

Traffic Classification for Portable Applications with Hardware Support

Abstract—Traffic filtering and classification is needed in many monitoring applications. To process large volumes of data, we need hardware support embedded in monitoring cards. The problem is that different cards have different resources for filtering and classification.

We propose an architecture that enables utilization of available hardware resources in different monitoring cards and which can be easily extended to support future types of monitoring cards.

Consequently, monitoring applications can run transparently and efficiently in different hardware and software environments of current and future monitoring devices.

Keywords—Traffic classification, network monitoring, programmable hardware, embedded systems, middleware

I. THE PROBLEM

Traffic filtering and classification is needed in many monitoring applications. For example, to reduce or distribute the volume of traffic to multiple processes or to compute statistics about specific traffic sources and destinations.

In *filtering*, we reduce the volume of traffic, while in *classification*, we mark packets as belonging to classes, which are then treated separately, for statistics or performance reasons.

Monitoring applications often need to run in varied hardware and software environments, such as different network link speeds, link types, network cards and operating systems. Filtering and classification of large volumes of data require support of hardware devices, such as FPGA or network processor-based monitoring cards. Examples of such devices are DAG [1] cards or COMBO [2] cards.

Therefore, there are two primary issues that need to be resolved:

1. Traffic filtering and classification should be portable to various hardware and software environments with minimal effort.
2. Hardware support should be utilized to the extent possible with the used monitoring cards and application requirements. Software replacement should be used otherwise.

II. CONTRIBUTIONS

In this paper we propose a generalized architecture of traffic filtering and classification that provides the following benefits:

- Applications can run completely transparently in different hardware and software environments.
- Dynamic libraries provide hardware acceleration with various monitoring cards.
- Filtering and classification specifications use abstract data structures applicable to current and future hardware.
- The architecture is easily extendable with replaceable backends to support future hardware.

The main added value of the work described in this paper is that monitoring applications using MAPI (described in the next section) can now have their filtering and classification requirements accelerated using various monitoring cards transparently to the application.

For example, we may have application to measure distribution of network traffic into protocols. For a single wireless network, inexpensive hardware with a regular Ethernet card and libpcap library to capture and classify packets can be used. On the other hand, to monitor a high-speed trunk link, we need support of a specialized monitoring card.

III. ARCHITECTURE

We describe the ideas of the proposed architecture in the following sections and we illustrate it in examples.

A. Filtering and classification transparency

An application programmer needs to specify filtering and classification using some notation and semantics. We decided to use the commonly known BPF (Berkeley Packet Filters) used by the libpcap library [3]. This allows easy porting of third-party libpcap-based application. We comment more on this decision in section V.

For distribution of packets into classes and for transparency of monitoring cards to applica-

tions, we use the concept of replaceable dynamic libraries in MAPI [4] middleware, which is a de facto standard for development of passive monitoring applications. We added to MAPI support for transparent hardware-accelerated filtering and classification.

MAPI allows multiple applications to run concurrently over the same set of network cards. When a new application is started, its filtering and classification requirements are passed to MAPI middleware. These new requirements are added to requirements of already running applications. For each type of a monitoring card used by the new application, the dynamic library responsible for communication with that type of monitoring cards evaluates what filtering and classification could be hardware-accelerated and what need to be implemented in software. This is done based on the number and order of filtering and classification requirements and on the hardware resources in the given type of a monitoring card.

Those filtering and classification requirements that fit into hardware resources are passed to the translation system. Some structures however cannot be hardware-accelerated. For example, when packet payload bytes should be compared to specified values or when a dynamically determined offset in a packet header is referred to. In that case the translation system returns an error code and the MAPI middleware uses software implementation instead.

The architecture is illustrated in Fig. 1. The initialization and packet processing works as follows:

1. Application uses `mapi_create_flow()` function to create one or more *flows*. Each flow is initially all packets arriving to one or more specified network interfaces.
2. Application then applies BPF strings to flows using `mapi_apply_function()`, thus implementing filtering and classification (depending on further packet processing).
3. Each type of supported network cards has a dynamic library implementing filtering and classification for that card (all standard Ethernet NICs are implemented as one type, specialized monitoring cards are other types).
4. An application uses `mapi_connect()` function to connect to MAPI middleware. At this point, MAPI makes the above described decision what

