

1. Introduction. `nfstat` is a program to process NetFlow v5 data in the format of Mark Fullmer's `flow-tools` and print statistics.

This program is written in **CWEB**—a software system for creating readable programs. **CWEB** is written by Silvio Levy and Donald E. Knuth. It allows one to combine C and \TeX to create programs that can be printed nicely on paper. Further information can be found at <http://www-cs-faculty.stanford.edu/~knuth/cweb.html>

This program is part of an effort to make NetFlow analysis of the kind found in a paper by S. Shalunov and B. Teitelbaum (available at <http://www.internet2.edu/abilene/tcp/>) operational and repeated daily (with weekly report generation). The amount of data involved is fairly large (over 4 GB per day) and growing, so the speed of processing is important. Given that fairly well-optimized Perl programs were taking several hours to process such data sets, we opted for writing the single pass of raw data in C. Subsequent data processing, consolidation of daily runs into weekly reports, pretty-printing, Web publishing, etc., will be handled in Perl. The output of this program is easy to read for machines, not for humans. It consists of two separate files: statistics and bulk TCP data.

Each line of the statistics output file has the form “ $\langle label \rangle: \langle flows \rangle \langle packets \rangle \langle octets \rangle$ ”. The *label* describes the kind of traffic this line summarizes and can be one of four things: (i) an alphanumeric string such as “`http`” or “`nntp`” (summarizes traffic of the type referred to by the *label*); (ii) “`protocols[number]`” (summarizes traffic with IP protocol field of the IPv4 header equal to *number*); (iii) “`toses[number]`” (summarizes traffic with IP TOS value equal to *number*); (iv) “`packets_of_size[number]`” (summarizes traffic that belonged to flows with average packet size rounded to the nearest integer equal to *number*). The remaining three numbers on the line that follow the semicolon after the *label* represent the number of flows, packets, and octets in the given traffic type. The statistics are written into a file named “*arg.stats*” where *arg* is the single argument supplied to `nfstat` (the name of the input file).

In the output file with bulk TCP flows each line represents a single flow and has the following format: “ $\langle packets \rangle \langle octets \rangle \langle duration \rangle \langle source\ ASN \rangle \langle destination\ ASN \rangle \langle source\ port \rangle \langle destination\ port \rangle \langle application\ label \rangle$ ”. Observe that the duration is given in milliseconds. The bulk TCP data are written into a file named “*arg.bulk*” where *arg* is the single command-line argument.

All numbers in the output files of this program are decimal unsigned whole numbers. Note that we are prepared to process sampled NetFlow data (see `SAMPLING_RATE`). The output is already scaled to take the sampling into account and no further adjustments are necessary.

The program may display a fatal error (a line prefixed with “**FATAL**”) or zero or more warnings (lines prefixed with “**WARNING**”). Warnings are generally displayed when significant work has been done and an error condition is encountered while fatal errors occur when something goes wrong in the beginning of the run. These message are printed on *stderr*.

This program is by Anatoly Karp (karp@internet2.edu) and Stanislav Shalunov (shalunov@internet2.edu) (mostly as an exercise in **CWEB**). Distributed **without any warranty**, express or implied.

Copyright 2002, Internet2.

2. Embed RCS id into the object file and the executable.

\langle Global variables [2](#) $\rangle \equiv$

```
static const char rcsid[] = "$Id: _nfstat.w,v_1.55_2004/05/20_23:23:06_shalunov_Exp$";
```

See also sections [5](#) and [10](#).

This code is used in section [3](#).

3. Program code. We use types from `sys/types.h`; they should be formatted as normal reserved words. The program has a typical C structure.

```
format u_int64_t int
format u_int32_t int
format u_int16_t int
format u_int8_t int
format ssize_t int
⟨Header files to include 4⟩
⟨Global variables 2⟩
⟨Functions 6⟩
⟨The main program 7⟩
```

4. ⟨Header files to include 4⟩ ≡

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>
#include <sys/stat.h>
```

See also section 9.

This code is used in section 3.

5. Abilene routers are normally configured to only process certain percentage of packets for the purposes of NetFlow accounting. `SAMPLING_RATE` is the reciprocal of the sampling rate of Abilene GSR external interfaces (“100” corresponds to 1% sampling, “1” means full feed).

`NUM_PACKET_SIZE` should be greater than the largest possible value of the average packet size of any flow; maximum POS (packet-over-SONET) packet size for Cisco and Juniper is 9182. `NUM_PROTOCOL` is the number of IP protocol numbers we can encounter (it’s an 8-bit value in IPv4 header); similarly for `NUM_TOS`.

`FLOW_ATTRIBUTES` is the number of attributes we are tracking for each flow. Do not change this value: treat arrays of this size as a **struct** with a slight efficiency advantage. `FLows`, `PACKETS`, and `OCTETS` are offsets into these arrays.

The array `apps` contains counters for various types of traffic. For example, `apps[NTP_APP][PACKETS]` always contains the number of NTP packets observed so far.

Similarly, for each possible average packet size, we count flows, packets, and octets and keep track of these values in `packets_of_size`; e.g., `packets_of_size[1500][OCTETS]` contains the number of octets contained in packets that belonged to flows with average packet size of 1500 (when rounded to nearest integer) seen so far. In addition, for each possible IP protocol and TOS value, we count flows, packets, and octets and keep track of these values in `protocols` and `toses`. The array `ports` contains port-based accounting of applications that are not otherwise identified (to help in identification of emerging popular applications).

