



Monitoring and Privacy



Herbert Bos

Vrije Universiteit Amsterdam

herbertb_AT_cs.vu.nl

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

Herbert Bos, VU, <http://www.cs.vu.nl/~herbertb>



Lobster infrastructure

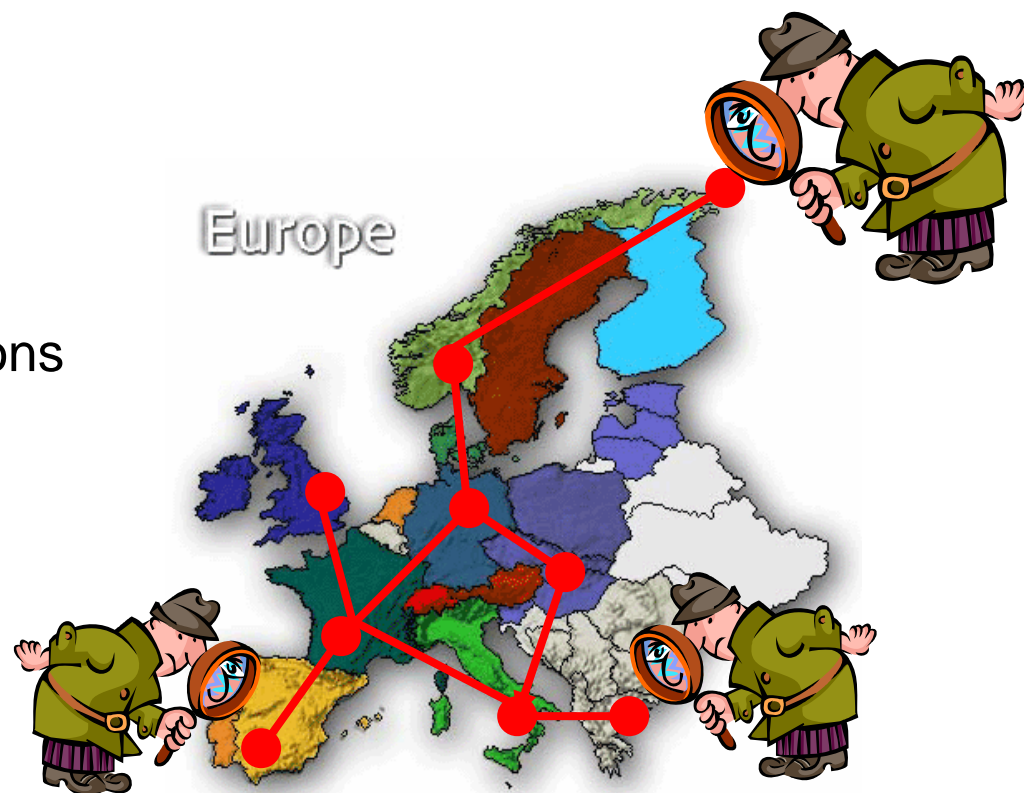


Information Society
Technologies

lobster An IST Project

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

- idea:
 - distributed monitoring
 - convenient access to network data
 - share data
 - within organisations
 - between organisations





MAPI – The Monitoring API

- core concept: *flow*
 - subset of network traffic corresponding to arbitrary user criteria
 - desired subset is expressed *functionally*
 - start with all incoming traffic
 - successively apply *functions* to it (filters, samplers, hashing, transcoding, etc.)

```
int fd = mapi_create_flow ("host1:/dev/eth0", "host2:/dev/eth1");
```

```
int funcid = mapi_apply_function (fd, "PKT_COUNTER");
```

```
mapi_connect (fd);
```

```
while (1) {
```

```
    resptr = mapi_read_results (fd, funcid, 0);
```

```
    sleep(1);
```

```
}
```

```
mapi_close (fd);
```