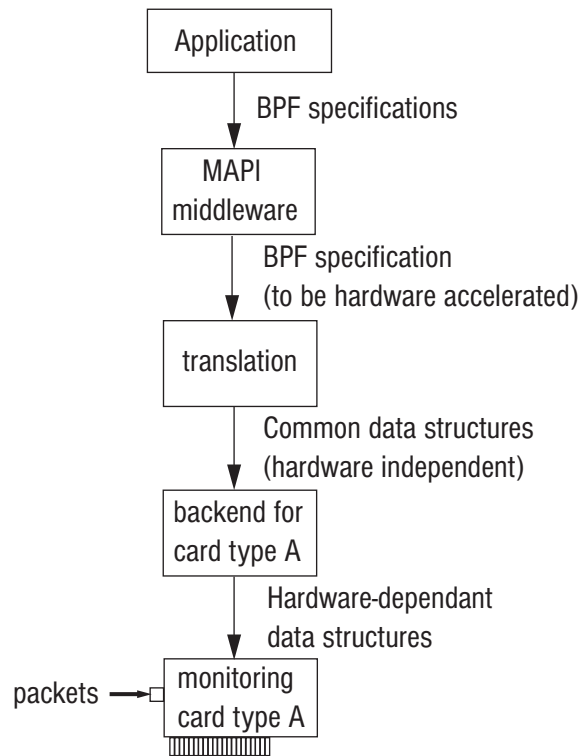


Fig. 1. Architecture of transparent and extensible packet classification

BPF filtering and classification will be offloaded to hardware and what will be done in software.

5. If hardware support is used, monitoring cards are configured to filter and classify packets.
6. As packets arrive from network cards to the MAPI middleware, they are either filtered or classified in software or no action is performed when filtering or classification was already done in hardware.
7. When the application needs results of classification, they are taken from software or hardware counters.

B. Common data structures and translation process

Libpcap library compiles BPF strings into a register-based instructions and uses interpreter to execute them for each incoming packet. This allows early decisions about packet rejection when a part of a complex condition is false without evaluating the rest of the condition.

However, the interpreter approach is slow for large volumes of packets. Moreover, the libpcap library can only filter packets. In order to classify them, individual classification conditions need to be interpreted for each packet sequentially until

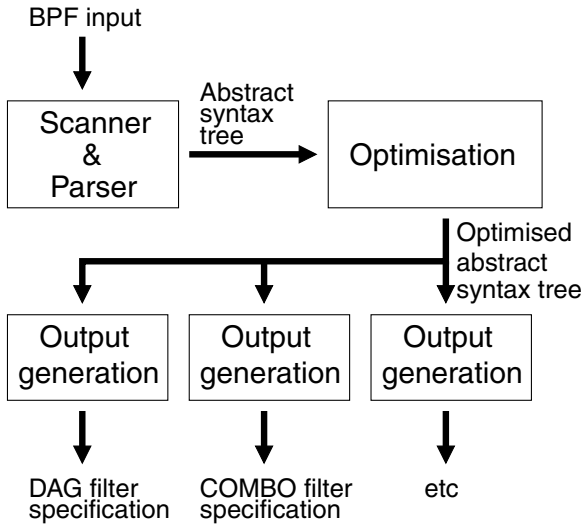


Fig. 2. Translation of classification conditions

```

expr: term
    | expr and term
    { $$ . b = new_and_node($1.b, $3.b);
      $$ . q = $3.q; }
...
other: VLAN pnum
    { $$ = gen_vlan($2);
      | VLAN
    { $$ = gen_vlan(-1); }
  
```

Fig. 3. Example grammer rule

the packet is classified.

Monitoring cards use hardware resources such as look-up tables and CAMs (Content Addressable Memory) that need a different type of filtering and classification implementation.

Therefore, we designed an abstract representation of classification conditions and the corresponding translation process keeping in mind hardware resources of monitoring cards, application transparency and extensibility for future hardware monitoring cards.

Structure of the translation process is depicted in Fig. 2. Two example translation rules are shown in Fig. 3. The input language is described by a context-free grammer using BNF (Backus-Naur-Form) notation annotated with semantic actions in C language in curly braces. Identifier \$\$ is a return value of a rule. It may represents a single variable, structure or union. Identifiers \$\$. b refer to a binary tree, \$\$. q is the direction qualifier. Identifiers \$1, \$2 and \$3 refer to the semantic values of the particular rule components.

- Syntax parser is implemented using *bison* and

Node type	Information
binary-node	AND or OR operator
link-node	link type (IP, ARP, etc.)
host-node	IP address and netmask
protocol-node	network protocol (TCP, UDP, SCTP, etc.)
port-node	port number
port-range-node	range of port numbers

TABLE I

NODE TYPES OF INTERNAL TREE REPRESENTATION

flex [5] tools.

- The abstract syntax tree uses several node types, see Table I chosen to represent entities that can be usually evaluated by embedded logic in monitoring cards.

• Optimisation removes sub-expressions that can never match and converts the tree representation into DNF (Disjunctive Normal Form) [10].

