



Implementing passive monitoring applications using MAPI

Arne Øslebø

UNINETT

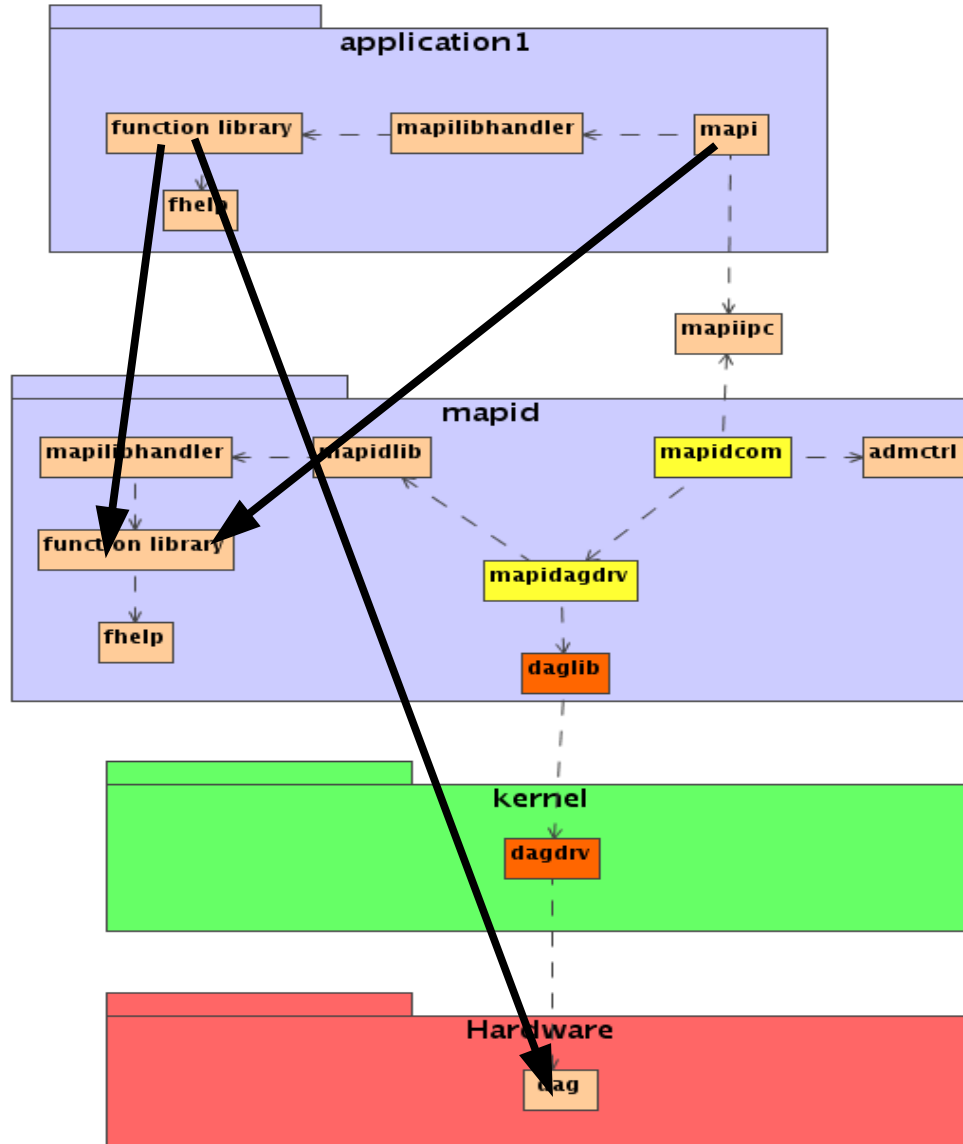
Arne.Oslebo@uninett.no

- General overview
 - What is MAPI?
 - Example
- Design
- Using MAPI
 - MAPI calls
 - MAPI functions
- Using DiMAPI
- Extending MAPI

- Monitoring Application Programming Interface
- Design goals:
 - Make it quick and easy to implement new monitoring applications
 - Low overhead
 - Support for multiple concurrent users and applications
 - Optional support for strong authentication.
 - Global optimization
 - Optimize processing of packets based on all applications from all users.
 - Transparent support for different hardware adapters
 - NIC, DAG, COMBO6
 - Easy to extend

