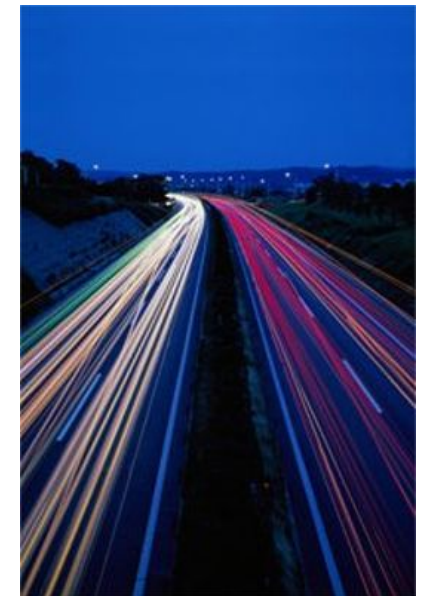




Evangelos Markatos
FORTH-ICS
markatos@ics.forth.gr

<http://www.ics.forth.gr/~markatos>
Institute of Computer Science (ICS)
Foundation for Research and Technology – Hellas (FORTH)

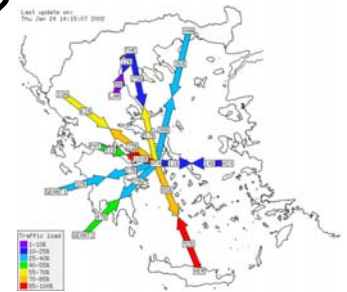
- Introduction
- What kinds of applications does MAPI support?
- Basic Abstraction:
 - The network flow
- Example applications
 - Counting packets in a network
 - Measuring used bandwidth
 - Detecting Intrusions
 - Passive FTP monitoring
- Summary



What kinds of applications does it support?



- Traffic Categorization/Accounting:
 - What % of my traffic is due to email?
 - Which subnet generates most outgoing traffic?
- Bandwidth Usage
 - What % of my bandwidth is available now?
 - What % of my bandwidth is being used?
- Study trends:
 - How does the application mix in the traffic changes with time?
 - ftp in the 80's, www in the 90's, p2p in the 00's
 - How does peer-to-peer traffic changes with time?
- Performance Debugging
 - Why is **my** application so sloooow?



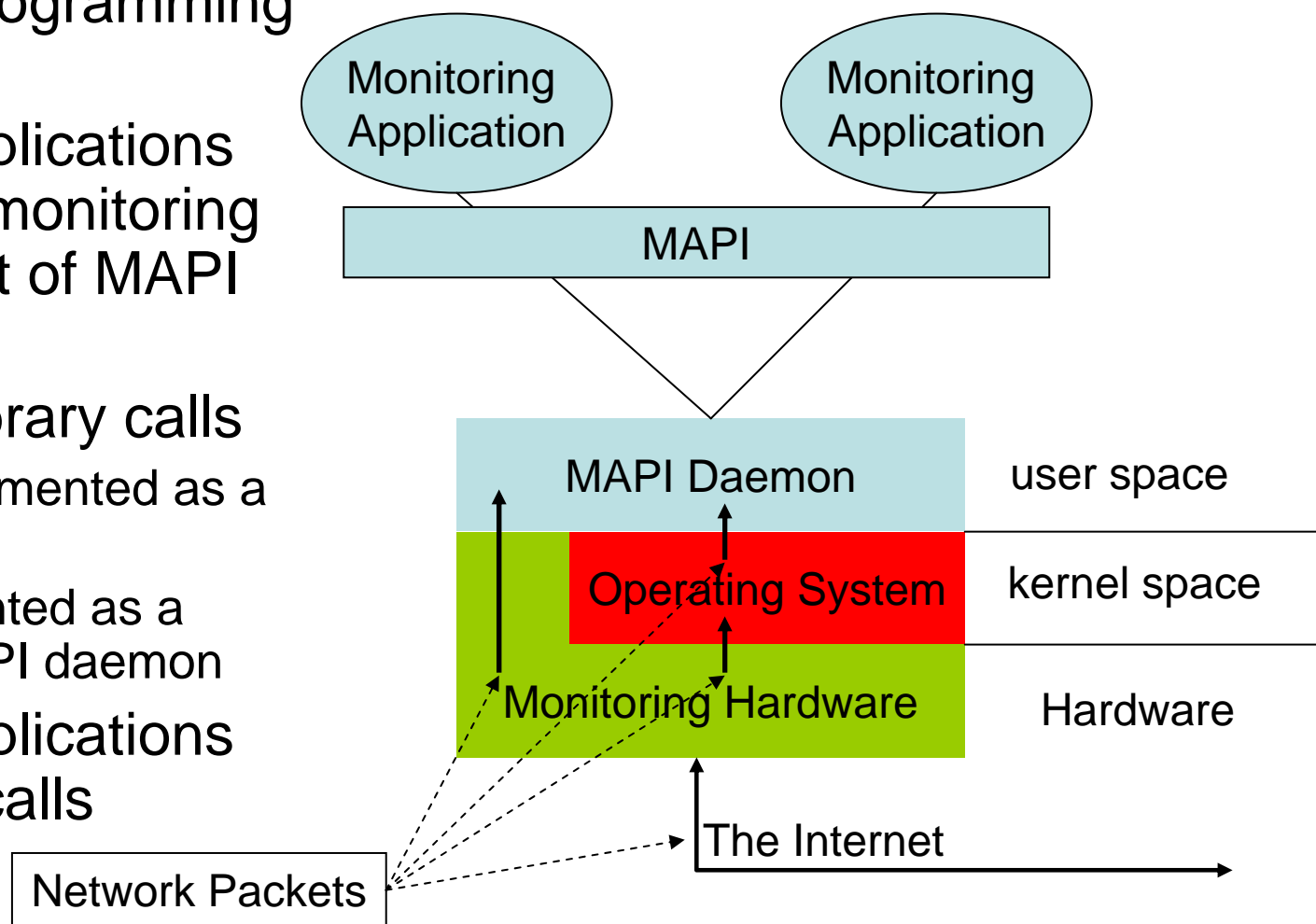
Passive Monitoring: What can it be used for? Security