The syntax parser accepts the following keywords:

- type qualifiers – host net port portrange
- direction qualifiers – src dst
- protocol qualifiers – proto arp rarp vlan mpls ip tcp sctp udp
- operators – and && or | | ()

These qualifiers allow to create most expressions with common network protocols, which can be evaluated statically, that is using comparisons only. Such expressions can be successfully translated into configuration of look-up tables and CAMs in monitoring cards. For example, the following are statically evaluated expressions:

```

"tcp port 80"
"ip host www.google.com and \
  tcp dst port 80"
  
```

However, the following expressions includes arithmetics that requires runtime evaluation (it filters all IPv4 HTTP packets to and from port 80 that contain data, not SYN, FIN and ACK-only packets):

```

"tcp port 80 and (
  (ip[2:2] - ((ip[0]&0xf)<<2)) - \
  ((tcp[12]&0xf0)>>2) \
  ) != 0 \
)"
  
```

Possible evaluation in hardware would require some sort of arithmetic coprocessor. This is not

the case with the supported monitoring cards, therefore this expression must be implemented in software.

For example, the BPF string `tcp port 80` is translated into the following abstract syntax tree:

```
[LINK IP] AND (
  [PROTO TCP] AND (
    [PORT SRC 80] OR [PORT DST 80]
  )
)
```

However, this tree is still not suitable for generating configuration for monitoring cards. Packet matching in these cards is usually based on CAMs, either real hardware CAM devices or firmware implementation of the CAM function.

A CAM consists of set of rows that are all matched in parallel to parts of the incoming packet in parallel. Parts to match are defined by a mask associated with each row. The output is the row number that matched the input (usually the first such row if more rows matched). CAMs can be seen as a logical disjunction of rows and the parts of rows to match as a logical conjunction. This is exactly a property of DNF expressions, which can therefore be easily written into CAMs.

The above abstract syntax tree is not in the DNF due to the nested OR within an AND conjunction. Transformation into the DNF in the optimization process uses distributive law [11] to eliminate inner disjunctions.

Transformed abstract syntax tree follows:

```
([LINK IP] AND [PROTO TCP]
 AND [PORT SRC 80])
OR
([LINK IP] AND [PROTO TCP]
 AND [PORT DST 80])
```

C. Supporting different hardware and software environments

We will illustrate example implementation for two common monitoring cards, DAG cards and COMBO cards. DAG cards are available in various types for Ethernet, PoS and ATM links up to 10 Gb/s. COMBO cards are available for 1 Gb/s and 10 Gb/s Ethernet. Filtering and classification requirements that can be hardware-accelerated with the DAG and COMBO cards are summarized in Table II. If such specification is used in a monitoring application, it will be hardware-accelerated. Those filtering or classification requirements that

Card type	Resources
DAG (cards with coprocessor)	One filter condition based on common fields in IP, TCP and UDP headers.
DAG (all cards with DSM [12] support)	Up to 8 independent classification conditions based on comparison of arbitrary bytes in the first 64 bytes of an Ethernet frame. When packets need to be passed to the host PC for further software processing, one of those 8 conditions needs to be used to filter packets to be passed packets to the PC.
COMBO (with NIFIC [13] firmware)	8 selected 32-bit words in L2-L4 packet headers are compared to an 8192-line CAM then a sequence of arithmetic comparisons can be executed depending on the required speed.

TABLE II
RESOURCES FOR HARDWARE ACCELERATION

do not satisfy these conditions will be returned by the translation process as not fitting into resources of a monitoring card and will be implemented in software of MAPI middleware transparently to the application.

When a DAG card with DSM classification is used, the translation process will convert the BPF string `TCP and port 80` into the following specification for the `dsm_loader` utility, which will configure the DSM unit in the DAG card:

```
<filter>
<name>filter0</name>
<number>0</number>
<ethernet>
  <ipv4>
    <tcp>
      <source-port>
        <port>80</port>
        <mask hex="true">FFFF</mask>
      </source-port>
    </tcp>
  </ipv4>
</ethernet>
</filter>
<filter>
<name>filter1</name>
```

```

<number>1</number>
<ethernet>
  <ipv4>
    <tcp>
      <dest-port>
        <port>80</port>
        <mask hex="true">FFFF</mask>
      </dest-port>
    </tcp>
  </ipv4>
</ethernet>
</filter>

```

When a COMBO card is used, the translation process will compile the above BPF string into the following specification for the `lupgen` utility [6], which will create a configuration bitstream to be uploaded into the COMBO card:

```

:rules
{@filter_0} @if_id=h0/h0 \
  @l3_reg=h0000/h8000 @protocol=h0006 @src_port=80;
{@filter_0} @if_id=h0/h0 \
  @l3_reg=h0000/h8000 @protocol=h0006 @dst_port=80;
{@default};

```

IV. EVALUATION

In this section we present comparison of packet classification performance of the same monitoring application for the 10 Gigabit Ethernet running over i) a regular Ethernet NIC, ii) a DAG card with software classification (DSM disabled) and iii) a DAG card with hardware classification (DSM enabled). The application used one classification condition on IP and UDP headers and counted packets that passed this condition. As a regular NIC, we used Myrinet 10GE PCI-E adapter. According to our other tests, this adapter appears to have highest performance of current 10 Gb/s NICs. The Myrinet card driver was version 1.3.0. The PC used was Supermicro X7DB8 mainboard with two dual-core 3 GHz Woodcrest Xeon processors with SuSe Linux and kernel version 2.6.19. However, different number of CPU cores were used in different tests depending on how packets could be distributed into multiple cores. With an Ethernet NIC and a DAG card with DSM classification, two cores were used (one for MAPI and one for application), while for the test with a DAG card and software classification, three cores were used (two for MAPI and one for application).