Worm detection:

```
1: fd=mapi_create_flow("/dev/dag0");
2: mapi_apply_function(fd,"BPF_FILTER","src port 1234");
3: ctr_id1=mapi_apply_function(fd,"PKT_COUNTER");
4: mapi_apply_function(fd,"STR_SEARCH","pattern",100,300);
5: ctr_id2=mapi_apply_function(fd,"PKT_COUNTER");
6: mapi_apply_function(fd,"TO_FILE",MFF_PCAP,"worm.trace",0);
7: mapi_connect(fd);
8:
9: while(1) {
10:     ctr_val1=mapi_read_results(fd,ctr_id1);
11:     ctr_val2=mapi_read_results(fd,ctr_id2);
12:
13:     printf("BPF match: %llu String match: %llu\n",
14:          *ctr_val1,*ctr_val2);
15:     sleep(10);
16: }
```





MAPI calls

- `mapi_create_flow`
- `mapi_create_offline_device`
- `mapi_start_offline_device`
- `mapi_delete_offline_device`
- `mapi_apply_function`
- `mapi_connect`
- `mapi_read_results`
- `mapi_get_next_pkt`
- `mapi_loop`
- `mapi_read_error`
- `mapi_close_flow`
- `mapi_set_authdata`
- `mapi_get_[device/library/flow/function]_info`
- `mapi_get_next_[device/library/low/function]_info`

- Collection of MAPI functions that can be applied to a MAPI flow
 - BPF, string search, packet counter, jitter, statistical function etc.
- Functions can:
 - filter packets
 - transform packets
 - return or process information from packets
 - read and process results from other functions
 - etc.
- Standard library: 24 functions



BINOP

- Adds or subtracts values from two other MAPI functions
- Arguments:
 - fd and fid for left function: int
 - fd and fid for right function: int
 - type:int (BINOP_ADD,BINOP_SUB)
- Return value:
 - unsigned long long
- Example:
 - `ts1=mapi_apply_function(fd,"PKT_INFO",PKT_TS);`
 - `ts2=mapi_apply_function(fd2,"PKT_INFO",PKT_TS);`
 - `binop=mapi_apply_function(fd,"BINOP",`
`fd2,ts2,fd1,ts1,BINOP_SUB);`

- Filters the packets according to a BPF filter expression
- Arguments
 - `bpf_filter` : `char*`
 - BPF filter
- Return value:
- Usage:
 - `mapi_apply_function(fd, "BPF_FILTER", "port 80");`

- Returns results from other MAPI functions in precise periodical intervals
- Arguments
 - interval : char*
 - fd : int
 - function : int
- Return value:
 - void *
- Usage:
 - `count=mapi_apply_function(fd, "PKT_COUNTER");`
 - `mapi_apply_function(fd, "BUCKET",
fd, count, "1s");`

- Counts number of bytes in a flow
- Arguments
- Return value:
 - unsigned long long
- Usage:
 - `mapi_apply_function(fd, "BYTE_COUNTER");`

- Processes the packets of a flow by performing IP defragmentation and TCP stream reassembly
- Arguments
 - threshold : int
 - minimum stream size, default 32Kb
 - timeout : int
 - timeout value for further processing since the first arrival of a packet. Default 30 seconds
- Return value:
- Usage:
 - `mapi_apply_function(fd, "COOKING", -1, -1);`

- Returns an array that represents the distribution of results from another MAPI function.
- Arguments
 - fd : int
 - function : int
 - min : char*
 - max : char*
 - interval : char*
- Return value:
 - unsigned long long[]
- Usage:
 - `gap=mapi_apply_function(fd,"GAP");`
 - `mapi_apply_function(fd,"DIST",fd,gap,
"1ms","2ms","1us");`

- Returns the time delay between two consecutive packets in a flow
- Arguments
- Return value:
 - unsigned long long
- Usage:
 - `mapi_apply_function(fd, "GAP");`

- Computes the additive hash function over the packets of a network flow.
- Arguments
 - layer : int
 - link, network, IP or application layer
- Return value:
 - unsigned int
- Usage:
 - `mapi_apply_function(fd, "HASH");`

- Hashing based sampling
- Only processes TCP and UDP packets
- Calculates hash on src/dst IP and src/dst port
- Arguments
 - range : int
 - keep : int
- Return value:
 - struct sample {
 unsigned int source_ip;
 unsigned int dest_ip;
 ..
}
- Usage:
 - `mapi_apply_function(fd, "HASHSAMP", 100, 1);`

