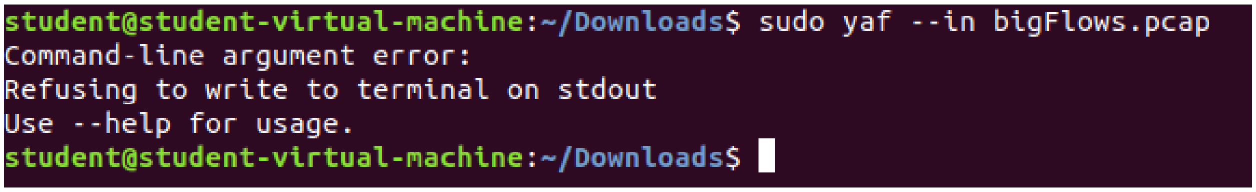
Network Security and Forensics
CSEC.462.01

Lab 4

Student: Wissam El Labban

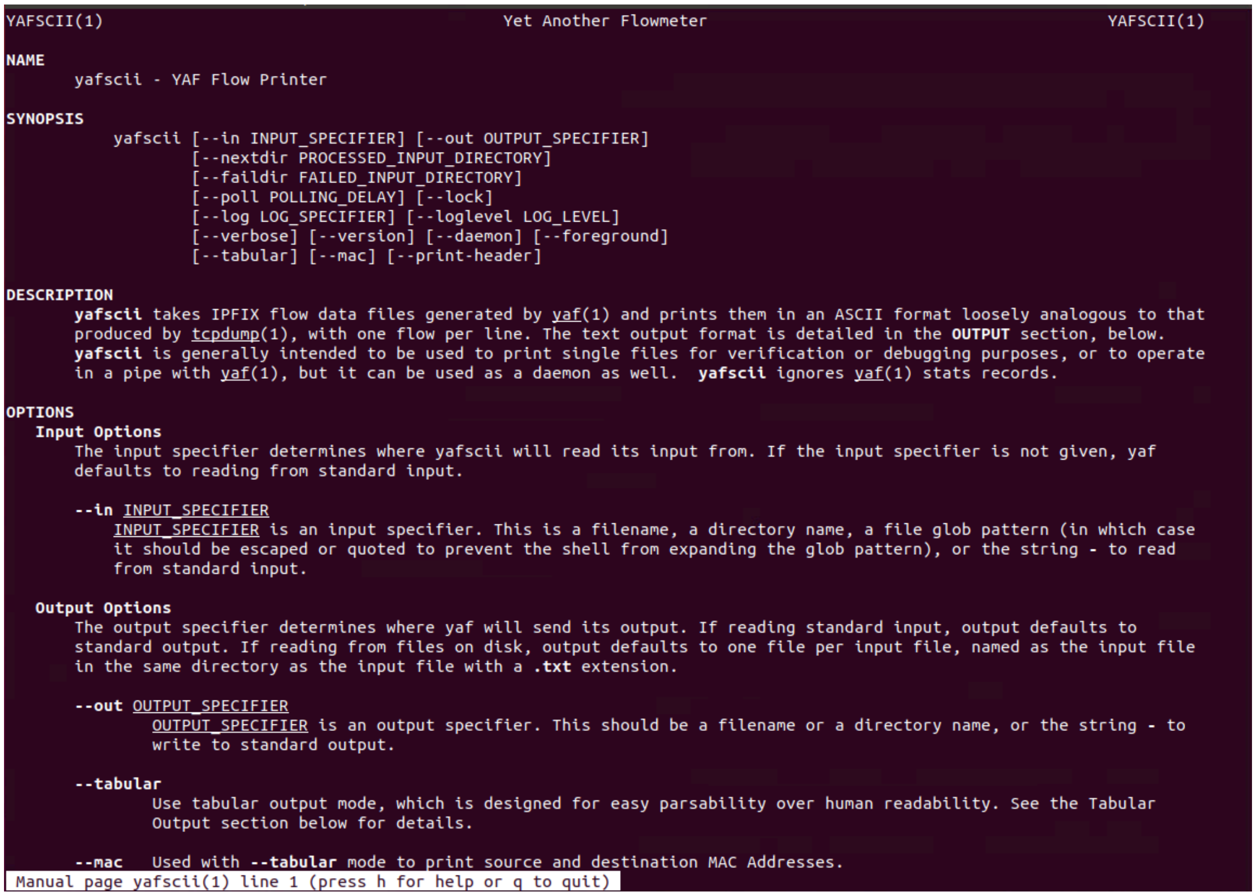
Table of Contents

Q1



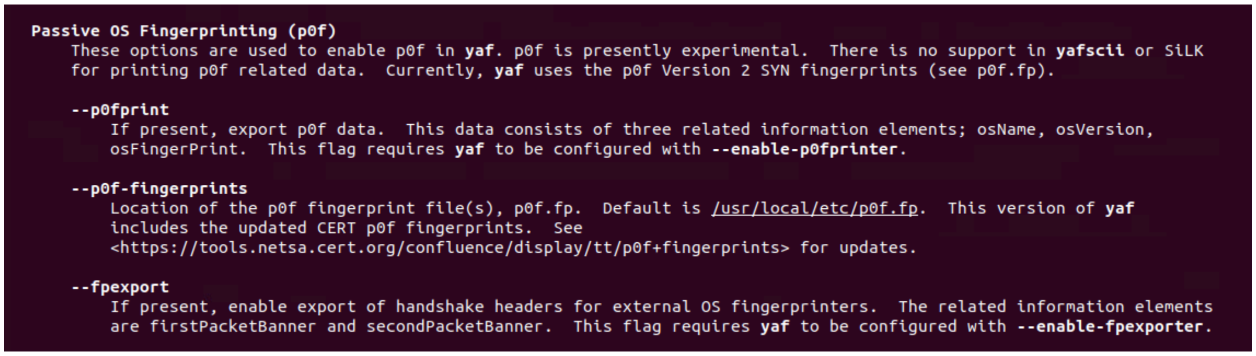
```
student@student-virtual-machine:~/Downloads$ sudo yaf --in bigFlows.pcap
Command-line argument error:
Refusing to write to terminal on stdout
Use --help for usage.
student@student-virtual-machine:~/Downloads$
```

It is refusing to write on stdout because no –out option was specified, and it cannot print in a format the terminal can display. This is because it outputs in binary format. A tool that can be used to print that format is the yafscii tool.



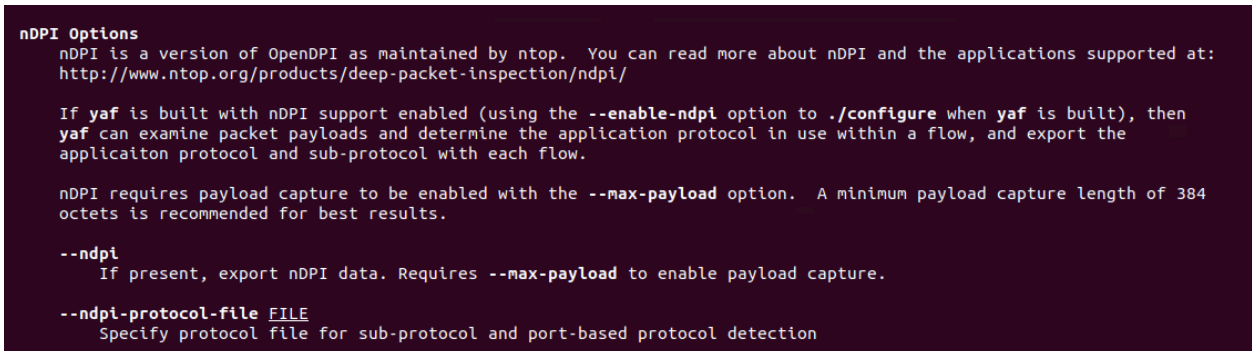
```
YAFSCII(1)                          Yet Another Flowmeter                          YAFSCII(1)

NAME
       yafscii - YAF Flow Printer

SYNOPSIS
           yafscii [--in INPUT_SPECIFIER] [--out OUTPUT_SPECIFIER]
                   [--nextdir PROCESSED_INPUT_DIRECTORY]
                   [--faildir FAILED_INPUT_DIRECTORY]
                   [--poll POLLING_DELAY] [--lock]
                   [--log LOG_SPECIFIER] [--loglevel LOG_LEVEL]
                   [--verbose] [--version] [--daemon] [--foreground]
                   [--tabular] [--mac] [--print-header]

DESCRIPTION
       yafscii takes IPFIX flow data files generated by yaf(1) and prints them in an ASCII format loosely analogous to that
       produced by tcpdump(1), with one flow per line. The text output format is detailed in the OUTPUT section, below.
       yafscii is generally intended to be used to print single files for verification or debugging purposes, or to operate
       in a pipe with yaf(1), but it can be used as a daemon as well.  yafscii ignores yaf(1) stats records.

OPTIONS
   Input Options
       The input specifier determines where yafscii will read its input from. If the input specifier is not given, yaf
       defaults to reading from standard input.

       --in INPUT_SPECIFIER
           INPUT_SPECIFIER is an input specifier. This is a filename, a directory name, a file glob pattern (in which case
           it should be escaped or quoted to prevent the shell from expanding the glob pattern), or the string - to read
           from standard input.

   Output Options
       The output specifier determines where yaf will send its output. If reading standard input, output defaults to
       standard output. If reading from files on disk, output defaults to one file per input file, named as the input file
       in the same directory as the input file with a .txt extension.

       --out OUTPUT_SPECIFIER
               OUTPUT_SPECIFIER is an output specifier. This should be a filename or a directory name, or the string - to
               write to standard output.

       --tabular
               Use tabular output mode, which is designed for easy parsability over human readability. See the Tabular
               Output section below for details.

       --mac   Used with --tabular mode to print source and destination MAC Addresses.
 Manual page yafscii(1) line 1 (press h for help or q to quit)
```