The achieved throughput and associated average CPU load (over the used CPU cores) in the host PC for various packet sizes is shown in Table III. We measured performance for Ethernet

Frame size	Myricom 10GE NIC		DAG without DSM		DAG with DSM	
	Througput	CPU load	Througput	CPU load	Througput	CPU load
1518	8100	76.19	10000	0.06	10000	0.06
1280	6770	73.03	10000	0.10	10000	0.06
1024	5440	68.56	10000	0.13	10000	0.06
512	3050	68.93	10000	13.87	10000	0.06
256	1620	74.26	10000	33.56	10000	0.06
128	1130	75.56	10000	54.97	10000	0.06
64	710	77.82	10000	78.94	10000	0.06

TABLE III
PERFORMANCE OF PACKET CLASSIFICATION -
THROUGHPUT IN MB/S AND CPU LOAD IN %

frame sizes recommended in [7]. We can see that a regular Ethernet NIC did not achieve full line rate. With the DAG card, we can classify packets at the full line rate. For software classification this was achieved due to much more effective packet transfer from the network to the memory in the host PC. However, the CPU load was high and therefore little further packet processing would be possible. With hardware classification, the CPU was used only to retrieve packet counters from the DAG card and therefore it remained almost fully available for further packet processing. We did not have 10 Gb/s COMBO card available for testing.

V. RELATED WORK

Other common languages for filtering or classification description include FPL-3 and Netfilter. FPL-3 [8] goes beyond packet header comparison by adding payload searching and it is based on a data flow-driven model for distribution of packet processing into parallel engines specialized for particular protocols or tasks. The implementation is targeted for network processor hardware. Netfilter [9] is a language and packet filtering framework for Linux kernels that is particularly designed for firewall and NAT applications.

We decided to use BPF, because it is the most commonly used language in passive monitoring applications that are our targeted use area. BPF also covers the range of expressions that can be implemented with the commonly available monitoring cards. We use FPGA because it can be

used for a broader range of applications than network processors, which are a more specialized platform. Also, boards with high-speed networks processors are more expensive than boards with fast FPGAs.

VI. CONCLUSION

We have proposed an architecture that allows monitoring applications to utilize as much as possible of hardware resources in various monitoring cards to accelerate packet filtering and classification. This allows applications to run transparently over various current and future hardware. Application programmers do not need to care about details of different monitoring cards (the programmers extending our environment for future monitoring cards of course need to do that). Applications can run in software only when monitoring moderate volumes of traffic, such as in wireless networks and in hardware-software environment for high-speed monitoring.

In our future work we plan to explore possibilities of hardware acceleration of more monitoring functions, such as payload searching or more complex statistics. This should be possible with some of the emerging high-speed (10 Gb/s) FPGA-based cards with open hardware interfaces, which allow users to write their own firmware.

REFERENCES

- [1] DAG cards, Endace corporation, www.endace.com.
- [2] COMBO cards, Liberouter project, www.liberouter.org.
- [3] Libpcap library, <http://www.tcpdump.org>.
- [4] MAPI - Monitoring Application Programmable Interface. <http://mapi.uninett.no>.
- [5] Bison and Flex tools, <http://www.gnu.org/software/bison>, <http://www.gnu.org/software/flex>.
- [6] LUP configuration tool, Liberouter project, https://www.liberouter.org/wiki/index.php/LUP_configuration_tool.
- [7] S. Bradner, J. McQuaid. *Benchmarking Methodology for Interconnect Devices*, RFC2544, March 1999.
- [8] Mihai Cristea, Willem de Bruijn, Herbert Bos. *FPL-3: Towards Language Support for Distributed Packet Processing*, IFIP Networking'05, 2005, Waterloo, Canada.
- [9] The netfilter.org project, www.netfilter.org.
- [10] Mendelson, E. *Introduction to Mathematical Logic*, 4th ed. London, Chapman & Hall, p. 30, 1997.
- [11] Distributive law from elementary algebra <http://en.wikipedia.org/wiki/Distributivity>.
- [12] Data Stream Management API, Endace Corporation, <http://www.endace.com>.
- [13] NIFIC firmware, Liberouter project, www.liberouter.org/nific.php.