SHARKFEST'14

WIRESHARK DEVELOPER AND USER CONFERENCE

JUNE 16-20 2014 · DOMINICAN UNIVERSITY

Monitoring Mobile Network Traffic (3G/LTE)

Luca Deri <deri@ntop.org>

Overview

- Introduction to mobile traffic monitoring
- Analysing mobile traffic with Wireshark
- Monitoring mobile traffic using ntop nProbe, an open-source traffic monitoring probe.
- Advanced mobile monitoring topics.

Why Mobile Traffic is Interesting?

- Plain (wired or wireless) traffic monitoring is a topic that has been covered since very long time and there are many solid monitoring tools available.
- Traffic generated by mobile devices is constantly increasing and it cannot be monitored using tools not designed for this task.
- Mobile traffic monitoring does not mean that this is a just topic for mobile operators: there are plenty of opportunities for companies already active in the network monitoring world.

Opportunities in Mobile Monitoring [1/2]

- Mobile network monitoring tools are usually derived from the telecommunication world.
- Often these tools produce aggregated metrics (i.e. number of bytes per cell) or radio-related statistics.
- Traffic monitoring tools are often not able to monitor mobile traffic in detail and usually rely on costly data capture cards or custom designed FPGA-based board.

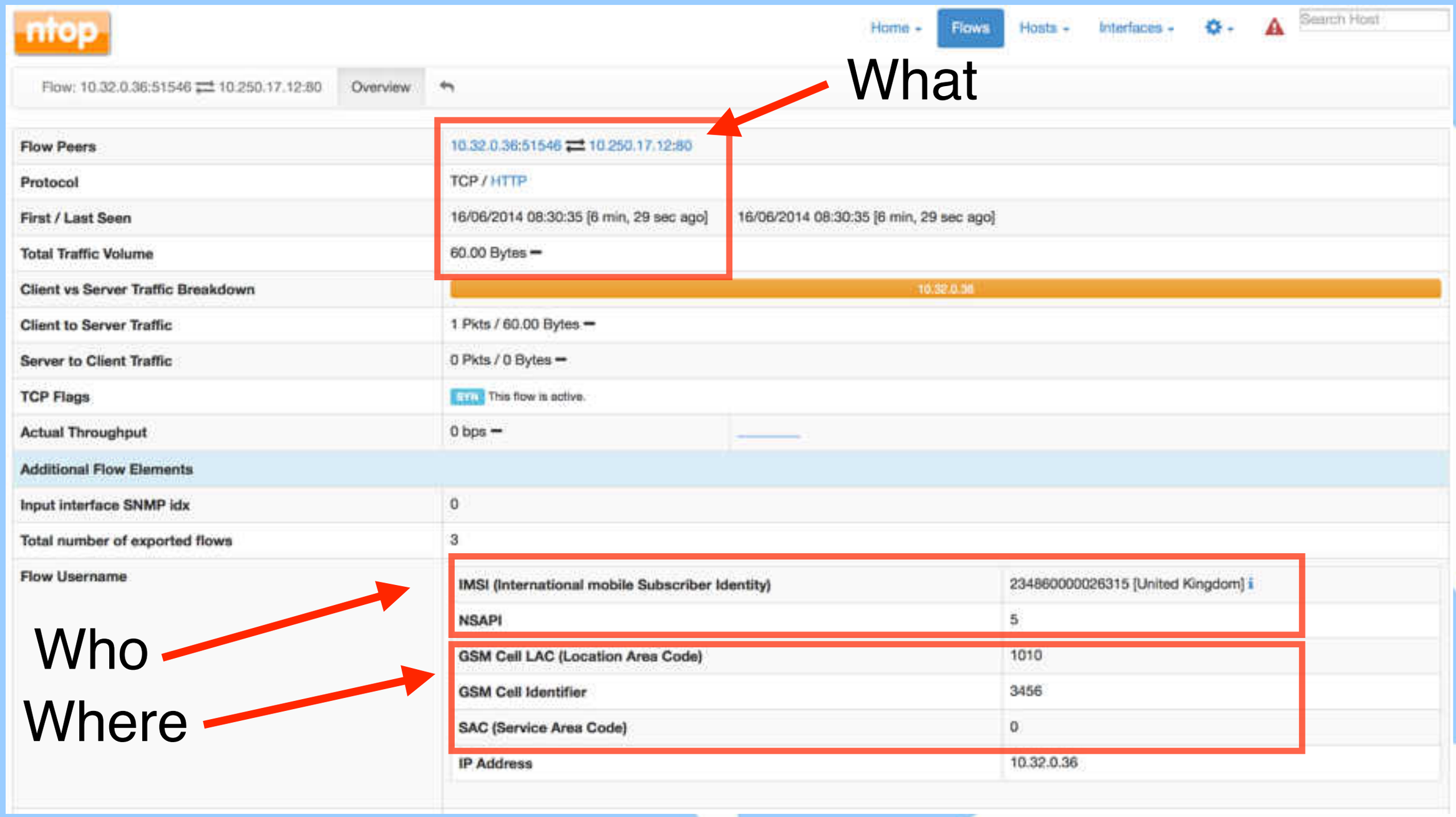
Opportunities in Mobile Monitoring [2/2]

- Monitoring tools are (usually) proprietary, and tight to the hardware platform used to capture data.
- Monitored data is stored in proprietary format and usually not available to third party apps (no open-data).
- Open source mobile traffic monitoring tools are rare, often just research tools (e.g. thesis outcome) and not production ready.

Motivation

- Bring benefits of open-source and open-data to mobile traffic monitoring.
- Exploit wireshark for mobile-traffic troubleshooting.
- Create comprehensive mobile traffic monitoring tools leveraging on code and lessons learnt on Internet traffic monitoring.
- In essence “open” the world of mobile traffic monitoring as happened with Internet monitoring.

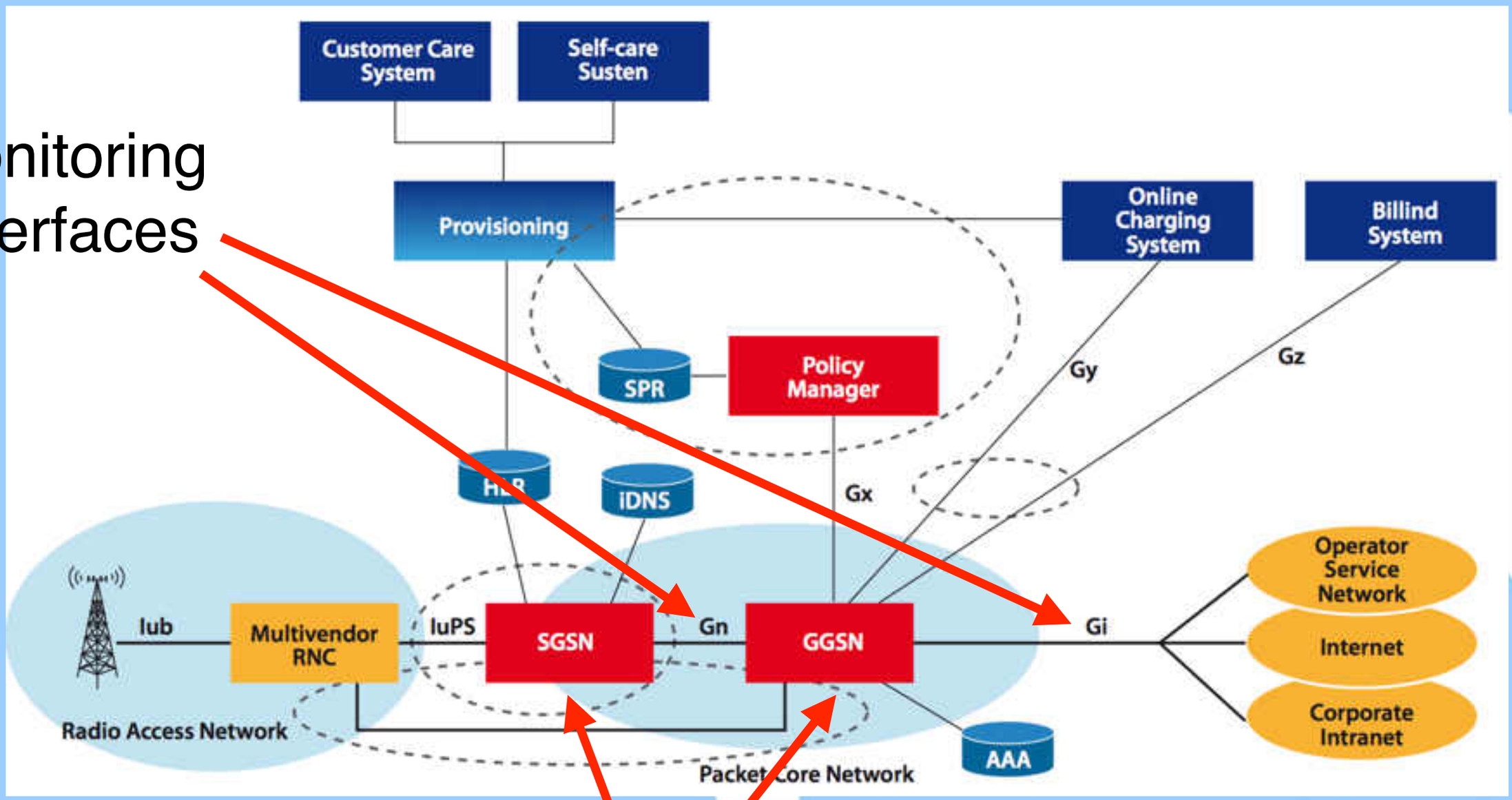
Let’s Start From The End…



Important: All realtime, using open source software, at 10 Gbit.

Mobile Network Architecture

Monitoring Interfaces



Core Gateways

GPRS Core Network Nodes [1/2]

SGSN (Serving GPRS [General Packet Radio System] Support Node) is a node responsible for:

- User authentication
- Data session management including QoS (based on subscription type).
- Gateway towards the radio access.
- Traffic routing towards GGSN.
- Management of roaming users.

GPRS Core Network Nodes [2/2]

GGSN (Gateway GPRS Support Node) is a node responsible for:

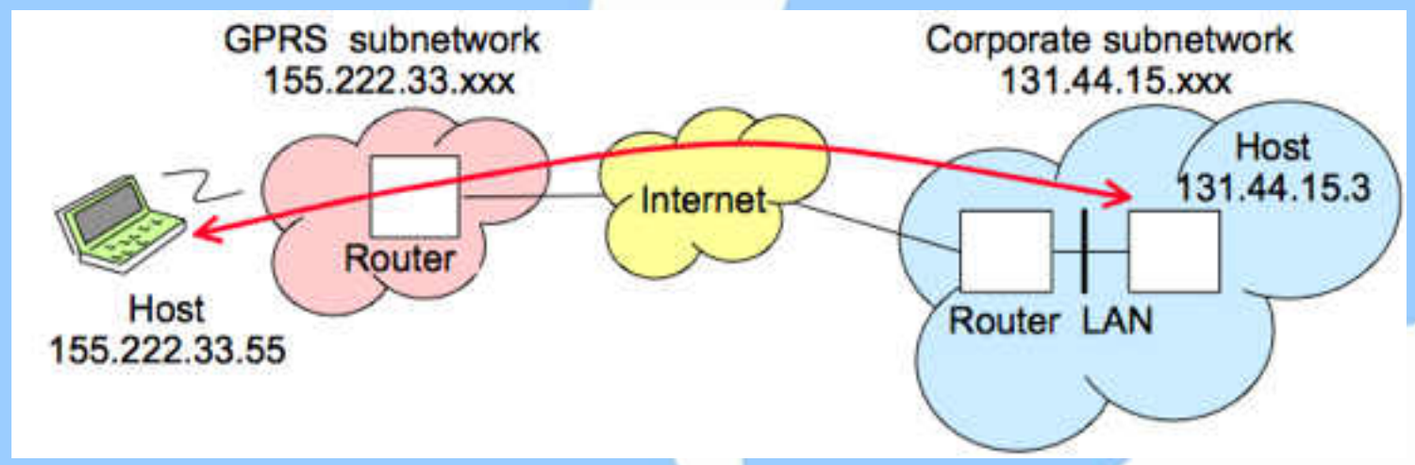
- Inter-networking between the GPRS network and external network such as the Internet.
- Gateway towards external networks (e.g. the Internet) identified by an APN (Access Point Name).
- IP assignment to mobile terminals.
- Billing and support for lawful interception.