```
#define SAMPLING_RATE 100
#define NUM_PACKET_SIZE 9216
#define NUM_PROTOCOL 256
#define NUM_TOS 256
#define NUM_PORT 65536
#define FLOW_ATTRIBUTES 3
#define FLOWS 0
#define PACKETS 1
#define OCTETS 2
#define TOTAL_APP 0
#define FTP_APP 1
#define NNTP_APP 2
#define HTTP_APP 3
#define HTTPS_APP 4
#define NFS_APP 5
#define MAIL_APP 6
#define SSH_APP 7
#define TELNET_APP 8
#define DNS_APP 9
#define NTP_APP 10
#define EDONKEY_APP 11
#define GNUTELLA_APP 12
#define FREENET_APP 13
#define IPSEC_IKE_APP 14
#define IPSEC_ESP_APP 15
#define IPSEC_AH_APP 16
#define MULTICAST_APP 17
#define GSIFTP_APP 18
#define BBFTP_APP 19
#define IPERF_APP 20
#define IRC_APP 21
#define MSWINDOWS_APP 22
#define REALPLAYER_APP 23
#define AFS_APP 24
#define X11_APP 25
```

```

#define WINMEDIA_APP 26
#define H323_APP 27
#define STREAMWORKS_APP 28
#define AUDIOGALAXY_APP 29
#define FASTTRACK_APP 30
#define BLUBSTER_APP 31
#define WINMX_APP 32
#define SHOUTCAST_APP 33
#define HOTLINE_APP 34
#define SNMP_APP 35
#define IDENT_APP 36
#define SQUID_APP 37
#define SOCKS_APP 38
#define BATTLENET_APP 39
#define QUAKE_APP 40
#define STARSIEGE_APP 41
#define CARRACHO_APP 42
#define PORTZERO_APP 43
#define HALFLIFE_APP 44
#define AIM_APP 45
#define RSYNC_APP 46
#define UNIDATA_LDM_APP 47
#define NEOMODUS_APP 48
#define BACKBONE_RADIO_APP 49
#define CAMARADES_APP 50
#define GAMESPYARCADE_APP 51
#define RTIP_APP 52
#define PORTMAPPER_APP 53
#define VOIP_APP 54
#define IBP_APP 55
#define MCIDAS_APP 56
#define ASHERON_APP 57
#define DIRECTX_APP 58
#define BITTORRENT_APP 59
#define BBCP_APP 60
#define SSM_APP 61
#define NUM_APP 62

< Global variables 2 > +=
u_int64_t packets_of_size[NUM_PACKET_SIZE][FLOW_ATTRIBUTES];
u_int64_t protocols[NUM_PROTOCOL][FLOW_ATTRIBUTES];
u_int64_t toses[NUM_TOS][FLOW_ATTRIBUTES];
u_int64_t ports[NUM_PORT][FLOW_ATTRIBUTES];
u_int64_t apps[NUM_APP][FLOW_ATTRIBUTES];
const char *app_label;
const char *app_names[] = {"total", "ftp", "nntp", "http", "https", "nfs", "mail", "ssh",
    "telnet", "dns", "ntp", "edonkey", "gnutella", "freenet", "ipsec_ike", "ipsec_esp",
    "ipsec_ah", "multicast", "gsiftp", "bbftp", "iperf", "irc", "mswindows", "realplayer",
    "afs", "x11", "winmedia", "h323", "streamworks", "audiogalaxy", "fasttrack", "blubster",
    "winmx", "shoutcast", "hotline", "snmp", "ident", "squid", "socks", "battlenet", "quake",
    "starsiege", "carracho", "portzero", "halflife", "aim", "rsync", "unidata_ldm", "neomodus",
    "backbone_radio", "camarades", "gamespyarcade", "rtip", "portmapper", "voip", "ibp",
    "mcidas", "asheron", "directx", "bittorrent", "bbcp", "ssm", "MISCONFIGURATION_FIX_SOURCE"};

```

6. `<Functions 6>` \equiv
`<Function persistent_read 33>`

This code is used in section 3.

7. The main function consists of necessary initialization, a single pass over available data while updating relevant counters and writing out bulk TCP records individually, and printing of the counters.

`<The main program 7>` \equiv

```

int main(int argc, char *argv[])
{
    <Initialize global variables and data structures 8>
    <Read NetFlow v5 header 16>
    <Main loop: read and process NetFlow records individually 15>
    <Print the results 25>
    <Clean up before exit 32>
    exit(0);
}

```

This code is used in section 3.

8. Process command line and zero out all the counters.

```

#define INIT_FPO(type) type[FLOWS] = type[PACKETS] = type[OCTETS] = 0

```

`<Initialize global variables and data structures 8>` \equiv

```

{ int i; <Process command line 11>
  for (i = 0; i < NUM_APP; i++) INIT_FPO(apps[i]);
  for (i = 0; i < NUM_PACKET_SIZE; i++) INIT_FPO(packets_of_size[i]);
  for (i = 0; i < NUM_PROTOCOL; i++) INIT_FPO(protocols[i]);
  for (i = 0; i < NUM_TOS; i++) INIT_FPO(toses[i]);
}

```

This code is used in section 7.

9. Extra header files.

`<Header files to include 4>` \equiv

```

#include <sys/param.h> /* For MAXPATHLEN. */
#include <fcntl.h> /* For open(). */

```

10. Input file descriptor (*input_fd*) is where we read all the NetFlow v5 data (it is a file descriptor and not an *stdio* file stream because we chose to implement our own buffering in `<Main loop: read and process NetFlow records individually 15>`). Bulk TCP file stream (*bulk_tcp_fs*) is where we write bulk TCP flows. Statistics (normal output) goes to *stdout*, which is redirected to the right file using *freopen*().

`<Global variables 2>` \equiv

```

int input_fd; /* Input file descriptor. */
FILE *bulk_tcp_fs; /* The associated stream. */

```

11. Set up input and output files based on a single command line argument.

