Deep Application Traffic Inspection real-time traffic analysis up to 20Gbps

EuroBSDcon 2013

Fabrizio Invernizzi
fabrizio.invernizzi@telecomitalia.it

TELECOM ITALIA

mPlane – an Intelligent Measurement Plane for Future Network and Application Management

Grant Agreement n. 318627

mPlane

SEVENTH FRAMEWORK PROGRAMME

TELECOM ITALIA

The Mplane project

- mPlane is an FP7 Integrated Project
  - Started in November 2012, 3 years project
  - 11.2+ M€ cost – 7.2 M€ EC funding
  - 16 partners (8 industrial, 8 research)

- Goal: design and demonstration of an "intelligent measurement plane for the Internet"
  - mPlane is about large scale network measurements,
  - and intelligent analysis for troubleshooting support
  - embedding measurement into the Internet as an additional capability

Agenda

- **Introduction**
- **Architecture**
- **Lab tests**
- **Real traffic**

DATI 3.0

INTRODUCTION

Introduction

- **Typical use of DPIs in ISP networks**
  - User/Service profiling
  - Lawful interception
  - Policy Enforcement (network nodes decongestion)
  - Security
  - **Traffic statistical analysis** ← We are interested only in this
- **Commercial solutions available for ISPs**
  - **Cisco, Sandvine, Allot, Procera, ...**

Why build a new DPI

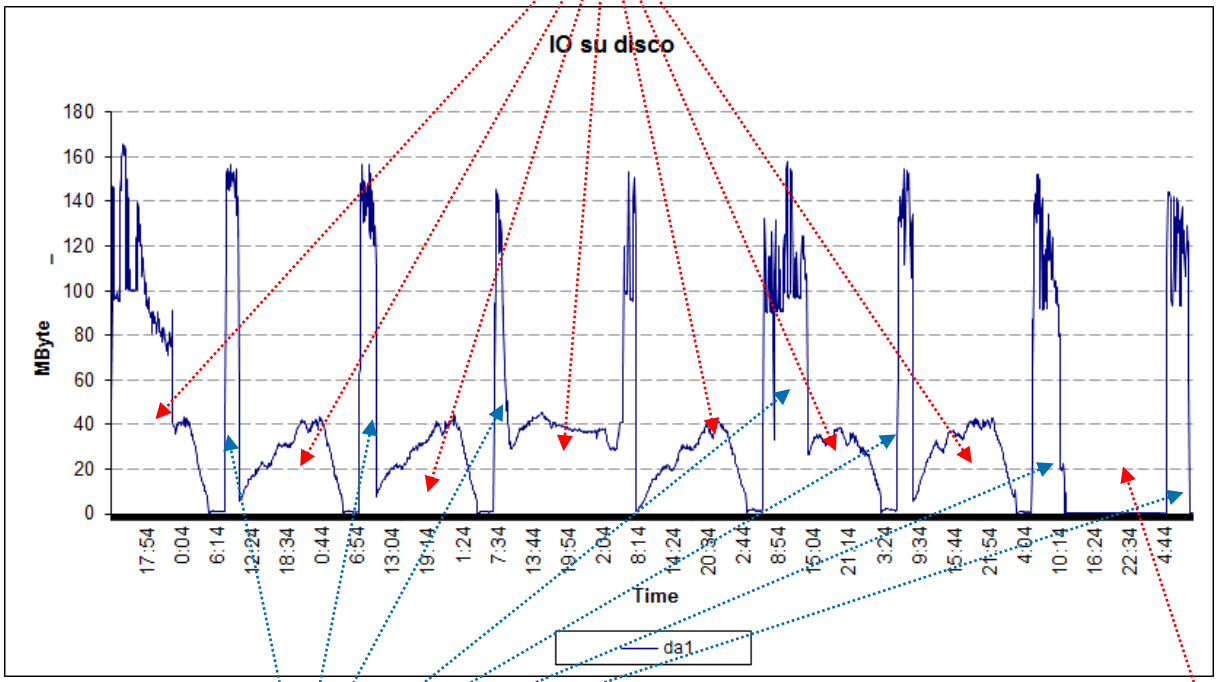
- **High Flexibility**
  - It is not (always) simple build complex, very specific traffic analysis on commercial solutions in "short" time
  - External data correlation (RIR db, Routing informations, OTT API, ...)
- **Specific (on-demand) analysis**
  - Targeted to specific traffic flows
  - Rebuild service logic
  - Reduce development time required for a new analysis

Previous versions lesson learned

- **Previous versions**
  - Kernel (netgraph) based
  - Batch processing (once a day work on the daily interesting traffic)
  - Multiple temporary files
- **Limits**
  - Kernel traffic handlings not efficient enough for high speed interfaces (~500Kpkts hard limit)
  - Batch processing uses only a limited portion of system resources
  - High disk I/O

Previous versions lesson learned – disk I/O

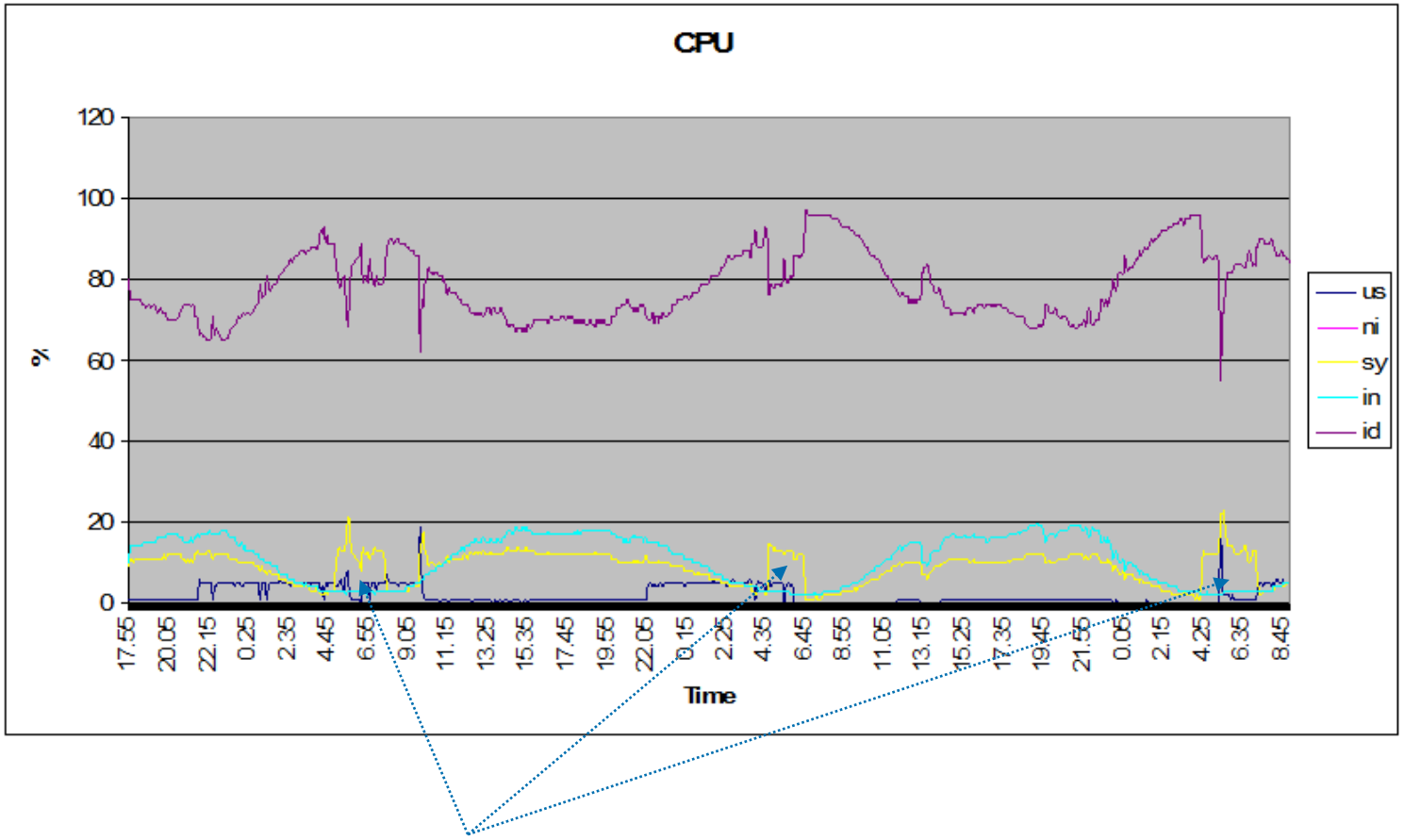
Daily work (Sqlite indexes)



Batch activities

Sqlite indexes disabled

Previous versions lesson learned – system load



Batch activities

DATI 3.0 – GOALS (1/2)

- Use general purpose hardware
  - No hardware accelerated traffic filter/disseption
  - Off-the-shelf server hardware
- Analyse traffic "on thy fly"
  - Reduce disks I/O
  - Take best advantage of system potentiality
  - No temp files to be managed
- Traffic throughput
  - 2 x 10 Gbe
  - Analyze at least 1% of the traffic

DATI 3.0 – GOALS (2/2)

- **Analysis programmability**
  - **No specific language (C, perl, javascript ...)**
  - **Multiple analysis on the same traffic should be possible**
  - **No specific knowledge of low level details needed for the developer**
  - **KISS approach**
- **Use FreeBSD**
  - **WE KNOWN IT**
  - **Solid and stable**
  - **BSD license**
  - **Accelerated network driver available (Netmap)**
  - **Everything is where it should be (even after a new major version release)**