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: }
```





Privacy



Information Society
Technologies

lobster An IST Project

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

- a shared monitoring infrastructure?
→ what about **privacy**?!



Before talking about privacy ask the following: would *you* share information?





Responses vary...

- No, **absolutely not**. I will not reveal any information
 - I will not reveal even the load of my network
- Maybe I can share **some general** information
 - what traffic goes in and out of my network
- I am willing to share the **headers** of my packets
 - IP addresses anonymized
 - just like NLANP
- I would like to share **some** information
 - all information with my local administrators
 - all information with associated researchers
 - some information with the rest of the world
- Yes, I am willing to share **everything!**
 - information wants to be free!

Lobster can provide all of the above



How does it work?



Information Society
Technologies

lobster An IST Project

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

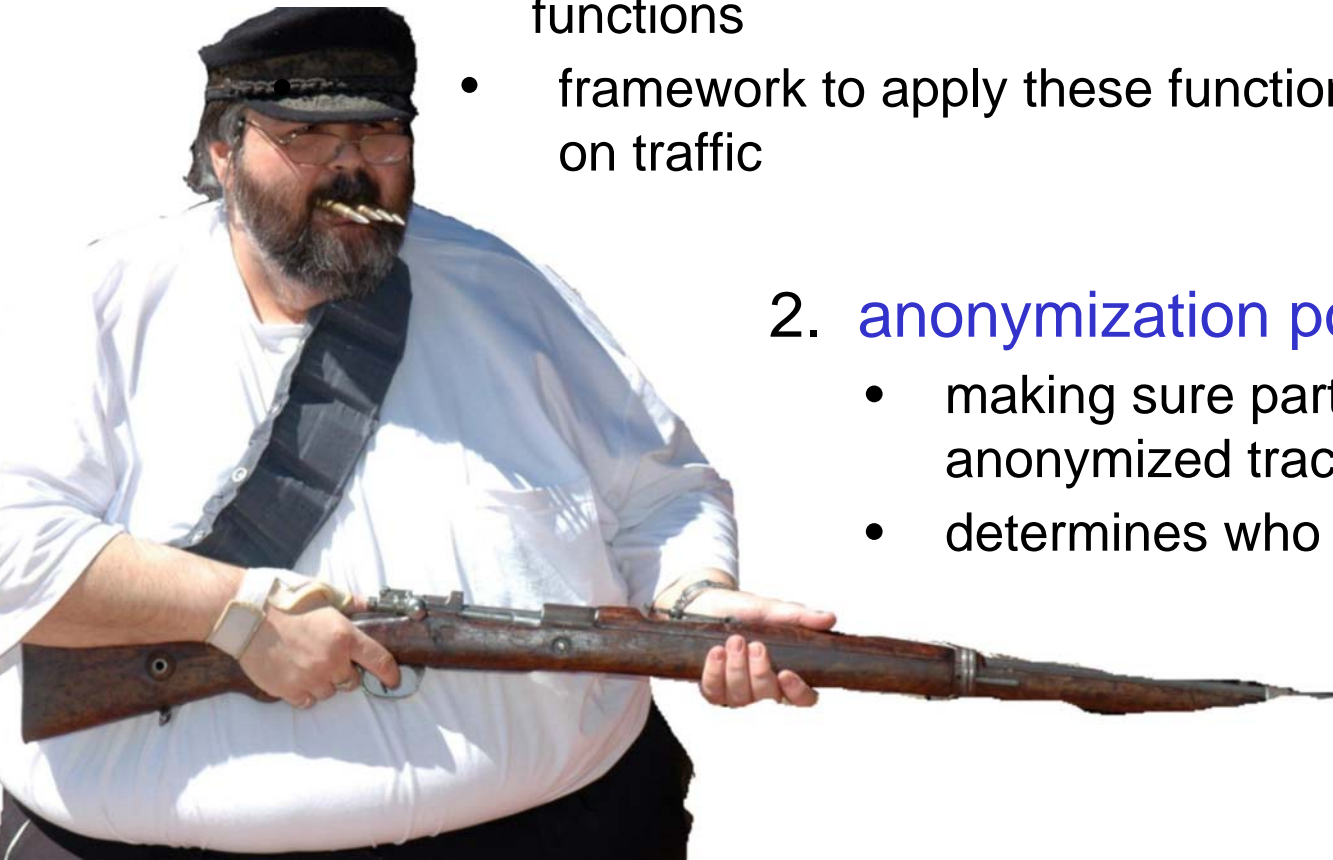
1. applying anonymisation functions

- functionality to strip/hash payloads, zero addresses, etc.
 - ➔ library of possible anonymisation functions