- Counts number of packets in a flow
- Arguments
- Return value:
 - unsigned long long
- Example:
 - `mapi_apply_function(fd, "PKT_COUNTER");`

- Returns information about a packet
- Arguments
 - info : int
 - PKT_TS – packet timestamp
 - PKT_SIZE – packet size
- Return value:
 - unsigned long long
- Usage:
 - `mapi_apply_function(fd, "PKT_INFO", PKT_SIZE);`

- Returns a specific protocol field
- Arguments
 - info : int
 - PI_TCPSEQ – TCP Sequence number
 - PI_TCPACK – TCP Ack number
 - Will be extended to support all fields in IP, TCP and UDP
- Return value:
 - unsigned long long
- Usage:
 - `mapi_apply_function(fd, "PROTINFO", PI_TCPSEQ);`

- Regular expression pattern matching
- Arguments
 - str : char*
- Return value:
- Example:
 - `mapi_apply_function(fd, "REGEXP", "[Aa]");`

- Stores results from other MAPI function to a file
- Arguments
 - functions : char*
 - list of functions to read results from
 - “<fid>@<fd>,<fid2>@<fd2>”
 - format : char*
 - R2F_RAW, R2F_ULLSTR, R2F_ULLSEC, R2F_STATS
 - header : char*
 - string to store in the beginning of the file
 - filename : char*
 - interval : char*
 - “-1” for always, “0” for once when flow closes, “1s”, “1.2ms” etc.

- Usage:

- `count=mapi_apply_function(fd, "PKT_COUNT");`
- `gap=mapi_apply_function(fd, "GAP");`
- `sprintf(fids, "%d@d, %d@d", count, fd, gap, fd);`
- `sprintf(format, "%d, %d", R2F_ULL, R2F_ULLSEC);`
- `mapi_apply_function(fd, "RES2FILE", fids, format, "Count Gap", "test.res", "-1")`

- Performs sampling
- Arguments
 - value : int
 - for DETERMINISTIC it specifies a sampling interval of 1 out of *value* packets. For probabilistic it specifies the probability in %.
 - mode : int
 - DETERMINISTIC or PROBABILISTIC
- Return value:
 - unsigned long long
- Usage:
 - `mapi_apply_function(fd, "SAMPLE", 100, DETERMINISTIC);`

- Starts and/or stops measurements at a specific time
- Arguments
 - start : char*
 - stop : char*
 - So far only relative time from the first captured packet is supported. Will be extended to support absolute time.
 - “0”, “1s”, “10ms”
- Return value:
- Example:
 - `mapi_apply_function(fd, "STARTSTOP", "10s", "0") ;`

- Returns statistical information about results from other MAPI functions
- Arguments
 - fd : int, fid : int
 - skip : int - number of packets to skip before reading first result
- Return value:
 - struct stats {
 unsigned long long count;
 long double sum;
 long double sum2;
 double max;
 double min}
- Usage:
 - gap=mapi_apply_function(fd, "GAP");
 - mapi_apply_function(fd, "STAT", fd, gap, 1);

- Search the payload of a packet for a specific pattern
- Arguments
 - pattern : char*
 - pattern to search for. Can contain non-printable characters or binary data using Snort syntax.
 - offset : int
 - offset into packet for starting the search
 - depth : int
 - maximum depth of the search
- Return value:
- Usage:
 - `mapi_apply_function(fd, "STR_SEARCH", "ab|63 64|", 0, 1500);`

- Signals when a certain threshold is reached.
- Arguments
 - type : int
 - CHAR, INT, ULL
 - fd : int
 - fid : in
 - boundary : int
 - EQ, GT, LT, EQ_D, GT_D, LT_D
 - threshold : unsigned long long
 - timeout : int
 - divider : int
 - count : int
- Return value:

- Returns top X values of a field
- Arguments
 - x : int
 - protocol : int
 - field : int
- Return value:
 - unsigned int[]
- Usage:
 - `mapi_apply_function(fd, "TOP", 5, TOPX_TCP, TOPX_TCP_DSTPORT);`

- Stores packets of a flow to a buffer so that they can be read using `mapi_get_next_packet`