Q2



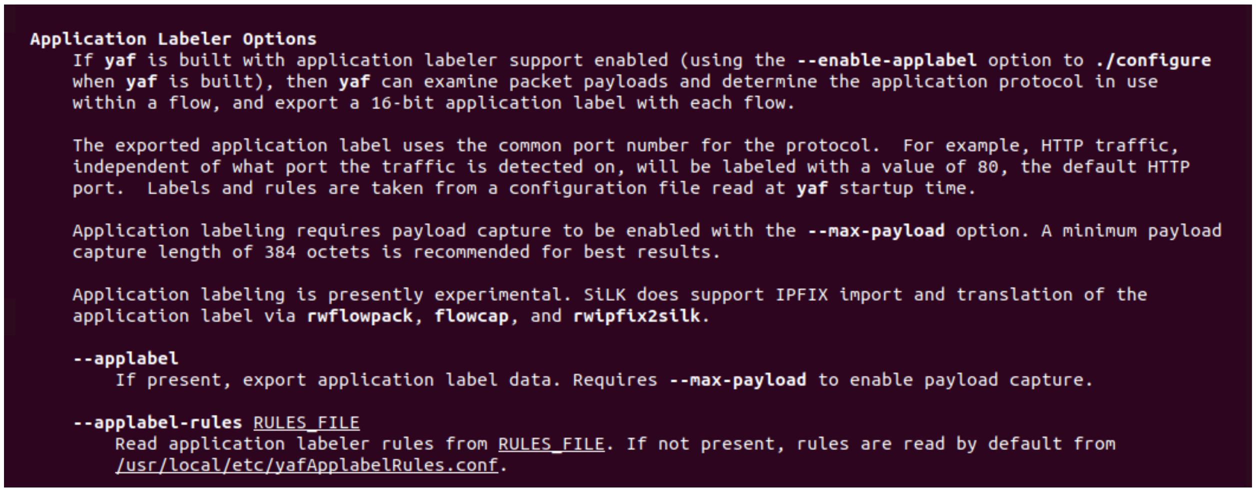
```
Passive OS Fingerprinting (p0f)
    These options are used to enable p0f in yaf. p0f is presently experimental.  There is no support in yafscii or SiLK
    for printing p0f related data.  Currently, yaf uses the p0f Version 2 SYN fingerprints (see p0f.fp).

    --p0fprint
        If present, export p0f data.  This data consists of three related information elements; osName, osVersion,
        osFingerPrint.  This flag requires yaf to be configured with --enable-p0fprinter.

    --p0f-fingerprints
        Location of the p0f fingerprint file(s), p0f.fp.  Default is /usr/local/etc/p0f.fp.  This version of yaf
        includes the updated CERT p0f fingerprints.  See
        <https://tools.netsa.cert.org/confluence/display/tt/p0f+fingerprints> for updates.

    --fpexport
        If present, enable export of handshake headers for external OS fingerprinters.  The related information elements
        are firstPacketBanner and secondPacketBanner.  This flag requires yaf to be configured with --enable-fpexporter.
```

The p0f option allows you to match the network traffic characteristics of a host against signatures of operating systems like their TCP/IP stack implementation. P0f also analyzes the network traffic passively.



```
nDPI Options
    nDPI is a version of OpenDPI as maintained by ntop.  You can read more about nDPI and the applications supported at:
    http://www.ntop.org/products/deep-packet-inspection/ndpi/

    If yaf is built with nDPI support enabled (using the --enable-ndpi option to ./configure when yaf is built), then
    yaf can examine packet payloads and determine the application protocol in use within a flow, and export the
    applicaiton protocol and sub-protocol with each flow.

    nDPI requires payload capture to be enabled with the --max-payload option.  A minimum payload capture length of 384
    octets is recommended for best results.

    --ndpi
        If present, export nDPI data. Requires --max-payload to enable payload capture.

    --ndpi-protocol-file FILE
        Specify protocol file for sub-protocol and port-based protocol detection
```

Deep packet inspection analyzes the payload packets rather than just the headers of the packet. It allows yaf to do a more advanced analysis of the network traffic. Predefined signatures are compared with traffic payload to help identify the type of traffic that it is.



```
Application Labeler Options
    If yaf is built with application labeler support enabled (using the --enable-applabel option to ./configure
    when yaf is built), then yaf can examine packet payloads and determine the application protocol in use
    within a flow, and export a 16-bit application label with each flow.

    The exported application label uses the common port number for the protocol.  For example, HTTP traffic,
    independent of what port the traffic is detected on, will be labeled with a value of 80, the default HTTP
    port.  Labels and rules are taken from a configuration file read at yaf startup time.

    Application labeling requires payload capture to be enabled with the --max-payload option. A minimum payload
    capture length of 384 octets is recommended for best results.

    Application labeling is presently experimental. SiLK does support IPFIX import and translation of the
    application label via rwflowpack, flowcap, and rwipfix2silk.

    --applabel
        If present, export application label data. Requires --max-payload to enable payload capture.

    --applabel-rules RULES_FILE
        Read application labeler rules from RULES_FILE. If not present, rules are read by default from
        /usr/local/etc/yafApplabelRules.conf.
```

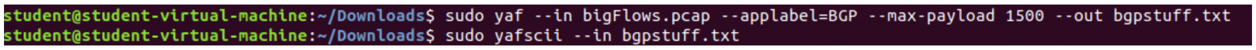
App labeling determines the application protocol that is in use to identify traffic in a network flow. It's a part of deep packet inspection since it analyzes the payload in each packet to determine the app that generated the examined traffic.

Q3

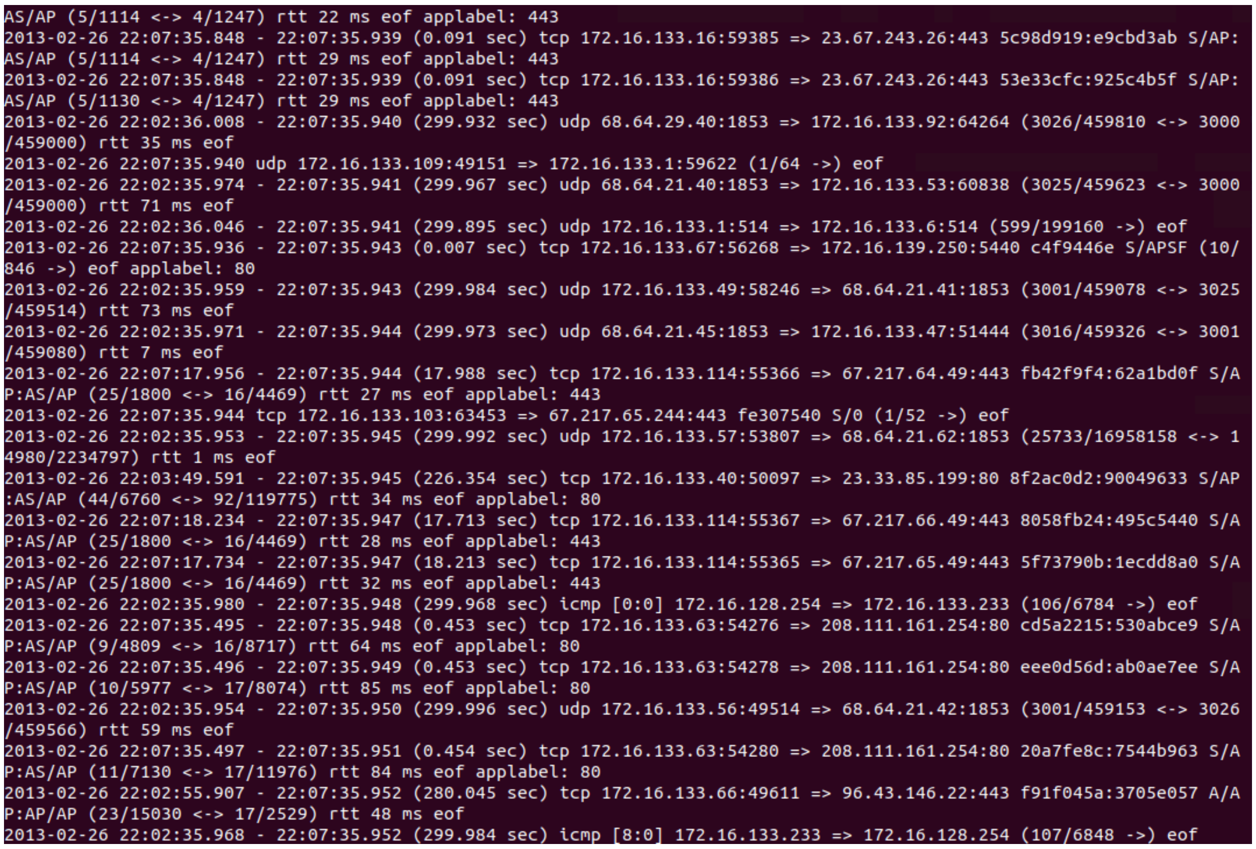
YAF does its labeling by inspecting network traffic and matches it to application protocols and signatures associated with those protocols. It has built in signatures for used applications, but can also use custom signatures or third party signatures.