- framework to apply these functions on traffic



2. anonymization policy enforcement

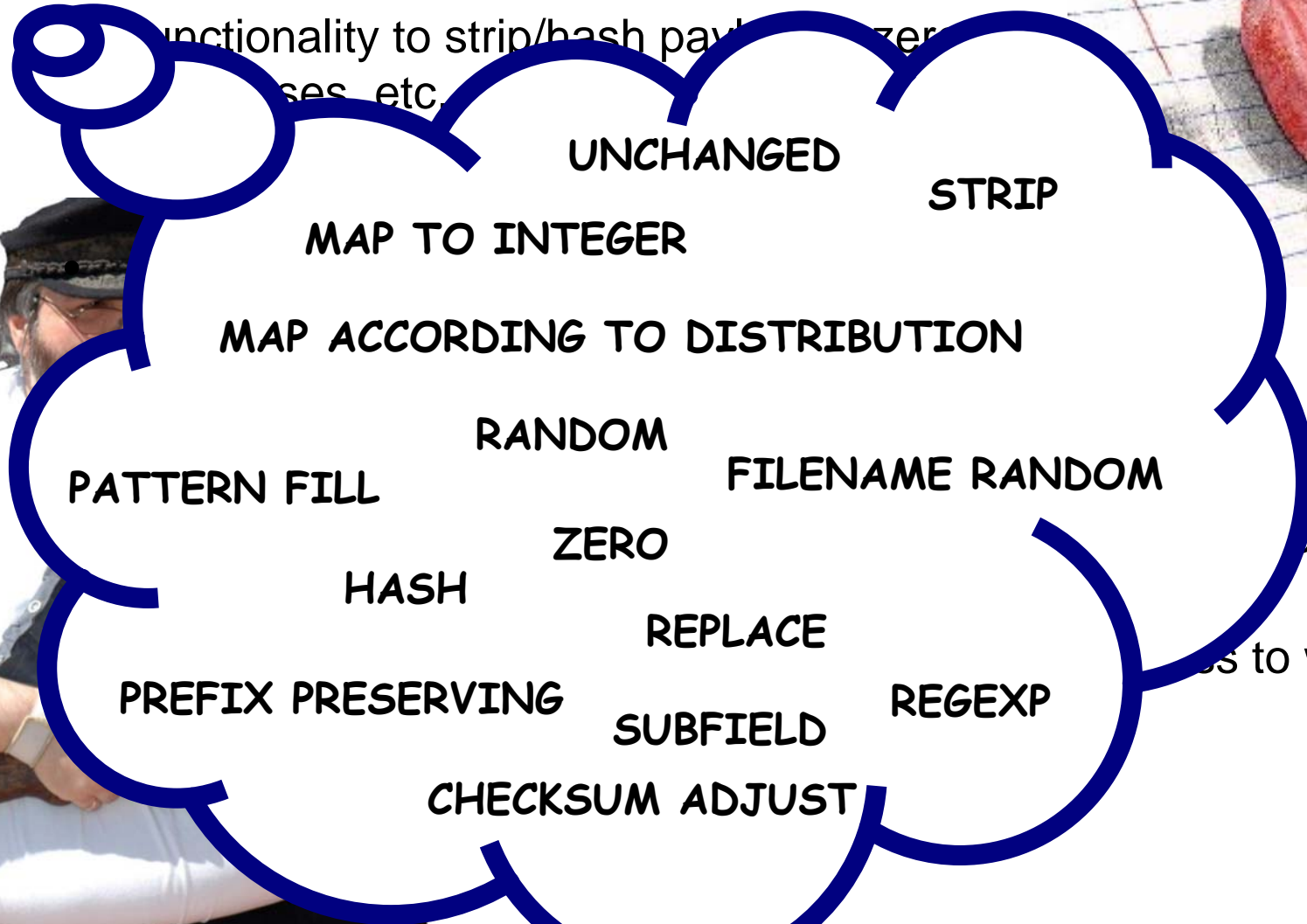
- making sure parties only get properly anonymized traces
- determines who gets access to what





How does it work?

1. applying anonymisation functions



ment
perly
ss to what



How does it work?

1. applying anonymisation functions

functionality to strip/hash payload, zero out headers, etc.



Ruler:

- special-purpose, high-level language
- for packet rewriting in general
- and anonymization in particular

What is so nice about it?

- easy to specify needs
- fast, efficient
- translates to low-level HW



ment
perly
ss to what



Information Society
Technologies

Authorisation





different policies



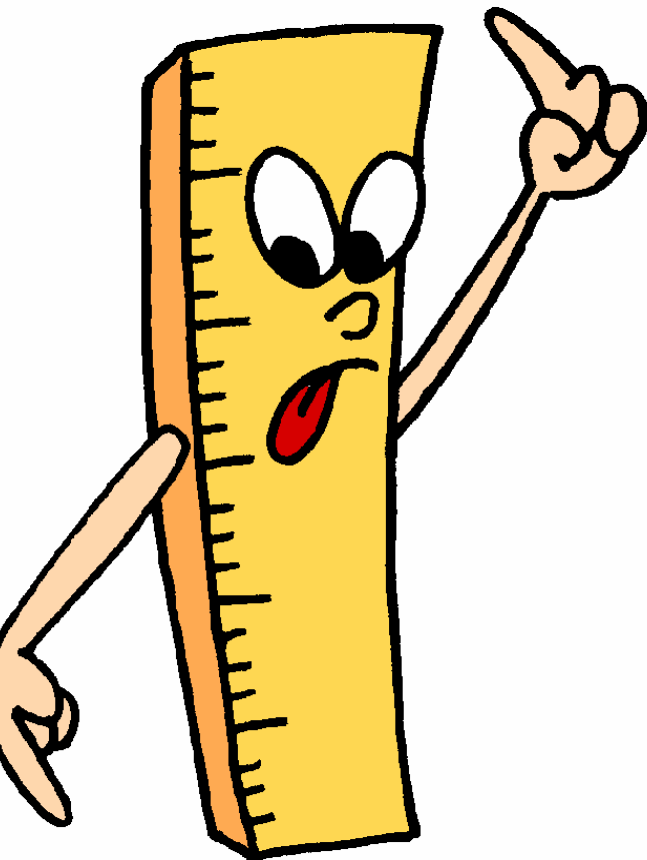
Information Society
Technologies

lobster An IST Project

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

- different anonymization rules for different users
- credentials determine what sort of access is allowed





a simple rule-based language

- Generic sanitization
- Efficient (DFA-based) implementation
- Design to allow for (future) formal reasoning
- Understands common network protocols
- Extensible
- Implemented on CPU, NPU, HW



- **Patterns** and **filters** consisting of **rules**

pattern:

```
pattern udpHeader: (  
    source: byte #2  
    dest: byte #2  
    len: byte #2  
    checksum: byte #2)
```

filter:

```
include layouts.rli
```

```
filter simple
```

```
    eh:Ethernet_header iph:IPv4_header *  
    => eh iph with [src=0, dest=0];
```





Matching language



Information Society
Technologies

lobster An IST Project

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

- Fixed number of bytes, fixed value

42#1

5

0x800#2

"url"

- Fixed number of bytes, arbitrary value

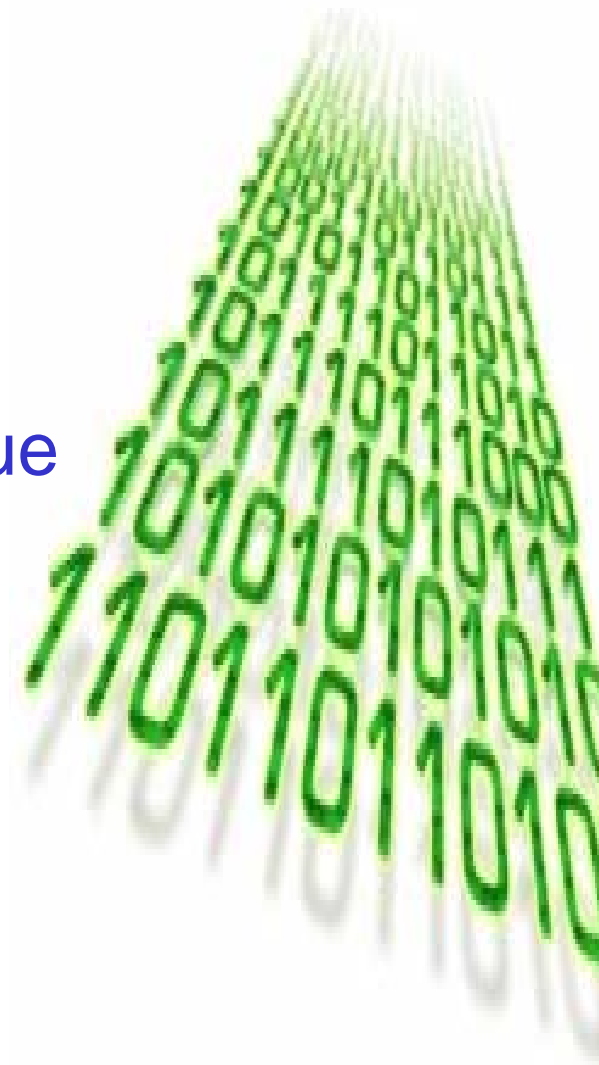
byte#2

byte#1

byte

- Arbitrary number of bytes and value

*





Matching language: bit patterns



Information Society
Technologies

lobster An IST Project

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

- Surrounded by { }
- Fixed number of bits, fixed value

{ 42#7 }

{ 0 }

{ 0x800#16 }

- Fixed number of bits,
arbitrary value

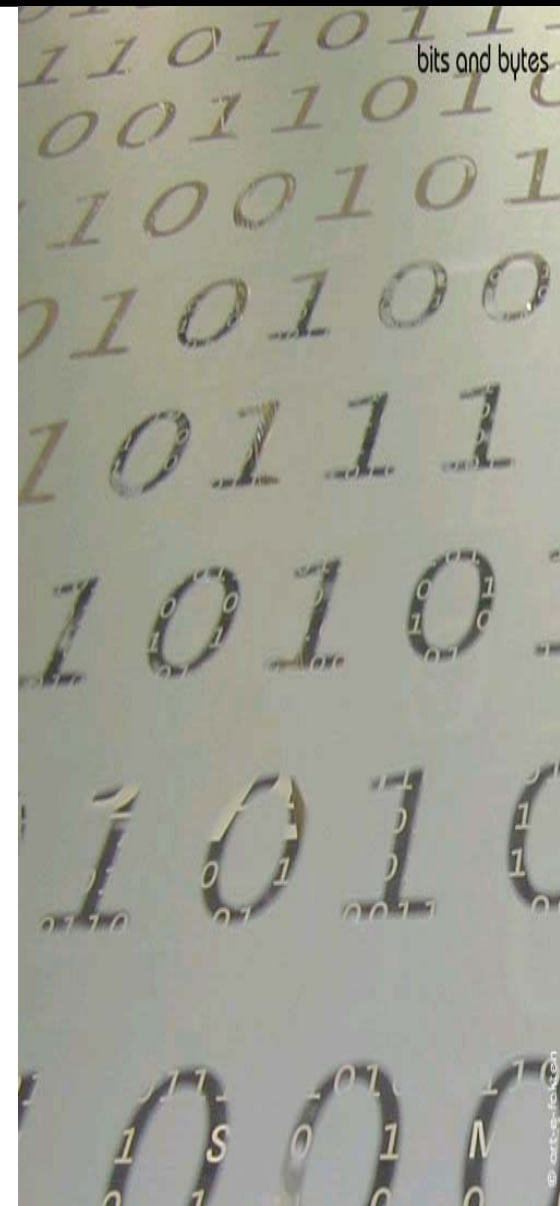
{ bit#2 }

{ bit#1 }

{ bit }

- Combination

{ bit#1 42#7 }





Matching language: labels

- Patterns can be labeled

`source: byte #6`

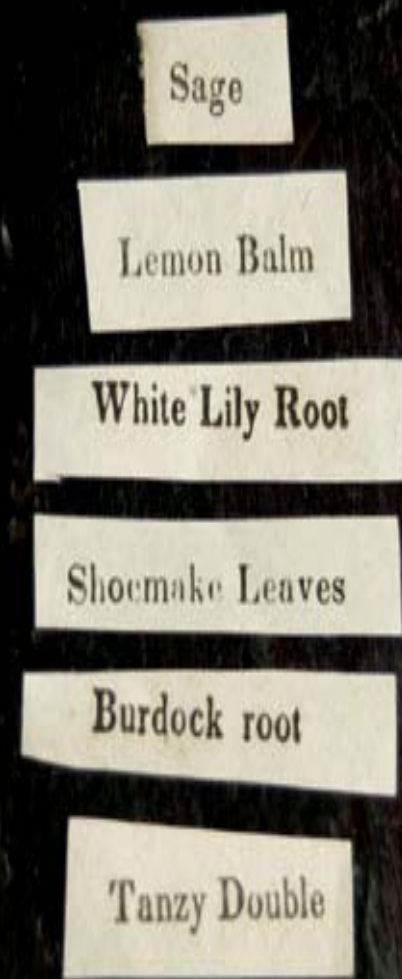
`dest: byte #6`

`payload: *`

- Patterns can be grouped with brackets

`ether: (s:byte#6 d:byte#6 p:byte#2)`

- Note the nested labels: `ether` refers to 14 bytes, `ether.s` refers to 6 bytes





Matching language: patterns



Information Society
Technologies

lobster An IST Project

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

- Patterns can be defined and named separately
pattern ether: (
 s:byte#6 d:byte#6 p:byte#2
)
- Allows re-use of common patterns





include files



Information Society
Technologies

lobster An IST Project

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

- Definitions can be included from another file
include "layouts.rli"
- Normally used for pattern definitions
(but can be used for filters too)
- `layouts.rli` defines many common header layouts:
ethernet, IP, TCP, UDP, etc.





use of labels



Information Society
Technologies

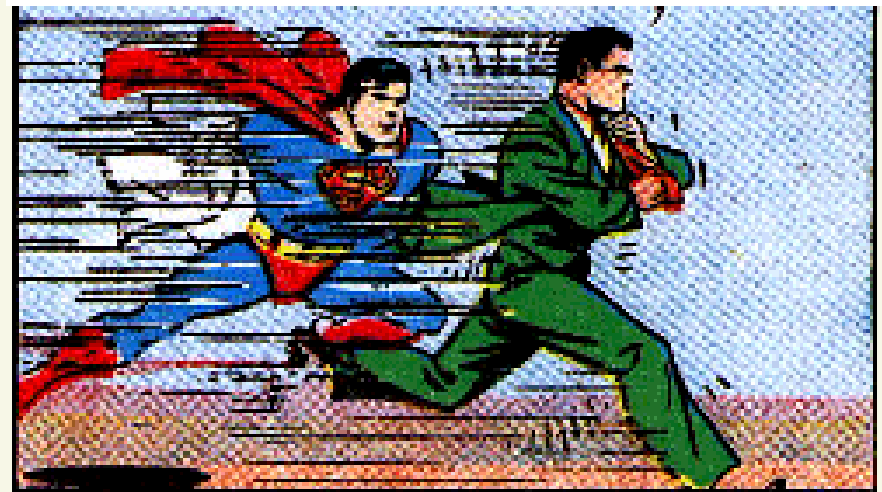
- Pattern names can now be used instead of the pattern they stand for

```
ether * "warez" *
```

- Labels can be used in rewrite actions:

```
a:byte#2 b:* => b a
```

- The trick is to do it fast





The *with* construct



Information Society
Technologies

lobster

An IST Project

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

- The `with` construct replaces named parts of a pattern with other patterns

```
ether with [p=0x0806#2]
```

- Restrictions: replacement pattern may not be longer or shorter than the original.

