Bytes and Counters
Deep Packet Inspection
System Introspection

Luca Deri <deri@ntop.org>
@lucaderi

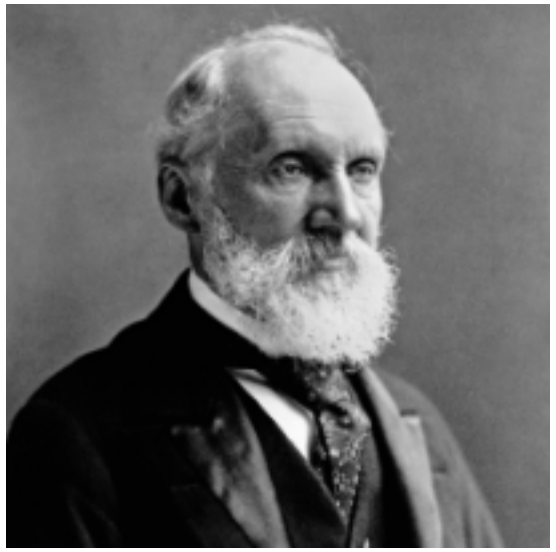
Samuele Sabella <sabella@ntop.org>

What is Network Traffic Monitoring?

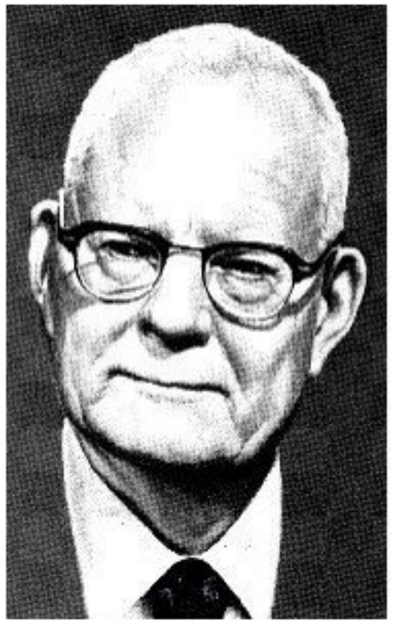
- The key objective behind network traffic monitoring is to *ensure availability and smooth operations on a computer network*. Network monitoring incorporates network sniffing and packet capturing techniques in monitoring a network. Network traffic monitoring generally requires reviewing each incoming and outgoing packet.