YAF adds a field to the flow record when it detects an application protocol.

The commands below give us the BGP traffic



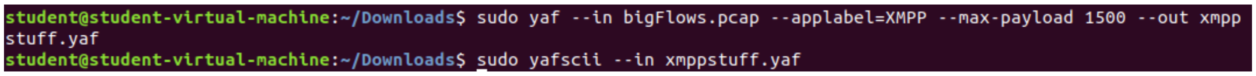
```
student@student-virtual-machine:~/Downloads$ sudo yaf --in bigFlows.pcap --applabel=BGP --max-payload 1500 --out bgpstuff.txt
student@student-virtual-machine:~/Downloads$ sudo yafscii --in bgpstuff.txt
```



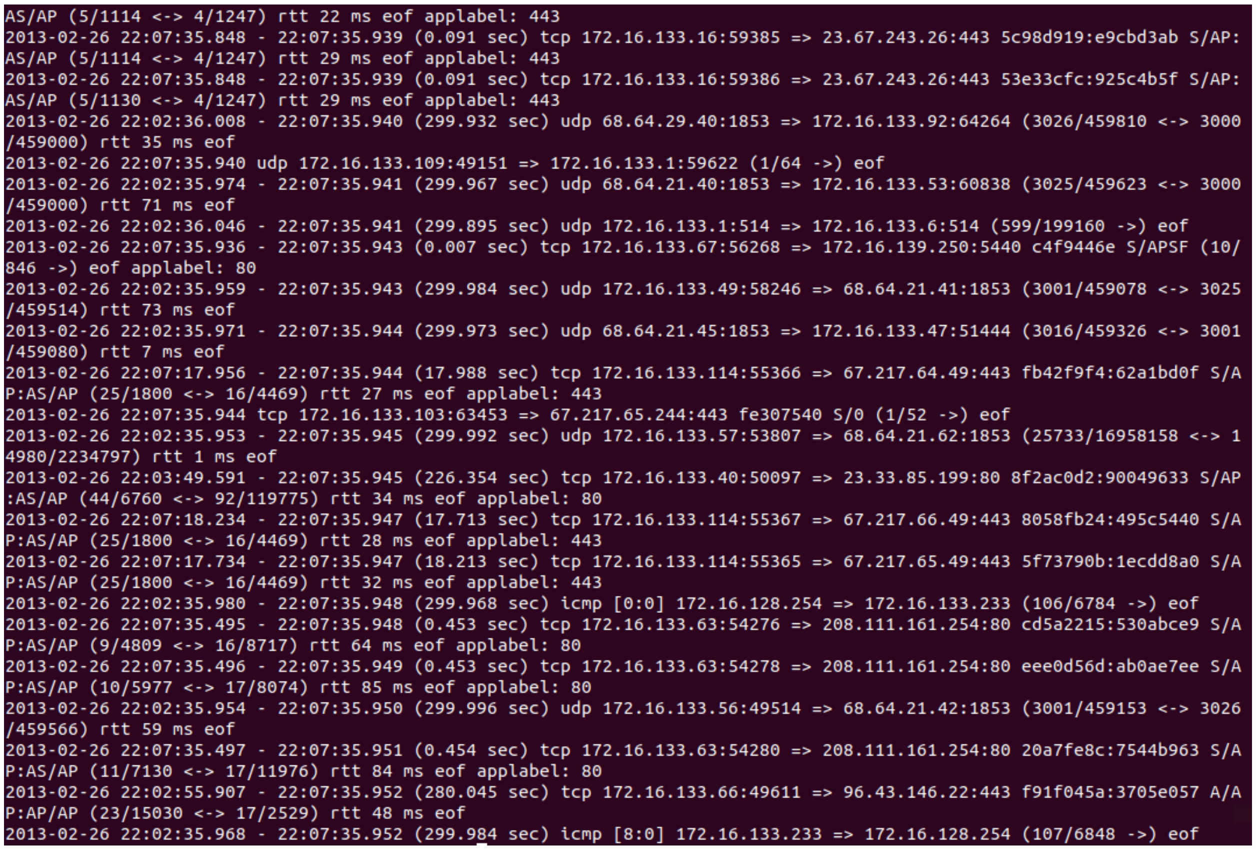
```
AS/AP (5/1114 <-> 4/1247) rtt 22 ms eof applabel: 443
2013-02-26 22:07:35.848 - 22:07:35.939 (0.091 sec) tcp 172.16.133.16:59385 => 23.67.243.26:443 5c98d919:e9cbd3ab S/AP:
AS/AP (5/1114 <-> 4/1247) rtt 29 ms eof applabel: 443
2013-02-26 22:07:35.848 - 22:07:35.939 (0.091 sec) tcp 172.16.133.16:59386 => 23.67.243.26:443 53e33cfc:925c4b5f S/AP:
AS/AP (5/1130 <-> 4/1247) rtt 29 ms eof applabel: 443
2013-02-26 22:02:36.008 - 22:07:35.940 (299.932 sec) udp 68.64.29.40:1853 => 172.16.133.92:64264 (3026/459810 <-> 3000
/459000) rtt 35 ms eof
2013-02-26 22:07:35.940 udp 172.16.133.109:49151 => 172.16.133.1:59622 (1/64 ->) eof
2013-02-26 22:02:35.974 - 22:07:35.941 (299.967 sec) udp 68.64.21.40:1853 => 172.16.133.53:60838 (3025/459623 <-> 3000
/459000) rtt 71 ms eof
2013-02-26 22:02:36.046 - 22:07:35.941 (299.895 sec) udp 172.16.133.1:514 => 172.16.133.6:514 (599/199160 ->) eof
2013-02-26 22:07:35.936 - 22:07:35.943 (0.007 sec) tcp 172.16.133.67:56268 => 172.16.139.250:5440 c4f9446e S/APSF (10/
846 ->) eof applabel: 80
2013-02-26 22:02:35.959 - 22:07:35.943 (299.984 sec) udp 172.16.133.49:58246 => 68.64.21.41:1853 (3001/459078 <-> 3025
/459514) rtt 73 ms eof
2013-02-26 22:02:35.971 - 22:07:35.944 (299.973 sec) udp 68.64.21.45:1853 => 172.16.133.47:51444 (3016/459326 <-> 3001
/459080) rtt 7 ms eof
2013-02-26 22:07:17.956 - 22:07:35.944 (17.988 sec) tcp 172.16.133.114:55366 => 67.217.64.49:443 fb42f9f4:62a1bd0f S/A
P:AS/AP (25/1800 <-> 16/4469) rtt 27 ms eof applabel: 443
2013-02-26 22:07:35.944 tcp 172.16.133.103:63453 => 67.217.65.244:443 fe307540 S/0 (1/52 ->) eof
2013-02-26 22:02:35.953 - 22:07:35.945 (299.992 sec) udp 172.16.133.57:53807 => 68.64.21.62:1853 (25733/16958158 <-> 1
4980/2234797) rtt 1 ms eof
2013-02-26 22:03:49.591 - 22:07:35.945 (226.354 sec) tcp 172.16.133.40:50097 => 23.33.85.199:80 8f2ac0d2:90049633 S/AP
:AS/AP (44/6760 <-> 92/119775) rtt 34 ms eof applabel: 80
2013-02-26 22:07:18.234 - 22:07:35.947 (17.713 sec) tcp 172.16.133.114:55367 => 67.217.66.49:443 8058fb24:495c5440 S/A
P:AS/AP (25/1800 <-> 16/4469) rtt 28 ms eof applabel: 443
2013-02-26 22:07:17.734 - 22:07:35.947 (18.213 sec) tcp 172.16.133.114:55365 => 67.217.65.49:443 5f73790b:1ecdd8a0 S/A
P:AS/AP (25/1800 <-> 16/4469) rtt 32 ms eof applabel: 443
2013-02-26 22:02:35.980 - 22:07:35.948 (299.968 sec) icmp [0:0] 172.16.128.254 => 172.16.133.233 (106/6784 ->) eof
2013-02-26 22:07:35.495 - 22:07:35.948 (0.453 sec) tcp 172.16.133.63:54276 => 208.111.161.254:80 cd5a2215:530abce9 S/A
P:AS/AP (9/4809 <-> 16/8717) rtt 64 ms eof applabel: 80
2013-02-26 22:07:35.496 - 22:07:35.949 (0.453 sec) tcp 172.16.133.63:54278 => 208.111.161.254:80 eee0d56d:ab0ae7ee S/A
P:AS/AP (10/5977 <-> 17/8074) rtt 85 ms eof applabel: 80
2013-02-26 22:02:35.954 - 22:07:35.950 (299.996 sec) udp 172.16.133.56:49514 => 68.64.21.42:1853 (3001/459153 <-> 3026
/459566) rtt 59 ms eof
2013-02-26 22:07:35.497 - 22:07:35.951 (0.454 sec) tcp 172.16.133.63:54280 => 208.111.161.254:80 20a7fe8c:7544b963 S/A
P:AS/AP (11/7130 <-> 17/11976) rtt 84 ms eof applabel: 80
2013-02-26 22:02:55.907 - 22:07:35.952 (280.045 sec) tcp 172.16.133.66:49611 => 96.43.146.22:443 f91f045a:3705e057 A/A
P:AP/AP (23/15030 <-> 17/2529) rtt 48 ms eof
2013-02-26 22:02:35.968 - 22:07:35.952 (299.984 sec) icmp [8:0] 172.16.133.233 => 172.16.128.254 (107/6848 ->) eof
```