- Arguments

- Return value:

```
- struct mapipkt {  
    unsigned long long ts;  
    unsigned short ifindex;  
    unsigned caplen;  
    unsigned wlen;  
    unsigned char pkt;  
}
```

- Usage:

```
- mapi_apply_function(fd, "TO_BUFFER" );
```

- Stores packets to a file
- Arguments
 - format : int
 - MFF_PCAP, MFF_ERF, MFF_RAW
 - file_name : char*
 - count : unsigned long long
 - number of packets to store. 0 means all packets will be stored.
- Return value:
- Usage:
 - `mapi_apply_function(fd, "TO_FILE", MFF_ERF, 1000);`

- DAG library
 - Store packets to file using DAG ERF format
 - Filter on interface
- Anonymization library
 - Versatile packet anonymization functions
- IPFIX library
 - Export Netflow v5, v9 or IPFIX flow records
- Protocol tracker library
 - Detect and track protocols that uses dynamic ports
 - P2P, FTP etc.

- <http://www.ist-lobster.org/downloads/>
- Makefile.in
 - DEBUG – turns on debug messages
 - WITH_DAG – adds support for DAG cards
 - WITH_COMBO6 – adds support for COMBO6 cards
 - WITH_ADMISSION_CONTROL – add support for admission control
 - WITH_FUNCTION_STATS – turns on counters for all functions to show how many packets where processed by each function
 - WITH_PRIORITIES – turns on support for flow priorities
 - WITH_MODIFY_PKTS – turns on support for functions that modifies packets.
 - WITH_ANONYMIZATION – compile anonymization library
 - WITH_TRACKING – compile tracking library
 - WITH_IPFIX – compile IPFIX library



Edit mapi.conf and test

```
libpath=.  
libs=mapidstdflib.so:dagflib.so  
drvpath=.
```

```
[driver]  
device=eth0  
driver=mapinicdrv.so
```

```
[driver]  
device=/dev/dag0  
driver=mapidagdrv.so  
description=DAG 4.3GE  
alias=trd-ntnu
```

```
[format]  
format=MFF_DAG_ERF  
driver=mapidagdrv.so
```

```
[format]  
format=MFF_PCAP  
driver=mapinicdrv.so
```

Start mapid: ./mapid

Run a test application:
cd tests
./test_off_pkt_counter tracefile

- Implement an application that can measure some qualities of a TCP stream
- We want to look at the following attributes:
 - Packets per second
 - Bit per second
 - Jitter
 - TCP congestion window
- Results should be written to a file for further study.
- Syntax:
 - `tcpanalyze <filename> <src_ip> <src_port>`
`<dst_ip> <dst_port>`

- PKT_COUNTER
 - To calculate packets per second
- BYTE_COUNTER
 - To calculate bit per second
- GAP
 - To measure jitter
- PROTINFO
 - TCP sequence and ack numbers
- BINOP
 - To calculate TCP congestion window
- STAT
 - Calculate statistical values
- RES2FILE
 - To store results to file

- Create a new subdirectory
 - mkdir tcpanalyze
 - cd tcpanalyze
- Create file tcpanalyze.c
- Include necessary headers:

```
#include <stdlib.h>
#include <stdio.h>
#include "mapi.h"
#include "stdlib/pktinfo.h"
#include "stdlib/res2file.h"
#include "stdlib/protinfo.h"
```



Create flow and apply BPF filter

```
int main(int argc, char *argv[]) {
    char *file=argv[0];

    int snd=mapi_create_flow("eth0");
    if(snd==-1) {
        printf("Could not create flow\n");
        exit(-1);
    }
    int rec=mapi_create_flow("eth0");

    char filter[1024];
    snprintf(filter,1024,"tcp and src host %s and src port %s and
                        dst host %s and dst port %s",
            argv[1],argv[2],argv[3],argv[4]);
    mapi_apply_function(snd,"BPF_FILTER",filter);

    snprintf(filter,1024,"tcp and src host %s and src port %s and
                        dst host %s and dst port %s",
            argv[3],argv[4],argv[1],argv[2]);
    mapi_apply_function(rec,"BPF_FILTER",filter);
}
```