DATI 3.0

ARCHITECTURE

Overall architecture

In memory Database

- I/O reduction
- Efficent, scalable

Programmability

- Languange independent programmability

Publish subscribe

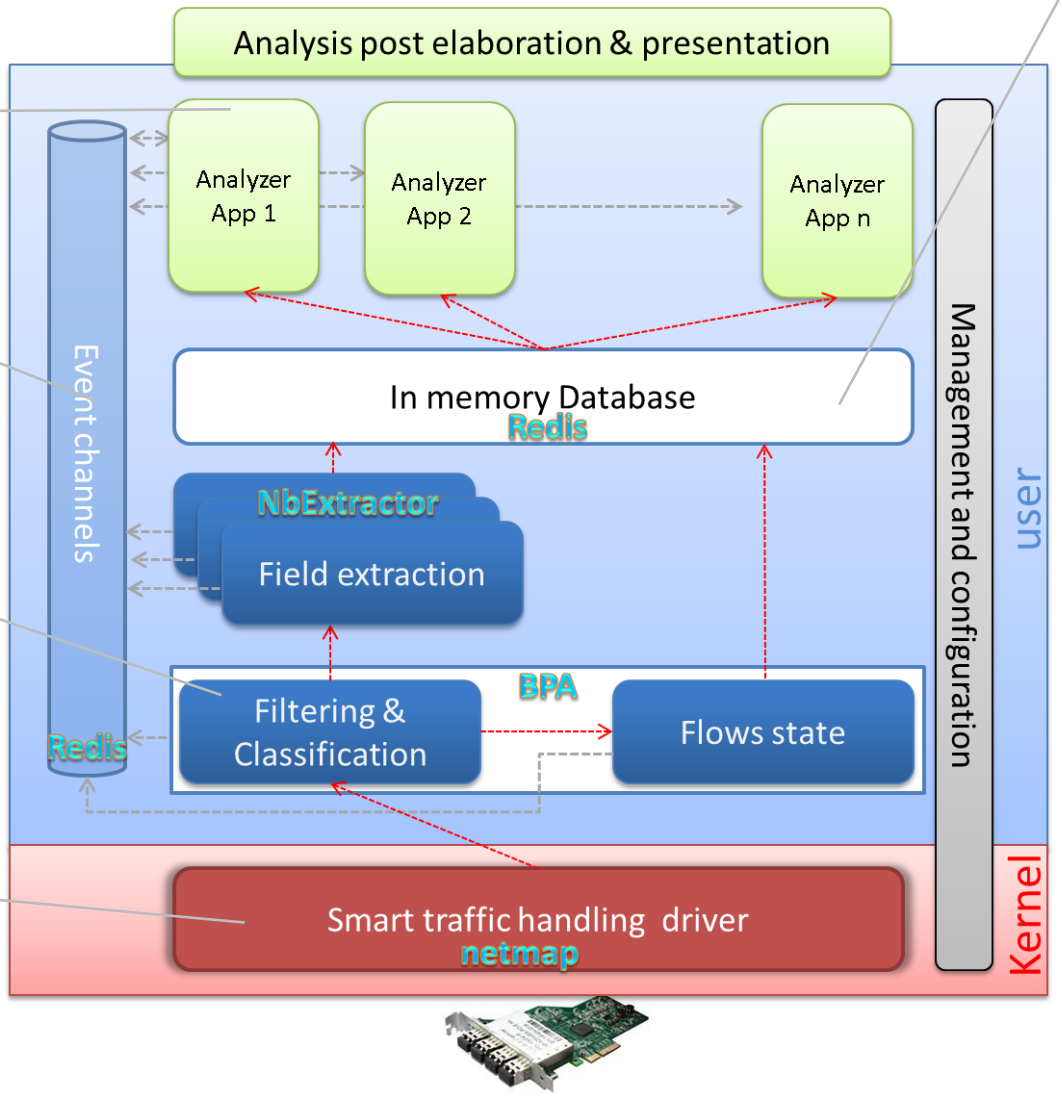
- Event driver inter module comunication

BPA

- BPF classificator
- Copy of traffic of interest
- (Efficient) HASH based multi-thread load distribution
- Flows state

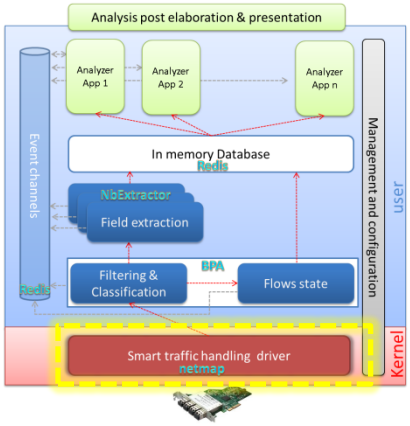
NETMAP

- Pointers to driver rings positions
- Coding best practises

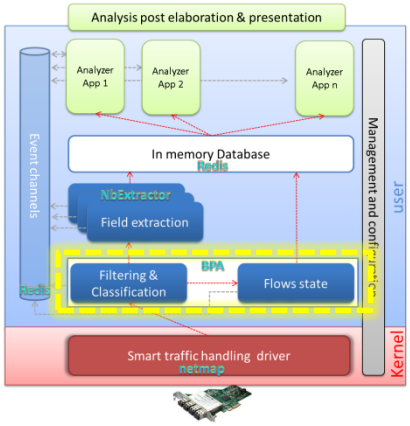


NETMAP

- ***See Luigi's talk***
  - *http://2013.eurobsdcon.org/eurobsdcon-2013/talks/#LuigiRizzo*

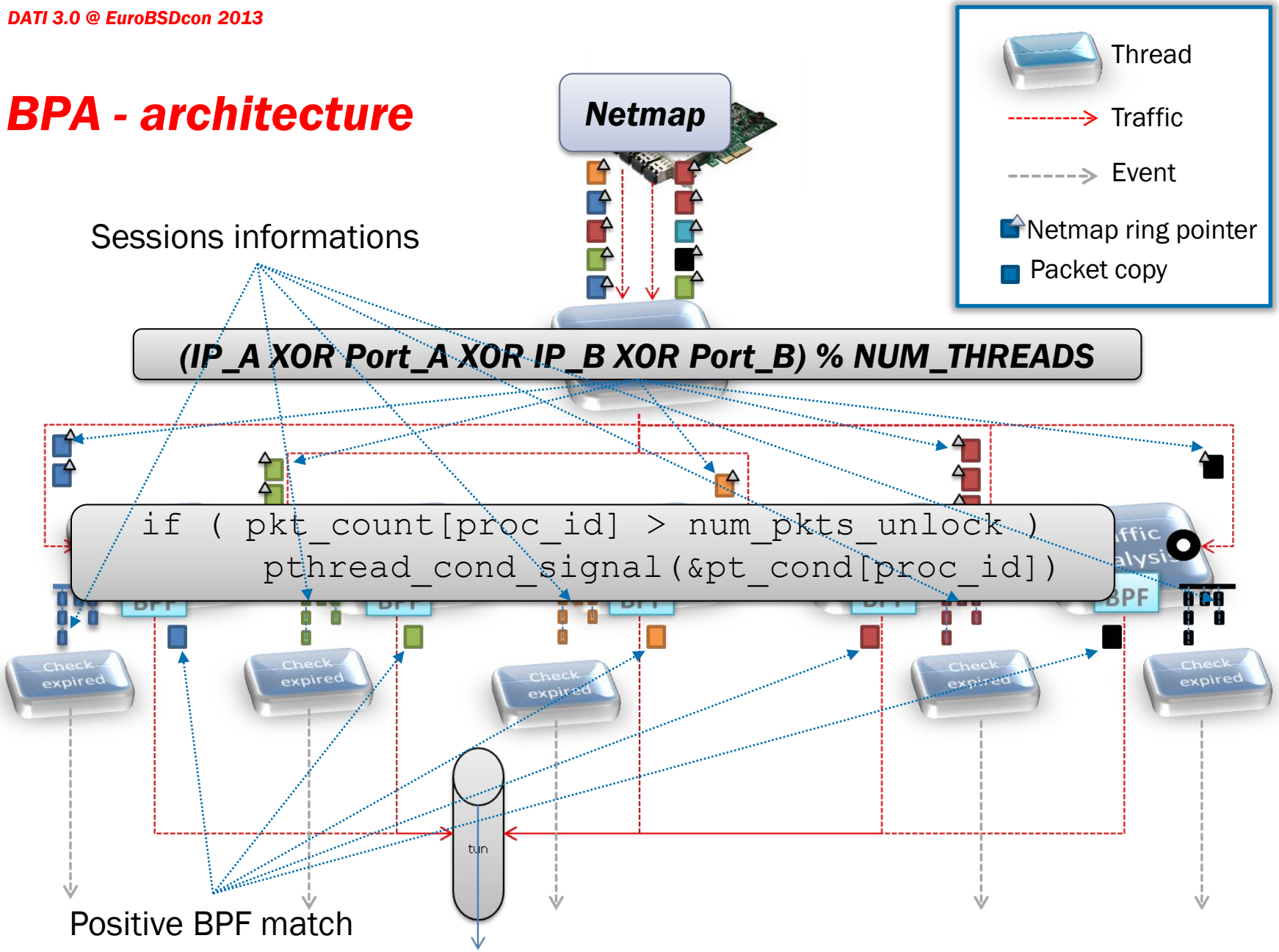


BPA



- ***Multithreaded user space application***
- ***Interface with Netmap***
  - ***Read traffic from Netmap rings***
- ***Filtering and Classification***
  - ***BPF static signature***
    - ***On match send the packet to a pool of tun interfaces (round robin)***
  - ***Flows (5-tuple) in memory***
- ***Flow management***
  - ***Creation: packets matches a BPF signature and flow is not present***
  - ***Update: any packets matching the identifying 5-tuple***
  - ***Removal: no packets with the identifying 5-tuple seen in PKT_TTL seconds (30s)***
    - ***On removal send a message on a specific redis channel with collected flow info***