https://www.techopedia.com/definition/29977/network-traffic-monitoring

Motivation For Traffic Monitoring



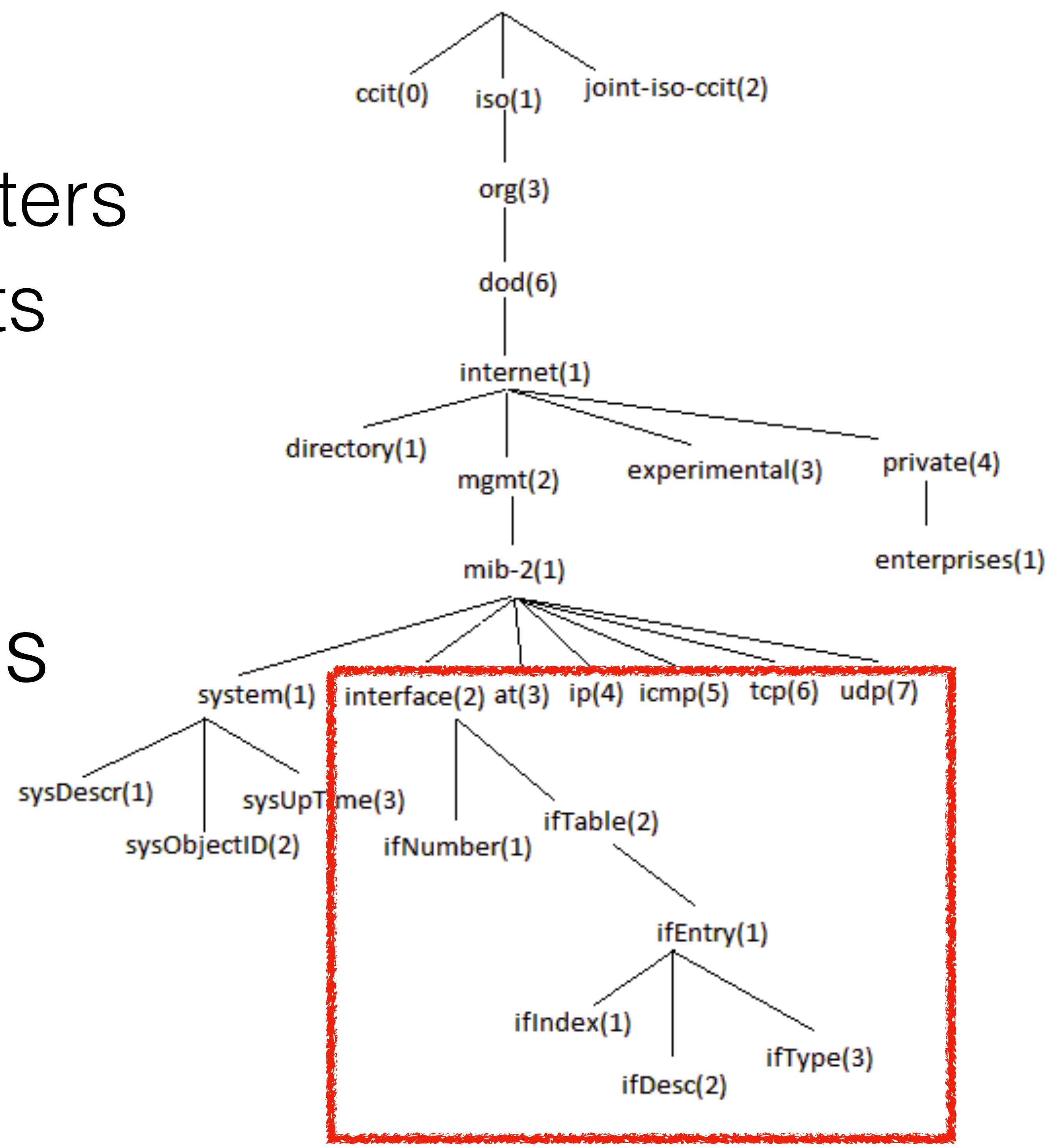
If you can't measure it, you can't improve it
(Lord Kelvin, 1824 – 1907)



Without data you're just another persone with an opinion
(W. Edwards Deming, 1900 – 1993)

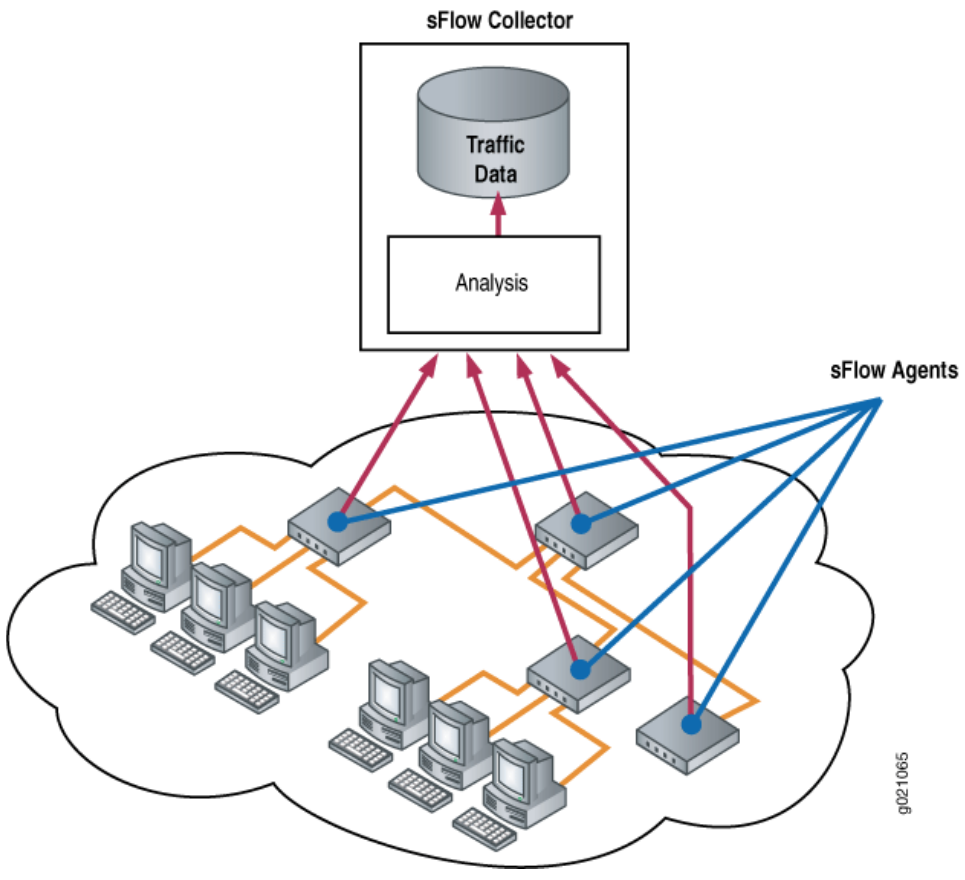
What Metrics? [1/3]