```

⟨Process command line 11⟩ ≡
{
    char *input_path; /* Input file name. */
    char output_path[MAXPATHLEN]; /* Output file name. */
    char bulk_tcp_path[MAXPATHLEN]; /* File to put all the bulk TCP flows. */
    if (argc ≠ 2) {
        fprintf(stderr, "Usage: _nfstat_ <filename>\n"); exit(1);
    }
    ⟨Set up input file 12⟩
    ⟨Set up statistics output 13⟩
    ⟨Set up bulk TCP flows output 14⟩
}

```

This code is used in section 8.

12. Input file name is passed to the program as *argv*[1].

```

⟨Set up input file 12⟩ ≡
    input_path = argv[1]; input_fd = open(input_path, O_RDONLY);
    if (input_fd ≡ -1) {
        perror(input_path); exit(1);
    }
}

```

This code is used in section 11.

```

13. ⟨Set up statistics output 13⟩ ≡
{
    FILE *f;
    snprintf(output_path, sizeof output_path, "%s.stats", input_path);
    f = freopen(output_path, "w", stdout);
    if (!f) {
        perror(output_path); exit(1);
    }
}
}

```

This code is used in section 11.

```

14. ⟨Set up bulk TCP flows output 14⟩ ≡
    snprintf(bulk_tcp_path, sizeof bulk_tcp_path, "%s.bulk", input_path);
    bulk_tcp_fs = fopen(bulk_tcp_path, "w");
    if (!bulk_tcp_fs) {
        perror(bulk_tcp_path); exit(1);
    }
}

```

This code is used in section 11.

15. We could have used `fread(..., RECORD_SIZE)` in the main loop, but it would introduce one extra memory-to-memory copy even if avoiding inefficient `read()` block sizes. We opt to implement buffering manually.

`RECORD_SIZE` is the NetFlow v5 record length in `flow-tools` format. `BUFFER_SIZE` must be an even multiple of `RECORD_SIZE`. It is the `read()` block size, so 8 KB or more is appropriate.

```
#define RECORD_SIZE 64
#define BUFFER_SIZE 128 * RECORD_SIZE
⟨Main loop: read and process NetFlow records individually 15⟩ ≡
{
    char buffer[BUFFER_SIZE];
    char *current_record;
    ssize_t bytes_read, bytes_left;
    while ((bytes_read = persistent_read(input_fd, buffer, BUFFER_SIZE)) > 0) {
        current_record = buffer; bytes_left = bytes_read;
        while (bytes_left ≥ RECORD_SIZE) {
            ⟨Process NetFlow v5 record starting from current_record 20⟩
            current_record += RECORD_SIZE; bytes_left -= RECORD_SIZE;
        }
        if (bytes_left > 0) fprintf(stderr,
            "WARNING: record of length %d discarded; truncated file?\n", bytes_left);
    }
    if (bytes_read < 0) perror("WARNING: reading data file");
}
```

This code is cited in section 10.

This code is used in section 7.

16. XXX: We should really take endianness into account.

```
⟨Read NetFlow v5 header 16⟩ ≡
{
    unsigned char header[8]; /* The first 8 bytes of the header. */
    u_int8_t magic1, magic2, byte_order, s_version;
    u_int32_t bytes_left;
    ⟨Read first 8 bytes of NetFlow v5 header 17⟩
    ⟨Header sanity check 18⟩
    ⟨Read and discard any remaining portion of the header 19⟩
}
```

This code is used in section 7.

17. The header consists of a 4-byte microheader (magic values, byte order indicator, and file format version), followed by 4-byte unsigned value representing the total header length (everything up to the first data record), followed by uninteresting TLVs (which we ignore). We subtract 8 from `bytes_left` because we read 8 bytes here.

```
⟨Read first 8 bytes of NetFlow v5 header 17⟩ ≡
if (persistent_read(input_fd, header, 8) ≠ 8) {
    perror("FATAL: reading header"); exit(1);
}
magic1 = header[0]; magic2 = header[1]; byte_order = header[2]; s_version = header[3];
bytes_left = ((u_int32_t) header[4]) - 8;
```

This code is used in section 16.

```

18. <Header sanity check 18> ≡
    if (magic1 ≠ #CF) {
        fprintf(stderr, "FATAL: incorrect magic number (0x%x, should be 0xCF)\n", magic1); exit(1);
    }
    if (s_version ≠ 3) {
        fprintf(stderr, "FATAL: incorrect file format version (%i, should be 3)\n", s_version);
        exit(1);
    }
    if (byte_order ≠ 1) {
        fprintf(stderr, "FATAL: I can't handle big-endian files. Sorry.\n"); exit(1);
    }

```

This code is used in section 16.

```

19. #define min(x,y) ((x) < (y) ? (x) : (y))
<Read and discard any remaining portion of the header 19> ≡
{
    char garbage[64];
    ssize_t rc = 0;
    while (bytes_left > 0 ∧ (rc = persistent_read(input_fd, garbage, min(bytes_left, sizeof garbage))) > 0)
        bytes_left -= rc;
    if (rc < 0) {
        perror("FATAL: reading header continuation"); exit(1);
    }
}

```

This code is used in section 16.

