The SCAMPI Approach

2nd SCAMPI Workshop
3 May 2004

Arne Øslebø
UNINETT

Overview

- Why do monitoring?
- High speed monitoring challenges
- Existing approaches
- SCAMPI approach
- Architecture
- How to extend the functionality

Why do monitoring?

- Detect network problems
- QoS
- Capacity planning and traffic engineering
- Security
  - DOS attacks
  - Intrusion detection
- Research

High speed monitoring challenges

- 10Gbps
  - 1250MB/s
  - Worst case: ~24Mpps
  - Normal network: 1-2Mpps
- PC limitations
  - Bus speed, Memory, CPU
- Too much data to process everything on the host PC
- Solutions:
  - Only headers
  - Filtering
  - Sampling
  - Aggregation
  - Packet processing on the network adapter

Existing approaches

- Libpcap
  - Available for all NICs and also specialized cards like DAG.
  - Only BPF filters
  - Only packets
- dagapi
  - Used for DAG cards
  - Header filters and string searches
  - Only packets
  - Proprietary
- CoralReef
  - Uniform interface for passive monitoring
  - Suite of different programming interfaces covering different layers.

The SCAMPI approach

- Expressiveness
- Process packets in hardware on the network adapter
- Hardware transparent for applications
- Push processing of packets automatically down to the hardware if possible
- Network flow:
  - A set of IP packets captured by a device
- Reuse of code
  - UNIX:
    - cat <file> | wc
  - Monitoring:
    - BPF filter | string search | store to file

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
  - Easy to extend

MAPI fundamentals

- Network flow
  - mapi_create_flow
  - Initially all packets seen on the network by the network adapter
- Apply functions to a flow
  - mapi_apply_function
  - BPF filter, string search, packet counter, byte counter, Netflow, jitter etc.
- Connect to flow
  - mapi_connect_flow
  - Packets starts being processed
- Read results
  - mapi_read_result
  - mapi_get_next_pkt

MAPI example

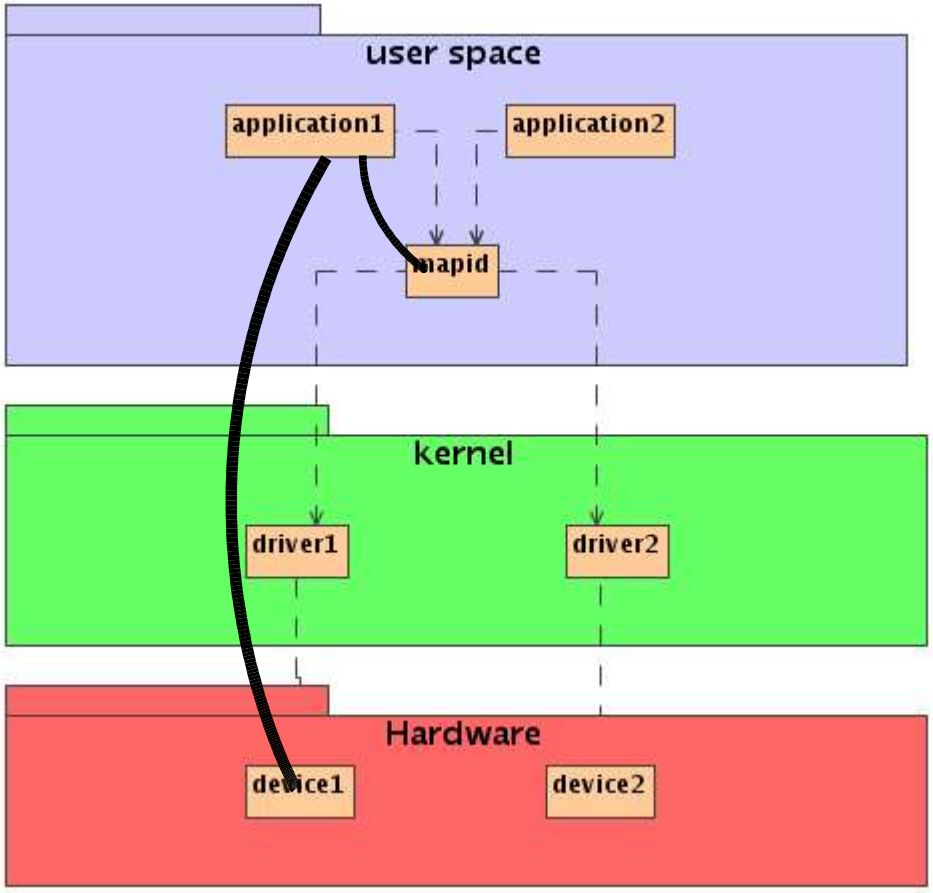
Worm detection:

```
fd=mapi_create_flow("/dev/dag0");
mapi_apply_function(fd,BPF_FILTER,"src port 1234");
ctr_id1=mapi_apply_function(fd,PKT_COUNTER);
mapi_apply_function(fd,STR_SEARCH,"pattern",100,300);
ctr_id2=mapi_apply_function(fd,PKT_COUNTER);
mapi_apply_function(fd,TO_FILE,MFF_TCPDUMP,"worm.trace",0);
mapi_connect(fd);

while(1) {
        mapi_read_results(fd,ctr_id1,&ctr_val1);
        mapi_read_results(fd,ctr_id2,&ctr_val2);

        printf("BPF match: %llu String match: %llu\n",
                ctr_val1,ctr_val2);
        sleep(10);
}
```