Core Network Interfaces

- SGSN-to-GGSN (Gn) Interface
  IP-based network interface between the SGSN and the GGSN. In 4G networks this interface is called S11.
- GGSN-to-PDN (Gi) Interface
  IP-based network interface between the GGSN and a public data network (PDN) such as the Internet. In 4G networks this interface is called S5.

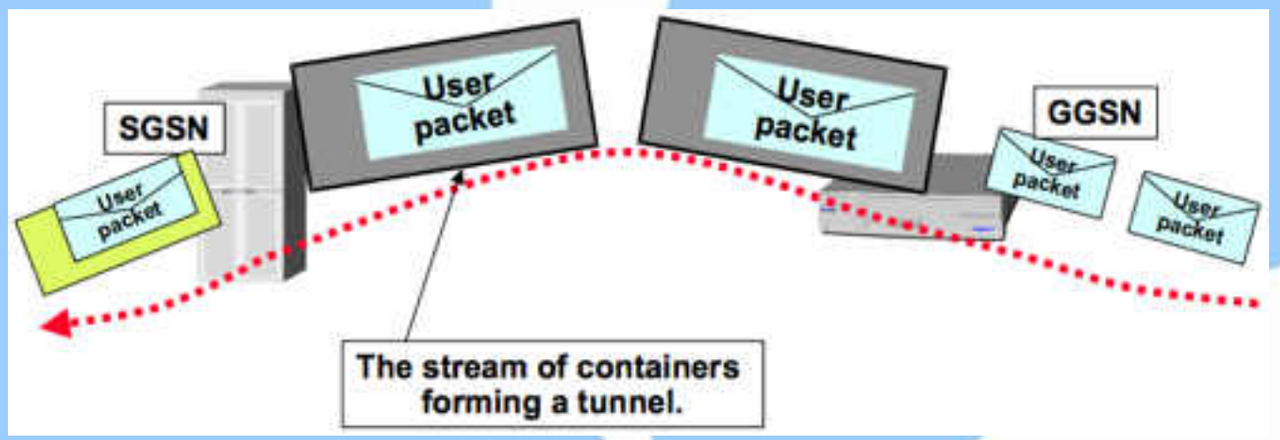
Traffic Routing in GPRS Networks [1/2]

- From the Internet, the GGSN is a router to an IP subnetwork: when incoming (from the Internet) traffic is received, it checks if the mobile address is active, and if so, it forwards the traffic to the SGSN gateway serving the mobile terminal.
- When egress traffic (to the Internet) is received from the GPRS code, it routes it to the correct external network.



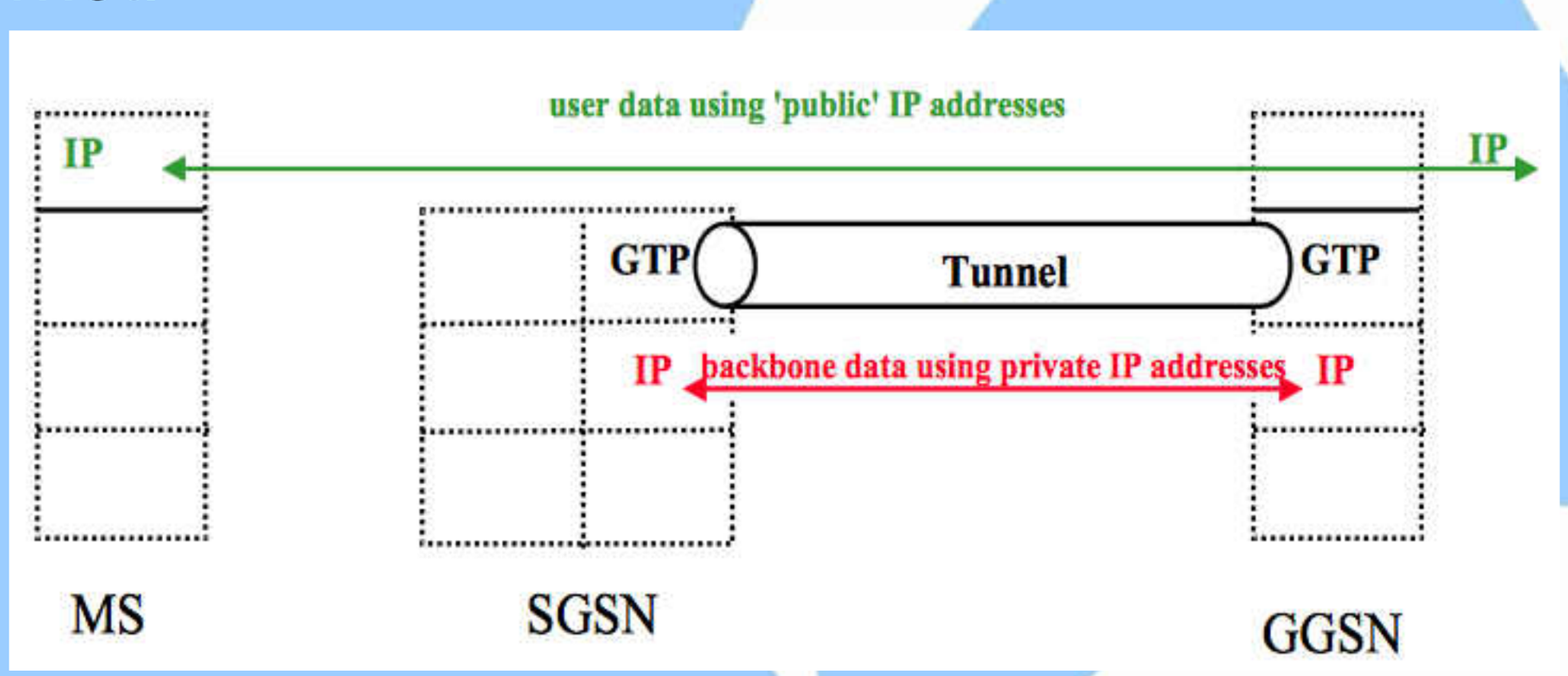
Traffic Routing in GPRS Networks [2/2]

- On the GPRS network, user traffic is sent in tunnels that traverse the network backbone.
- For mobile terminal, it looks as if they are connected via a router (the GGSN) to the Internet.
- Properly handling tunnels, allows mobile users to move inside the mobile network while connected to the Internet with a permanent IP address (i.e. it does not change overtime).



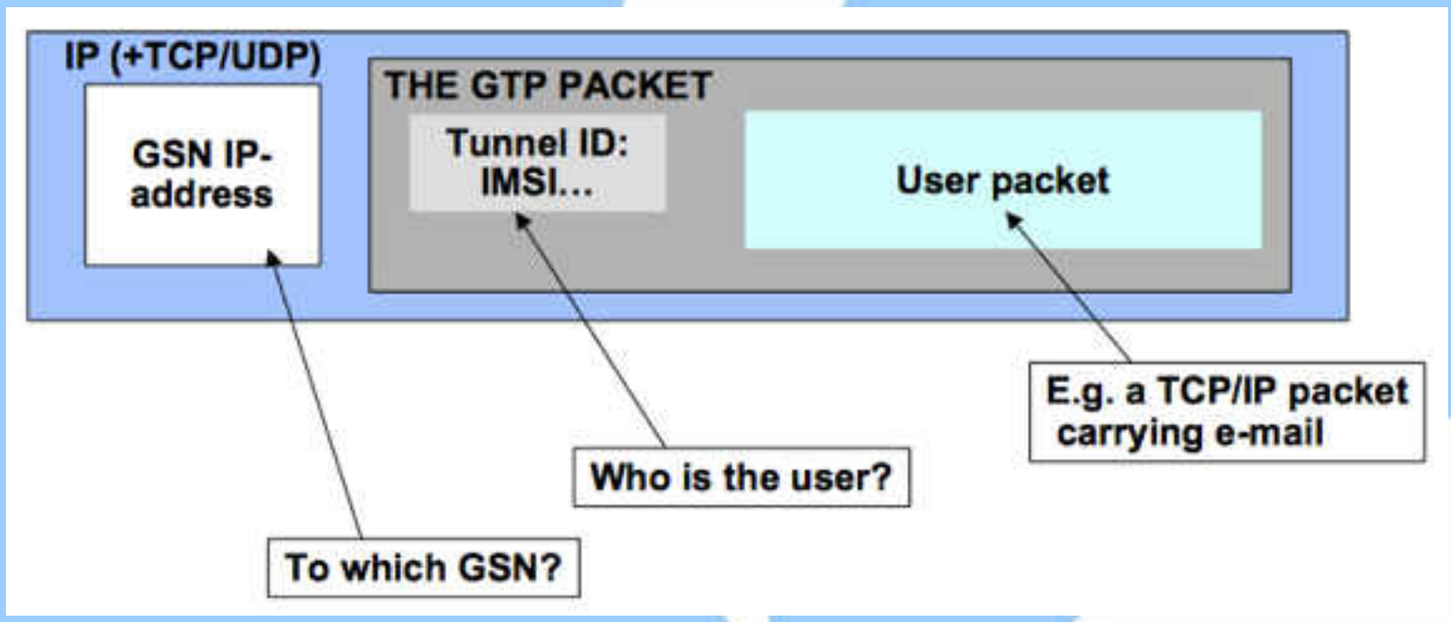
GTP Tunnelling [1/3]

- Every user packet is encapsulated in GTP tunnels (Gn Interface).
- Through the use of GTP tunnels, the mobile subscriber (MS) traffic traverses the GPRS backbone up to the Internet.