BPA - architecture



BPA – sessions management

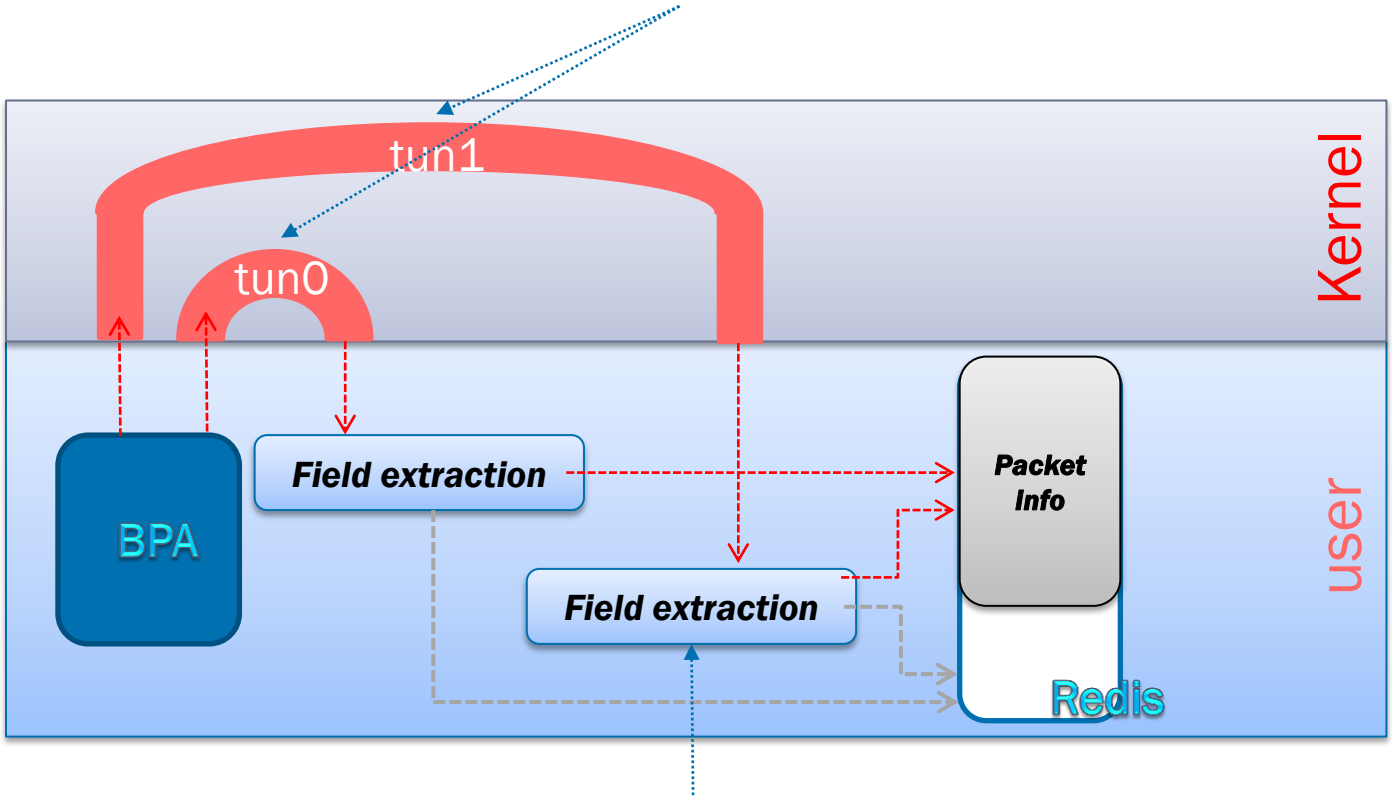
- **Stored in NUM_TAB pre-allocated simple linked lists (TABS)**
  - **NUM_TAB = 60 M (~1Gb memory)**
  - **Slots taken from pre-allocated memory for each thread (~400Mb memory)**
- **Use the hash of the packet to select the list**
  - `tab_id = packet_hash%NUM_TABS;`
- **Check expired thread**
  - **One for each elaboration thread**
  - **Continuously check all session and mark a flag in expired ones. Send a message on redis channel**
  - **If it finds less than EXPIRED_THRESOLD (100) stops for 1 second**
  - **Expired session will be removed by the elaboration thread and memory returned to the pool**

BPA – Flows

- **Flow identification**
  - Protocol [TCP,UDP], IP source, IP destination, Port source, Port destination
- **Flow information**
  - 5-tuple, Creation timestamp
  - Packets, Volume, Last match
  - Expired
- **Multiple BPF match in a flow**
  - Example: HTTP pipelining
  - Only one flow information is kept
  - All matching packets will be sent to the corresponding tun for elaboration
  - All interested analyzer will receive the redis message

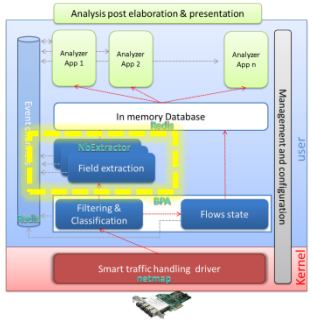
Traffic from BPA to field extraction

BPA can load balance the matching packets over multiple tuns



Any application capable of reading from tun interfaces

Field extraction

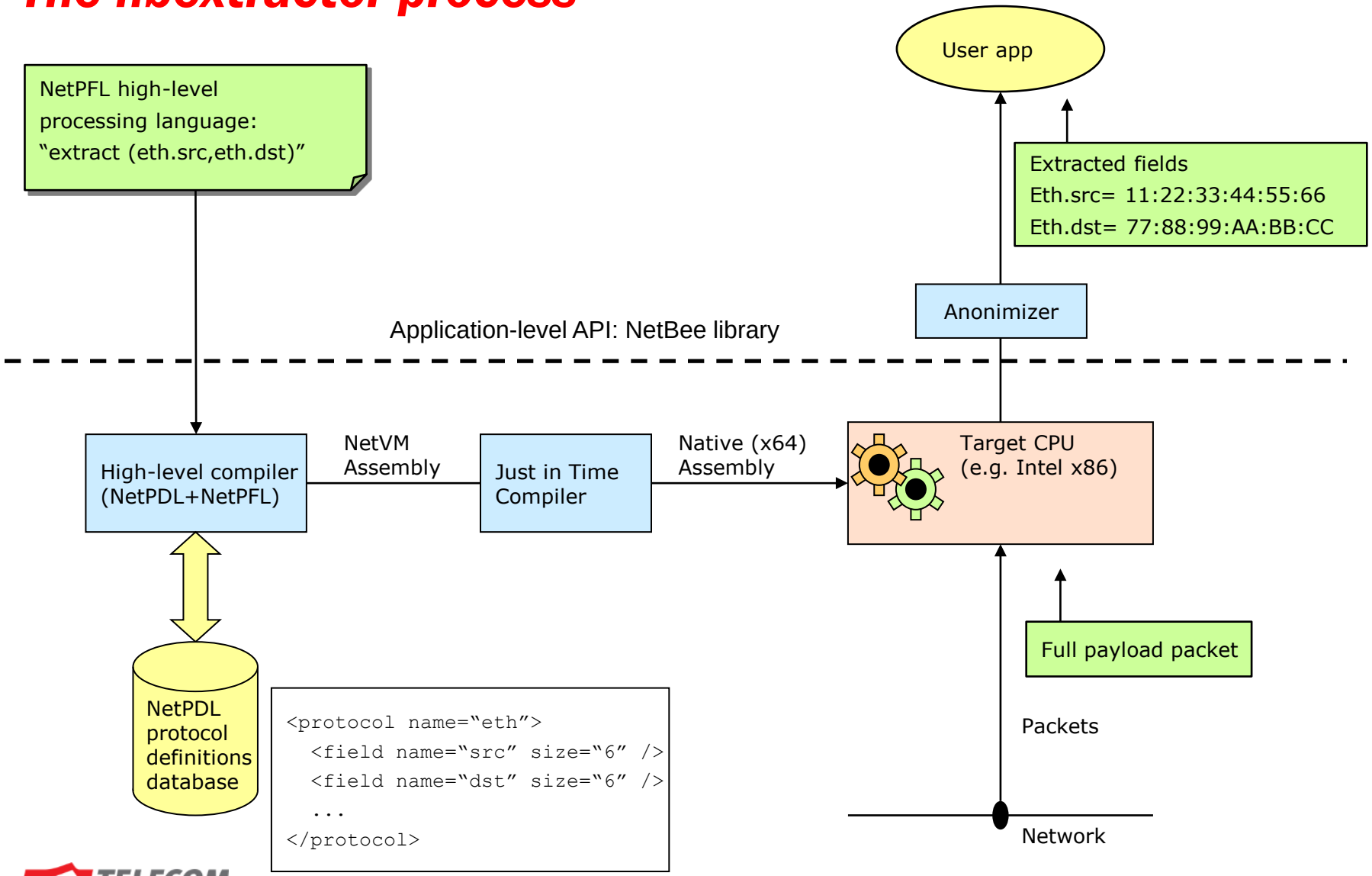


- ***Receives traffic from configured tun interfaces***
- ***Extract desired traffic fields from packets***
- ***Store information in redis***
  - ***Gets a unique id from redis (INCRBY)***
  - `HMSET <key> Timestamp ts field1 value1 … fieldn valuen`
  - ***Publish the availability of the information on a specific redis channel***
- ***Currently using NBEXTRACTOR***