```

20. #define COUNT_FPO(type) do
    {
        type[FLAWS]++; type[PACKETS] += packets; type[OCTETS] += octets;
    }
    while (0)
<Process NetFlow v5 record starting from current_record 20> ≡
{
    u_int32_t src_ip, dst_ip;
    u_int32_t packets, octets;
    u_int16_t src_port, dst_port, src_as, dst_as;
    u_int8_t prot, tos;
    u_int32_t start_time, end_time; /* In milliseconds since router boot. */
    <Parse NetFlow v5 record using constant offsets 23>
    <Update non-application-specific counters 21>
    <Recognize individual protocols and update corresponding counters 24>
    <Maybe output this record as a bulk TCP flow 22>
}

```

This code is used in section 15.

21. Update counters other than those associated with protocols peculiar to specific applications. Note that *average_packet_size* is calculated using integer arithmetic in such a fashion that it is rounded to the nearest integer.

```

⟨Update non-application-specific counters 21⟩ ≡
{
    u_int32_t average_packet_size;
    if (packets ≡ 0) {
        fprintf(stderr, "WARNING: packets is 0, replacing with 1\n"); packets = 1;
    }
    average_packet_size = (octets + packets/2)/packets;
    if (average_packet_size < NUM_PACKET_SIZE) COUNT_FPO(packets_of_size[average_packet_size]);
    else fprintf(stderr, "WARNING: average_packet_size = %u >= %u "NUM_PACKET_SIZE=" %u\n",
        average_packet_size, NUM_PACKET_SIZE);
    COUNT_FPO(apps[TOTAL_APP]); COUNT_FPO(protocols[prot]); COUNT_FPO(toses[tos]);
}

```

This code is used in section 20.

22. We call TCP connections that transferred ACTUAL_BULKNESS_THRESHOLD or more octets “bulk TCP flows.” BULKNESS_THRESHOLD is adjusted by SAMPLING_RATE so that the recorded number of octets may be compared against it.

```

#define ACTUAL_BULKNESS_THRESHOLD 10000000
#define BULKNESS_THRESHOLD ACTUAL_BULKNESS_THRESHOLD/SAMPLING_RATE
⟨Maybe output this record as a bulk TCP flow 22⟩ ≡
if (octets > BULKNESS_THRESHOLD ^ prot ≡ 6) fprintf(bulk_tcp_fs,
    "%11u %11u %u %u %u %u %u %u %s\n", ((u_int64_t) packets) * SAMPLING_RATE, ((u_int64_t)
    octets) * SAMPLING_RATE, end_time - start_time, src_as, dst_as, src_port, dst_port, app_label);

```

This code is used in section 20.

23. Rip out the values we are interested in from the binary format. Offsets are fixed.

```

#define FIELD_AT_OFFSET(offset, type) * (type *) (current_record + offset)
⟨Parse NetFlow v5 record using constant offsets 23⟩ ≡
/* IP numbers are anonymized, but useful for things like multicast detection. */
src_ip = FIELD_AT_OFFSET(16, u_int32_t); dst_ip = FIELD_AT_OFFSET(20, u_int32_t);
packets = FIELD_AT_OFFSET(32, u_int32_t); octets = FIELD_AT_OFFSET(36, u_int32_t);
start_time = FIELD_AT_OFFSET(40, u_int32_t); end_time = FIELD_AT_OFFSET(44, u_int32_t);
src_port = FIELD_AT_OFFSET(48, u_int16_t); dst_port = FIELD_AT_OFFSET(50, u_int16_t);
prot = FIELD_AT_OFFSET(52, u_int8_t); tos = FIELD_AT_OFFSET(53, u_int8_t);
src_as = FIELD_AT_OFFSET(60, u_int16_t); dst_as = FIELD_AT_OFFSET(62, u_int16_t);

```

This code is used in section 20.

24. Port number snooping. Caveat emptor: all we know are the port and protocol numbers. Identification only works if applications don't masquerade.

```
#define PROTO_TCP 6
#define PROTO_UDP 17
#define PORT_IS(x) (src_port == x ∨ dst_port == x)
#define PORT_IN_RANGE(x, y) ((src_port ≥ x ∧ src_port ≤ y) ∨ (dst_port ≥ x ∧ dst_port ≤ y))
#define IS_TCP (prot == PROTO_TCP)
#define IS_UDP (prot == PROTO_UDP)
#define APP_IS(x) do
{
    COUNT_FPO(apps[x]); app_label = app_names[x];
}
while (0)
```

(Recognize individual protocols and update corresponding counters 24) ≡