The above is what some of the lines for the BGP look like.

The commands below give us the XMPP traffic



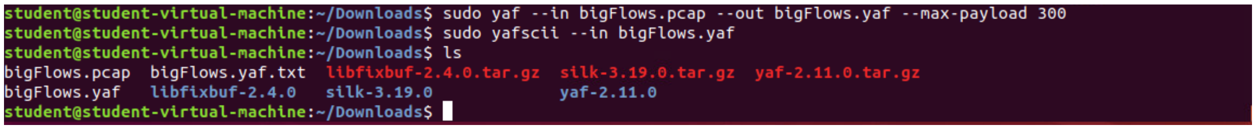
```
student@student-virtual-machine:~/Downloads$ sudo yaf --in bigFlows.pcap --applabel=XMPP --max-payload 1500 --out xmpp
stuff.yaf
student@student-virtual-machine:~/Downloads$ sudo yafscii --in xmppstuff.yaf
```



```
AS/AP (5/1114 <-> 4/1247) rtt 22 ms eof applabel: 443
2013-02-26 22:07:35.848 - 22:07:35.939 (0.091 sec) tcp 172.16.133.16:59385 => 23.67.243.26:443 5c98d919:e9cbd3ab S/AP:
AS/AP (5/1114 <-> 4/1247) rtt 29 ms eof applabel: 443
2013-02-26 22:07:35.848 - 22:07:35.939 (0.091 sec) tcp 172.16.133.16:59386 => 23.67.243.26:443 53e33cfc:925c4b5f S/AP:
AS/AP (5/1130 <-> 4/1247) rtt 29 ms eof applabel: 443
2013-02-26 22:02:36.008 - 22:07:35.940 (299.932 sec) udp 68.64.29.40:1853 => 172.16.133.92:64264 (3026/459810 <-> 3000
/459000) rtt 35 ms eof
2013-02-26 22:07:35.940 udp 172.16.133.109:49151 => 172.16.133.1:59622 (1/64 ->) eof
2013-02-26 22:02:35.974 - 22:07:35.941 (299.967 sec) udp 68.64.21.40:1853 => 172.16.133.53:60838 (3025/459623 <-> 3000
/459000) rtt 71 ms eof
2013-02-26 22:02:36.046 - 22:07:35.941 (299.895 sec) udp 172.16.133.1:514 => 172.16.133.6:514 (599/199160 ->) eof
2013-02-26 22:07:35.936 - 22:07:35.943 (0.007 sec) tcp 172.16.133.67:56268 => 172.16.139.250:5440 c4f9446e S/APSF (10/
846 ->) eof applabel: 80
2013-02-26 22:02:35.959 - 22:07:35.943 (299.984 sec) udp 172.16.133.49:58246 => 68.64.21.41:1853 (3001/459078 <-> 3025
/459514) rtt 73 ms eof
2013-02-26 22:02:35.971 - 22:07:35.944 (299.973 sec) udp 68.64.21.45:1853 => 172.16.133.47:51444 (3016/459326 <-> 3001
/459080) rtt 7 ms eof
2013-02-26 22:07:17.956 - 22:07:35.944 (17.988 sec) tcp 172.16.133.114:55366 => 67.217.64.49:443 fb42f9f4:62a1bd0f S/A
P:AS/AP (25/1800 <-> 16/4469) rtt 27 ms eof applabel: 443
2013-02-26 22:07:35.944 tcp 172.16.133.103:63453 => 67.217.65.244:443 fe307540 S/0 (1/52 ->) eof
2013-02-26 22:02:35.953 - 22:07:35.945 (299.992 sec) udp 172.16.133.57:53807 => 68.64.21.62:1853 (25733/16958158 <-> 1
4980/2234797) rtt 1 ms eof
2013-02-26 22:03:49.591 - 22:07:35.945 (226.354 sec) tcp 172.16.133.40:50097 => 23.33.85.199:80 8f2ac0d2:90049633 S/AP
:AS/AP (44/6760 <-> 92/119775) rtt 34 ms eof applabel: 80
2013-02-26 22:07:18.234 - 22:07:35.947 (17.713 sec) tcp 172.16.133.114:55367 => 67.217.66.49:443 8058fb24:495c5440 S/A
P:AS/AP (25/1800 <-> 16/4469) rtt 28 ms eof applabel: 443
2013-02-26 22:07:17.734 - 22:07:35.947 (18.213 sec) tcp 172.16.133.114:55365 => 67.217.65.49:443 5f73790b:1ecdd8a0 S/A
P:AS/AP (25/1800 <-> 16/4469) rtt 32 ms eof applabel: 443
2013-02-26 22:02:35.980 - 22:07:35.948 (299.968 sec) icmp [0:0] 172.16.128.254 => 172.16.133.233 (106/6784 ->) eof
2013-02-26 22:07:35.495 - 22:07:35.948 (0.453 sec) tcp 172.16.133.63:54276 => 208.111.161.254:80 cd5a2215:530abce9 S/A
P:AS/AP (9/4809 <-> 16/8717) rtt 64 ms eof applabel: 80
2013-02-26 22:07:35.496 - 22:07:35.949 (0.453 sec) tcp 172.16.133.63:54278 => 208.111.161.254:80 eee0d56d:ab0ae7ee S/A
P:AS/AP (10/5977 <-> 17/8074) rtt 85 ms eof applabel: 80
2013-02-26 22:02:35.954 - 22:07:35.950 (299.996 sec) udp 172.16.133.56:49514 => 68.64.21.42:1853 (3001/459153 <-> 3026
/459566) rtt 59 ms eof
2013-02-26 22:07:35.497 - 22:07:35.951 (0.454 sec) tcp 172.16.133.63:54280 => 208.111.161.254:80 20a7fe8c:7544b963 S/A
P:AS/AP (11/7130 <-> 17/11976) rtt 84 ms eof applabel: 80
2013-02-26 22:02:55.907 - 22:07:35.952 (280.045 sec) tcp 172.16.133.66:49611 => 96.43.146.22:443 f91f045a:3705e057 A/A
P:AP/AP (23/15030 <-> 17/2529) rtt 48 ms eof
2013-02-26 22:02:35.968 - 22:07:35.952 (299.984 sec) icmp [8:0] 172.16.133.233 => 172.16.128.254 (107/6848 ->) eof
```

The above is what some of the lines for the xmpp look like.