- 1988: SNMPv1
  - Interface byte/counters
  - TCP/UDP/ICMP stats
- 2000: RMON
- Aggregated values



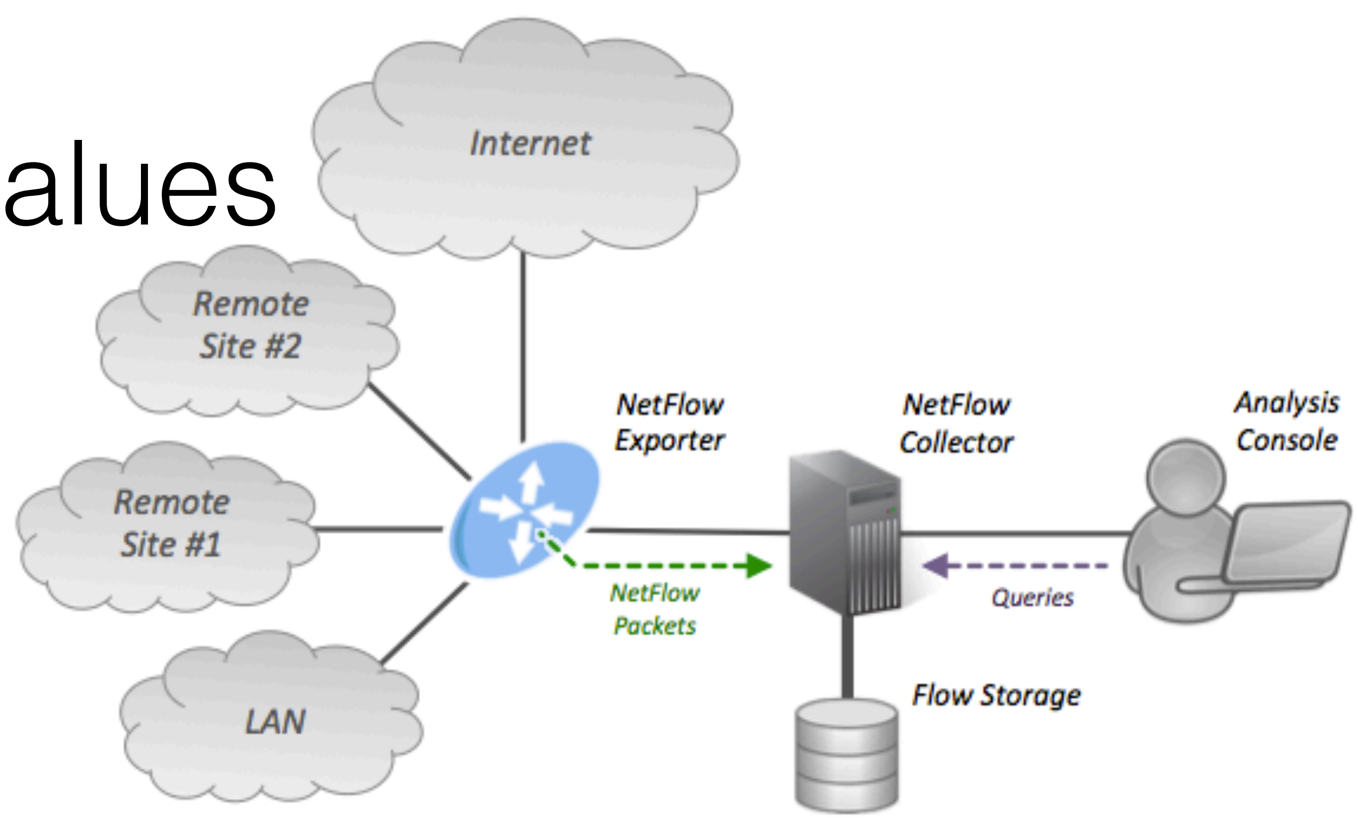
What Metrics? [2/3]