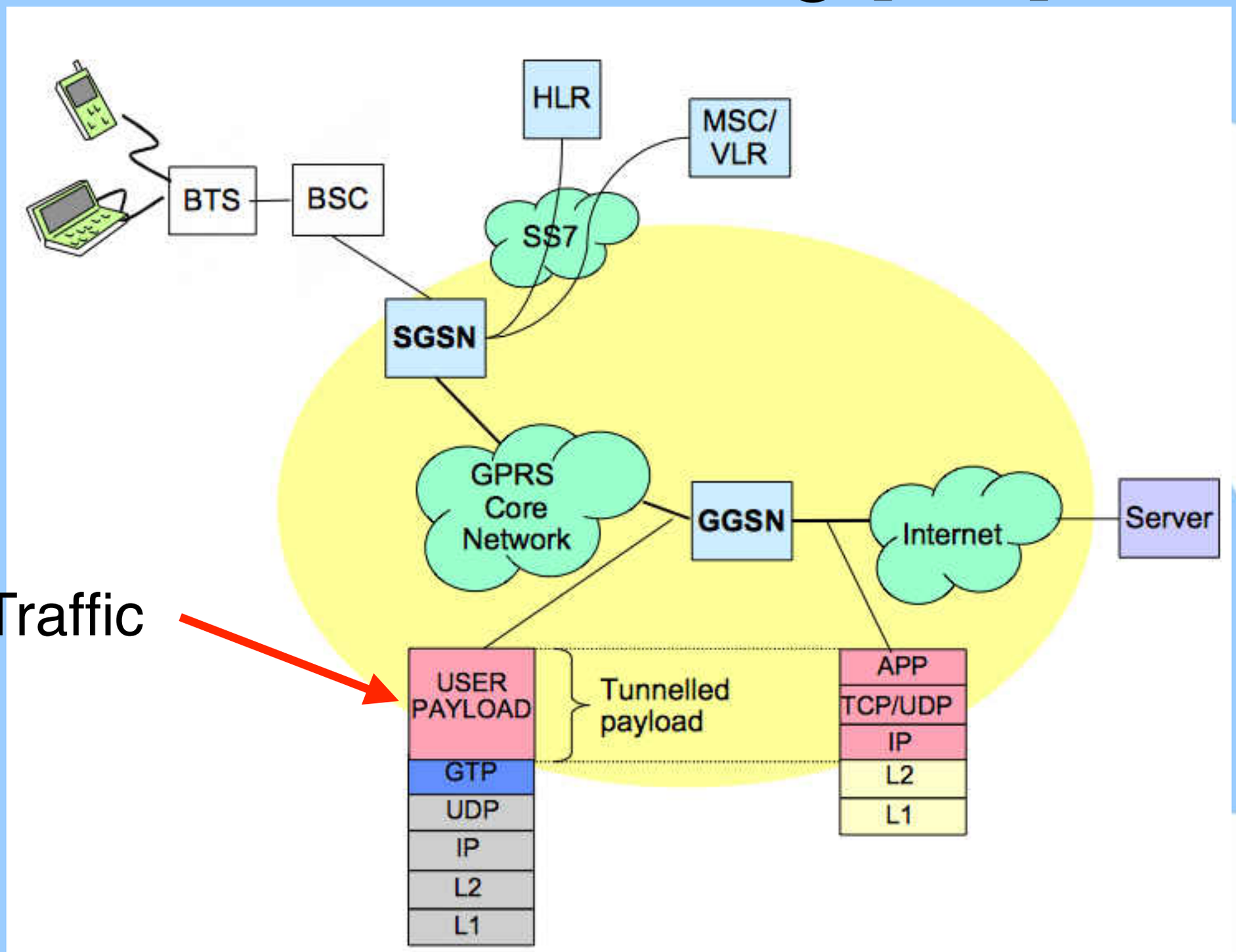


GTP Tunnelling [2/3]

- GTP packets contain several encapsulation layers.
  - The external IP are those of the GGSN and SGSN.
  - The GTP packet contains the tunnel-Id that identifies the user IMSI (International Mobile Subscriber Identity) stored in the phone SIM.



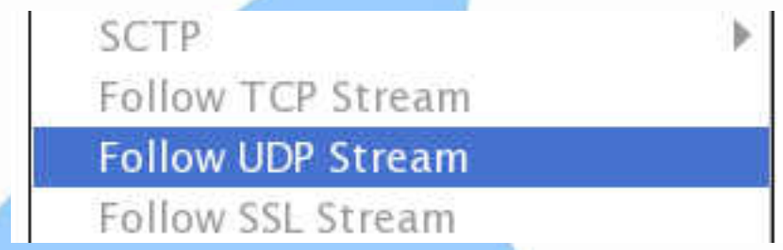
GTP Tunnelling [3/3]



User Traffic

GTP Support in Wireshark

- Wireshark contains native dissectors for GTP (v0, v1, and v2).
- It is able to follow GTP tunnels via “Follow UDP Stream”.



- Wireshark can associate GTP requests with responses as well decode GTP fields. [Response In: 5922]
- Each GTP message has a fixed header message format (e.g. where the IMSI is stored), and a set of variable fields, each identified by a unique numeric identifier.

GTP-C Context Creation [1/2]

SGSN GGSN

```
▷ Internet Protocol Version 4, Src: [illegible]6.197 ([illegible]26.197), Dst: [illegible]01.137 ([illegible]1.137)
▷ User Datagram Protocol, Src Port: gtp-control (2123), Dst Port: gtp-control (2123)
▽ GPRS Tunneling Protocol
  ▷ Flags: 0x32
    Message Type: Create PDP context request (0x10)
    Length: 178
    TEID: 0x00000000
    Sequence number: 0x890e
    IMSI: [illegible]
  ▷ Routing Area Identity
    .... ..01 = Selection mode: MS provided APN, subscription not verified (1)
    TEID Data I: 0x000225ae
    TEID Control Plane: 0x2c00ef8c
  ▷ NSAPI: 5
  ▷ End user address (IETF/IPv4)
  ▷ Access Point Name: t[illegible]
  ▷ Protocol configuration options
  ▷ GSN address : [illegible]26.197
  ▷ GSN address : [illegible]26.197
  ▷ MS international PSTN/ISDN number
  ▷ Quality of Service
  ▷ RAT Type: UTRAN
  ▷ User Location Information
  ▷ MS Time Zone: GMT + 10 hours 0 minutes
  ▷ IMEI(SV): [illegible]
    [Response In: 5922]
```

Associate Request with Response

Unique User (SIM) Identifier

GTP-U Tunnel Id (SGSN -> GGSN)

GTP-C Tunnel Id (SGSN -> GGSN)

Session Identifier (0-15)

Requested Address Type (IPv4 or IPv6)

Mobile Terminal Phone Number

Bandwidth Assigned to the Mobile Terminal (Contract)

Radio Cell to which this Mobile Terminal is connected

International Mobile Equipment Identity
(Unique Mobile Device Identifier)

GTP-C Context Creation [2/2]

GGSN SGSN (IP Swap)

```
▷ Internet Protocol Version 4, Src: [redacted].137 ([redacted].137), Dst: [redacted]126.197 ([redacted]126.197)
▷ User Datagram Protocol, Src Port: gtp-control (2123), Dst Port: gtp-control (2123)
▽ GPRS Tunneling Protocol
  ▷ Flags: 0x32
    Message Type: Create PDP context response (0x11)
    Length: 93
    TEID: 0x2c00ef8c
    Sequence number: 0x890e
    Cause: Request accepted (128)
    Reordering required: False
    Recovery: 70
    TEID Data I: 0x3c38[redacted]14d
    TEID Control Plane: 0x2d9e644d
    Charging ID: 0x9ac1fda3
  ▷ End user address (IETF/IPv4) : [redacted]9.22
  ▷ Protocol configuration options
  ▷ GSN address : [redacted]1.137
  ▷ GSN address : [redacted].137
  ▷ Quality of Service
  ▷ Charging Gateway address : [redacted].146
    [Response To: 4926]
    [Time: 0.032000000 seconds]
```

Same as Request
Positive/Negative Response

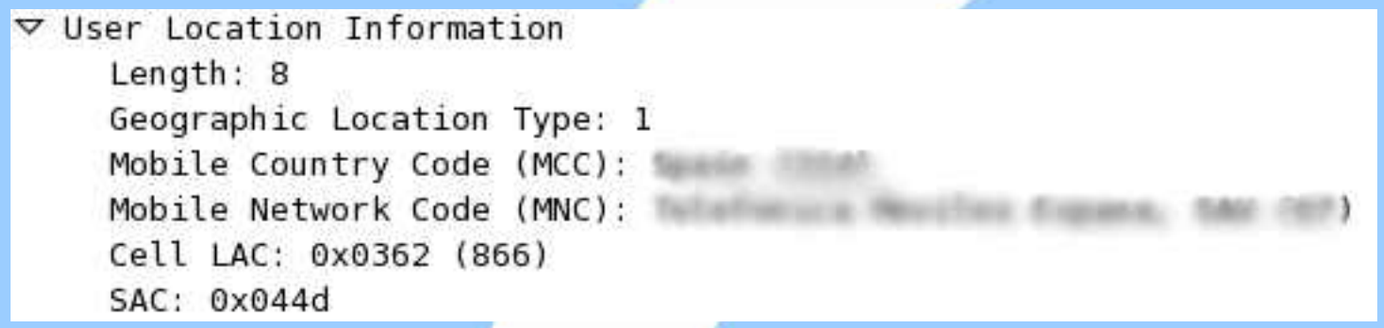
GTP-U Tunnel Id (GGSN -> SGSN)

GTP-C Tunnel Id (GGSN -> SGSN)

Assigned IP Address (DHCP-like)

GTP-C Mobile Terminal Location

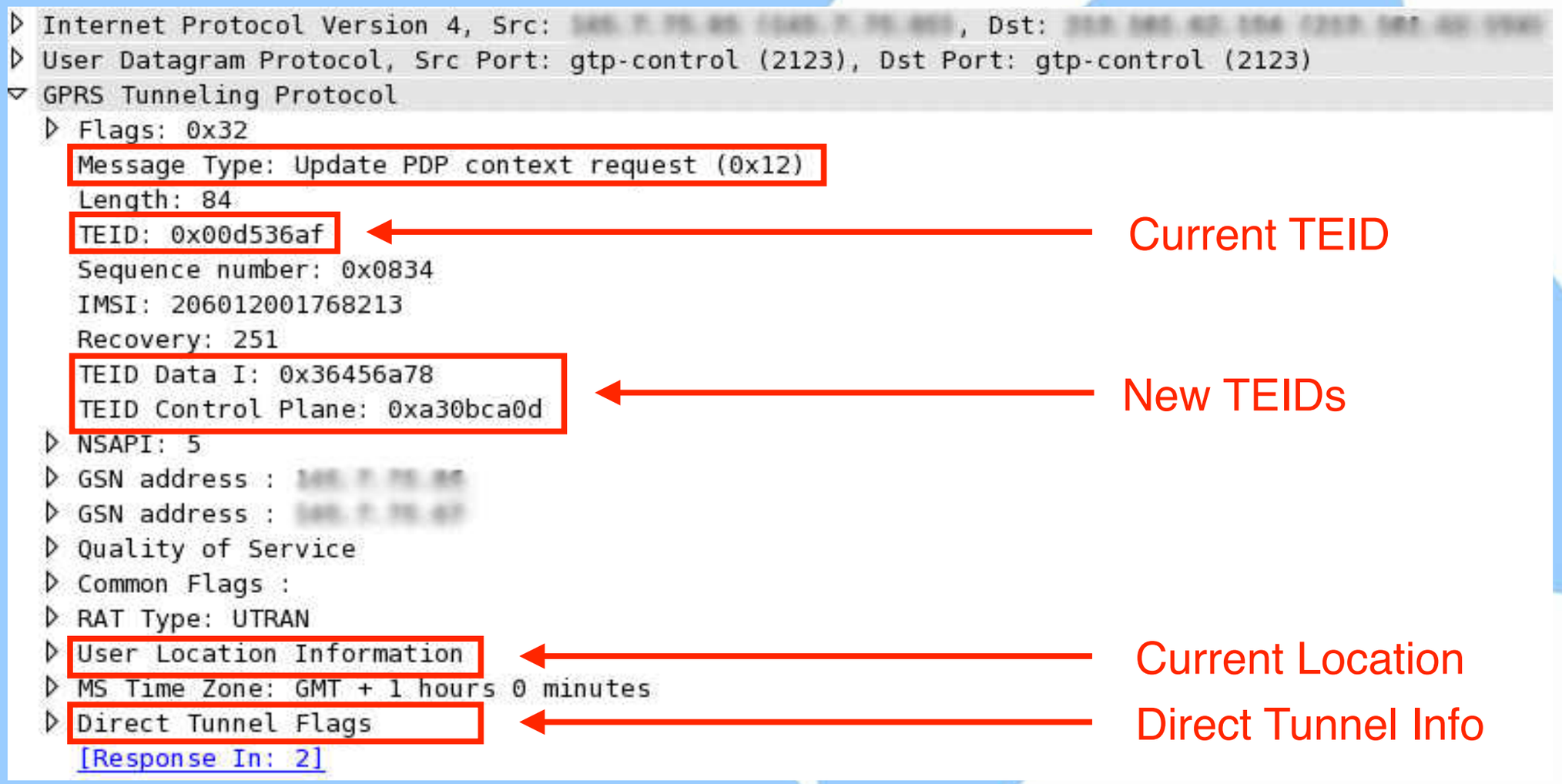
- During GTP-C context creation, the mobile terminal reports the cell to which it is currently connected.