Field extraction - nbextractor

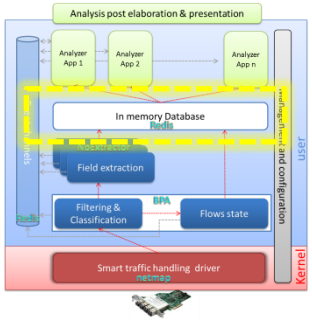
- **Deployed by Politecnico di Torino**
  - http://www.nbee.org/doku.php
- **Based on library for generic packet processing (netPDL)**
  - **XML description of packets format**
    - **Fixed position fields**
    - **Delimited fields**
    - **Regex based fields (performance warning!)**
- **Just In time compiler available**
    - **Boost performance**
    - **Freebsd @64 expected for 1Q2014**

The nbextractor process




```
<protocol name="eth">
  <field name="src" size="6" />
  <field name="dst" size="6" />
  ...
</protocol>
```

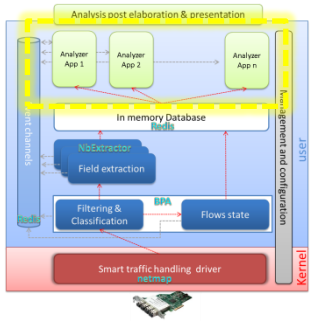
REDIS



- ***Extracted information storage***
  - ***Packets fields***
  - ***Aging of data using policy_TTL***
  - ***Effective memory usage***
  - ***Performance***
  - ***Clustering (when available)***

- ***PUB/SUB system***
  - ***All analyzers interested in an "application" subscribe a specific channel***
  - ***All analyzer interested in an application sessions information subscribe to a specific channel***

Analyzer example (nodeJS)




```
var redis = require("redis");

redisMessages = redis.createClient('/tmp/redis.sock');

redisSessionInfo = redis.createClient('/tmp/redis.sock');

redisHash = redis.createClient('/tmp/redis.sock');


redisMessages.subscribe(configuration.protocolChannel);
redisSessionInfo.subscribe(configuration.BPASessionChannel);

var sessions = 0;

var requests  = 0;


setInterval(publishStats , configuration.publishStatsPeriod);


redisSessionInfo.on("message", function (channel, message) {

        sessions = (sessions *1) + 1;

}
```

Analyzer example (nodeJS)

```
redisMessages.on("message", function (channel, message) {
        requests = (requests *1)+1;
        redisHash.hgetall(message, function (err, obj) {
        if (!obj){
                console.log("Error reading from REDIS DB");
                return;
        }
        console.log(obj.http_referer);
}

function publishStats(){
        console.log("Requests:"+requests+" Sessions:"+sessions);
        requests = 0;    sessions = 0;
}
```

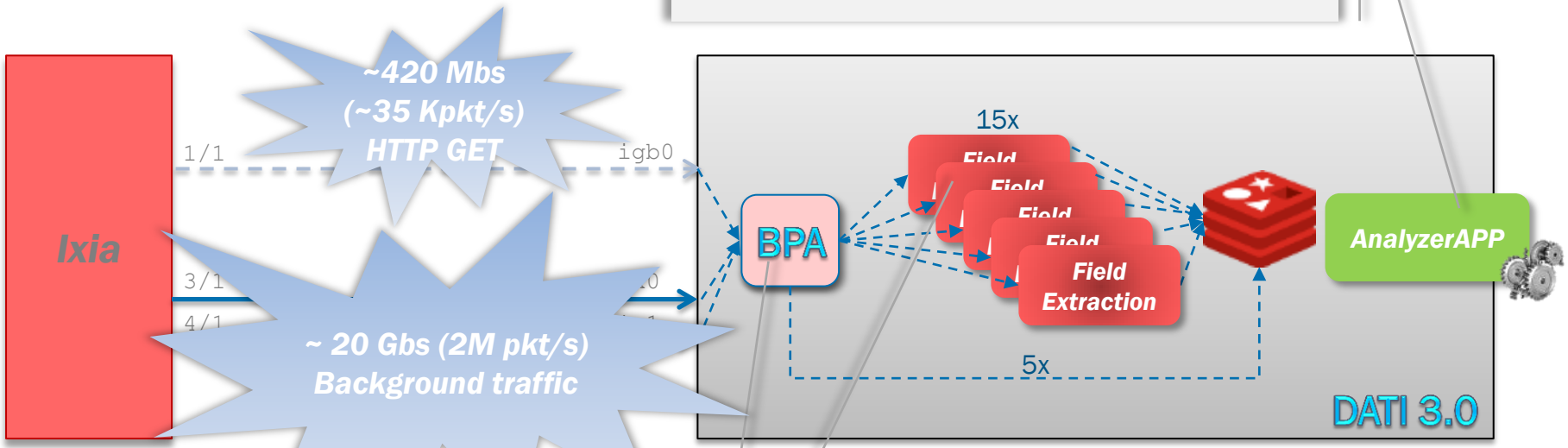
DATI 3.0

LAB TESTS

Test architecture

ANALYSIS

- INPUT: http_GET
- PUBLISH_PERIOD: 10sec
- PUBLISH_FORMAT:STDOUT



BPF

```
FILTER_NAME http
# GET
BPF tcp[(tcp[12] >> 4)*4:4]=0x47455420
TAP_IF tun0
…
TAP_IF tun9
```

FIELD EXTRACTION

- INPUT: tun0 – tun14
- PROTOCOL DEF: netPDL_HTTP_noeth.xml
- FIELDS:(ip.src,ip.dst,tcp.sport,tcp.dport,http.url,http.referer,http.host)
- REDIS_EXPIRE: 300
- REDIS_CHANNEL: http_GET

10 Gbe
1 Gbe

System Under Test details (1/3)

Hardware

SondaRedis# dmesg | grep memory

real memory = 68719476736 (65536 MB)

avail memory = 66232705024 (63164 MB)

SondaRedis# dmesg | grep CPU

CPU: Intel(R) Xeon(R) CPU E5620 @ 2.40GHz (2400.13-MHz K8-class CPU)

FreeBSD/SMP: Multiprocessor System Detected: 16 CPUs

Software

SondaRedis# uname -mrp

9.1-PRERELEASE amd64

SondaRedis# redis-cli info | grep version

redis_version:2.4.4

System Under Test details (2/3)

BPA config

```
#Capture Interface
IFNAME  ix0
IFNAME  ix1
IFNAME  igb0
#With witch frequency (in seconds) print statistics to syslog
TIME_STATS 10

#Redis Unix Socket
REDIS_SOCK      /tmp/redis.sock

#Filters
FILTER_NAME http
BPF tcp[(tcp[12] >> 4) * 4 : 4] = 0x47455420
TAP_IF tun0
…
TAP_IF tun14
```

System Under Test details (3/3)

Redis config

unixsocket /tmp/redis.sock

maxmemory 32000000000
maxmemory-policy volatile-ttl

vm-enabled no

no persistence

Traffic composition

- **Background traffic (each interface)**
  - ~1.031 Kpkt/sec
  - Packet size: uniform distribution from 500 bytes to 1518 bytes
  - ~ 9.791 Mbs
  - Random source and destination (on last 16 bits)
  - Valid TCP header, random payload
- **HTTP traffic**
  - ~35 Kpkt/sec
  - Packet size: 1500 bytes
  - ~420 Mbs
  - Random source and destination (100 destinations, last 16 bits for source)
  - Valid TCP header with 3 different HTTP GET as payload

NbExtractor pseudo-config (1/2)

```
<protocol name="http" longname="HTTP Protocol" showsumtemplate="http">
    <format>
        <fields>
                    <switch expr="buf2int($packet[$currentoffset:4])">
                              <case value="0x47455420">
                                        <includeblk name="GET_payload" />
                              </case>
                              [...]
                    </switch>
                    </fields>
        <block name="GET_payload"     longname="HTTP GET payload">
                    <field type="fixed" name="method" longname="HTTP Method" size="4"
showtemplate="FieldAsciiLineFast"/>
                    <field type="line" name="url" showtemplate="FieldAsciiLineFast"/>
                    <field type="pattern" name="host" pattern="(?&lt;=Host: )[^\\n]*"
showtemplate="FieldAsciiLineFast"/>
                    <field type="pattern" name="useragent" pattern="(?&lt;=User-Agent:
)[^\\n]*" showtemplate="FieldAsciiLineFast"/>
```