```
if (PORT_IS(1214) ∧ IS_TCP) APP_IS(FASTTRACK_APP);
else if (PORT_IN_RANGE(6881, 6889) ∧ IS_TCP) APP_IS(BITTORRENT_APP);
else if (PORT_IN_RANGE(6346, 6350) ∧ IS_TCP) APP_IS(GNUTELLA_APP);
else if (PORT_IN_RANGE(20, 21) ∧ IS_TCP) APP_IS(FTP_APP);
else if ((PORT_IN_RANGE(80, 81) ∨ PORT_IS(8080)) ∧ IS_TCP) APP_IS(HTTP_APP);
else if ((PORT_IS(119) ∨ PORT_IS(563)) ∧ IS_TCP) APP_IS(NNTP_APP);
else if (PORT_IS(443) ∧ IS_TCP) APP_IS(HTTPS_APP);
else if (PORT_IS(2049) ∨ PORT_IS(1110)) APP_IS(NFS_APP);
else if ((PORT_IS(25) ∨ PORT_IN_RANGE(109, 110) ∨ PORT_IS(143) ∨ PORT_IS(220) ∨ PORT_IS(465) ∨
    PORT_IS(585) ∨ PORT_IS(587) ∨ PORT_IS(993)) ∧ IS_TCP) APP_IS(MAIL_APP);
else if (PORT_IS(22) ∧ IS_TCP) APP_IS(SSH_APP);
else if (PORT_IS(23) ∧ IS_TCP) APP_IS(TELNET_APP);
else if (PORT_IS(123) ∧ IS_UDP) APP_IS(NTP_APP);
else if (PORT_IS(53)) APP_IS(DNS_APP);
else if (PORT_IN_RANGE(4661, 4665)) APP_IS(EDONKEY_APP);
else if ((PORT_IS(6690) ∨ PORT_IS(19114)) ∧ IS_TCP) APP_IS(FREENET_APP);
else if (PORT_IS(500) ∧ IS_UDP) APP_IS(IPSEC_IKE_APP);
else if (prot == 50) APP_IS(IPSEC_ESP_APP);
else if (prot == 51) APP_IS(IPSEC_AH_APP);
else if (dst_ip >> 24 == 232) /* XXX: Endianness. */
    APP_IS(SSM_APP);
else if (dst_ip >> 28 == 14)
    /* XXX: Add left shift by 24 bits (“(dst_ip << 24) >> 28”) for different endianness. */
    APP_IS(MULTICAST_APP);
else if (PORT_IS(2811) ∧ IS_TCP) APP_IS(GSIFTP_APP);
else if (PORT_IN_RANGE(5020, 5022) ∧ IS_TCP) APP_IS(BBFTP_APP);
else if (PORT_IN_RANGE(5031, 5033) ∧ IS_TCP) APP_IS(BBCP_APP);
else if (PORT_IN_RANGE(5000, 5009)) APP_IS(IPERF_APP);
else if ((PORT_IS(194) ∨ PORT_IN_RANGE(6666, 6670)) ∧ IS_TCP) APP_IS(IRC_APP);
else if (PORT_IS(135) ∨ PORT_IN_RANGE(137, 139) ∨ PORT_IS(445) ∨ PORT_IN_RANGE(568,
    569) ∨ PORT_IS(1512)) APP_IS(MSWINDOWS_APP);
else if (PORT_IN_RANGE(6970, 6973) ∨ PORT_IN_RANGE(7070, 7071) ∨ PORT_IS(554))
    APP_IS(REALPLAYER_APP);
else if (PORT_IN_RANGE(7000, 7006)) APP_IS(AFS_APP);
else if ((PORT_IN_RANGE(6000, 6005) ∨ PORT_IS(7100) ∨ PORT_IS(1024) ∨ PORT_IS(6016)) ∧ IS_TCP)
    APP_IS(X11_APP);
else if (PORT_IS(1755) ∨ PORT_IS(7007) ∨ PORT_IS(135)) APP_IS(WINMEDIA_APP);
else if ((PORT_IS(1720) ∨ PORT_IS(1503)) ∧ IS_TCP) APP_IS(H323_APP);
```

```

else if (PORT_IS(1558)) APP_IS(STREAMWORKS_APP);
else if (PORT_IS(41170) ^ IS_UDP) APP_IS(BLUBSTER_APP);
else if (PORT_IS(6699) ^ PORT_IS(6257)) APP_IS(WINMX_APP);
else if (PORT_IN_RANGE(8000, 8005)) APP_IS(SHOUTCAST_APP);
else if (PORT_IN_RANGE(5500, 5503)) APP_IS(HOTLINE_APP);
else if (PORT_IS(161)) APP_IS(SNMP_APP);
else if (PORT_IS(113) ^ IS_TCP) APP_IS(IDENT_APP);
else if (PORT_IS(3128)) APP_IS(SQUID_APP);
else if (PORT_IS(1080) ^ IS_TCP) APP_IS(SOCKS_APP);
else if (PORT_IS(4000) ^ PORT_IN_RANGE(6112, 6119)) APP_IS(BATTLENET_APP);
else if (PORT_IS(26000) ^ PORT_IN_RANGE(27910, 27961)) APP_IS(QUAKE_APP);
else if (PORT_IN_RANGE(28000, 28008)) APP_IS(STARSIEGE_APP);
else if (PORT_IN_RANGE(6700, 6702)) APP_IS(CARRACHO_APP);
else if (PORT_IS(0) ^ (IS_TCP ^ IS_UDP)) APP_IS(PORTZERO_APP);
else if (PORT_IS(27005) ^ PORT_IS(27015)) APP_IS(HALFLIFE_APP);
else if (PORT_IS(5190)) APP_IS(AIM_APP);
else if (PORT_IS(873) ^ IS_TCP) APP_IS(RSYNC_APP);
else if (PORT_IS(388)) APP_IS(UNIDATA_LDM_APP);
else if (PORT_IN_RANGE(412, 413)) APP_IS(NEOMODUS_APP);
else if ((PORT_IS(2048) ^ IS_TCP) ^ (PORT_IS(2050) ^ IS_UDP)) APP_IS(BACKBONE_RADIO_APP);
else if ((PORT_IN_RANGE(2047, 2048) ^ PORT_IS(1972)) ^ IS_UDP) APP_IS(CAMARADES_APP);
else if (PORT_IS(27900) ^ PORT_IS(28900) ^ PORT_IN_RANGE(29900,
    29901) ^ ((PORT_IS(13193) ^ PORT_IS(6515)) ^ IS_UDP)) APP_IS(GAMESPYARCADE_APP);
else if (PORT_IS(771)) APP_IS(RTIP_APP);
else if (PORT_IS(111)) APP_IS(PORTMAPPER_APP);
else if (PORT_IN_RANGE(49606, 49609) ^ IS_UDP) APP_IS(VOIP_APP);
else if (PORT_IS(6714) ^ IS_TCP) APP_IS(IBP_APP);
else if (PORT_IS(112)) APP_IS(MCIDAS_APP);
else if (PORT_IS(9000) ^ IS_UDP) APP_IS(ASHERON_APP);
else if ((PORT_IS(47624) ^ IS_TCP) ^ (PORT_IS(6073) ^ IS_UDP) ^ PORT_IN_RANGE(2300, 2400))
    APP_IS(DIRECTX_APP);
else if (PORT_IN_RANGE(41000, 42000) ^ IS_TCP) APP_IS(AUDIOGALAXY_APP);
else {
    COUNT_FPO(ports[src_port]); COUNT_FPO(ports[dst_port]); app_label = "unidentified";
}
}

```

This code is used in section 20.