- Using the Cell LAC (Location Area Code) or Cell CI (Cell Identifier) it is possible to know approximately where the terminal is currently located (no GPS is needed).

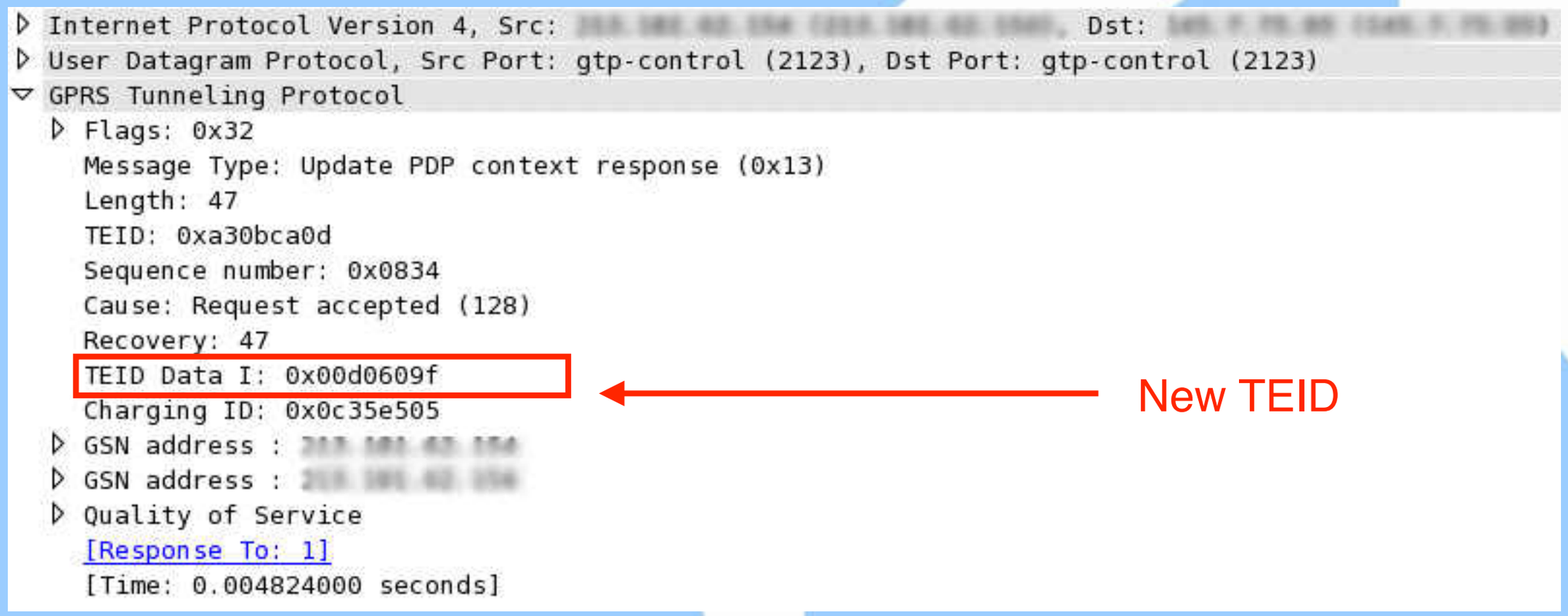
GTP-C Context Update [1/2]

- When a mobile terminal changes SGSN (move inside the mobile network), the TEID can change.
- GTP Direct Tunnel (see later).



GTP-C Context Update [2/2]

- In the response (if positive) there are the new tunnels for the reverse direction.



GTP-U

- GTP-U are GTP messages whose type is T-PDU.
- GTP-U carries user traffic tunnelled in GTP data tunnels previously negotiated with GTP-C.
- Due to GTP tunnelling, unless the interface MTU is enlarged, GTP-U might lead to strong traffic fragmentation.

```
▷ Internet Protocol Version 4, Src: [redacted], Dst: [redacted]
▷ User Datagram Protocol, Src Port: trp (2156), Dst Port: gtp-user (2152)
▽ GPRS Tunneling Protocol
  ▷ Flags: 0x30
    Message Type: T-PDU (0xff)
    Length: 1064
    TEID: 0x366090ff
  T-PDU Data 1064 bytes
▷ Internet Protocol Version 4, Src: 67.201.54.210 (67.201.54.210), Dst: 10.167.143.192 (10.167.143.192)
▷ Transmission Control Protocol, Src Port: http (80), Dst Port: 50770 (50770), Seq: 4205529448, Ack: 258092658, Len: 1024
▷ Hypertext Transfer Protocol
```

GTP: Monitoring Challenges [1/2]

Due to the nature of GTP, monitoring tools need to face with many challenges:

- For no reason GTP-C traffic can be dropped as this would lead to inability to map user/cells with user traffic.
- Traffic Fragmentation. In general traffic can be fragmented, but with GTP this can happen much more often in networks where the MTU has not been enlarged. Fragmentation increases load on probes and slows down operations.

GTP: Monitoring Challenges [2/2]

- Due to the distributed nature of mobile networks, it is unlikely that all the traffic can be observed from the same monitoring point.
- The increasing network speed (also promoted by 4G networks) requires traffic to be balanced across multiple monitoring probes.
- Traffic partition creates an additional issue if monitored data need to be consolidated into a single location.

Merging+Balancing GTP Traffic [1/3]

- Interface merging is necessary whenever traffic is split across interfaces (e.g. network tap) or when multiple interfaces (e.g. master and fail-over interfaces) are monitored.
- Once traffic is merged, it needs to be balanced across monitoring applications in order to share the load.
- Due to the nature of GTP traffic, it is necessary to properly balance traffic in order to send both direction of the flow (src-to-dst and dst-to-src) to the same monitoring probe.

Merging+Balancing GTP Traffic [2/3]

- Depending on GTP PDUs we need to balance on the inner or outer IP addresses.

GTP-U

```
▷ Internet Protocol Version 4, Src: [redacted], Dst: [redacted]
▷ User Datagram Protocol, Src Port: trp (2156), Dst Port: gtp-user (2152)
▽ GPRS Tunneling Protocol
  ▷ Flags: 0x30
    Message Type: T-PDU (0xff)
    Length: 1064
    TEID: 0x366090ff
  T-PDU Data 1064 bytes
▷ Internet Protocol Version 4, Src: 67.201.54.210 (67.201.54.210), Dst: 10.167.143.192 (10.167.143.192)
▷ Transmission Control Protocol, Src Port: http (80), Dst Port: 50770 (50770), Seq: 4205529448, Ack: 258092658, Len: 1024
▷ Hypertext Transfer Protocol
```

GTP-C

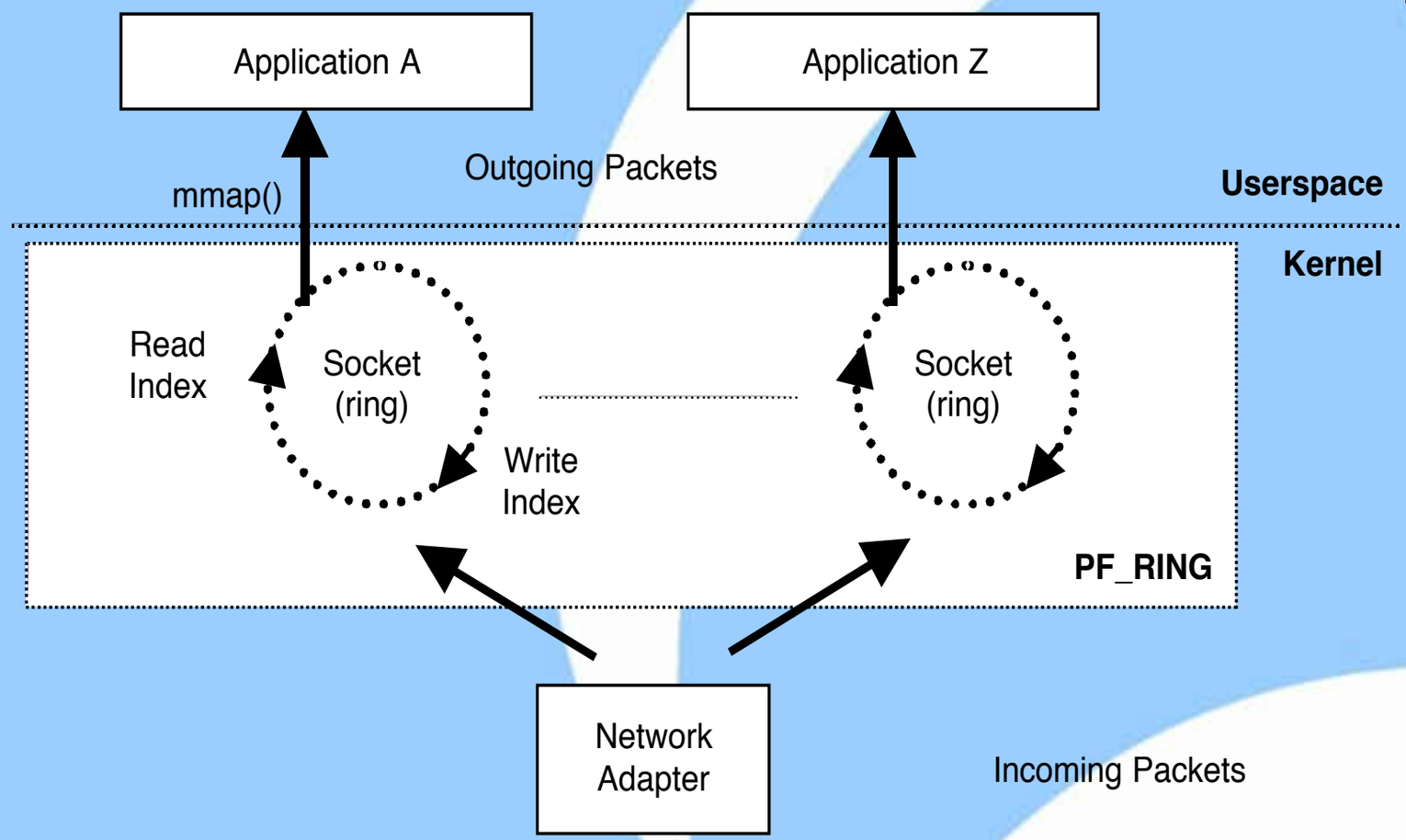
```
▷ Internet Protocol Version 4, Src [redacted], Dst: [redacted]
▷ User Datagram Protocol, Src Port: gtp-control (2123), Dst Port: gtp-control (2123)
▽ GPRS Tunneling Protocol
  ▷ Flags: 0x32
    Message Type: Delete PDP context response (0x15)
    Length: 6
    TEID: 0x8ebc7a0e
    Sequence number: 0xb435
    Cause: Request accepted (128)
    [Response To: 1]
    [Time: 0.023881000 seconds]
```

Merging+Balancing GTP Traffic [3/3]

- Balancing using the tunnelId as key is not a good idea as each direction of the tunnel as a different TEID. Thus tunnel coherency should be implemented by keeping the tunnel association of each direction via a stateful system.
- Better (as it's a stateless activity) to use IPs for balancing traffic across probes, and send each probe a portion of the traffic.
- If necessary, in the market there are products (e.g. Gigamon) that allow to send GTP-C/U of selected IMSIs to a specific egress interface.

PF_RING [1/2]

- PF_RING is a home-grown open source packet processing framework for Linux.
- It is split in a kernel module and user-space libs.