```
ether with [p=42] // WRONG (1 byte)
```





Rule actions



Information Society
Technologies

lobster An IST Project

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

- Action is one of:
 - reject
 - accept, accept 5
 - result pattern





Tagged Deterministic Finite Automata



Information Society
Technologies

lobster

An IST Project

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

- We compile to a **Tagged Deterministic Finite Automaton (TDFA)**
- Tagged because we need the position of sub-patterns.
- **States:**
 - **stop** (we know what action to take)
 - **inspect** (look at a byte, branch)
 - **tag** (register positions in the input)
 - **jump** (skip irrelevant input bytes)
 - **memory inspect** (look at previous input)





- Ruler is extensible: it may call external functions

```
filter zap_url
```

```
    head:(Ethernet_header IPv4_header * "GET ") url:* tail: (0x0D 0x0A *)
```

```
    => head @hash(url) tail ;
```

- allows us to use libraries of previously developed anonymisation functions
- in Lobster we have integrated Ruler with MAPI
 - a Ruler program can be applied to packets of arbitrary flows
 - Ruler may use MAPI's default anonymisation functions
- also: implementations directly on
 - pcap
 - DAG API





Information Society
Technologies

Performance





Used filters

name	action	description
aa	accept/reject	match anything
ha	accept/reject	match on header
hi	in-place	match on header
hr	rewrite	match on header
pa	accept/reject	match on entire packet
pi	in-place	match on entire packet
pr	rewrite	match on entire packet
la	accept/reject	large FSM, match on entire packet
all	in-place	Select IP, ARP traffic, zero addr., drop payload
www	in-place	Select & anonymise TCP traffic to port 80
snort	accept/reject	Select packets matching Snort `backdoor' rules





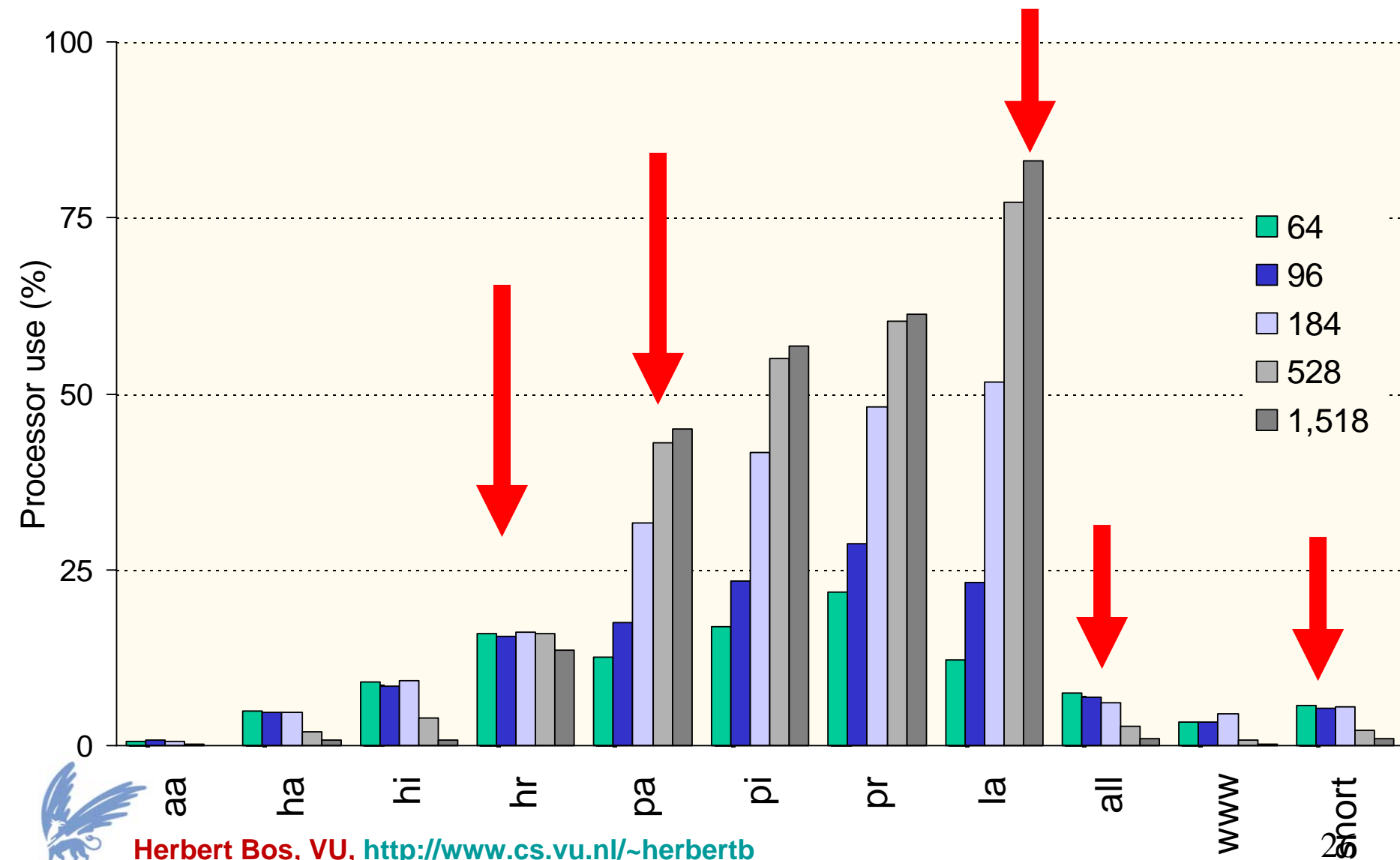
Synthetic traffic



Information Society
Technologies

lobster An IST Project

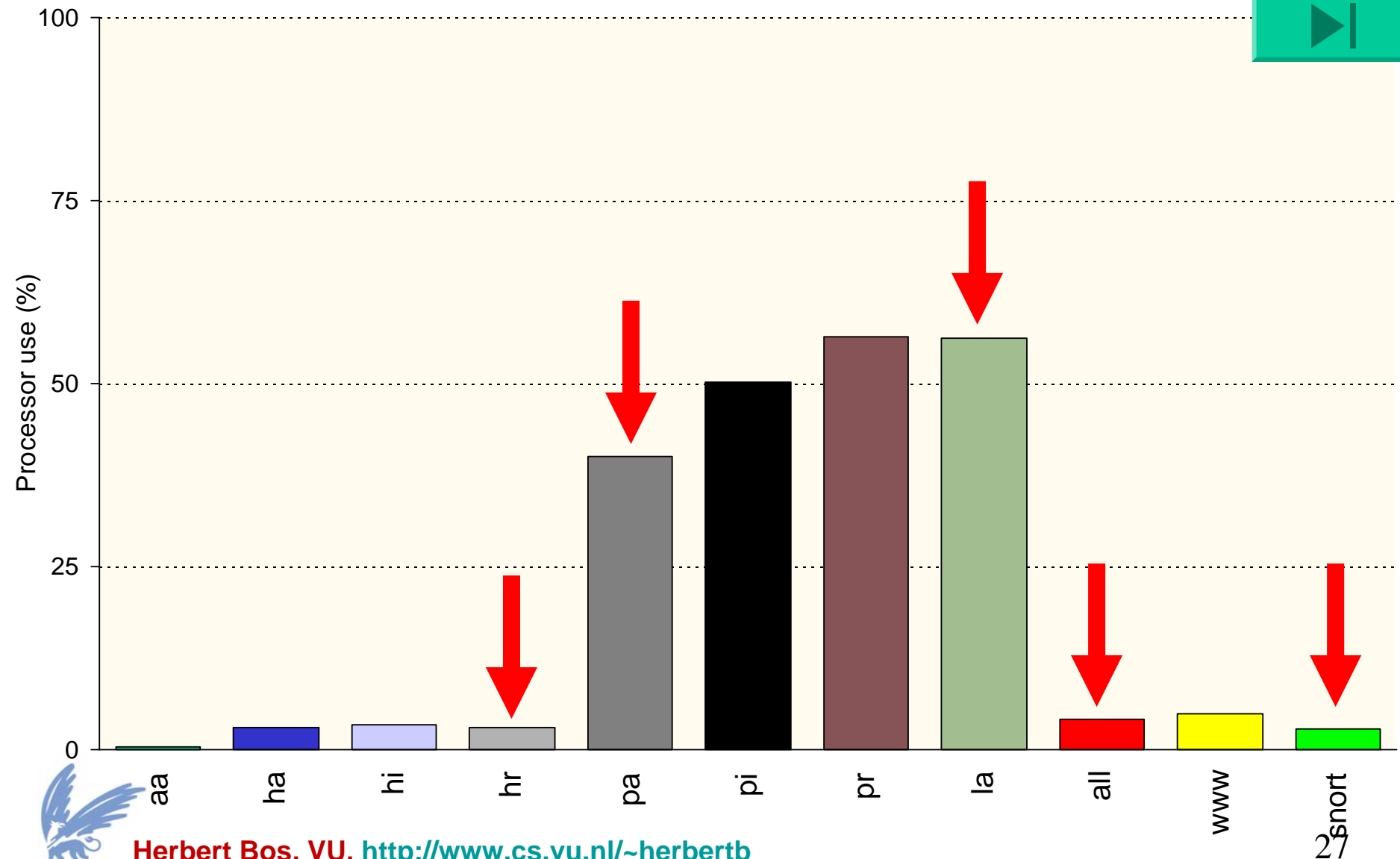
<http://www.ist-lobster.org/>



Herbert Bos, VU, <http://www.cs.vu.nl/~herbertb>



Real traffic





applications



Information Society
Technologies

lobster An IST Project

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

- besides anonymisation we also applied Ruler to intrusion detection
- developed a snort2ruler compiler
 - translates majority of Snort rules automatically