NbExtractor pseudo-config (2/2)

```
                    <field type="pattern" name="referer" pattern="(?&lt;=Referer:
)[^\\n]*" showtemplate="FieldAsciiLineFast"/>
                    <field type="pattern" name="cookie" pattern="(?&lt;=Cookie:
)[^\\n]*" showtemplate="FieldAsciiLineFast"/>
                    <field type="variable" name="get" longname="GET options"
expr="$payloadlength-$currentoffset" showtemplate="FieldAsciiLineFast"/>
          </block>
          [...]
          </format>
          [...]
</protocol>
```

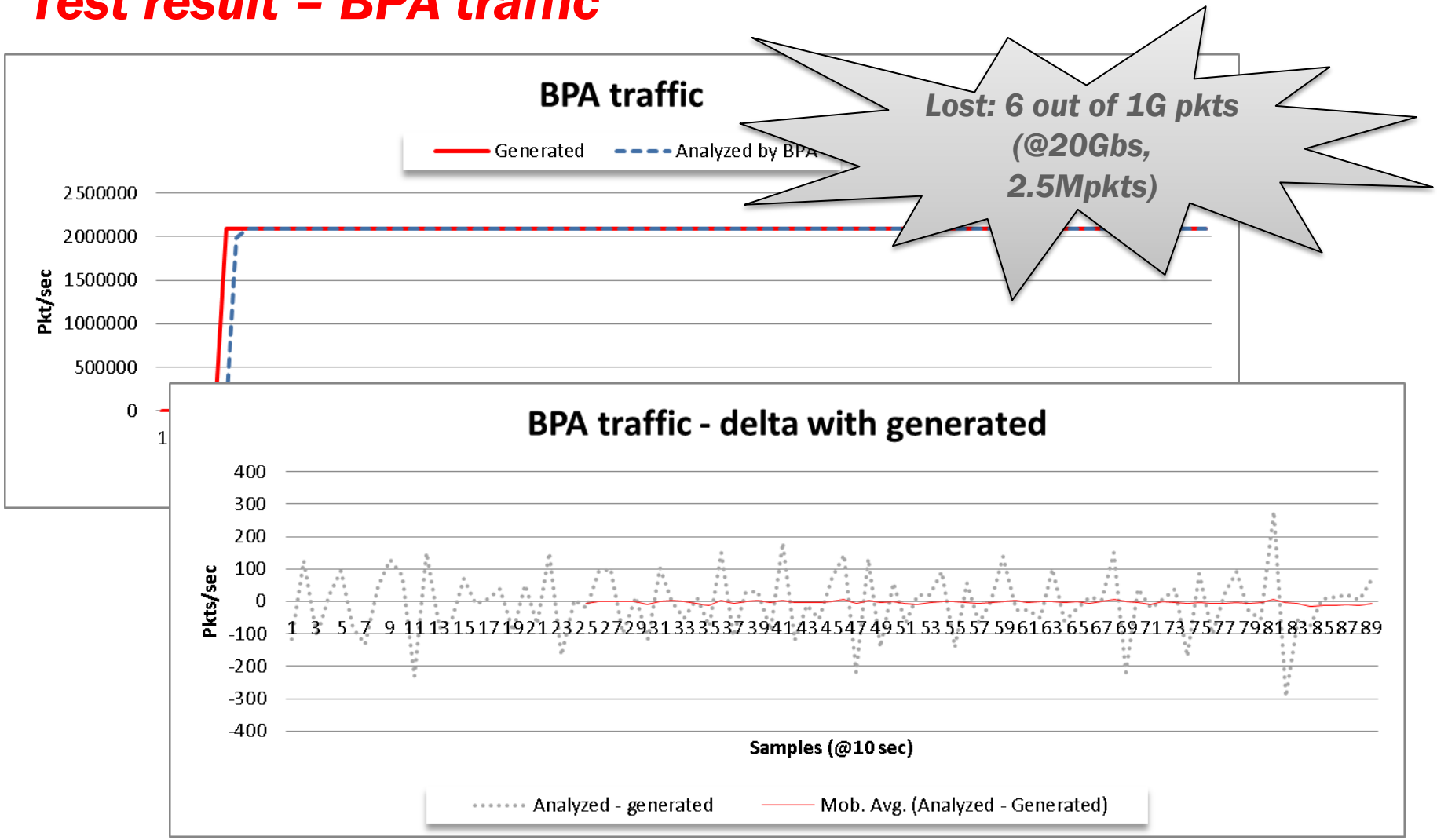
analyzer app pseudo code (nodejs) – 1/2

```
redisMessages = redis.createClient(configuration.redisSocket);
redisSessionInfo = redis.createClient(configuration.redisSocket);
var requests = 0;
Var sessions = 0;

// Publish statistics
setInterval(publishGlobalStats , configuration.publishGlobalStatsPeriod);

redisMessages.on("message", function (channel, message) {
          requests = (requests  *1) +1;
});
redisSessionInfo.on("message", function (channel, message) {
          sessions = (sessions *1) + 1;
});
```

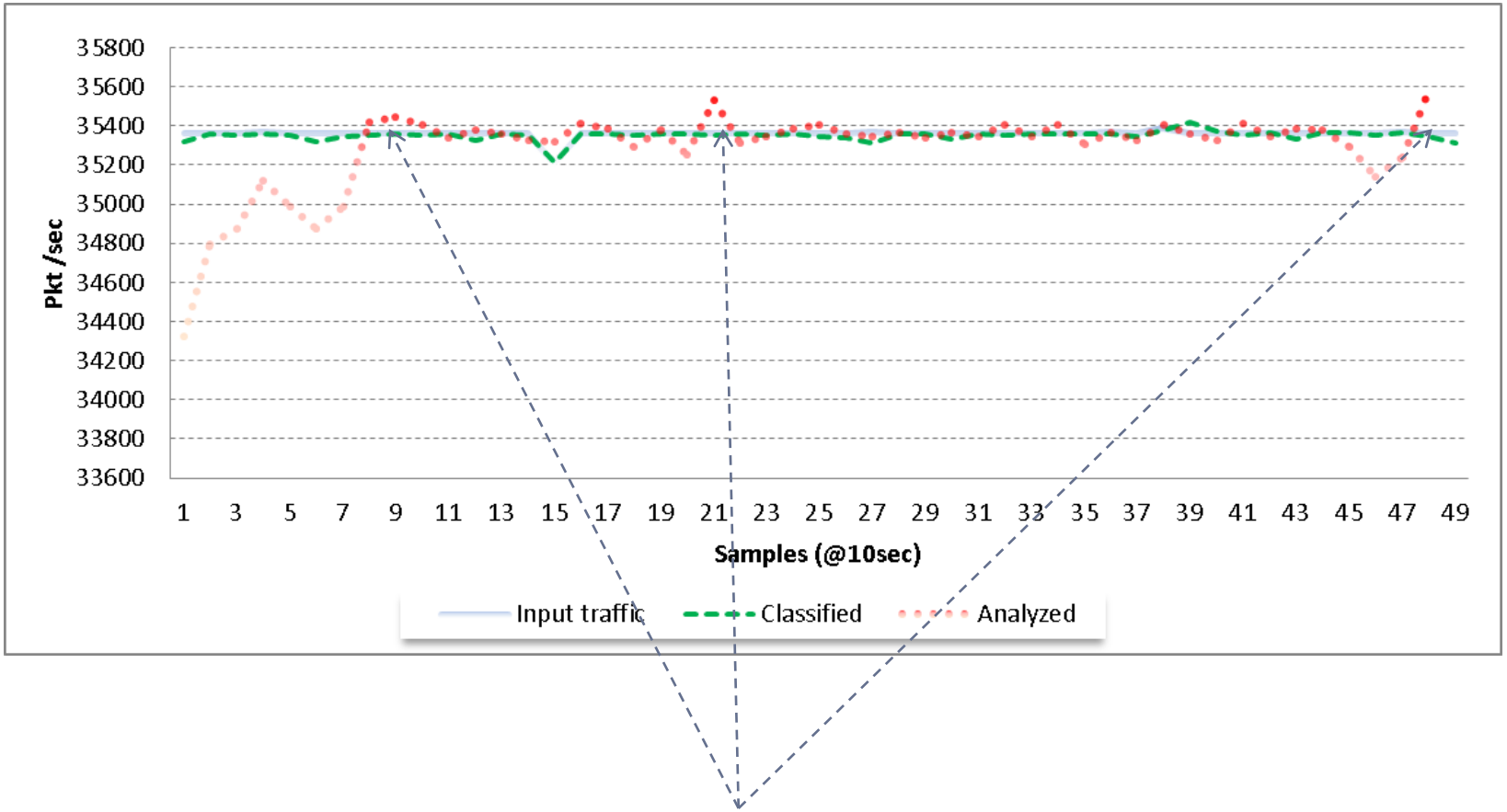
analyzer app pseudo code (nodejs) – 2/2

```
function publishGlobalStats(){
          console.log(requests+":"+sessions);
          requests = 0;
          sessions = 0;
}

// Subscribe to get fields channel
redisMessages.subscribe(configuration.protocolChannel);

// Subscribe to the sessions info channel
redisSessionInfo.subscribe(configuration.BPASessionChannel);
```