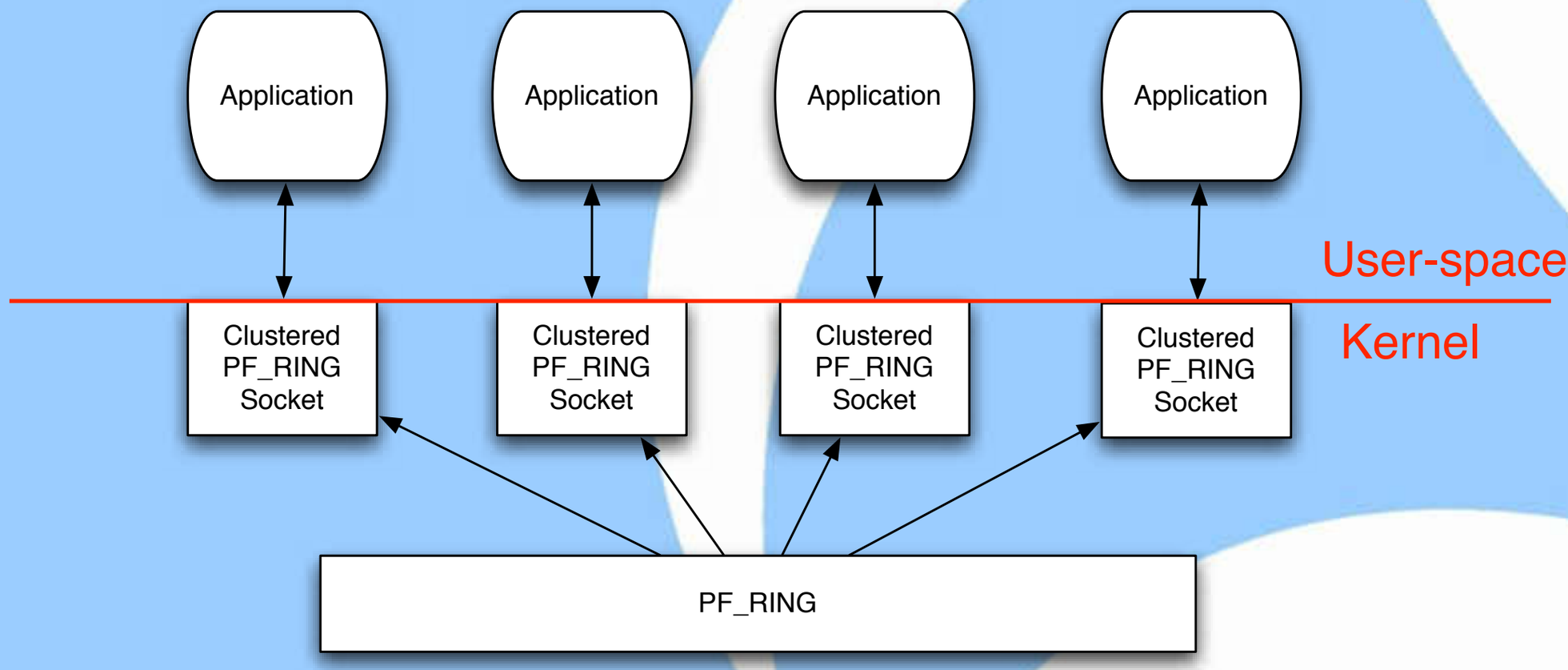
PF_RING [2/2]

It comes in two flavours:

- In-kernel packet capture: packets are received and processed inside the Linux kernel, then are dispatched using memory-map to user-space. This operating mode is good for1G interfaces.
- Zero-copy: once the device is open, packets are ready directly by user-space applications without passing through the kernel. This is the preferred solution for aggregate traffic over 4/5 Gbit.

PF_RING and GTP [1/5]

- The PF_RING kernel module comes with support for “ring clustering” that is the ability to federate applications each analysing a portion of the traffic.

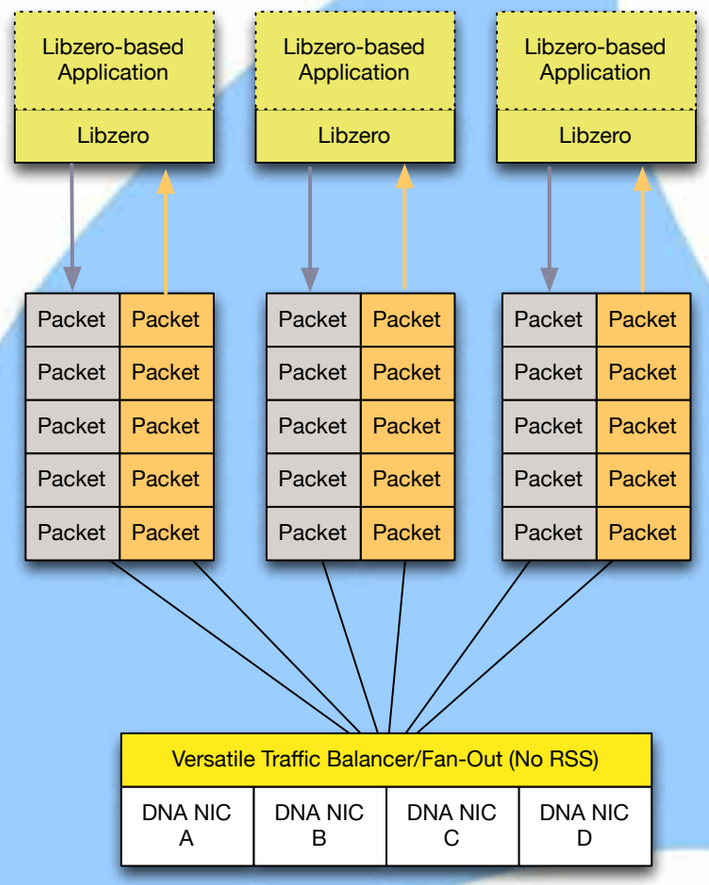


PF_RING and GTP [2/5]

- Applications (e.g. GTP probes) opening PF_RING sockets can merge (via PF_RING), traffic from multiple interfaces.
- Ingress traffic can be balanced in many ways (per-flow, round robin…) including 5 tuple (protocol, ip/port src/dst) that:
  - In case of GTP-U uses the user IP/ports
  - In case of GTP-C uses the GSNs IP/ports.

PF_RING and GTP [3/5]

- For 10G interfaces, in-kernel packet capture is overkilling and thus it is necessary to bypass it.
- On top of PF_RING ZC (Zero Copy) it is possible to efficiently read packets in zero-copy and distribute them to applications.



PF_RING and GTP [4/5]

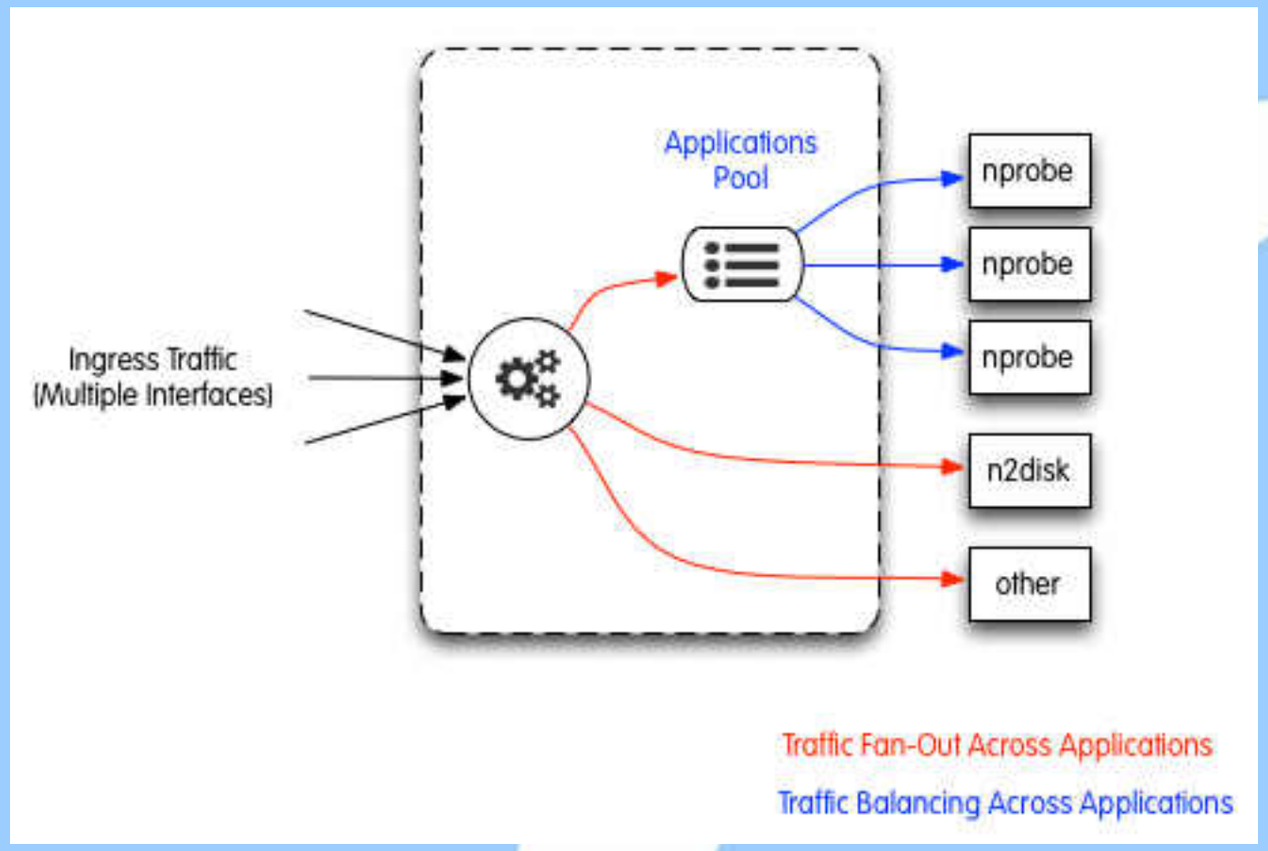
- Our software GTP balancer can aggregate and balance coherently GTP traffic from multiple 10G interfaces. It can support packet fan-out in zero-copy to multiple applications if necessary.

```
# DNALoadBalancer -h
Usage:
     DNABalancer [-h][-d][-a][-q] -c <id> -i <device>
     [-n <num app> | -o <devices>] [-f <path>]
     [-l <path>] [-p <path>] [-t <level>]
     [-g <core_id>]

-h              | Help
-c <id>         | Cluster id
-i <devices>    | Capture device names (comma-separated list) [Default: dna0]
-n <num app>    | Max number of slave applications (max 32) [Default: 1]
-o <devices>    | Egress device names (comma-separated list) for balancing packets across interfaces
-m <dissector>  | Dissector to enable:
                | 0 - n-tuple [Default]
                | 1 - GTP+RADIUS
                | 2 - GRE
-q              | Drop non GTP traffic (valid for GTP dissector only)
-r              | Reserve first application for signaling only
-s <type>       | n-Tuple type used by the hash algorithm:
                | 0 - <Src IP, Dst IP> [Default]
                | 1 - <Src IP, Dst IP, Src Port, Dst Port, Proto, VLAN>
                | 2 - <Src IP, Dst IP, Src Port, Dst Port, Proto, VLAN> with TCP, <Src IP, Dst IP> otherwise
-u <mode>       | Behaviour with tunnels:
                | 0 - Hash on tunnel content [Default]
                | 1 - Hash on tunnel ID
                | 2 - Hash on outer headers
-d              | Run as a daemon
-l <path>       | Log file path
-p <path>       | PID file path [Default: /tmp/dna_balancer.pid]
-t <level>      | Trace level (0..6) [Default: 0]
-g <core_id>    | Bind to a core
-a              | Active packet wait

Example:
     DNABalancer -i 'dna0,dna1' -c 5 -n 4
```