 - some rules need manual help
 - some rules cannot be translated *as is*
 - implemented in parallel on an Intel IXP2400 network processor





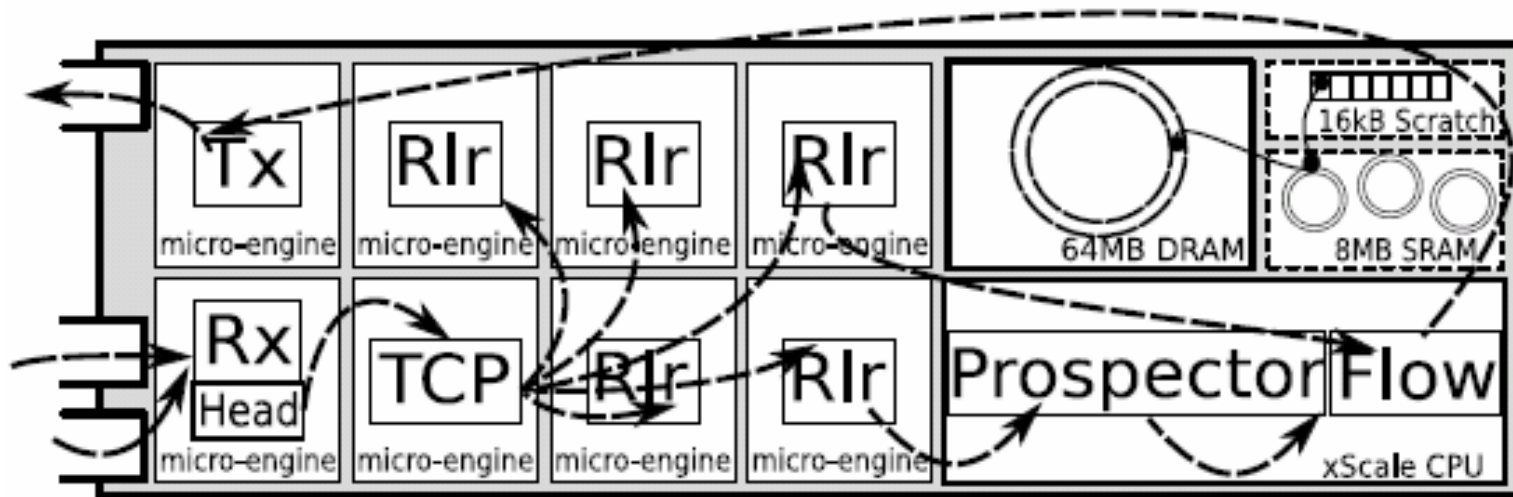
SafeCard: Ruler for IDS/IPS



Information Society
Technologies

lobster An IST Project

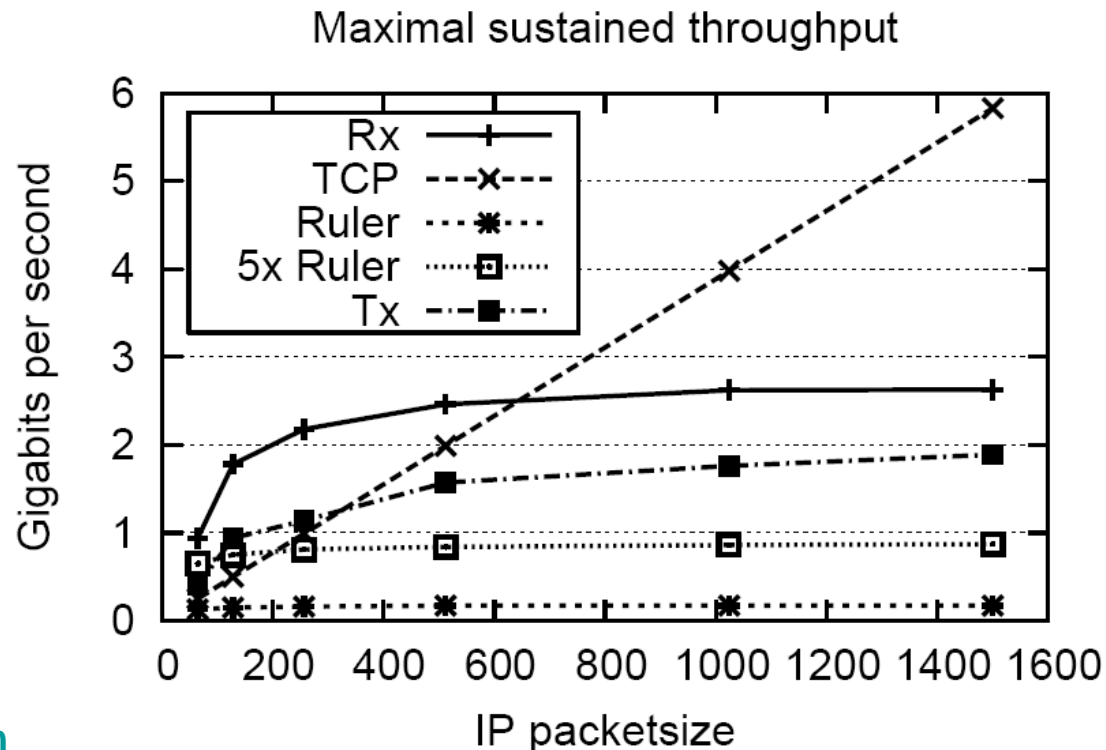
<http://www.ist-lobster.org/>





Preliminary results

- able to keep up with gigabit scanning (only small number of rules tested)
- large number of rules:
 - large number of states
 - *long* compilation
- Gigabit rates: even on IXP with real traffic





conclusions

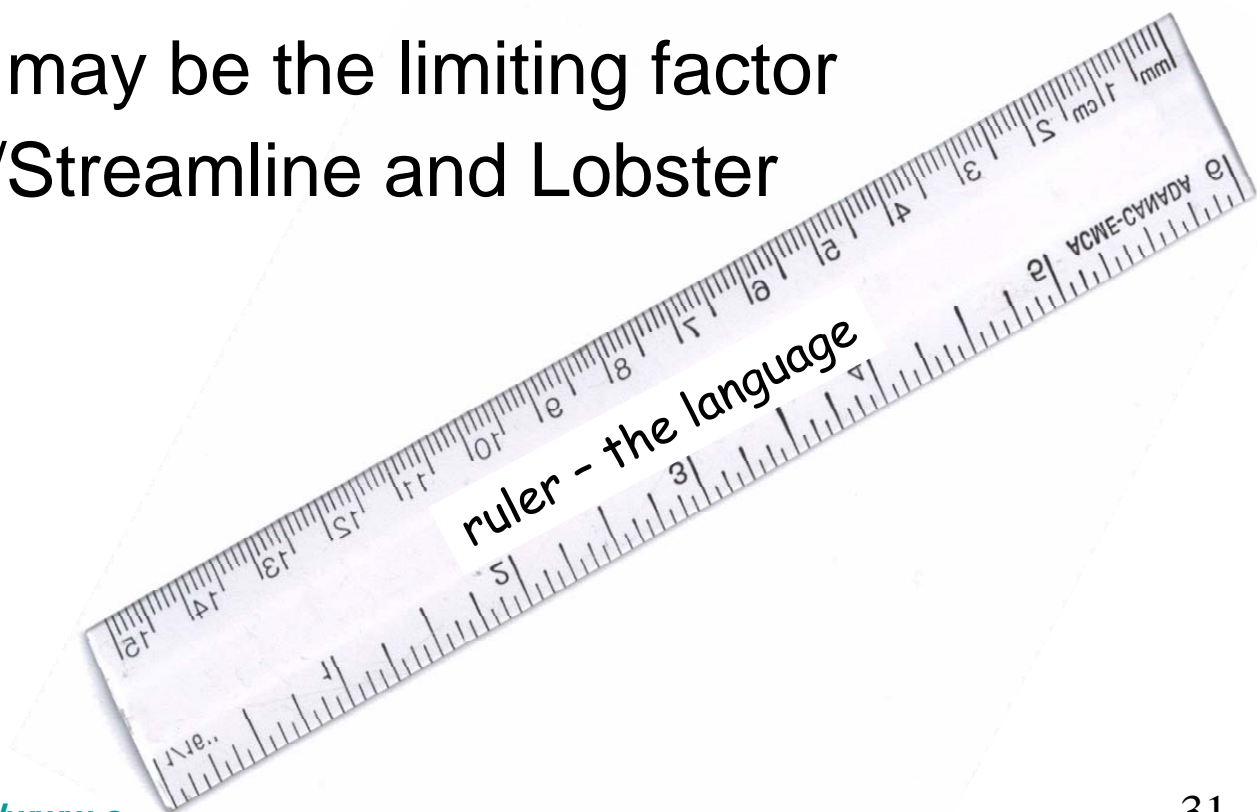


Information Society
Technologies

lobster An IST Project

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

- useful
- efficient
- multiple application domains
- state space may be the limiting factor
- fits in FFPPF/Streamline and Lobster



Herbert Bos, VU, <http://www.cs.>



availability

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





filters



Information Society
Technologies

The `hr` filter reverses the order of the packet headers and payload. This requires all bytes in the packet to be copied to an output buffer.