- 2001: sFlow, a network monitoring technology for switch (and also routed) network based on statistical sampling.
- Exports:
  - Flow samples
  - SNMP Interface counters



What Metrics? [3/3]

- 2004: NetFlow v9
  - Flow: 5-tuple with unique key
  - Flow byte/counters
- 2008: IPFIX
- Average counter values (Flow duration 1 min+)



Poor Metrics: The nProbe Way

- nProbe (2002-today) is a home-grown flow probe/collector that tried to provide better metrics leveraging on NetFlow/IPFIX extensibility. Metrics include:
    - Tunnelling/encapsulation information
    - Geolocation: AS, city
    - Behaviour: fragments, per-second stats
    - TCP stats: window scaling, OOO, retransmissions
    - Packet distribution: size, TTL

More Visibility: DPI

- Deep packet inspection is a technique that identifies the application protocol by looking at packet payload.
- Since 2012 ntop developed nDPI, an opensource extensible DPI toolkit featuring ~240 protocols.

```
05/May/2019 11:28:03 [engine.c:3074] Emitting Flow: [<-][tcp] 185.60.216.53:443 ->
192.168.2.29:49674 [149 pkt/13624 bytes][ifIdx 0->0][0.0 sec][SSL.WhatsAppFiles/242][AS: 32934 -
> 0]
05/May/2019 11:28:03 [engine.c:3047] Emitting Flow: [->][tcp] 127.0.0.1:59604 -> 127.0.0.1:11211
[6 pkt/327 bytes][ifIdx 0->0][0.0 sec][Memcached/40][init 127.0.0.1][AS: 0 -> 0]
05/May/2019 11:28:03 [engine.c:3074]Emitting Flow: [->][tcp] 192.168.12.114:41517 ->
54.192.27.217:443 [11 pkt/2744 bytes][ifIdx 0->0][0.0 sec][SSL.Nintendo/173][init
192.168.12.114][AS: 0 -> 0]
```