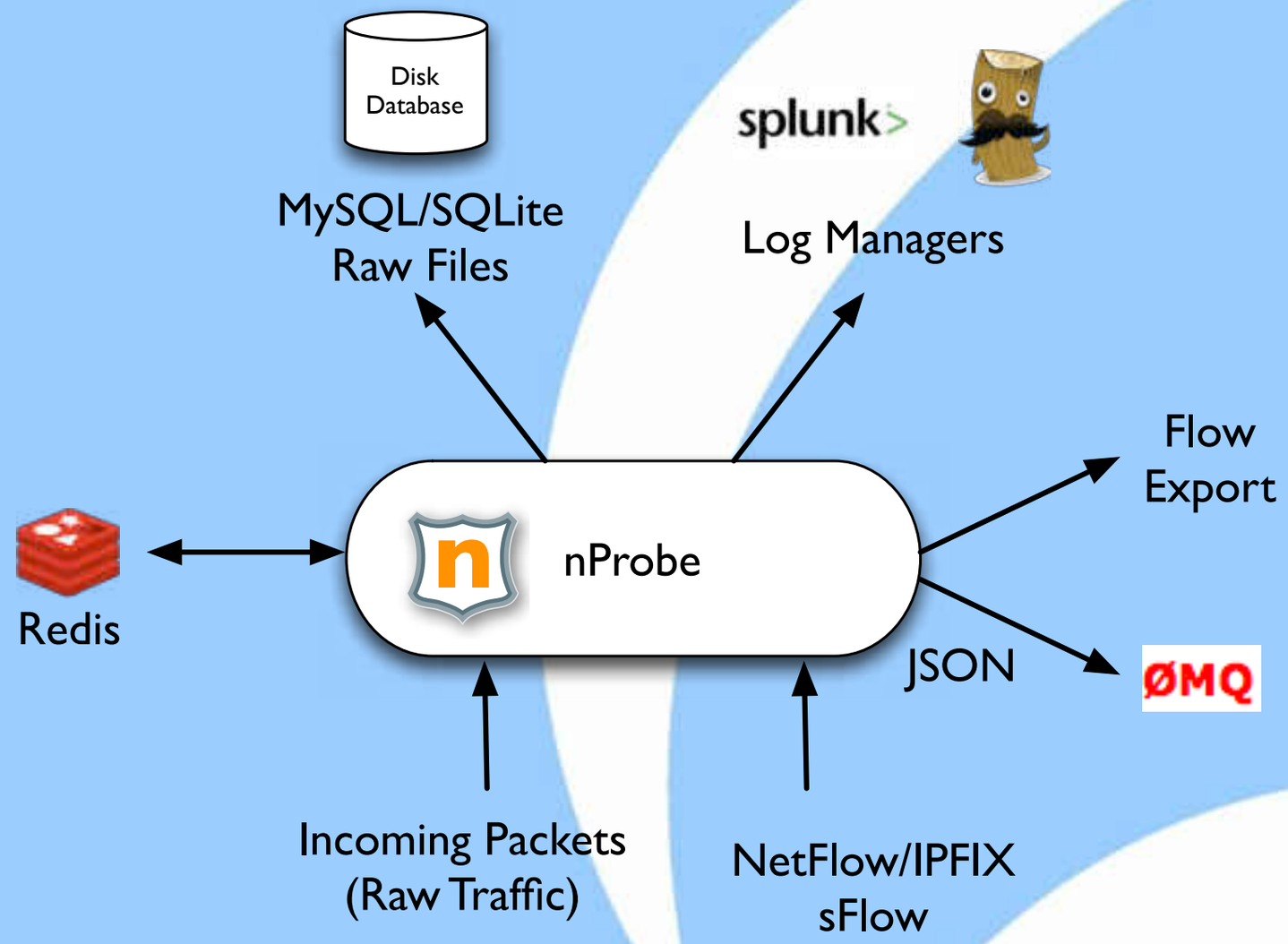
PF_RING and GTP [5/5]



- balancer -i “dna0,dna1,dna2” -c 10 -n 3,1,1 -r 0
- nprobe -i dnacluster:10@0 -g 1
- nprobe -i dnacluster:10@1 -g 2
- nprobe -i dnacluster:10@2 -g 3
- n2disk -i dnacluster:10@3 ….
- wireshark -i dnacluster:10@4 …

nProbe [1/3]

- nProbe is a high-speed (multi 10G) open source traffic probe/collector developed by ntop.



nProbe [2/3]

- Originally designed as a drop-in replacement of a physical NetFlow probe, currently it can:
  - Convert flow format (sFlow-to-NetFlow/IPFIX) or version (e.g. v5 to v9).
  - High-speed packet-to-flow processing.
  - Leverage on in-memory-databases to maintain flow state coherency.
  - Ability to pre-compute data for realtime traffic aggregation.

nProbe [3/3]

- It has an open architecture extensible by means of plugins that include:
  - GTP (v0, v1, v2) plugins.
  - VoIP (SIP and RTP) plugins for analysing voice signalling (who's calling who/when) and voice quality (Jitter and pseudo-MOS/R-Factor).
  - HTTP(S), Email (SMTP, IMAP, POP3), Radius, Database (Oracle and MySQL), FTP, DHCP, and BGP.

nProbe Flow Format [1/2]

- nProbe supports flexible NetFlow, that allows data export format to be customised at runtime.
- nProbe allow users to define a template on the command line.
- In addition to the standard fields (IP, port…), nProbe can export many other fields such as packet stats (TTL and size distribution), network/application latency, geolocation, packets retransmitted/out-of-order, tunnel information, and DPI (Deep Packet Inspection).

nProbe Flow Format [2/2]

- nProbe plugins also define datatypes that can be exported via NetFlow v9/IPFIX.
- Example of GTPv1 Datatypes:

```
Plugin GTPv1 Signaling Protocol templates:
[NFv9 57692][IPFIX 35632.220] %GTPV1_REQ_MSG_TYPE          GTPv1 Request Msg Type
[NFv9 57693][IPFIX 35632.221] %GTPV1_RSP_MSG_TYPE          GTPv1 Response Msg Type
[NFv9 57694][IPFIX 35632.222] %GTPV1_C2S_TEID_DATA         GTPv1 Client->Server TunnelId Data
[NFv9 57695][IPFIX 35632.223] %GTPV1_C2S_TEID_CTRL         GTPv1 Client->Server TunnelId Control
[NFv9 57696][IPFIX 35632.224] %GTPV1_S2C_TEID_DATA         GTPv1 Server->Client TunnelId Data
[NFv9 57697][IPFIX 35632.225] %GTPV1_S2C_TEID_CTRL         GTPv1 Server->Client TunnelId Control
[NFv9 57698][IPFIX 35632.226] %GTPV1_END_USER_IP           GTPv1 End User IP Address
[NFv9 57699][IPFIX 35632.227] %GTPV1_END_USER_IMSI         GTPv1 End User IMSI
[NFv9 57700][IPFIX 35632.228] %GTPV1_END_USER_MSISDN       GTPv1 End User MSISDN
[NFv9 57701][IPFIX 35632.229] %GTPV1_END_USER_IMEI         GTPv1 End User IMEI
[NFv9 57702][IPFIX 35632.230] %GTPV1_APN_NAME              GTPv1 APN Name
[NFv9 57703][IPFIX 35632.231] %GTPV1_RAI_MCC               GTPv1 RAI Mobile Country Code
[NFv9 57704][IPFIX 35632.232] %GTPV1_RAI_MNC               GTPv1 RAI Mobile Network Code
[NFv9 57814][IPFIX 35632.342] %GTPV1_RAI_LAC               GTPv1 RAI Location Area Code
[NFv9 57815][IPFIX 35632.343] %GTPV1_RAI_RAC               GTPv1 RAI Routing Area Code
[NFv9 57816][IPFIX 35632.344] %GTPV1_ULI_MCC               GTPv1 ULI Mobile Country Code
[NFv9 57817][IPFIX 35632.345] %GTPV1_ULI_MNC               GTPv1 ULI Mobile Network Code
[NFv9 57705][IPFIX 35632.233] %GTPV1_ULI_CELL_LAC          GTPv1 ULI Cell Location Area Code
[NFv9 57706][IPFIX 35632.234] %GTPV1_ULI_CELL_CI           GTPv1 ULI Cell CI
[NFv9 57707][IPFIX 35632.235] %GTPV1_ULI_SAC               GTPv1 ULI SAC
[NFv9 57804][IPFIX 35632.332] %GTPV1_RESPONSE_CAUSE        GTPv1 Cause of Operation
```

nDPI

- nDPI is a ntop-maintained GPLv3 deep packet inspection library used in nProbe to dissect traffic and thus classify application protocols.
- Supported protocols (~170) include:
  - P2P (Skype, BitTorrent)
  - Messaging (Viber, Whatsapp, MSN, The Facebook)
  - Multimedia (YouTube, Last.gm, iTunes)
  - Conferencing (Webex, CitrixOnLine)
  - Streaming (Zattoo, Icecast, Shoutcast, Netflix)
  - Business (VNC, RDP, Citrix, *SQL)

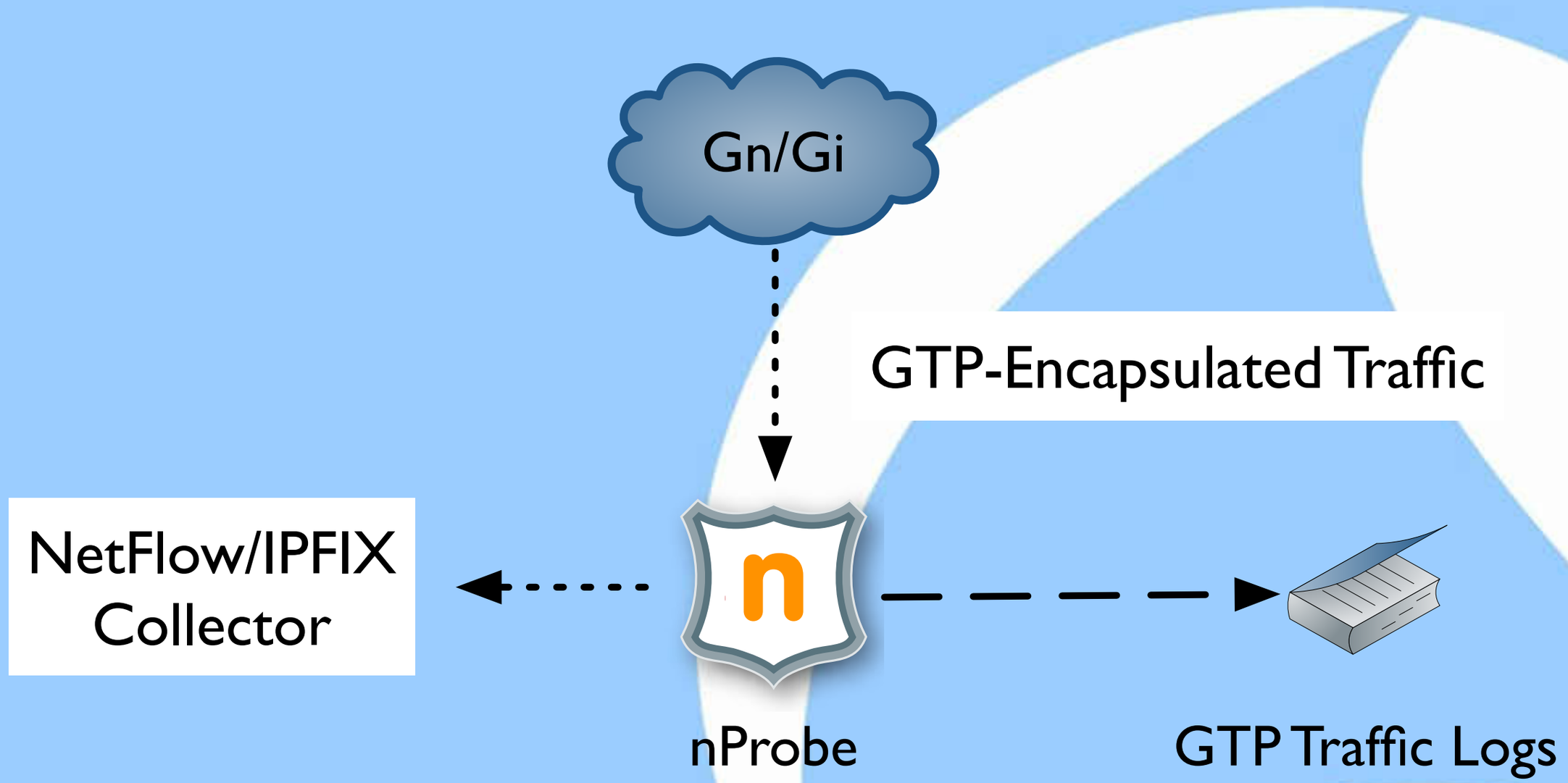
nProbe and GTP [1/5]

- The nProbe core is able to natively handle and decode tunnelled (e.g. PPP, PPPoE, GRE, L2TP, Mobile IP) and tagged packets (e.g. 802.1Q, MPLS), and thus generate flows on GTP-U packets.
- The GTP plugins are responsible for decoding GTP-C signalling packets, and save on in-memory-database (redis) signalling information such as TEID, Cell-Id, and IMSI.