```

25. #define PRINT_FPO(type, label) printf("%s: %llu %llu %llu\n", label, type[FLOWS],
    type[PACKETS] * SAMPLING_RATE, type[OCTETS] * SAMPLING_RATE)

```

⟨Print the results 25⟩ ≡

```

{
    int i;
    char label[64];
    ⟨Print application-specific counters 26⟩
    ⟨Print IP protocol counters 27⟩
    ⟨Print TOS counters 28⟩
    ⟨Print average packet size counters 29⟩
    ⟨Print data from unidentified ports 30⟩
}

```

This code is used in section 7.

26. `<Print application-specific counters 26> ≡`
`for (i = 0; i < NUM_APP; i++) PRINT_FPO(apps[i], app_names[i]);`

This code is used in section 25.

27. The `PRINT_FPO` needs a string label. We produce it using `snprintf()`.

`<Print IP protocol counters 27> ≡`
`for (i = 0; i < NUM_PROTOCOL; i++)`
`if (protocols[i][FLOWS]) {`
`snprintf(label, sizeof label, "protocols[%i]", i); PRINT_FPO(protocols[i], label);`
`}`

This code is used in section 25.

28. `<Print TOS counters 28> ≡`
`for (i = 0; i < NUM_TOS; i++)`
`if (toses[i][FLOWS]) {`
`snprintf(label, sizeof label, "toses[%i]", i); PRINT_FPO(toses[i], label);`
`}`

This code is used in section 25.

29. `<Print average packet size counters 29> ≡`
`for (i = 0; i < NUM_PACKET_SIZE; i++)`
`if (packets_of_size[i][FLOWS]) {`
`snprintf(label, sizeof label, "packets_of_size[%i]", i); PRINT_FPO(packets_of_size[i], label);`
`}`

This code is used in section 25.

30. `#define NUMWIN 10`

`<Print data from unidentified ports 30> ≡`
`{`
`u_int64_t winners[NUMWIN][FLOW_ATTRIBUTES];`
`u_int16_t winports[NUMWIN];`
`u_int64_t minwin = 0;`
`for (i = 0; i < NUMWIN; i++) INIT_FPO(winners[i]);`
`for (i = 1; i < NUM_PORT; i++) {`
`if (ports[i][OCTETS] > minwin) <Insert new winner port into the appropriate place 31>`
`}`
`for (i = 0; i < NUMWIN; i++)`
`if (winners[i][FLOWS]) {`
`snprintf(label, sizeof label, "ports[%i]", winports[i]); PRINT_FPO(winners[i], label);`
`}`
`}`

This code is used in section 25.

```

31.  ⟨Insert new winner port into the appropriate place 31⟩ ≡
    {
      int newpos, j, k;
      for (newpos = NUMWIN - 1; newpos ≥ 0; newpos--)
        if (ports[i][OCTETS] > winners[newpos][OCTETS]) break;
      for (j = 1; j ≤ newpos; j++) {
        for (k = 0; k < FLOW_ATTRIBUTES; k++) winners[j - 1][k] = winners[j][k];
        winports[j - 1] = winports[j];
      }
      for (k = 0; k < FLOW_ATTRIBUTES; k++) winners[newpos][k] = ports[i][k];
      winports[newpos] = i; minwin = winners[0][OCTETS];
    }

```

This code is used in section 30.

```

32.  ⟨Clean up before exit 32⟩ ≡
      fclose(bulk_tcp_fs); fclose(stdout);

```

This code is used in section 7.

33. A wrapper for `read()` system call that makes short reads impossible. Short reads generally do not happen with regular files, but could be an issue if input comes from a (named) pipe or a socket. Behavior and return values are generally similar to `read()` otherwise. If `read()` ever returns an error in this function, an error is passed up even if previous reads were successful (this can only happen with short reads).