- **Intrusion Detection**
 - Are any of my computers compromised?
 - Is there any attacker trying to intrude into my network?
- **Large-scale Attack Detection – Detection of Epidemics**
 - DoS Attack detection
 - e.g. Detect sharp increases in TCP/SYN packets
 - Zero-day worm detection
 - e.g. Detect lots of identical packets, never seen before, from several sources to several destinations
 - e.g. Detect worm characteristics
 - such as NOP sleds: long sequences of executable code
- **Network Telescopes**
 - They monitor unused IP addresses
 - Observe victims of DoS attacks
 - “back-scatter” traffic
 - Observe infected hosts
 - Port scans



- MAPI: **M**onitoring **A**pplication **P**rogramming **I**nterface
- Monitoring applications express their monitoring needs as a set of MAPI calls
- It is a set of library calls
 - It is not implemented as a library
 - It is implemented as a separate MAPI daemon
- Monitoring applications invoke these calls



- **Portability**
 - Write your application once
 - Run it everywhere
 - On ordinary cards, on specialized hardware, etc.
- **Re-use of code**
 - There exist frequently used useful functions
 - String searching, packet/byte counters, etc.
 - Put them in MAPI
- **Efficiency**
 - Applications express their monitoring needs to MAPI
 - MAPI pushes functionality all the way down to the hardware (in the network card)
 - e.g. filtering, string searching, etc.
 - MAPI delivers to applications only the needed packets
 - Filtering is done down at the hardware

The major abstraction:

– The Network Flow

- What is a network flow?

- a subset of the traffic, e.g.



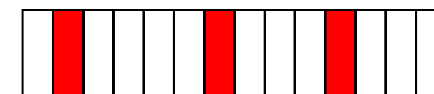
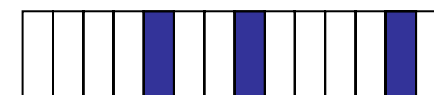
- all packets destined to my web server



- all packets destined to my web server which contain the string “/bin/sh”



- All packets to port 8080 which contain the CODE-RED worm



The Network Flow

What can you do with it?