Test result – BPA traffic

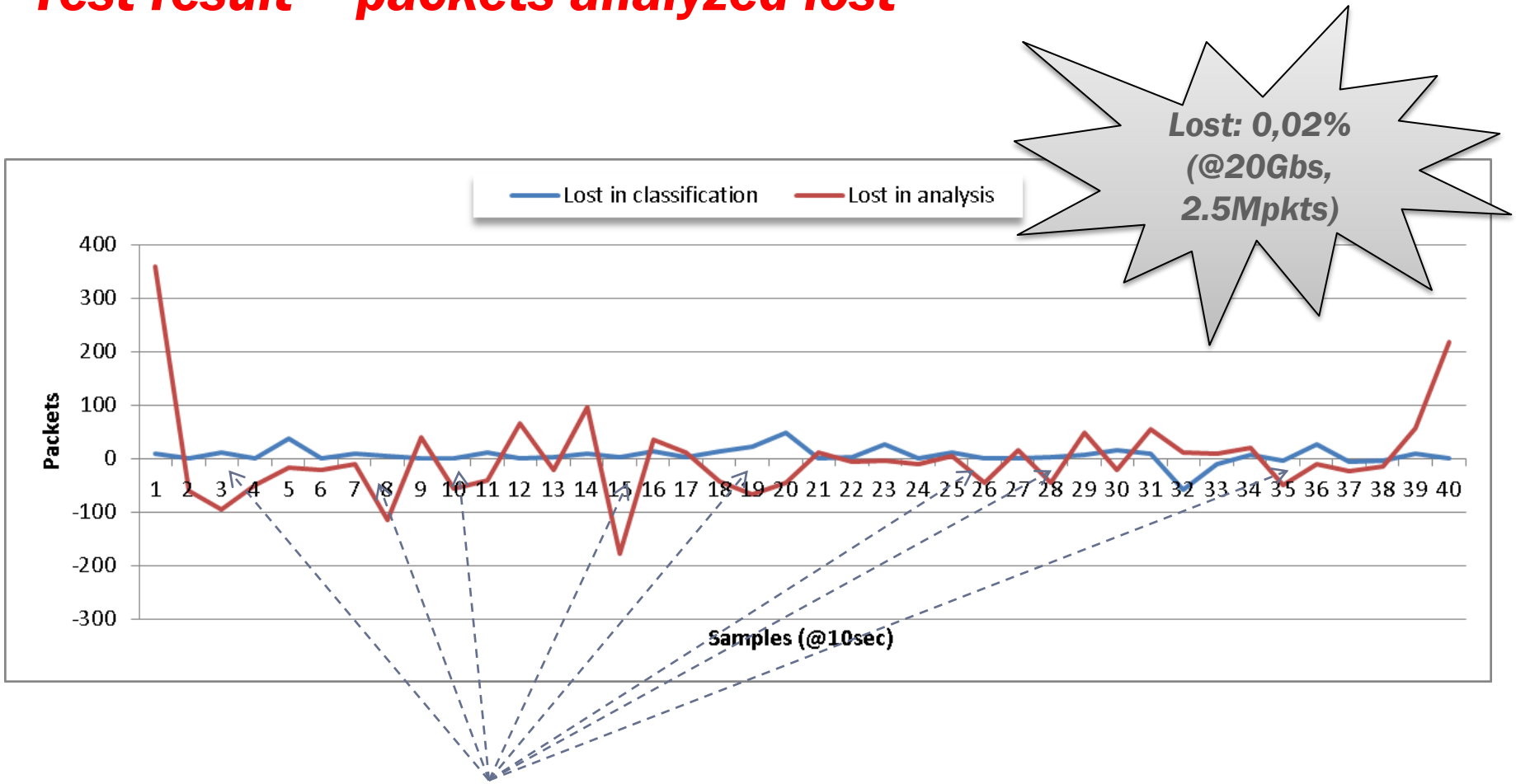


Test result – packets analyzed



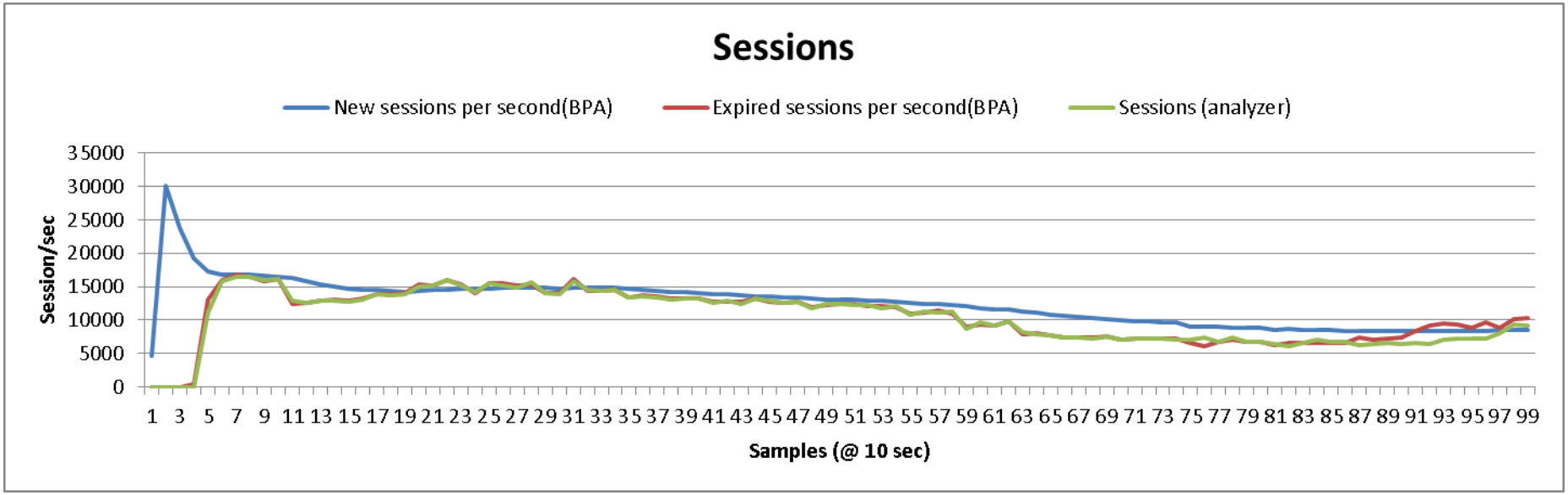
Packets arrives to the analyzer with some variable delay

Test result – packets analyzed lost



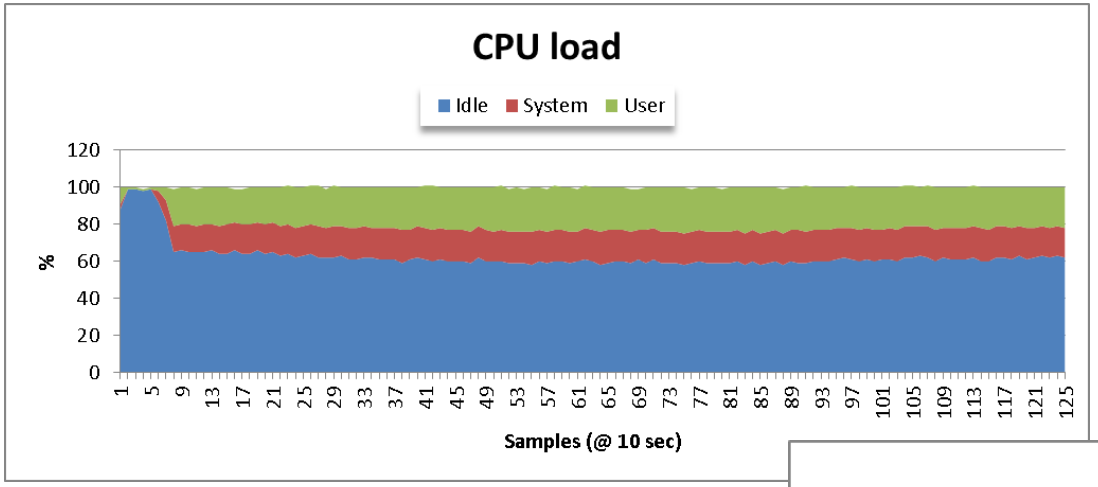
Packets arrives to the analyzer with some variable delay

Test result – BPA sessions



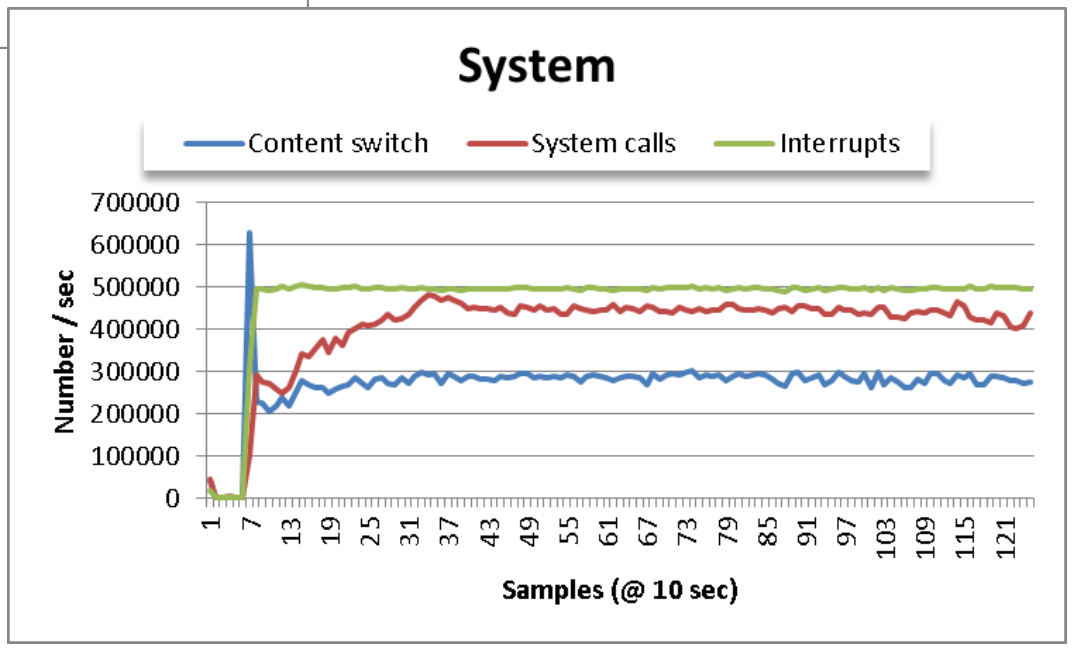
- ***Session identification***
  - ***IP (src, dst) and port (src, dst)***
- ***Session expiration***
  - ***No match on a session for (at least) 30 seconds***