```

⟨Function persistent_read 33⟩ ≡
ssize_t persistent_read(int d, void *buf, size_t nbytes)
{
  ssize_t rc = 0;
  size_t bytes_read = 0;
  while (bytes_read < nbytes ^ (rc = read(d, (char *) buf + bytes_read, nbytes - bytes_read)) > 0) {
    bytes_read += rc;
  }
  return rc ≡ 0 ? bytes_read : rc;
}

```

This code is used in section 6.

ACTUAL_BULKNESS_THRESHOLD: 22 .	BITTORRENT_APP: 5 , 24 .
AFS_APP: 5 , 24 .	BLUBSTER_APP: 5 , 24 .
AIM_APP: 5 , 24 .	buf: 33 .
APP_IS: 24 .	buffer: 15 .
app_label: 5 , 22 , 24 .	BUFFER_SIZE: 15 .
app_names: 5 , 24 , 26 .	bulk TCP: 1 , 22 .
apps: 5 , 8 , 21 , 24 , 26 .	bulk_tcp_fs: 10 , 14 , 22 , 32 .
argc: 7 , 11 .	bulk_tcp_path: 11 , 14 .
argv: 7 , 12 .	BULKNESS_THRESHOLD: 22 .
ASHERON_APP: 5 , 24 .	byte_order: 16 , 17 , 18 .
atexit: 32 .	bytes_left: 15 , 16 , 17 , 19 .
AUDIOGALAXY_APP: 5 , 24 .	bytes_read: 15 , 33 .
average_packet_size: 21 .	CAMARADES_APP: 5 , 24 .
BACKBONE_RADIO_APP: 5 , 24 .	CARRACHO_APP: 5 , 24 .
BATTLETNET_APP: 5 , 24 .	copyright: 1 .
BBCP_APP: 5 , 24 .	COUNT_FPO: 20 , 21 , 24 .
BBFTP_APP: 5 , 24 .	current_record: 15 , 23 .
big-endian: 16 .	d: 33 .

DIRECTX_APP: [5](#), [24](#).
 DNS_APP: [5](#), [24](#).
dst_as: [20](#), [22](#), [23](#).
dst_ip: [20](#), [23](#), [24](#).
dst_port: [20](#), [22](#), [23](#), [24](#).
 EDONKEY_APP: [5](#), [24](#).
end_time: [20](#), [22](#), [23](#).
exit: [7](#), [11](#), [12](#), [13](#), [14](#), [17](#), [18](#), [19](#).
f: [13](#).
 FASTTRACK_APP: [5](#), [24](#).
 FATAL: [1](#).
 FATAL: I can't handle big-endian...: [18](#).
 FATAL: incorrect file format...: [18](#).
 FATAL: incorrect magic number...: [18](#).
 FATAL: reading header: [17](#).
 FATAL: reading header continuation: [19](#).
fclose: [32](#).
 FIELD_AT_OFFSET: [23](#).
 FLOW_ATTRIBUTES: [5](#), [30](#), [31](#).
 FLOWS: [5](#), [8](#), [20](#), [25](#), [27](#), [28](#), [29](#), [30](#).
fopen: [14](#).
fprintf: [11](#), [15](#), [18](#), [21](#), [22](#).
fread: [15](#).
 FREENET_APP: [5](#), [24](#).
freopen: [10](#), [13](#).
 FTP_APP: [5](#), [24](#).
 GAMESPYARCADE_APP: [5](#), [24](#).
garbage: [19](#).
 GNUTELLA_APP: [5](#), [24](#).
 GSIFTP_APP: [5](#), [24](#).
 HALFLIFE_APP: [5](#), [24](#).
header: [16](#), [17](#).
 HOTLINE_APP: [5](#), [24](#).
 HTTP_APP: [5](#), [24](#).
 HTTPS_APP: [5](#), [24](#).
 H323_APP: [5](#), [24](#).
i: [8](#), [25](#).
 IBP_APP: [5](#), [24](#).
 IDENT_APP: [5](#), [24](#).
 INIT_FPO: [8](#), [30](#).
input_fd: [10](#), [12](#), [15](#), [17](#), [19](#).
input_path: [11](#), [12](#), [13](#), [14](#).
 IPERF_APP: [5](#), [24](#).
 IPSEC_AH_APP: [5](#), [24](#).
 IPSEC_ESP_APP: [5](#), [24](#).
 IPSEC_IKE_APP: [5](#), [24](#).
 IRC_APP: [5](#), [24](#).
 IS_TCP: [24](#).
 IS_UDP: [24](#).
j: [31](#).
k: [31](#).
label: [25](#), [27](#), [28](#), [29](#), [30](#).
 little-endian: [16](#).
magic1: [16](#), [17](#), [18](#).
magic2: [16](#), [17](#).
 MAIL_APP: [5](#), [24](#).
main: [7](#).
 MAXPATHLEN: [9](#), [11](#).
 MCIDAS_APP: [5](#), [24](#).
min: [19](#).
minwin: [30](#), [31](#).
 MSWINDOWS_APP: [5](#), [24](#).
 MULTICAST_APP: [5](#), [24](#).
nbytes: [33](#).
 NEOMODUS_APP: [5](#), [24](#).
newpos: [31](#).
 NFS_APP: [5](#), [24](#).
 NNTP_APP: [5](#), [24](#).
 NTP_APP: [5](#), [24](#).
 NUM_APP: [5](#), [8](#), [26](#).
 NUM_PACKET_SIZE: [5](#), [8](#), [21](#), [29](#).
 NUM_PORT: [5](#), [30](#).
 NUM_PROTOCOL: [5](#), [8](#), [27](#).
 NUM_TOS: [5](#), [8](#), [28](#).
 NUMWIN: [30](#), [31](#).
 O_RDONLY: [12](#).
 OCTETS: [5](#), [8](#), [20](#), [25](#), [30](#), [31](#).