```
filter hr
  eh: Ethernet_IPv4 ih: IPv4
  uh: UDP with [dest=0x7AAA~2]
  d: *
  =>
  d uh ih eh ;
```





filters



Information Society
Technologies

lobster

An IST Project

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

```
filter pa [accept_reject]
  eh:Ethernet_IPv4 ih:IPv4 uh:UDP
  * "REJECT" *
  =>
  reject;
  * => accept;
```





```
filter la [accept_reject]
  eh:Ethernet_IPv4 ih:IPv4 uh:UDP
  * (0x00 byte#8 "z") *
  =>
  reject;
  eh:Ethernet_IPv4 ih:IPv4 uh:UDP
  * (0xff byte#8 0x11) *
  =>
  reject;
  eh:Ethernet_IPv4 ih:IPv4 uh:UDP
  * ("R" byte#4 "T") *
  =>
  reject;
  * => accept;
```





```
filter all [in_place]
  eh: Ethernet_IPv4 ih: IPv4 d: *
  =>
  eh with [e_dest=0#6,e_src=0#6]
  ih with [src=0~4,dest=0~4];

  eh: Ethernet_ARP ah: ARP d: *
  =>
  eh with [e_dest=0#6,e_src=0#6]
  ah with [sender_ha=0#6,sender_ip=0#4,
          target_ha=0#6,target_ip=0#4];
```





Information Society
Technologies

lobster

An IST Project

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

EXAMPLES



Herbert Bos, VU, <http://www.cs.vu.nl/~herbertb>



1

```
include layouts.rli
```

```
filter two_rules
```

```
eh: Ethernet_header iph: IPv4_header payload:(* "/bin/sh" *)
```

```
=> reject ;
```

```
eh: Ethernet_header iph: IPv4_header payload:*
```

```
=> eh iph with [src=0, dest=0] ;
```

2

```
include layouts.rli
```

```
filter rewrite
```

```
eh:Ethernet_header iph:IPv4_header with [ proto=17] udph:udpHeader *
```

```
=> iph.src iph.dest udph.source udph.dest udph.len ;
```





3

```
include layouts.rli
filter zap_url
    head:(Ethernet_header IPv4_header * "GET ") url:* tail: (0x0D 0x0A *)
=> head "XXX" tail;
```

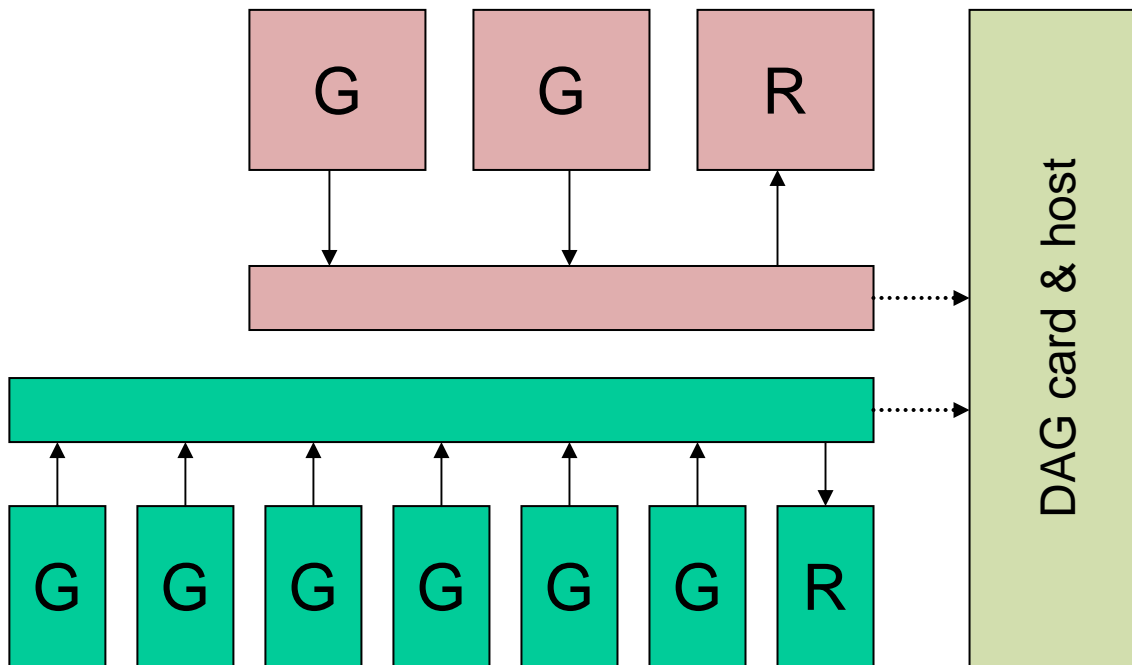
4

```
include layouts.rli
filter zap_url
    head:(Ethernet_header IPv4_header * "GET ") url:* tail: (0x0D 0x0A *)
=> head @hash(url) tail ;
```





Measurement setup



- 2 times 1Gb link
- Port mirroring

G = generator

R = receiver