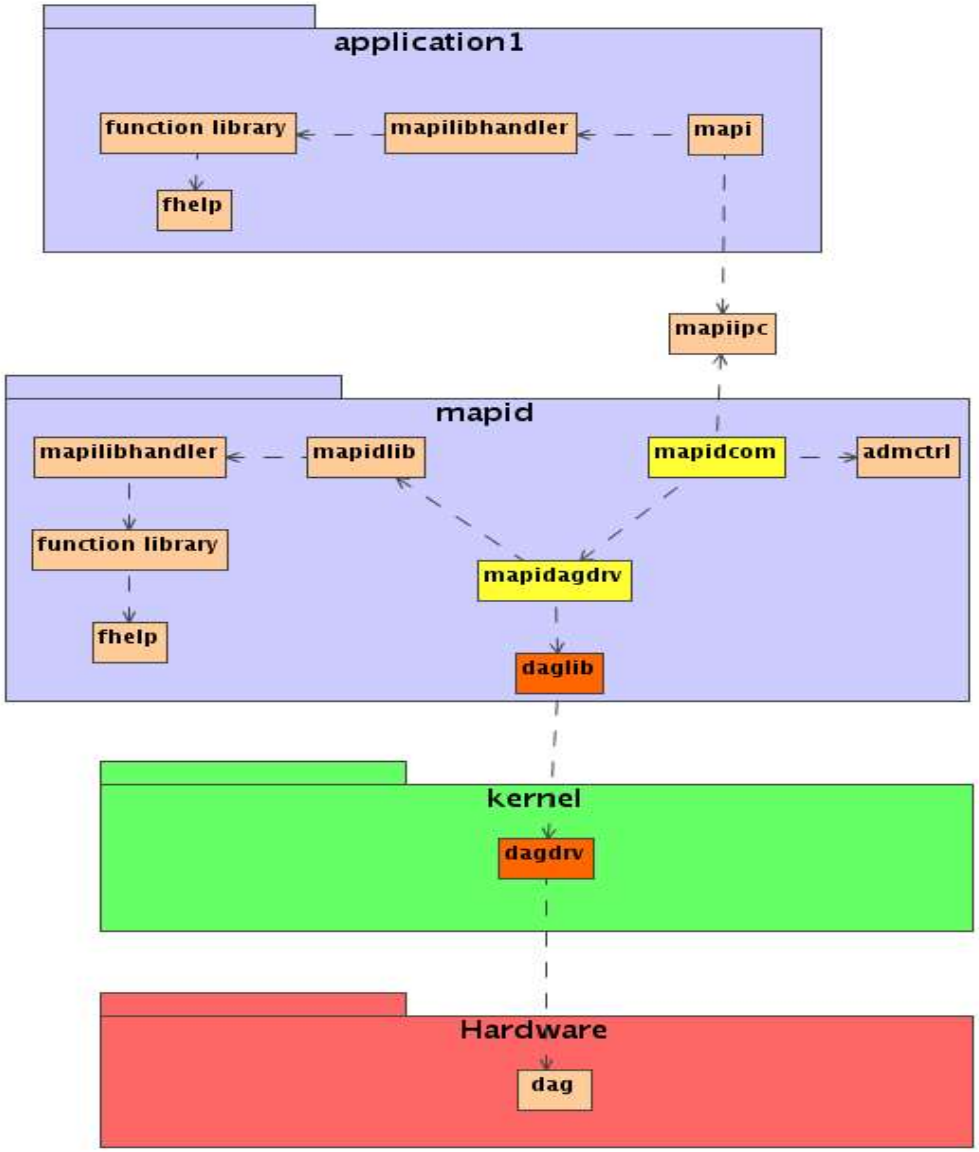
Available MAPI functions

- BPF
- Byte counter
- Cooking
- Ethereal
- Hash
- Packet counter
- Sample
- String search
- To file
- Get packet
- Netflow/IPFIX
- Consecutive packet delay
- Packet size
- Histogram
- Statistics
- Periodical results