octets: [20](#), [21](#), [22](#), [23](#).
offset: [23](#).
open: [9](#), [12](#).
 output files formats: [1](#).
output_path: [11](#), [13](#).
packets: [20](#), [21](#), [22](#), [23](#).
 PACKETS: [5](#), [8](#), [20](#), [25](#).
packets_of_size: [5](#), [8](#), [21](#), [29](#).
 performance: [1](#).
 Perl: [1](#).
perror: [12](#), [13](#), [14](#), [15](#), [17](#), [19](#).
persistent_read: [15](#), [17](#), [19](#), [33](#).
 PORT_IN_RANGE: [24](#).
 PORT_IS: [24](#).
 PORTMAPPER_APP: [5](#), [24](#).
ports: [5](#), [24](#), [30](#), [31](#).
 PORTZERO_APP: [5](#), [24](#).
 PRINT_FPO: [25](#), [26](#), [27](#), [28](#), [29](#), [30](#).
printf: [25](#).
prot: [20](#), [21](#), [22](#), [23](#), [24](#).
 PROTO_TCP: [24](#).
 PROTO_UDP: [24](#).
protocols: [5](#), [8](#), [21](#), [27](#).
 QUAKE_APP: [5](#), [24](#).
rc: [19](#), [33](#).
 RCS: [2](#).
resid: [2](#).

read: 15, 33.
REALPLAYER_APP: 5, 24.
RECORD_SIZE: 15.
RSYNC_APP: 5, 24.
RTIP_APP: 5, 24.
s_version: 16, 17, 18.
SAMPLING_RATE: 1, 5, 22, 25.
SHOUTCAST_APP: 5, 24.
SNMP_APP: 5, 24.
snprintf: 13, 14, 27, 28, 29, 30.
SOCKS_APP: 5, 24.
SQUID_APP: 5, 24.
src_as: 20, 22, 23.
src_ip: 20, 23.
src_port: 20, 22, 23, 24.
SSH_APP: 5, 24.
ssize_t: 15, 19, 33.
SSM_APP: 5, 24.
STARSIEGE_APP: 5, 24.
start_time: 20, 22, 23.
stderr: 1, 11, 15, 18, 21.
stdio: 10.
stdout: 10, 13, 32.
STREAMWORKS_APP: 5, 24.
TELNET_APP: 5, 24.
tos: 20, 21, 23.
toses: 5, 8, 21, 28.
TOTAL_APP: 5, 21.
type: 8, 20, 23, 25.
u_int16_t: 20, 23, 30.
u_int32_t: 16, 17, 20, 21, 23.
u_int64_t: 5, 22, 30.
u_int8_t: 16, 20, 23.
UNIDATA_LDM_APP: 5, 24.
VOIP_APP: 5, 24.
WARNING: 1.
WARNING: reading data file: 15.
WARNING: record...discarded...: 15.
WINMEDIA_APP: 5, 24.
WINMX_APP: 5, 24.
winners: 30, 31.
winports: 30, 31.
XXX: 16, 24.
X11_APP: 5, 24.
OxCF: 18.

- ⟨ Clean up before exit 32 ⟩ Used in section 7.
- ⟨ Function *persistent_read* 33 ⟩ Used in section 6.
- ⟨ Functions 6 ⟩ Used in section 3.
- ⟨ Global variables 2, 5, 10 ⟩ Used in section 3.
- ⟨ Header files to include 4, 9 ⟩ Used in section 3.
- ⟨ Header sanity check 18 ⟩ Used in section 16.
- ⟨ Initialize global variables and data structures 8 ⟩ Used in section 7.
- ⟨ Insert new winner port into the appropriate place 31 ⟩ Used in section 30.
- ⟨ Main loop: read and process NetFlow records individually 15 ⟩ Cited in section 10. Used in section 7.
- ⟨ Maybe output this record as a bulk TCP flow 22 ⟩ Used in section 20.
- ⟨ Parse NetFlow v5 record using constant offsets 23 ⟩ Used in section 20.
- ⟨ Print IP protocol counters 27 ⟩ Used in section 25.
- ⟨ Print TOS counters 28 ⟩ Used in section 25.
- ⟨ Print application-specific counters 26 ⟩ Used in section 25.
- ⟨ Print average packet size counters 29 ⟩ Used in section 25.
- ⟨ Print data from unidentified ports 30 ⟩ Used in section 25.
- ⟨ Print the results 25 ⟩ Used in section 7.
- ⟨ Process NetFlow v5 record starting from *current_record* 20 ⟩ Used in section 15.
- ⟨ Process command line 11 ⟩ Used in section 8.
- ⟨ Read NetFlow v5 header 16 ⟩ Used in section 7.
- ⟨ Read and discard any remaining portion of the header 19 ⟩ Used in section 16.
- ⟨ Read first 8 bytes of NetFlow v5 header 17 ⟩ Used in section 16.
- ⟨ Recognize individual protocols and update corresponding counters 24 ⟩ Used in section 20.
- ⟨ Set up bulk TCP flows output 14 ⟩ Used in section 11.
- ⟨ Set up input file 12 ⟩ Used in section 11.
- ⟨ Set up statistics output 13 ⟩ Used in section 11.
- ⟨ The main program 7 ⟩ Used in section 3.
- ⟨ Update non-application-specific counters 21 ⟩ Used in section 20.