Test result – system load

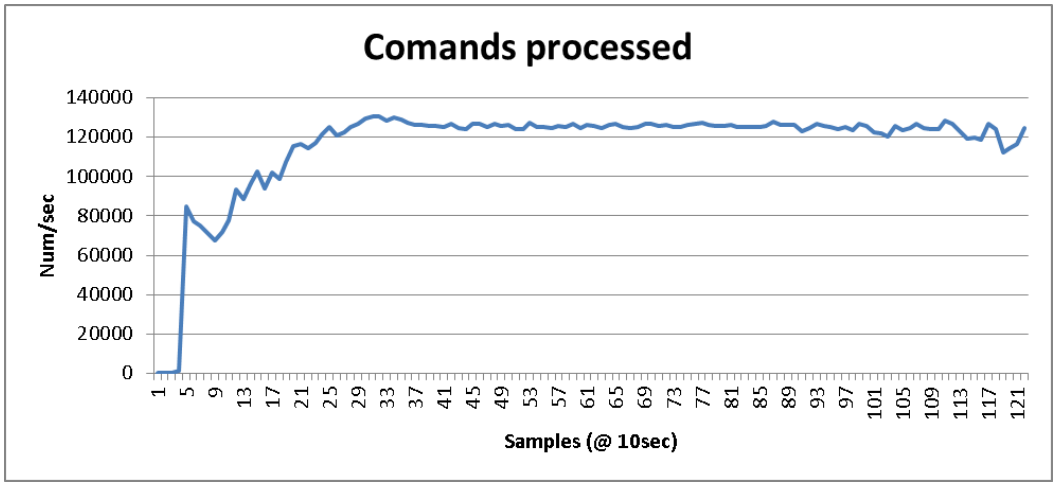


- ***60% system idle***

- ***Stable trend***



Redis



20 clients

- ***15 field extractors***
- ***5 BPA threads for session info***

Pipelining

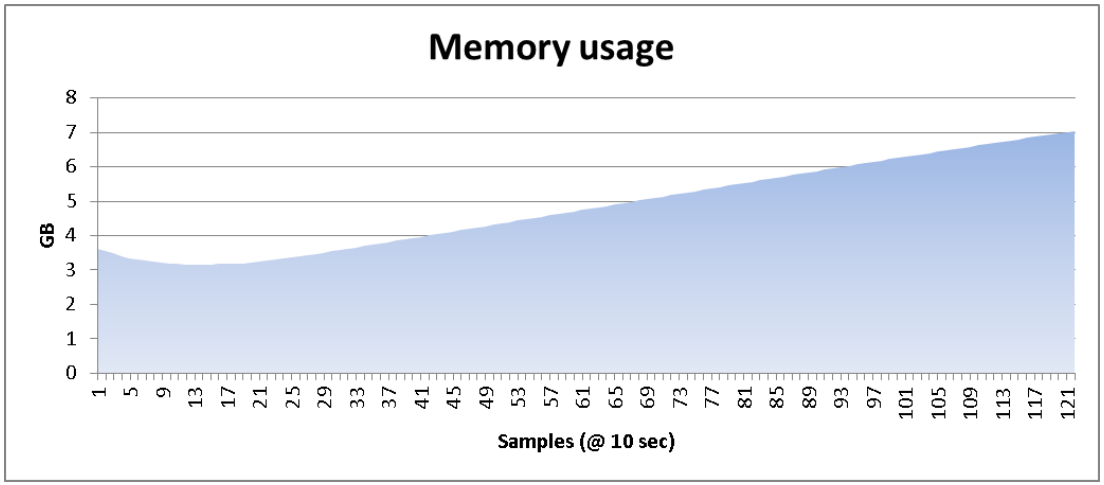
- ***Commands of 50 packets before send***

Socket connections

No persistence

A great amount of memory is required

Real traffic tests demonstrate how good is Redis in memory handling



LAB test conclusions

- The system can handle traffic up to 20 Gbs
  - ~2.5Mpkt/sec handled during tests
  - 35 Kpkt/sec analyzed (1.5 % of total traffic)
  - Resources available for “real”, more complex analyzers
  - 32 GB of memory can be considered enough, 64 GB safe if analyzers use redis for storing temporary information
  - The system is stable
  - 0.02% packets lost in the field extraction/analysis

LAB test conclusions - improvements

- Eliminate the usage of tun interfaces
  - Is this a real limit?
- Improve field extraction
  - We need 0% loss @2.5Mpkts
  - We need to increase the number of packets per Field extractor instance
    - Too many process can lead to contention problems and reduce scalability
    - Complex to manage
  - Trying to go beyond 100Kpts/sec (~5% or total traffic) analyzed
  - Profiling of interaction with redis is required to better understand howto optimize messages and information storage.
- A single redis instance could not be enough
  - Redis clustering?

DATI 3.0

REAL TRAFFIC

System details (1/2)

Hardware

```
DATI-TO1_TEMP2#  dmesg | grep memory
        real memory  = 137438953472 (131072 MB)
        avail memory = 132546859008 (126406 MB)

DATI-TO1_TEMP2# dmesg | grep CPU
CPU: Intel(R) Xeon(R) CPU        X5675  @ 3.07GHz (3066.83-MHz K8-class CPU)
FreeBSD/SMP: Multiprocessor System Detected: 24 CPUs
```

Software

```
DATI-TO1_TEMP2# uname -mrp
9.1-PRERELEASE amd64

DATI-TO1_TEMP2#  redis-cli info | grep version
redis_version:2.4.4
```

System details (2/2)

Redis config

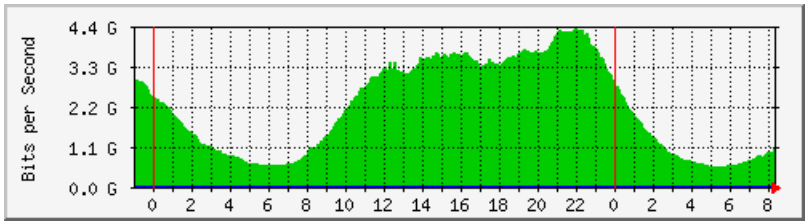
```
unixsocket /tmp/redis.sock
maxmemory 25000000000
maxmemory-policy volatile-ttl
vm-enabled no
no persistence
```

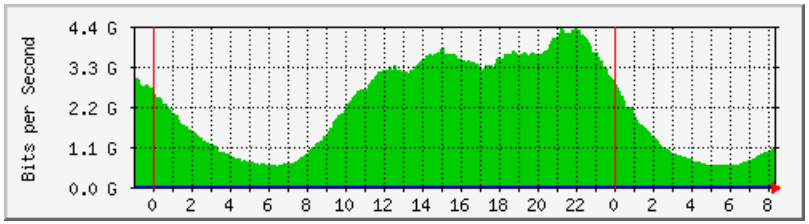
BPA configuration

```
#Capture Interface
IFNAME  ix0
IFNAME  ix1
TIME_STATS 300

FILTER_NAME http
BPF tcp[(tcp[12] >> 4) * 4 : 4] = 0x47455420
TAP_IF tun0
```