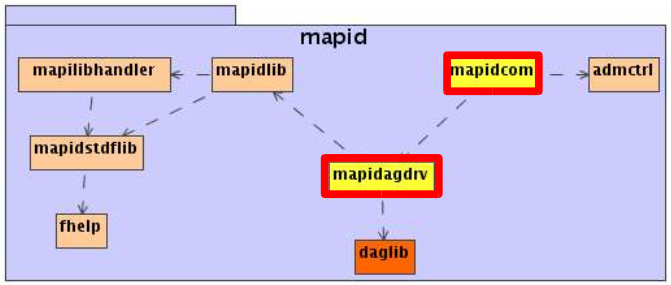
MAPI architecture overview



MAPI detailed architecture

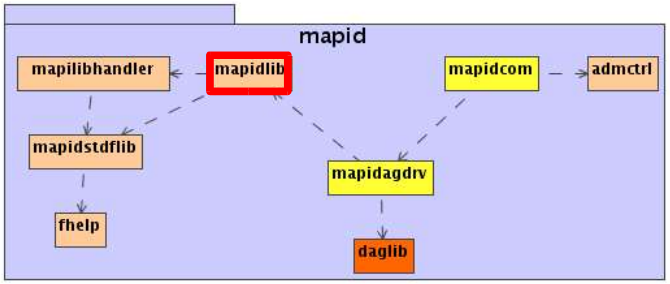


- MAPId
- Mapidcom
  - Communicates with applications through IPC
  - Based on sockets
  - Forwards received messages to correct driver
  - Only used when creating a new flow and applying functions
- MAPId driver
  - Read packets from the adapter
  - Creates new thread for processing packets
  - Similar interface as MAPI
  - Supports multiple devices



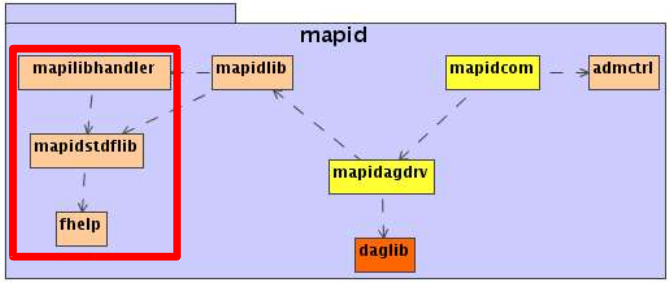
mapidlib

- Contains generic code that can be used by all drivers.
- All functions are optional.
  - Drivers decide if they want to use functions available in mapidlib or to implement optimized versions themselves.
- Contains functions for:
  - Keeping track of active flows and applied functions
  - Loading and using function libraries



Function libraries

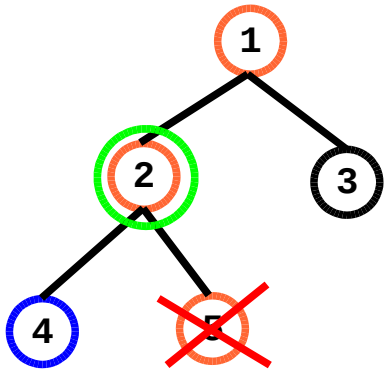
- Function libraries can be loaded/unloaded dynamically
- mapilibhandler
  - loads and manages libraries
  - finds optimal function version for a given adapter
- library
  - Contains multiple MAPI functions
  - Can be multiple versions of the same function type optimized for different adapters.
- fhelp
  - common functions that can be used by all functions to make implementation easier.



Finding the correct function

- Each supported adapter is assigned unique device OID (devOID)
- Each function defines a list of devOIDs that this function supports.
  - Supports the specific devOID and all its descendants
- Generic software functions has a devOID=1
- The most optimized function is the one with the longest devOID that matches the devOID of the adapter being used.

Finding correct function example



- Adapter devOID=1.2.4
- Function 1 devOID=1
- Function 2 devOID=1.2
- Function 3 devOID=1.2.5

Implementing a new MAPI function

- All code in one single source file
- Header file only needed if function returns complex data.
- Script automatically creates source file for libraries

- **instance**
- **init**
- process
- get_result
- reset
- change_args
- get_args
- **cleanup**

- client_init
- client_get_result
- client_cleanup

MAPI performance

- Measured number of cycles to process a packet in the using the MAPI NIC driver
- Perfctrs + PAPI
- Captured traffic @ 1000 packets/s
- Measured cycles to capture 75.000 packets
- Tests:
  - libpcap without MAPI
  - One flow with 1 empty function
  - One flow with 2 empty functions
  - One flow with 3 empty functions
  - Etc.
- Overhead: 200+40*#functions

Summary

- What is new in MAPI
  - Multi user monitoring platform
  - Transparent hardware support
  - Expressiveness
  - High level functions
  - Function libraries
  - Easier reuse of code
- http://www.ist-scampi.org