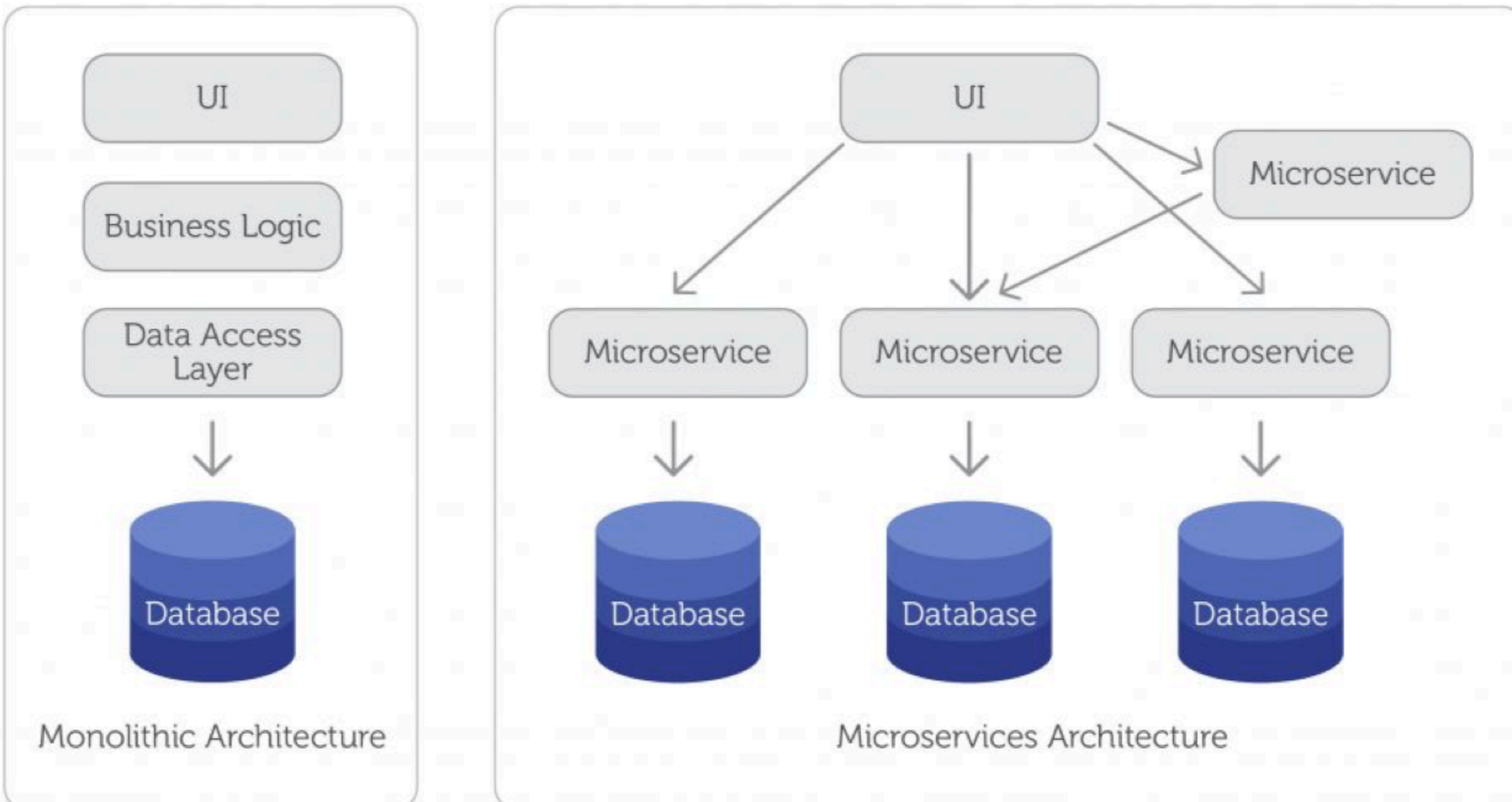
Is DPI The Ultimate Visibility Tool?

- While DPI can make application developers aware of the traffic flowing into their networks it does not solve many problems including:
  - What application has generated a given traffic? An IDS that identifies nasty traffic without being able to traceback the problem is not that useful.
  - What user has spawn the above application?
  - What are the real metrics the system sees (e.g. latency) and thus tunes its speed according to them?

Our Challenges

- Bind network events to system activities.
- Provide application visibility to network flows. BTW this improves DPI (e.g. if skype.exe generated traffic X, then DPI(X)=Skype).
- Understand system interactions while performing a network action (e.g. open a web page).
- In essence: can we merge network and system monitoring to improve visibility ?

From Monolith to Microservices [1/2]



From Monolith to Microservices [2/2]

- Code of *each microservice* is stored in an *isolated container*, runs its own memory space, and functions independently.
- Scaling of one component is possible.
- Clearly organised architecture. Decoupled units have their specific jobs, can be reconfigured without affecting the entire application.
- Deployments don't require downtime.
- If a microservice crashes, the rest of the system keeps going.
- Each microservice can be scaled individually according to its needs.
- Services can use different tech stacks (developers are *free* to code in any language).

What's Wrong with Packets on Containerised Environments?