Q4

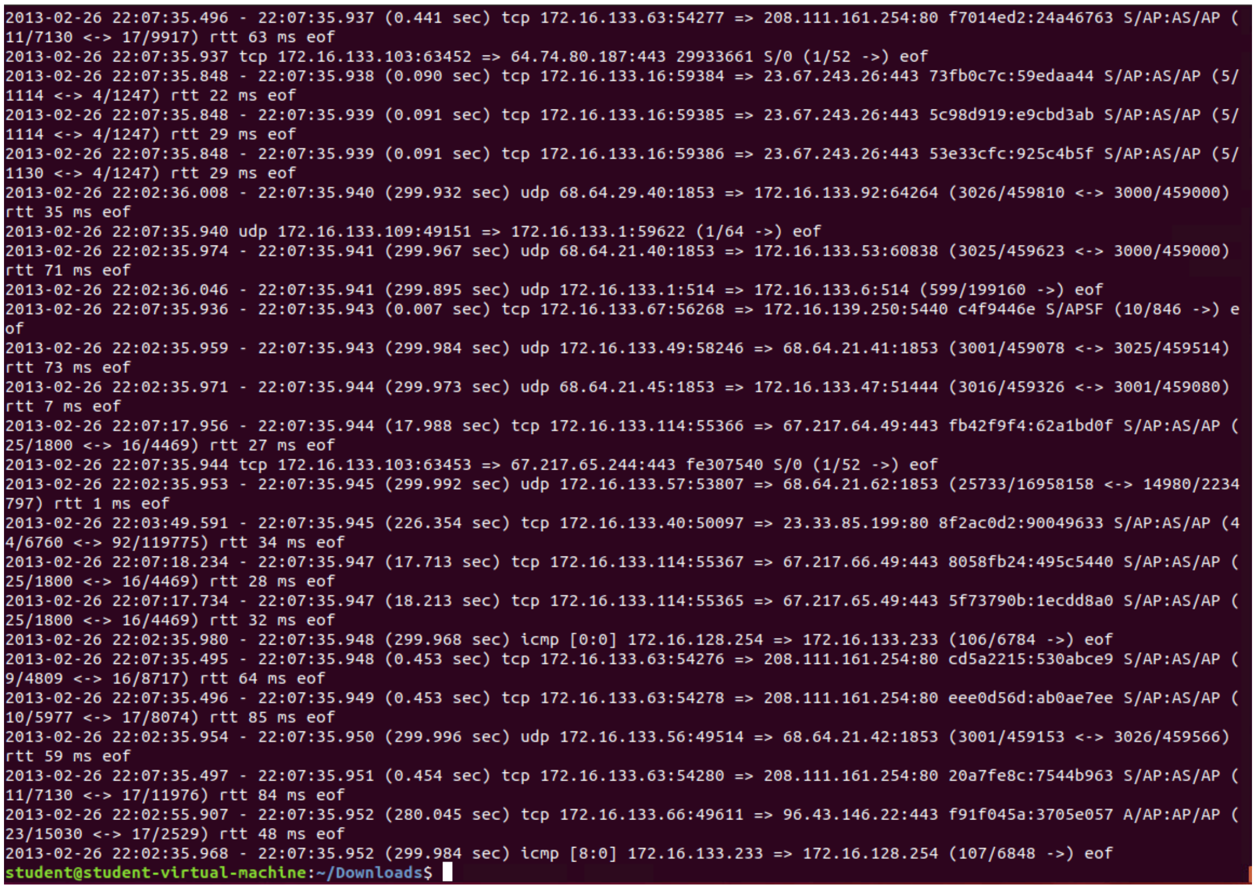


```
student@student-virtual-machine:~/Downloads$ sudo yaf --in bigFlows.pcap --out bigFlows.yaf --max-payload 300
student@student-virtual-machine:~/Downloads$ sudo yafscii --in bigFlows.yaf
student@student-virtual-machine:~/Downloads$ ls
bigFlows.pcap  bigFlows.yaf.txt  libfixbuf-2.4.0.tar.gz  silk-3.19.0.tar.gz  yaf-2.11.0.tar.gz
bigFlows.yaf   libfixbuf-2.4.0   silk-3.19.0             yaf-2.11.0
student@student-virtual-machine:~/Downloads$ 
```

I can now see a bigFlows.yaf and a bigFlows.yaf.txt.

```
[illegible]
student@student-virtual-machine:~/Downloads$
```

Cat bigFlows.yaf gives gibberish text.

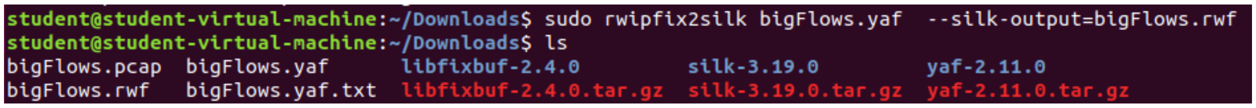


```
2013-02-26 22:07:35.496 - 22:07:35.937 (0.441 sec) tcp 172.16.133.63:54277 => 208.111.161.254:80 f7014ed2:24a46763 S/AP:AS/AP (
11/7130 <-> 17/9917) rtt 63 ms eof
2013-02-26 22:07:35.937 tcp 172.16.133.103:63452 => 64.74.80.187:443 29933661 S/0 (1/52 ->) eof
2013-02-26 22:07:35.848 - 22:07:35.938 (0.090 sec) tcp 172.16.133.16:59384 => 23.67.243.26:443 73fb0c7c:59edaa44 S/AP:AS/AP (5/
1114 <-> 4/1247) rtt 22 ms eof
2013-02-26 22:07:35.848 - 22:07:35.939 (0.091 sec) tcp 172.16.133.16:59385 => 23.67.243.26:443 5c98d919:e9cbd3ab S/AP:AS/AP (5/
1114 <-> 4/1247) rtt 29 ms eof
2013-02-26 22:07:35.848 - 22:07:35.939 (0.091 sec) tcp 172.16.133.16:59386 => 23.67.243.26:443 53e33cfc:925c4b5f S/AP:AS/AP (5/
1130 <-> 4/1247) rtt 29 ms eof
2013-02-26 22:02:36.008 - 22:07:35.940 (299.932 sec) udp 68.64.29.40:1853 => 172.16.133.92:64264 (3026/459810 <-> 3000/459000)
rtt 35 ms eof
2013-02-26 22:07:35.940 udp 172.16.133.109:49151 => 172.16.133.1:59622 (1/64 ->) eof
2013-02-26 22:02:35.974 - 22:07:35.941 (299.967 sec) udp 68.64.21.40:1853 => 172.16.133.53:60838 (3025/459623 <-> 3000/459000)
rtt 71 ms eof
2013-02-26 22:02:36.046 - 22:07:35.941 (299.895 sec) udp 172.16.133.1:514 => 172.16.133.6:514 (599/199160 ->) eof
2013-02-26 22:07:35.936 - 22:07:35.943 (0.007 sec) tcp 172.16.133.67:56268 => 172.16.139.250:5440 c4f9446e S/APSF (10/846 ->) e
of
2013-02-26 22:02:35.959 - 22:07:35.943 (299.984 sec) udp 172.16.133.49:58246 => 68.64.21.41:1853 (3001/459078 <-> 3025/459514)
rtt 73 ms eof
2013-02-26 22:02:35.971 - 22:07:35.944 (299.973 sec) udp 68.64.21.45:1853 => 172.16.133.47:51444 (3016/459326 <-> 3001/459080)
rtt 7 ms eof
2013-02-26 22:07:17.956 - 22:07:35.944 (17.988 sec) tcp 172.16.133.114:55366 => 67.217.64.49:443 fb42f9f4:62a1bd0f S/AP:AS/AP (
25/1800 <-> 16/4469) rtt 27 ms eof
2013-02-26 22:07:35.944 tcp 172.16.133.103:63453 => 67.217.65.244:443 fe307540 S/0 (1/52 ->) eof
2013-02-26 22:02:35.953 - 22:07:35.945 (299.992 sec) udp 172.16.133.57:53807 => 68.64.21.62:1853 (25733/16958158 <-> 14980/2234
797) rtt 1 ms eof
2013-02-26 22:03:49.591 - 22:07:35.945 (226.354 sec) tcp 172.16.133.40:50097 => 23.33.85.199:80 8f2ac0d2:90049633 S/AP:AS/AP (4
4/6760 <-> 92/119775) rtt 34 ms eof
2013-02-26 22:07:18.234 - 22:07:35.947 (17.713 sec) tcp 172.16.133.114:55367 => 67.217.66.49:443 8058fb24:495c5440 S/AP:AS/AP (
25/1800 <-> 16/4469) rtt 28 ms eof
2013-02-26 22:07:17.734 - 22:07:35.947 (18.213 sec) tcp 172.16.133.114:55365 => 67.217.65.49:443 5f73790b:1ecdd8a0 S/AP:AS/AP (
25/1800 <-> 16/4469) rtt 32 ms eof
2013-02-26 22:02:35.980 - 22:07:35.948 (299.968 sec) icmp [0:0] 172.16.128.254 => 172.16.133.233 (106/6784 ->) eof
2013-02-26 22:07:35.495 - 22:07:35.948 (0.453 sec) tcp 172.16.133.63:54276 => 208.111.161.254:80 cd5a2215:530abce9 S/AP:AS/AP (
9/4809 <-> 16/8717) rtt 64 ms eof
2013-02-26 22:07:35.496 - 22:07:35.949 (0.453 sec) tcp 172.16.133.63:54278 => 208.111.161.254:80 eee0d56d:ab0ae7ee S/AP:AS/AP (
10/5977 <-> 17/8074) rtt 85 ms eof
2013-02-26 22:02:35.954 - 22:07:35.950 (299.996 sec) udp 172.16.133.56:49514 => 68.64.21.42:1853 (3001/459153 <-> 3026/459566)
rtt 59 ms eof
2013-02-26 22:07:35.497 - 22:07:35.951 (0.454 sec) tcp 172.16.133.63:54280 => 208.111.161.254:80 20a7fe8c:7544b963 S/AP:AS/AP (
11/7130 <-> 17/11976) rtt 84 ms eof
2013-02-26 22:02:55.907 - 22:07:35.952 (280.045 sec) tcp 172.16.133.66:49611 => 96.43.146.22:443 f91f045a:3705e057 A/AP:AP/AP (
23/15030 <-> 17/2529) rtt 48 ms eof
2013-02-26 22:02:35.968 - 22:07:35.952 (299.984 sec) icmp [8:0] 172.16.133.233 => 172.16.128.254 (107/6848 ->) eof
student@student-virtual-machine:~/Downloads$
```