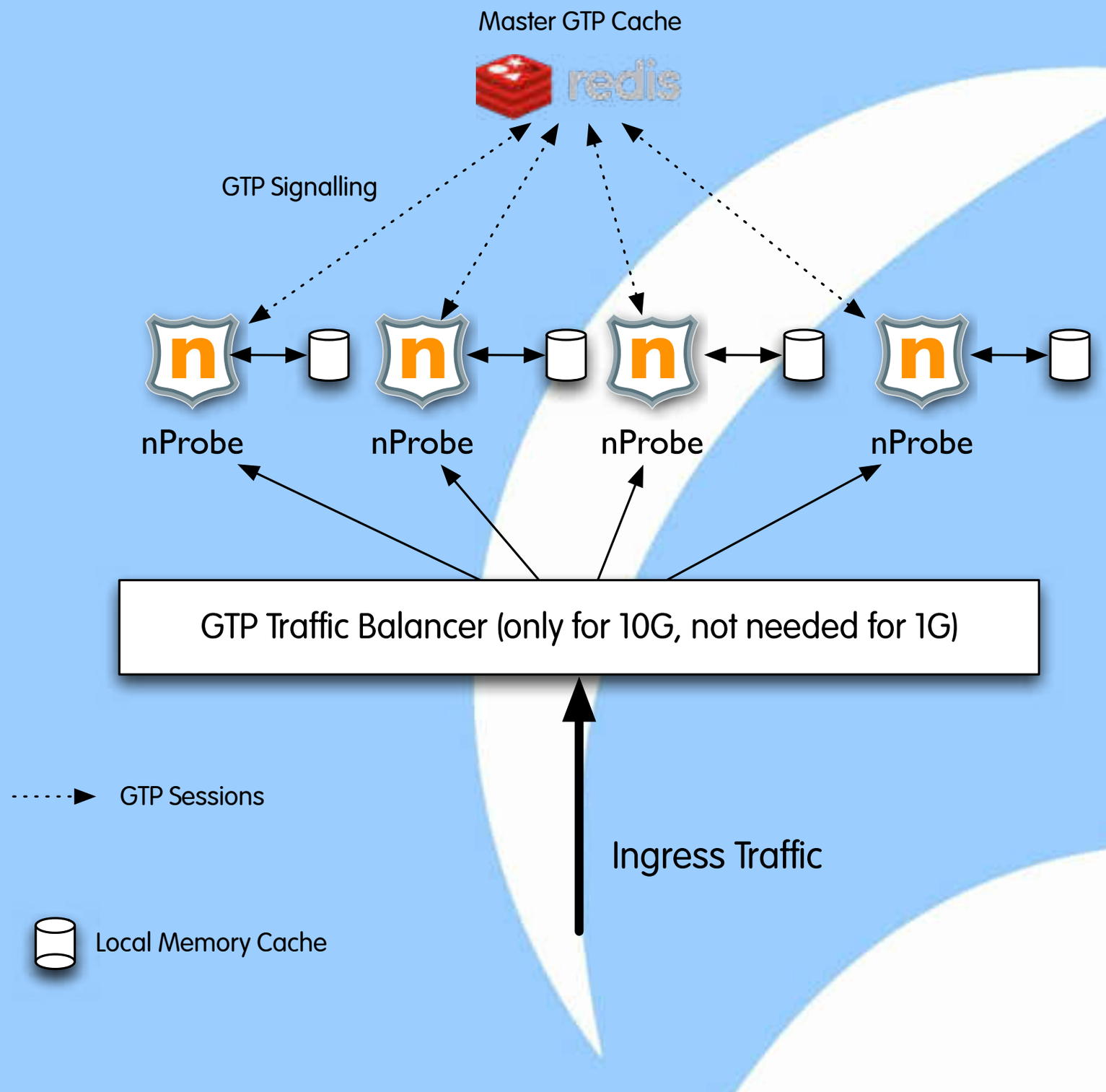
nProbe and GTP [2/5]

- User-to-traffic correlation is performed in realtime by the probe, and not (as often happens) by the collector often in post-processing (e.g. searching on a database) and thus not in realtime.
- GTP-C traffic is spread across all probes (i.e. it can be processed in parallel with respect to sending it to one probe) as well as GTP-U.
- GTP tunnel-to-user association is kept in redis and cached in individual probes for reducing the number of redis communications.

nProbe and GTP [3/5]

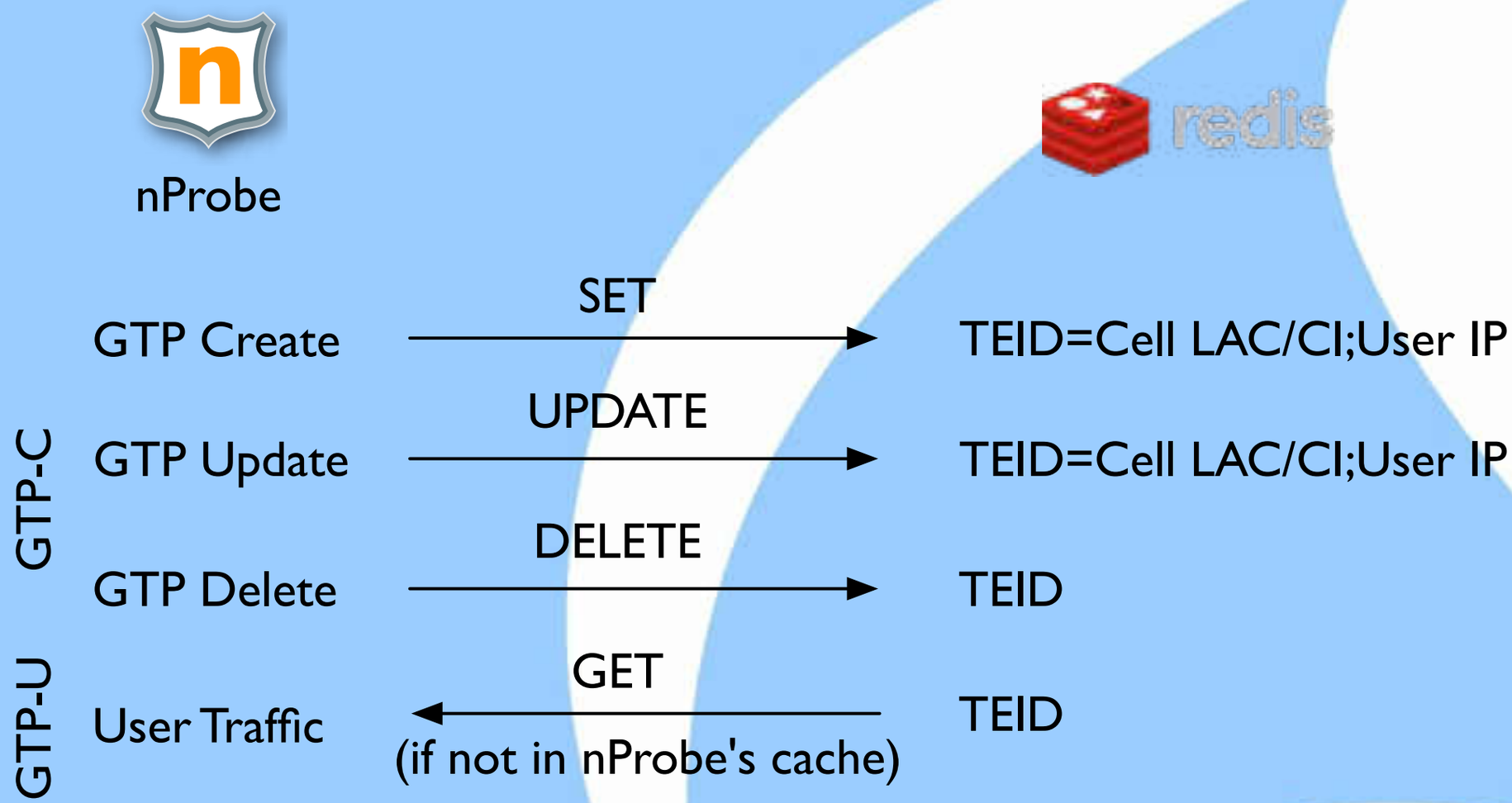


nProbe and GTP [4/5]



nProbe and GTP [5/5]

- Example of GTP cache manipulation:



nProbe and Redis [1/4]

- nProbe opens several connections to the redis cache.
  - Write to cache: these operations have lower priority and are queued and executed in batches.
  - Read from cache: high-priority synchronous (i.e. the probe cannot continue the operations until an answer is received) operation performed when the probe needs to export GTP-U data.

nProbe and Redis [2/4]

- In average a single redis process can handle up to 50/70k requests/sec in total.
- It is a good practice to run redis on the same host where the probes are working to avoid putting network latency into the equation and thus reduce the number of redis operations.
- In order to add redundancy and increase resiliency it is possible to setup a redis cluster or use a proxy (e.g. https://github.com/twitter/twemproxy) to share keys across the various nodes.

nProbe and Redis [3/4]

Advantages of using redis:

- The GTP state is kept on a central location regardless of the number of probes.
- It can be used to aggregate the traffic monitored by each individual probe. For instance if it is necessary to compute the total amount of traffic per cellId, nProbe can increment the cell-bytes counter after a flow is expired.

nProbe and Redis [4/4]

Disadvantages of using redis:

- It is a central point of failure thus it is important to use a cluster/proxy.
- Read operations are synchronous and thus they are influenced by the network latency: keep redis and the nProbe instances on the same host and do not put them on a WAN.

GTP Traffic Reports [1/3]

The probe can produce two type of reports:

- GTP-C reports that contain aggregate signalling information.
- GTP-U traffic reports that contain standard flow fields (e.g. IPs/ports, bytes/packets) in addition to mobile terminal information (e.g. IMSI and cell Id).

In essence nProbe handles GTP signalling similar to what happens with Radius: it is use to map a user (IMSI) to a traffic (flow) along with metadata (e.g cell Id).