Get statistical data and start flow

```
int pkts=mapi_apply_function(snd,"PKT_COUNTER");
int bytes=mapi_apply_function(snd,"BYTE_COUNTER");
int gap=mapi_apply_function(snd,"GAP");
int gapstat=mapi_apply_function(snd,"STAT",snd,gap,1);
int seq=mapi_apply_function(snd,"PROTINFO",PI_TCPSEQ);
int ack=mapi_apply_function(rec,"PROTINFO",PI_TCPACK);
int cwnd=mapi_apply_function(snd,"BINOP",BINOP_SUB,
                             snd,seq,rec,ack);
int cwndstat=mapi_apply_function(snd,"STAT",snd,gap,1);

char fids[64],types[64];
snprintf(fids,64,"%d@d,%d@d,%d@d,%d@d",
         pkts,snd,bytes,snd,gapstat,snd,cwndstat,snd);
snprintf(types,64,"%d,%d,%d,%d",R2F_ULLSTR,R2F_ULLSTR,
         R2F_STAT,R2F_STAT);
mapi_apply_function(fd,"RES2FILE",fids,types,file,"1s");

mapi_connect(snd);
mapi_connect(rec);

while(1);
```



Makefile

```
SOURCES=$(wildcard *.c)
TARGETS=$(SOURCES:.c=)

CFLAGS=-g -O2 $(C_WARNINGS) -DDEBUG -I..
LDFLAGS= ../mapi.so -lpcap -lpthread

all: $(TARGETS)

%:%.c ../mapi.so ../mapi.h
    $(CC) $(CFLAGS) -o $@ $< $(LDFLAGS)

clean:
    rm -rf $(TARGETS)
```

- What do you do if you have multiple probes and want to measure the TCP quality at all of them?
- Answer: use DiMAPI!
- DiMAPI is the distributed version of MAPI
- The SCOPE abstraction:
 - SCOPE is a set of lines (interfaces) to monitor
 - `mapi_create_flow("host1:eth2, host2:/dev/dag0, host3:eth1");`
 - `mapi_read_result` returns an array of results
- Fully compatible with MAPI
 - All previous functions work as usual



Implementing new MAPI functions



<http://www.ist-lobster.org>

- All code in one single source file
- Header file only needed if function returns complex data.
- Script automatically creates source file for libraries
- MAPI function interfaces:
 - instance
 - init
 - process
 - get_result
 - reset
 - cleanup
 - client_init
 - client_get_result
 - client_cleanup



MAPI function definition

```
typedef struct mapidflib_function_def {
    char* libname;
    char* name;
    char* descr;
    char* argdescr;
    char* devtype;
    mapi_result_method_t restype;
    int shm_size;
    short modifies_pkts;
    mapidflib_optimize_t optimize;
    int (*instance)(....);
    int (*init)(....);
    int (*process)(....);
    int (*get_result)(....);
    int (*change_args)(....);
    int (*reset)(....);
    int (*cleanup)(....);
    int (*client_init)(....);
    int (*client_read_result)(....);
    int (*client_cleanup)(....);
} mapidflib_function_def t;
```



PKT_COUNTER example

```
static mapidflib_function_def_t finfo={
    "", //libname
    "PKT_COUNTER", //name
    "Counts number of packets\n\tReturn value: unsigned long
long", //descr
    "", //argdescr
    MAPI_DEVICE_ALL, //devtype
    MAPIRES_SHM, //Use shared memory to return results
sizeof(unsigned long long), //shm size
    0, //modifies_pkts
    MAPIOPT_NONE, //global optimization
    NULL, //instance
    NULL, //init
    pktc_process,
    NULL, //get_result,
    pktc_reset,
    NULL, //cleanup
    NULL, //client_init
    NULL, //client_read_result
    NULL //client_cleanup
};
```



PKT_COUNTER example



<http://www.ist-lobster.org>

```
static int pktc_process(  
    mapidflib_function_instance_t *instance,  
    const unsigned char* dev_pkt,  
    const unsigned char* link_pkt,  
    mapid_pkthdr_t* pkt_head)  
{  
    (*(unsigned long long*)instance->result.data)++;  
    return 1;  
}  
  
static int pktc_reset(  
    mapidflib_function_instance_t *instance)  
{  
    (*(unsigned long long*)instance->result.data)=0;  
    return 0;  
}
```



Current status of MAPI

- Still under development
- Stable if used as intended
- Can segfault if wrong arguments are passed
- Working hard on making it more robust
- Available from:
 - <http://www.ist-lobster.org/downloads/>



More information?



<http://www.ist-lobster.org>

<http://www.ist-lobster.org>