Cat bigflows.yaf.txt gives readable text.

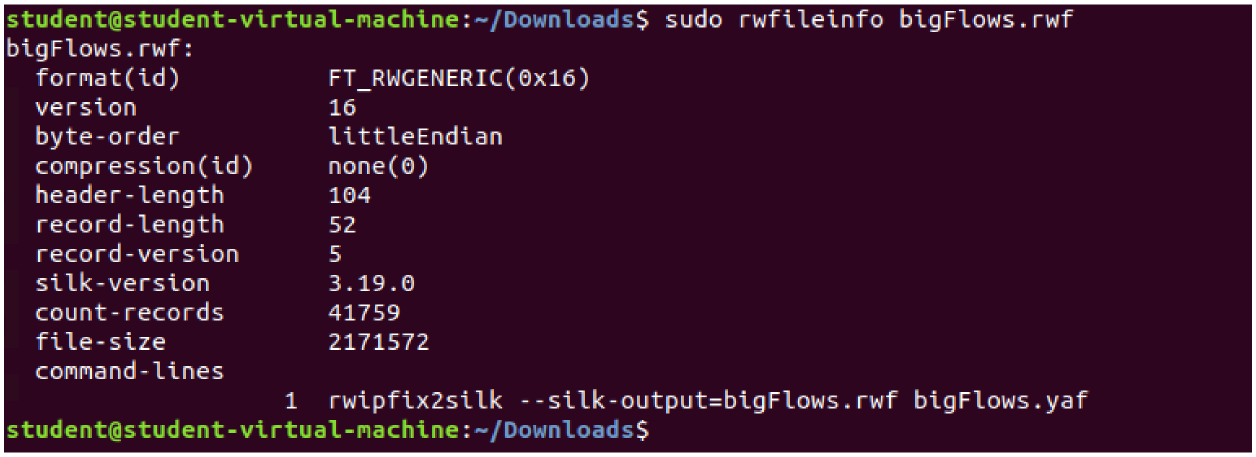
In terms of the directionality of flow, we can see that there are tcp, udp, and icmp and the directionality of where the packets are traveling (source to destination). We can even see some parenthesis with arrows pointing to both sides, possibly indicating bidirectional flow.

Q5



```
student@student-virtual-machine:~/Downloads$ sudo rwipfix2silk bigFlows.yaf  --silk-output=bigFlows.rwf
student@student-virtual-machine:~/Downloads$ ls
bigFlows.pcap  bigFlows.yaf      libfixbuf-2.4.0         silk-3.19.0         yaf-2.11.0
bigFlows.rwf   bigFlows.yaf.txt  libfixbuf-2.4.0.tar.gz  silk-3.19.0.tar.gz  yaf-2.11.0.tar.gz
```

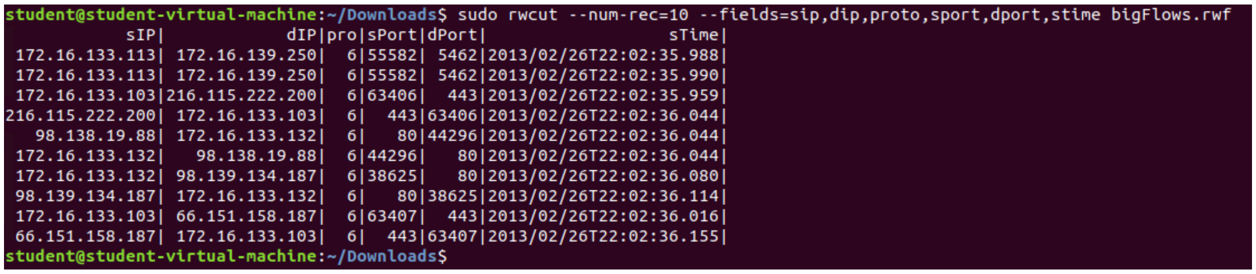
I used the command above to convert the YAF file into an rwf format.



```
student@student-virtual-machine:~/Downloads$ sudo rwfileinfo bigFlows.rwf
bigFlows.rwf:
  format(id)          FT_RWGENERIC(0x16)
  version             16
  byte-order          littleEndian
  compression(id)     none(0)
  header-length       104
  record-length       52
  record-version      5
  silk-version        3.19.0
  count-records       41759
  file-size           2171572
  command-lines
                   1  rwipfix2silk --silk-output=bigFlows.rwf bigFlows.yaf
student@student-virtual-machine:~/Downloads$
```

We can see the metadata in the image above.

Q6

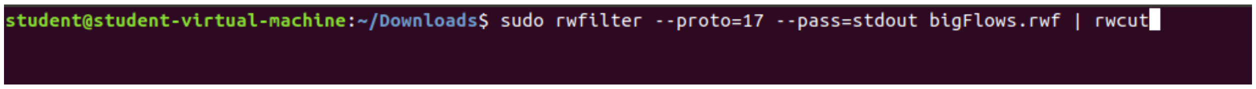


```
student@student-virtual-machine:~/Downloads$ sudo rwcut --num-rec=10 --fields=sip,dip,proto,sport,dport,stime bigFlows.rwf
            sIP|            dIP|pro|sPort|dPort|                  sTime|
 172.16.133.113| 172.16.139.250|  6|55582| 5462|2013/02/26T22:02:35.988|
 172.16.133.113| 172.16.139.250|  6|55582| 5462|2013/02/26T22:02:35.990|
 172.16.133.103|216.115.222.200|  6|63406|  443|2013/02/26T22:02:35.959|
216.115.222.200| 172.16.133.103|  6|  443|63406|2013/02/26T22:02:36.044|
   98.138.19.88| 172.16.133.132|  6|   80|44296|2013/02/26T22:02:36.044|
 172.16.133.132|   98.138.19.88|  6|44296|   80|2013/02/26T22:02:36.044|
 172.16.133.132| 98.139.134.187|  6|38625|   80|2013/02/26T22:02:36.080|
 98.139.134.187| 172.16.133.132|  6|   80|38625|2013/02/26T22:02:36.114|
 172.16.133.103| 66.151.158.187|  6|63407|  443|2013/02/26T22:02:36.016|
 66.151.158.187| 172.16.133.103|  6|  443|63407|2013/02/26T22:02:36.155|
student@student-virtual-machine:~/Downloads$
```

Rwcut is used to extract and display specific fields from a rwf flow file. –num-rec is referring to the number of records to be processed from the bigFlows.rwf file which in our case are the first 10 records. The –fields option specifies the fields to display which in our case are the sip (source ip address), dip (destination ip address), proto (protocol used), sport (source port number used), dport (destination port number used), and stime (starting time of flow).