GTP Traffic Reports [2/3]

- GTP-C report

```
#
# StartTime[epoch]  Duration(ms)[float] GTP_version[uint]   Peers[ascii:64]  SeqId[hex:4]
RspCause[ascii:64]  c2s_s2c_msg_type[ascii:64] c2s_s2c_teid[hex:20]  c2s_s2c_teid_data[hex:20]
c2s_s2c_teid_ctrl[ascii:32]   c2s_gsn_addr[ascii:32]    APN[ascii:64]    IMSI[ascii:32]
MSISDN[ascii:32] IMEI[ascii:32]   NSAPI[uint]   rai_mcc[uint]    rai_mnc[uint]    rai_lac[uint]
rai_rac[uint]uli_mcc[uint]uli_mnc[uint]uli_cell_lac[uint]  uli_cell_ci[uint]uli_sac[uint]
s2c_gsn_addr[ascii:32]  s2c_end_user_ip[ascii:32]   s2c_charging_gw[ascii:32] s2c_charging_id[uint]
Req_QoS[ascii:255]  Rsp_QoS[ascii:255]
#
1399325474   0.000  1  194.33.24.58,194.33.27.26   45  Request accepted(128)
CreateContextRequest(16),CreateContextResponse(17)  00000000,0600170A06001908,000000010600170A,
00000044  194.33.24.58,194.33.24.52 orange 23486000002XXXX  +44797335XXXX        5   0   0   0   0
   0   0   0   0   0   194.33.26.17,194.33.26.17  10.32.0.36   0.0.0.0  12481267
delay=4,reliability=3,peak=6,precedence=2,mean=31,class=4,del_order=2,del_err_sdu=3,max_sdu=1500,max
_ul=64,max_dl=384,res_ber=7,err_ratio=4,transfer_delay=2,traf_prio=3,guar_ul=16,guar_dl=64,src_stat_
desc=0,sig_ind=0
delay=4,reliability=3,peak=6,precedence=2,mean=31,class=4,del_order=2,del_err_sdu=3,max_sdu=1500,max
_ul=64,max_dl=384,res_ber=7,err_ratio=4,transfer_delay=2,traf_prio=3,guar_ul=16,guar_dl=64,src_stat_
desc=0,sig_ind=0
1399325474   0.000  1  194.33.24.58,194.33.26.17   46  Request accepted(128)
DeleteContextRequest(20),DeleteContextResponse(21)  00000044,0600170A00000000,00000000
00000000,00000000   0.0.0.0,0.0.0.0        0.0.0.0,0.0.0.0  0.0.0.0  0.0.0.0  0
delay=0,reliability=0,peak=0,precedence=0,mean=0,class=0,del_order=0,del_err_sdu=0,max_sdu=0,max_ul=
0,max_dl=0,res_ber=0,err_ratio=0,transfer_delay=0,traf_prio=0,guar_ul=0,guar_dl=0,src_stat_desc=0,si
g_ind=0
delay=0,reliability=0,peak=0,precedence=0,mean=0,class=0,del_order=0,del_err_sdu=0,max_sdu=0,max_ul=
0,max_dl=0,res_ber=0,err_ratio=0,transfer_delay=0,traf_prio=0,guar_ul=0,guar_dl=0,src_stat_desc=0,si
g_ind=0
```

GTP Traffic Reports [3/3]

- GTP-U Report (HTTP)

```
#
# Client[ascii:32]  Server[ascii:32] Protocol[ascii:8]  Method[ascii:8]  URL[ascii:255]
HTTPReturnCode[uint]   Location[ascii:255] Referer[ascii:255]   UserAgent[ascii:255]
ContentType[ascii:96]  Bytes[uint]  BeginTime[epoch]   EndTimeWithPayload[epoch]
FlowHash[ascii:16]  Cookie[ascii:255]   Terminator[ascii:4] ApplLatency(ms)[uint]
ClientLatency(ms)[uint]ServerLatency(ms)[uint] ApplicationID[uint]  Application[ascii:32]
BalancerHost[ascii:32]  ServerIP[ascii:32]  RehttpPkts[uint] Client2Server_TEID[ascii:8]
Server2Client_TEID[ascii:8]   FlowUserName (User or IMSI/LAC/CCI/CSAC)[ascii:32]
AdditionalInfo[ascii:32]   OOO_Cli2Srv[uint]    OOO_Svr2Cli[uint]    POSTParams[ascii:256]
#

10.68.78.155 web.icq.com  http  GET   302   www.icq.com/whitepages/online?
icq=601004214&img=5 www.google.com  Mozilla/5.0 (Linux; U; Android 4.1.2; ru-ru; GT-I9300
Build/JZO54K) AppleWebKit/534.30 (KHTML, like Gecko) Version/4.0 Mobile Safari/534.30 text/
html  1170  1399325662   1399325664   838964263  0   U   40.173  23.091  20.200  7
HTTP     205.188.95.190  1  86048DB0  0001D69D   "25001686073XXXX;592;54073;0;183493424"
      0  0
```

Advanced GTP Monitoring [1/2]

- The use of redis allows the simple creation of application that would have been much more difficult to create otherwise.
- Suppose you want to have every 5 minutes the total amount of application protocol traffic per IMSI. Namely you want to know for all active network users, what are the protocols in use (e.g. HTTP, email, Whatspp, Skype) and the traffic volume for each of it.

Advanced GTP Monitoring [2/3]

- Using a specified command line option (—imsi-aggregation) it is possible to tell nProbe that when a flow expires, it must increment in redis for the <5 mins epoch><IMSI> hash key, the number of bytes of the subkey <application protocol>.
- Redis guarantees that increment operations are properly executed even if two simultaneous clients want to increment the same key.
- Periodically a simple python script can access redis and read the aggregated values.

Advanced GTP Monitoring [3/3]

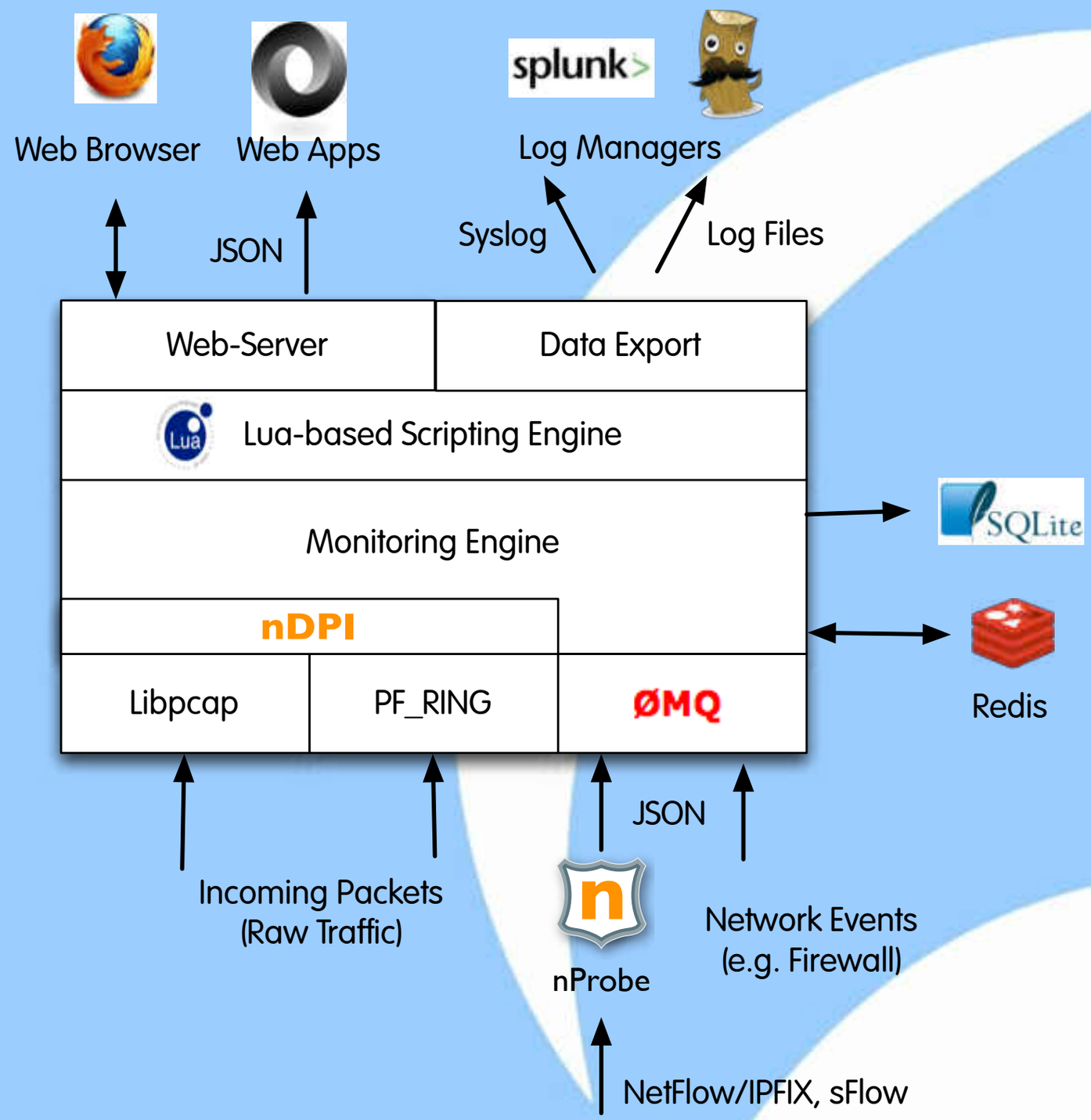
- Companion script

```
$ crontab -l|grep ggrega
*/5 * * * * /home/imsi/imsiAggregator.py --redis localhost
--epoch -2 --outdir /export/working_dir/imsi
```

- Example of traffic reports

```
#
# Timestamp IMSI Granularity Protocol Packets Bytes Flows Duration
#
1374938100 XXXXX2001106796 300 Unknown 3 298 2 2
1374938100 XXXXX1100485374 300 HTTP 393 283553 13 114
1374938100 XXXXX2001110729 300 SSL 49 14269 10 18
1374938100 XXXXX2001338233 300 Skype 15 1411 1 7
1374938100 XXXXX1101335045 300 DNS 2 385 1 1
1374938100 XXXXX2001931139 300 Viber 17 1487 4 35
```

Using ntopng as Monitoring Console [1/2]



Using ntopng as Monitoring Console [2/2]

Active Flows

10 Applications

Info	Application	L4 Proto	Client	Server	Duration	Breakdown	Throughput	Total Bytes
Info	GTP	UDP	194.33.24.58 :2123	194.33.27.26 :2123	2 sec	Client Server	0 bps	264 Bytes
Info	HTTP	TCP	10.32.0.36 :51546	10.250.17.12:80	1 sec	Client	0 bps	60 Bytes

ntop

Home Flows Hosts Interfaces

Flow: 10.32.0.36:51546 10.250.17.12:80 Overview

Flow Peers	10.32.0.36:51546 10.250.17.12:80	
Protocol	TCP / HTTP	
First / Last Seen	16/06/2014 08:30:35 [6 min, 29 sec ago]	16/06/2014 08:30:35 [6 min, 29 sec ago]
Total Traffic Volume	60.00 Bytes	
Client vs Server Traffic Breakdown	10.32.0.36	
Client to Server Traffic	1 Pkts / 60.00 Bytes	
Server to Client Traffic	0 Pkts / 0 Bytes	
TCP Flags	SYN This flow is active.	
Actual Throughput	0 bps	
Additional Flow Elements		
Input interface SNMP idx	0	
Total number of exported flows	3	
Flow Username		

IMSI (International mobile Subscriber Identity)	234860000026315 [United Kingdom]
NSAPI	5
GSM Cell LAC (Location Area Code)	1010
GSM Cell Identifier	3456
SAC (Service Area Code)	0
IP Address	10.32.0.36

Final Remarks

In this talk we have learnt:

- Introduction to mobile traffic monitoring.
- How to use Wireshark to analyse mobile traffic.
- Versatile software traffic merging/balancing using PF_RING.
- Permanent GTP traffic monitoring using the open-source nProbe application.

  Thanks to Wireshark and the open source ntop tools presented, it is possible to effectively analyse mobile traffic without using costly proprietary tools.