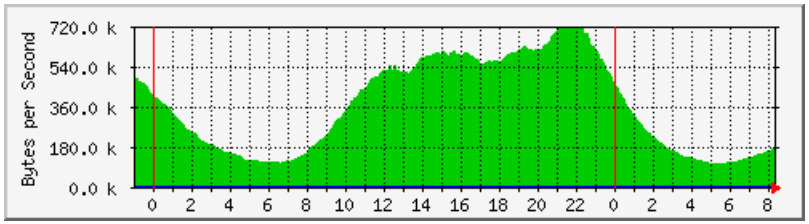
Traffic trend

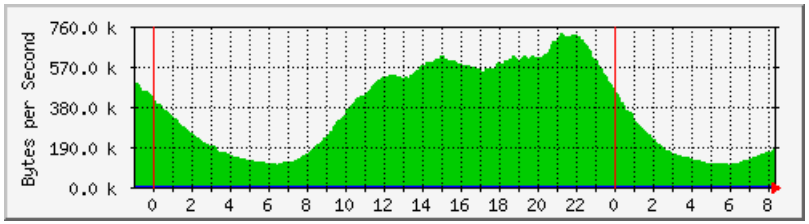
Traffic volume



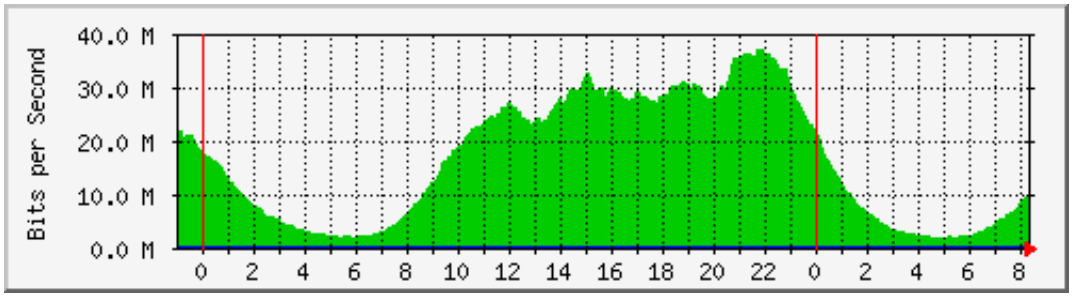


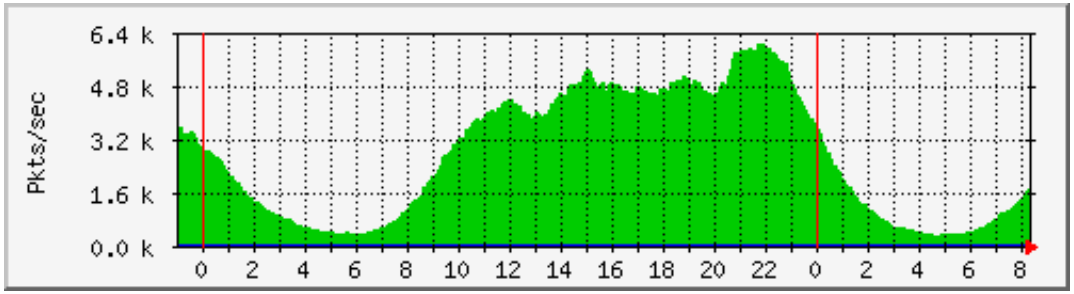
Packet rate





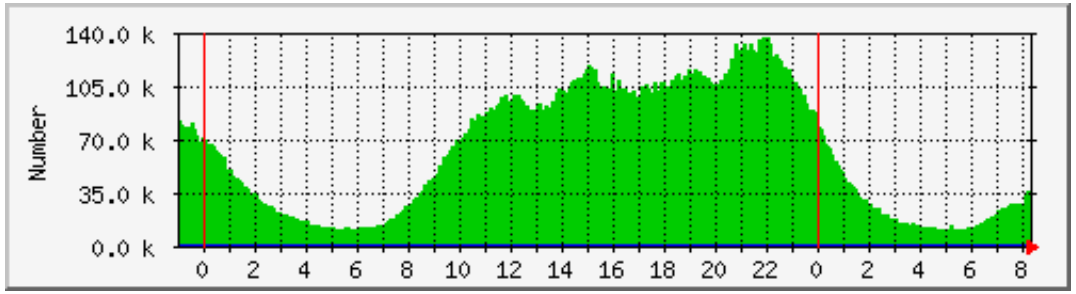
Filtered traffic (HTTP GET)



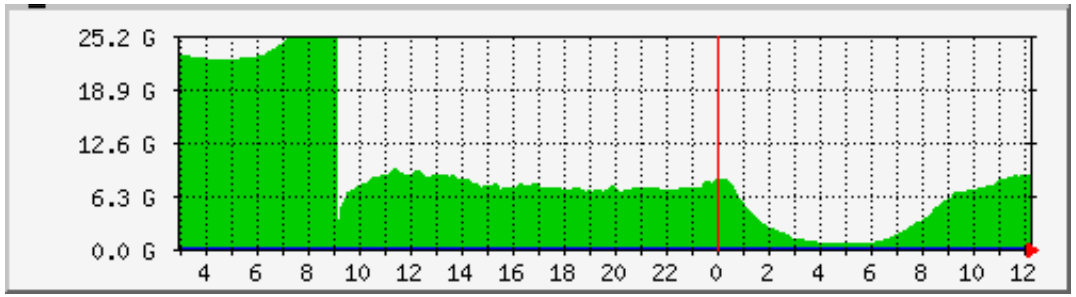


BPA filtered

BPA active sessions

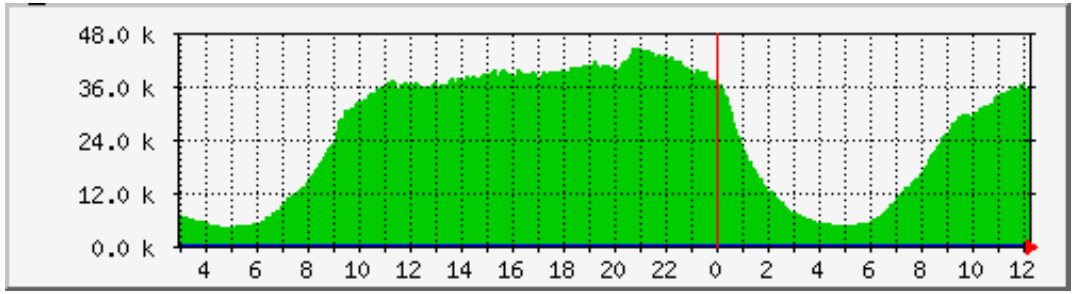


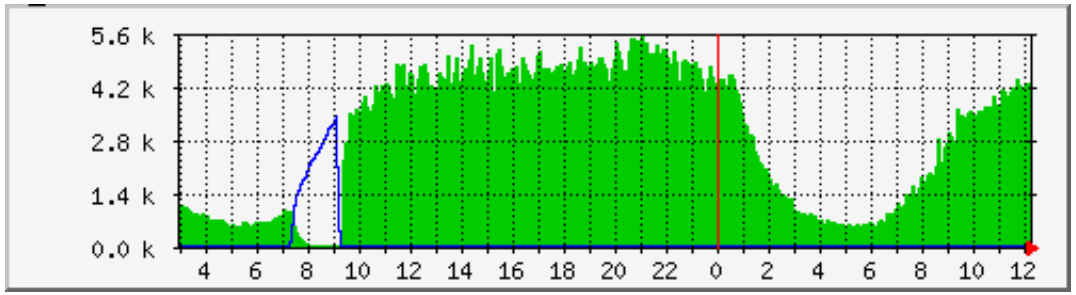
REDIS stats



Redis Memory usage

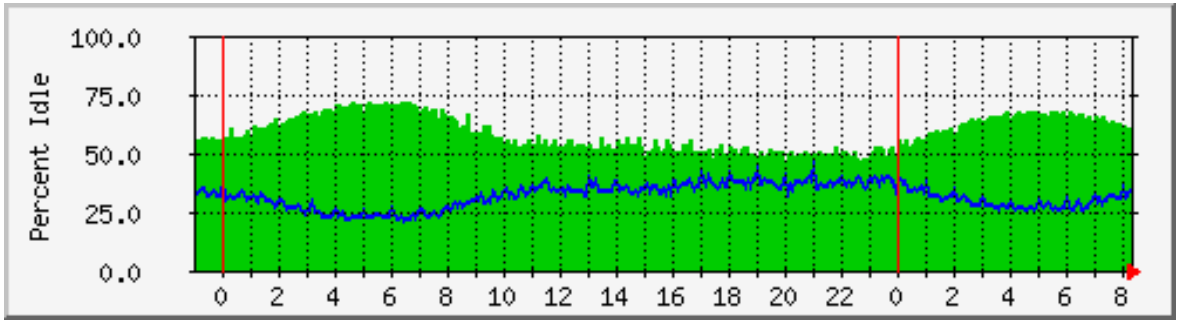
Redis comands processed





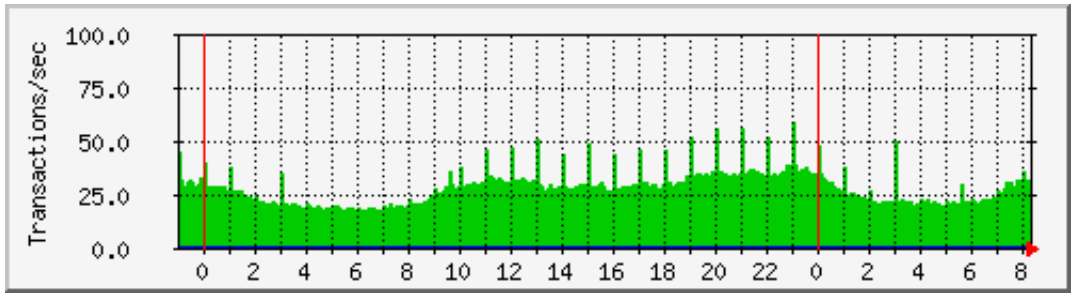
Redis Expired vs Evicted keys

System



Idle CPU (green)
User CPU (blu)

Disk transations



Real traffic conclusions – possible improvements

- ***A lot of work to do***
- ***No so simple to use flow states***
  - ***Analyzers should keep track of interesting flows***
  - ***You should re implement something already in BPA***
- ***Improve BPA classification***
  - ***Statistical classification***
  - ***Analyzers/Field extractors feedback***
- ***Implement a "feedback channel" toward the BPA***
  - ***Analyzers/field extractors can suggest per flow tags***
  - ***These tags will be sent by BPA associated to usual info***

QUESTIONS