- Create a network flow
 - int **mapi_create_flow** (char *dev)
- Destroy (close) a network flow
 - int **mapi_close_flow** (int fd)
- Connect to a network flow
 - int **mapi_connect** (int fd)

- You can apply functions to it
 - int **mapi_apply_function** (int fd, char *funct, ...)
 - **mapi_apply_function** (fd, "BPF_FILTER", "tcp and dst port 80"
 - We are only interested in receiving **TCP** packets going to **port 80**
 - **mapi_apply_function** (fd, "BYTE_COUNTER")
 - We are not interested in the packets themselves – we are only interested in counting the number of bytes in them

- Get information from a network flow
- Reading packets from a network flow
 - `struct mapipkt * mapi_get_next_pkt (int fd, int fid)`
- Reading function results
 - `void * mapi_read_results (int fd, int fid, int type)`

- **Create** a network flow: `mapi_create_flow`
- **Apply functions** to it : `mapi_apply_function`
- **Connect** to the network flow: `mappi_connect`
- **FOREVER**
 - **Read packets** or **function results** from the flow:
`mapi_get_next_pkt`, `mapi_read_results`
 - Print/store them

- Create an application which
 - Counts the number of received packets by our web server in the past 10 seconds
- How?
 - Create a flow (consisting of all packets destined to our web server)
 - Apply the function “PACKET_COUNT” to it
 - Go to sleep for 10 seconds
 - Wake up and print the counter

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <unistd.h>
4 #include "mapi.h"
5
6 int main() {
7
8     int fd, fid;
9     unsigned long long *cnt;
10
11     /* create a flow using the eth0 interface */
12     fd = mapi_create_flow("eth0");
13     if (fd < 0) {
14         printf("Could not create flow\n");
15         exit(EXIT_FAILURE);
16     }
17
18     /* keep only the packets directed to the web server */
19     mapi_apply_function (fd, "BPF_FILTER", "tcp and dst port 80");
20
```



Packet Count Example (2)



```
21 /* and just count them */
22 fid = mapi_apply_function (fd, "PKT_COUNTER");
23
24 /* connect to the flow */
25 if( mapi_connect (fd) < 0) {
26     printf("Could not connect to flow %d\n", fd);
27     exit(EXIT_FAILURE);
28 }
29
30 sleep(10);
31
32 /* read the results of the applied PKT_COUNTER function */
33 cnt = (unsigned long long *) mapi_read_results (fd, fid, MAPI_COPY);
34 printf("pkts:%llu\n", *cnt);
35
36 mapi_close_flow (fd);
37 return 0;
38 }
```

- Create a network flow
 - Consisting of all incoming traffic to my computer
- Apply the function `BYTE_COUNT` to it
- Go to sleep
- Every one second wake up and
 - Print the counter

Link Utilization Example: periodically report incoming traffic in Mbps

```
int fd, fid ;
unsigned long long * cnt, prev = 0 ;
fd = mapi_create_flow("eth0");
if ((fd < 0)) {
    printf("Could not create flow\n");
    exit(EXIT_FAILURE);
}
mapi_apply_function (fd, "BPF_FILTER", "dst host 139.91.145.84");
/* count the bytes of the flow */
fid = mapi_apply_function (fd, "BYTE_COUNTER");
/* connect to the flow */
if (mapi_connect (fd) < 0) {
    printf("Could not connect to flow %d\n", fd); exit(-1) }
/* get reference to the memory where the results are stored */
while(1) { /* forever, report the load */
    int * new;
    sleep(1);
    new = (unsigned long long *) mapi_read_results (fd, fid, MAPI_COPY);
    printf("incoming: %.2f Mbit/s (%llu bytes)\n",
        (*new-prev)*8/1000000.0, (*new-prev));
    prev = *new;
}
```

```
int fd, fid ;
unsigned long long * cnt, prev = 0 ;
fd = mapi_create_flow("eth0");
if ((fd < 0)) {
    printf("Could not create flow\n");
    exit(EXIT_FAILURE);}
mapi_apply_function (fd, "BPF_FILTER", "dst host 139.91.145.84");
/* count the bytes of the flow */
fid = mapi_apply_function (fd, "BYTE_COUNTER");
/* connect to the flow */
If (mapi_connect (fd) < 0) {
    printf("Could not connect to flow %d\n", fd); exit(-1) }
/* get reference to the memory where the results are stored */
cnt=(unsigned long long *)mapi_read_results(fd, in_fid,MAPI_REF);
while(1) { /* forever, report the load */
    int new ;
    sleep(1);
    new = *cnt ;
    printf("incoming: %.2f Mbit/s (%llu bytes)\n",
        (new-prev)*8/1000000.0, (new-prev));
    prev = new;
```

- Find the slapper worm
 - UDP packets from port 2002 to web servers
 - Contains substring “|00 00|E|00 00|E|00 00|@|00” at the first 100 bytes

- SNORT signature:

```
alert udp $EXTERNAL_NET 2002 -> $HTTP_SERVERS $HTTP_PORTS (msg:"MISC
  slapper worm admin traffic";
content:"|00 00|E|00 00|E|00 00|@|00|"; depth:100;
reference:url,isc.incidents.org/analysis.html?id=167;
reference:url,www.cert.org/advisories/CA-2002-27.html;
classtype:trojan-activity; sid:1889; rev:5;)
```

- MAPI application:

- Create a network flow consisting of
 - All packets coming from port 2002
 - Destined to the \$HTTP_PORTS of our \$HTTP_SERVERS
 - Which contain the substring “|00 00|E|00 00|E|00 00|@|00”
 - In the first 100 bytes
- Wait for ever
 - To receive a packet from this network flow

```
if ((fd = mapi_create_flow("eth0")) < 0) {  
    printf("Could not create flow\n");  
    exit(EXIT_FAILURE);} 
```

```
/* apply the BPF part of the signature */
```

```
mapi_apply_function (fd, "BPF_FILTER", "udp and src port 2002 and dst net  
139.91.23 and dst port 80");
```

```
mapi_apply_function (fd, "STR_SEARCH", "|00 00|E|00 00|E|00 00|@|00|", 0, 100);
```

```
fid = mapi_apply_function (fd, "TO_BUFFER");
```

```
if( mapi_connect (fd) < 0) {
```

```
    printf("Could not connect to flow %d\n", fd);
```

```
    exit(EXIT_FAILURE);} 
```

```
while(1) { /* forever, wait for matching packets */
```

```
    pkt = mapi_get_next_pkt (fd, fid);
```

```
    printf("\nSlapper worm packet!\n");
```

```
    print_IP_packet(pkt);
```

```
}
```

- Find all passive FTP traffic
 - Challenge:
 - Data transfers are carried over dynamically generated ports (i.e. not over static ports such as port 80 web traffic)
 - We need to find these dynamic ports and monitor them
- Implementation
 - Create a flow that monitors “control port” 21
 - Capture all “port 21” packets and find dynamic ports
 - Create a new network flow for each dynamic port
 - Monitor these new flows and report traffic

```
fd = mapi_create_flow("eth0"),  
mapi_apply_function (fd, "BPF_FILTER", "tcp and port 21");  
mapi_apply_function (fd, "STR_SEARCH", "227 Entering Passive Mode", 0, 0,  
1500);  
buf_fid = mapi_apply_function (fd, "TO_BUFFER");  
mapi_connect (fd);  
while(1){  
    pkt = mapi_get_next_pkt (fd, bufid);  
    extract_port (pkt, &port); filter = create_filter (port);  
  
    /* Create and set up a flow for the data transfer */  
    flow_struct = allocate_new_flow_structure() ;  
    flow_struct->fd = mapi_create_flow ("eth0");  
    mapi_apply_function (flow_struct.fid, "BPF_FILTER", filter);  
    flow_struct->pc = mapi_apply_function (flow_struct.fid, "PKT_COUNTER");  
    flow_struct->bc = mapi_apply_function (flow_struct.fid, "BYTE_COUNTER");  
    mapi_connect (flow_struct->fd); add_to_flow_list(flow_struct)  
  
    while(list_not_empty(flow_list)){  
        aggregate_counter_results(); }  
    report_results();  
}
```

- Main Abstraction
 - Network Flow: a subset of packets
- Main program skeleton of MAPI
 - Create a flow
 - Apply some functions to it
 - Read the resulting packets/statistics
- Advantages:
 - Portability (runs everywhere)
 - Ease of use (high-level abstraction)
 - Performance (pushes functionality downwards)
 - Library of pre-existing functions



Evangelos Markatos
FORTH-ICS
markatos@ics.forth.gr

<http://www.ics.forth.gr/~markatos>
Institute of Computer Science (ICS)
Foundation for Research and Technology – Hellas (FORTH)