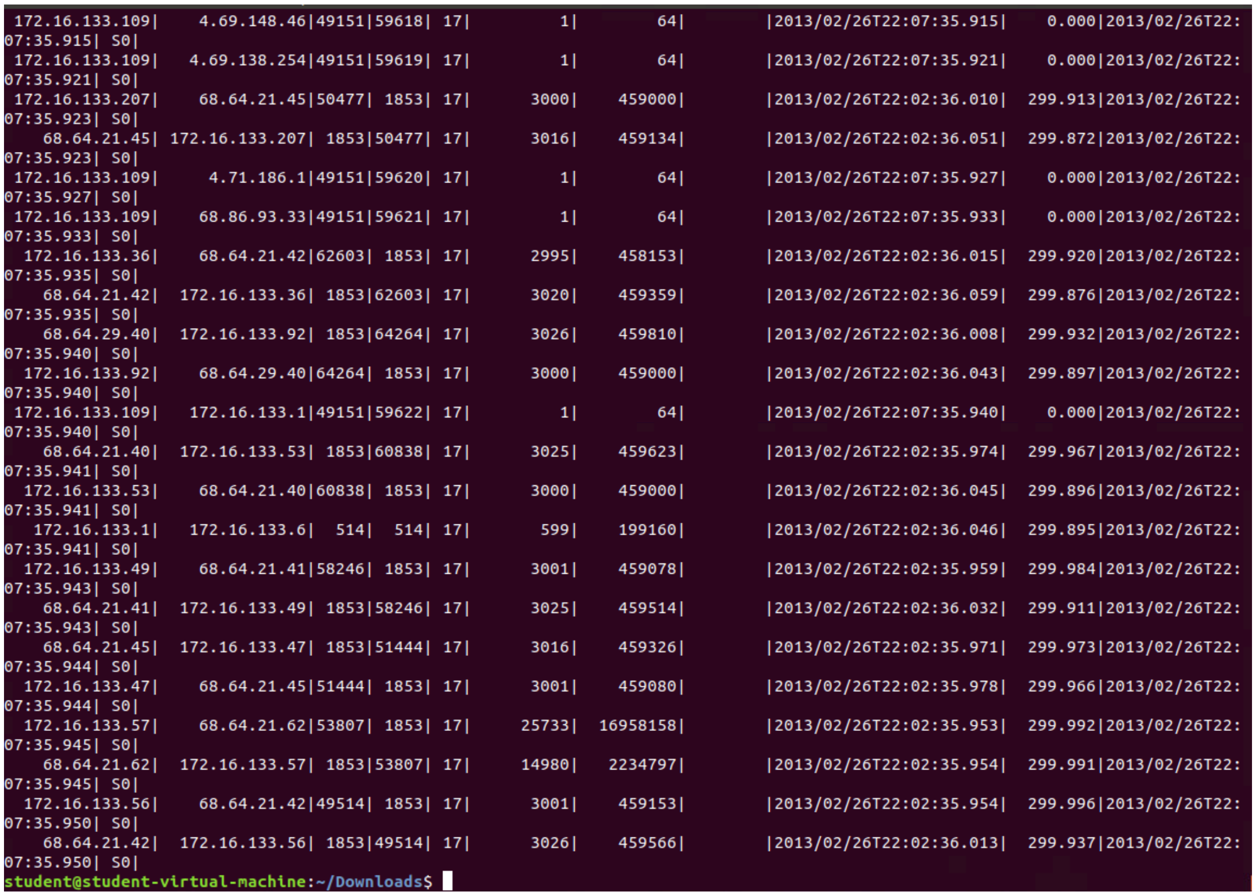
We can see in the given image that this is all displayed in a neat looking table.

Q7



```
student@student-virtual-machine:~/Downloads$ sudo rwfilter --proto=17 --pass=stdout bigFlows.rwf | rwcut
```

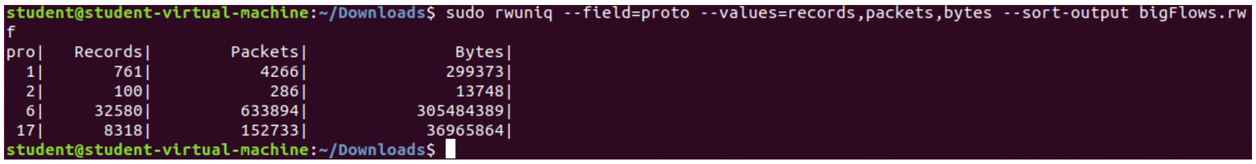
The command is above and it's what we want if we wanted to focus on UDP traffic since the specified protocol is 17 (number corresponding to the UDP protocol).



```
 172.16.133.109|    4.69.148.46|49151|59618| 17|         1|        64|        |2013/02/26T22:07:35.915|    0.000|2013/02/26T22:
07:35.915| S0|
 172.16.133.109|   4.69.138.254|49151|59619| 17|         1|        64|        |2013/02/26T22:07:35.921|    0.000|2013/02/26T22:
07:35.921| S0|
 172.16.133.207|    68.64.21.45|50477| 1853| 17|      3000|    459000|        |2013/02/26T22:02:36.010|  299.913|2013/02/26T22:
07:35.923| S0|
    68.64.21.45| 172.16.133.207| 1853|50477| 17|      3016|    459134|        |2013/02/26T22:02:36.051|  299.872|2013/02/26T22:
07:35.923| S0|
 172.16.133.109|     4.71.186.1|49151|59620| 17|         1|        64|        |2013/02/26T22:07:35.927|    0.000|2013/02/26T22:
07:35.927| S0|
 172.16.133.109|    68.86.93.33|49151|59621| 17|         1|        64|        |2013/02/26T22:07:35.933|    0.000|2013/02/26T22:
07:35.933| S0|
  172.16.133.36|    68.64.21.42|62603| 1853| 17|      2995|    458153|        |2013/02/26T22:02:36.015|  299.920|2013/02/26T22:
07:35.935| S0|
    68.64.21.42|  172.16.133.36| 1853|62603| 17|      3020|    459359|        |2013/02/26T22:02:36.059|  299.876|2013/02/26T22:
07:35.935| S0|
    68.64.29.40|  172.16.133.92| 1853|64264| 17|      3026|    459810|        |2013/02/26T22:02:36.008|  299.932|2013/02/26T22:
07:35.940| S0|
  172.16.133.92|    68.64.29.40|64264| 1853| 17|      3000|    459000|        |2013/02/26T22:02:36.043|  299.897|2013/02/26T22:
07:35.940| S0|
 172.16.133.109|   172.16.133.1|49151|59622| 17|         1|        64|        |2013/02/26T22:07:35.940|    0.000|2013/02/26T22:
07:35.940| S0|
    68.64.21.40|  172.16.133.53| 1853|60838| 17|      3025|    459623|        |2013/02/26T22:02:35.974|  299.967|2013/02/26T22:
07:35.941| S0|
  172.16.133.53|    68.64.21.40|60838| 1853| 17|      3000|    459000|        |2013/02/26T22:02:36.045|  299.896|2013/02/26T22:
07:35.941| S0|
   172.16.133.1|   172.16.133.6|  514|  514| 17|       599|    199160|        |2013/02/26T22:02:36.046|  299.895|2013/02/26T22:
07:35.941| S0|
  172.16.133.49|    68.64.21.41|58246| 1853| 17|      3001|    459078|        |2013/02/26T22:02:35.959|  299.984|2013/02/26T22:
07:35.943| S0|
    68.64.21.41|  172.16.133.49| 1853|58246| 17|      3025|    459514|        |2013/02/26T22:02:36.032|  299.911|2013/02/26T22:
07:35.943| S0|
    68.64.21.45|  172.16.133.47| 1853|51444| 17|      3016|    459326|        |2013/02/26T22:02:35.971|  299.973|2013/02/26T22:
07:35.944| S0|
  172.16.133.47|    68.64.21.45|51444| 1853| 17|      3001|    459080|        |2013/02/26T22:02:35.978|  299.966|2013/02/26T22:
07:35.944| S0|
  172.16.133.57|    68.64.21.62|53807| 1853| 17|     25733|  16958158|        |2013/02/26T22:02:35.953|  299.992|2013/02/26T22:
07:35.945| S0|
    68.64.21.62|  172.16.133.57| 1853|53807| 17|     14980|   2234797|        |2013/02/26T22:02:35.954|  299.991|2013/02/26T22:
07:35.945| S0|
  172.16.133.56|    68.64.21.42|49514| 1853| 17|      3001|    459153|        |2013/02/26T22:02:35.954|  299.996|2013/02/26T22:
07:35.950| S0|
    68.64.21.42|  172.16.133.56| 1853|49514| 17|      3026|    459566|        |2013/02/26T22:02:36.013|  299.937|2013/02/26T22:
07:35.950| S0|
student@student-virtual-machine:~/Downloads$ 
```

A portion of its output is above

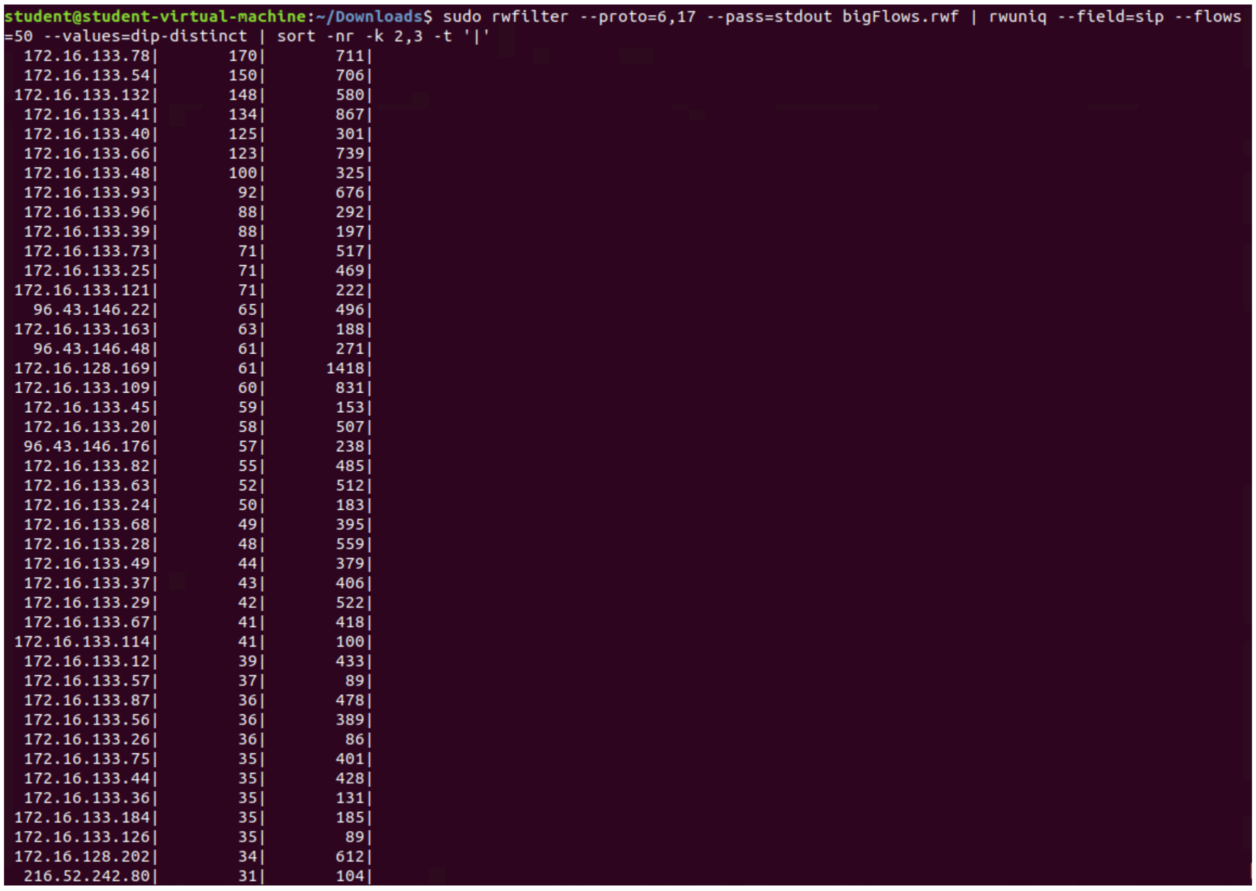
Q8



```
student@student-virtual-machine:~/Downloads$ sudo rwuniq --field=proto --values=records,packets,bytes --sort-output bigFlows.rw
f
pro|   Records|        Packets|              Bytes|
  1|       761|           4266|             299373|
  2|       100|            286|              13748|
  6|     32580|         633894|          305484389|
 17|      8318|         152733|           36965864|
student@student-virtual-machine:~/Downloads$ 
```

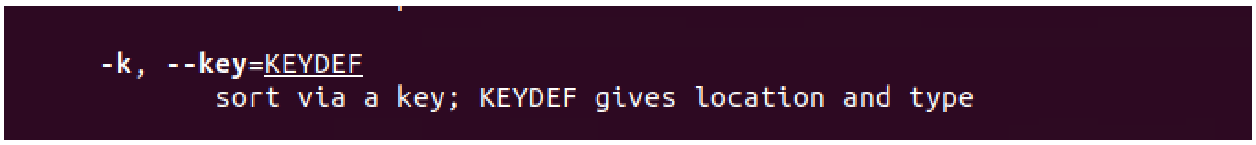
This rwuniq command is used to count and display the unique values of a specified field in the silk file. The options that are used were –field=proto which indicates that the protocol field should be used to count the the unique values (querying by protocol basically), –values=records,packets,bytes (specifies that the number of records, packets, and bytes should be the displayed), –sort-output (sorts output in ascending order based on the field values).

Q9



```
student@student-virtual-machine:~/Downloads$ sudo rwfilter --proto=6,17 --pass=stdout bigFlows.rwf | rwuniq --field=sip --flows
=50 --values=dip-distinct | sort -nr -k 2,3 -t '|'
  172.16.133.78|       170|       711|
  172.16.133.54|       150|       706|
 172.16.133.132|       148|       580|
  172.16.133.41|       134|       867|
  172.16.133.40|       125|       301|
  172.16.133.66|       123|       739|
  172.16.133.48|       100|       325|
  172.16.133.93|        92|       676|
  172.16.133.96|        88|       292|
  172.16.133.39|        88|       197|
  172.16.133.73|        71|       517|
  172.16.133.25|        71|       469|
 172.16.133.121|        71|       222|
   96.43.146.22|        65|       496|
 172.16.133.163|        63|       188|
   96.43.146.48|        61|       271|
 172.16.128.169|        61|      1418|
 172.16.133.109|        60|       831|
  172.16.133.45|        59|       153|
  172.16.133.20|        58|       507|
  96.43.146.176|        57|       238|
  172.16.133.82|        55|       485|
  172.16.133.63|        52|       512|
  172.16.133.24|        50|       183|
  172.16.133.68|        49|       395|
  172.16.133.28|        48|       559|
  172.16.133.49|        44|       379|
  172.16.133.37|        43|       406|
  172.16.133.29|        42|       522|
  172.16.133.67|        41|       418|
 172.16.133.114|        41|       100|
  172.16.133.12|        39|       433|
  172.16.133.57|        37|        89|
  172.16.133.87|        36|       478|
  172.16.133.56|        36|       389|
  172.16.133.26|        36|        86|
  172.16.133.75|        35|       401|
  172.16.133.44|        35|       428|
  172.16.133.36|        35|       131|
 172.16.133.184|        35|       185|
 172.16.133.126|        35|        89|
 172.16.128.202|        34|       612|
  216.52.242.80|        31|       104|
```

The command filters flows and passes the filtered output to standard output. The only selected flows are TCP (6) and UDP (17). The pipe then takes that output and puts into the second command “rwuniq” where with its options, it takes the unique count of source IP addresses and counts the number of distinct destination ip addresses. The output of that command is then passed to the sort command which sorts the output in descending order based on the second and third fields and separates them by the symbol “|”. The second field is the flow count for every source IP and the third field is the count of destination IPs corresponding to its respective source IP.



```
    -k, --key=KEYDEF
           sort via a key; KEYDEF gives location and type
```

The above screenshot from the sort man page supports the sorting by field explanation.

Q10

The rwsetbuild tool is a silk data analysis tool used to create IP sets of IP addresses and subnets. It reads a list if IP addresses and subnets from a file and creates a new set file containing the IP addresses and subnets listed from a selected text.

The rwsettool is used for the manipulation of IP sets. It can be used if one wants to add or remove IPs or subnets. It reads from a set file and a list of commands and and performs the specified action. For example the “add” option takes an IP address as an argument and adds that IP address.

The rwbagbuild tool is used to make IP bags which are like sets, but stores the instance count of IP addresses or subnets from a file. It creates and stores them in a bag file.

The rwbagtool manipulates IP bags by adding or removing IP addresses and subnets. It even has the ability to update their instance count. For example, add [ipaddress] will increment the instance count for that ip address by 1.

Resources

https://tools.netsa.cert.org/yaf/applabel.html#:~:text=yaf%20can%20examine%20packet%20payloads,port%20number%20for%20the%20protocol.

https://tools.netsa.cert.org/p0f/libp0f.html

https://tools.netsa.cert.org/yaf/yafdpi.html https://tools.netsa.cert.org/yaf/applabel.html

https://tools.netsa.cert.org/silk/rwsetbuild.html

https://tools.netsa.cert.org/silk/rwsettool.html

https://tools.netsa.cert.org/silk/rwbagbuild.html

https://tools.netsa.cert.org/silk/rwbagtool.html