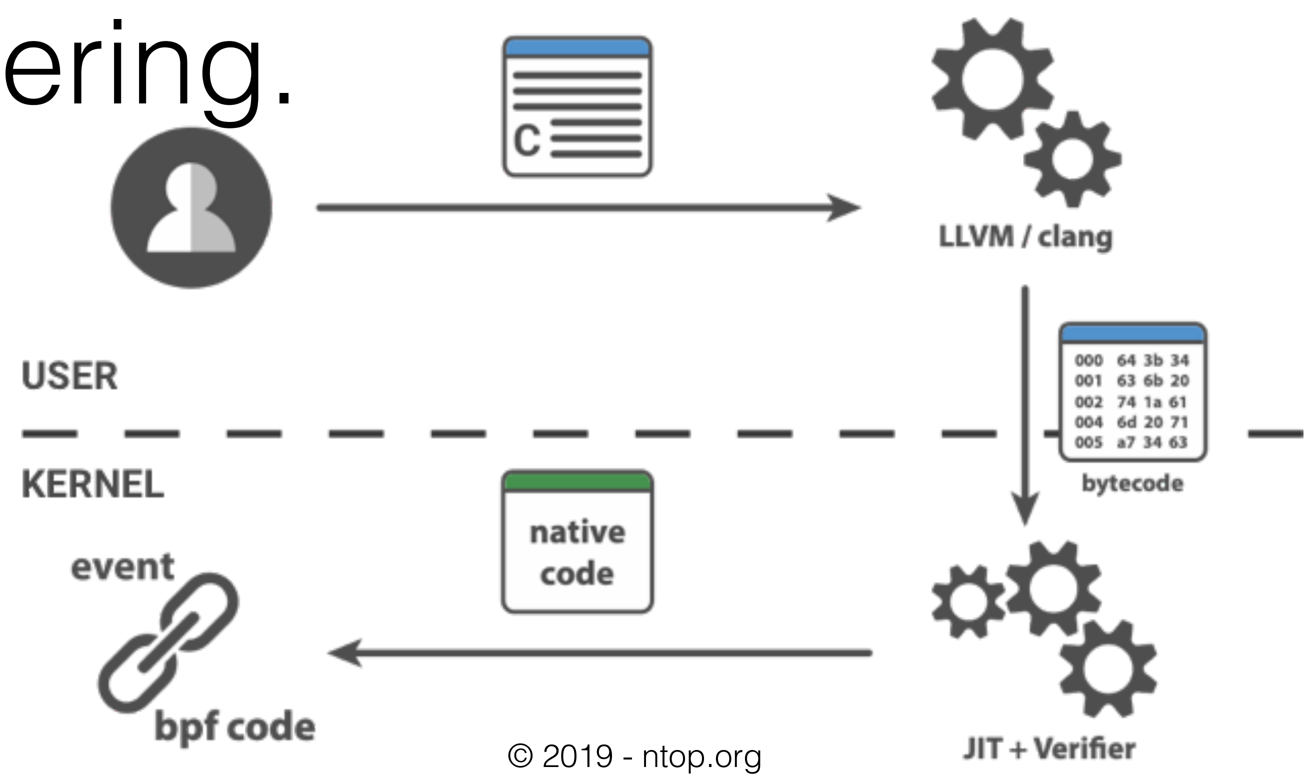
- Container lifecycle and cardinality changes according to the workload.
- Each container has a virtual ethernet interface so commands such as "tcpdump -i veth40297a6" won't help as devops think in terms of container name, pod and namespace rather than veth.
- Intra-container traffic stays inside the system without hitting the wire, thus monitoring traffic from/to the host won't help.

From Challenges to Solutions

- Enhance network visibility with system introspection.
- Handle virtualisation as first citizen and don't be blind (yes we want to see containers interaction).
- Complete our monitoring journey and…
    - System Events: processes, users, containers.
    - Flows
    - Packets
- …bind system events to network traffic for enabling continuous drill down: system events uncorrelated with network traffic are basically useless.

Welcome to eBPF

- In 1997, it was introduced in Linux kernel as a technology for in-kernel packet filtering.
- eBPF extended the original BPF virtual machine, allowing it to process other kind of events execute various actions other than packet filtering.



Why eBPF is Interesting for Monitoring

- It gives the ability to avoid sending everything to user-space but perform in kernel computations and send metrics to user-space.
- We can track more than system calls (i.e. be notified when there is a transmission on a TCP connection without analyzing packets).
- It is part of modern Linux systems (i.e. no kernel module needed).

libebpflow: eBPF for System Visibility

- Our aim has been to create an open-source library that offers a simple way to interact with eBPF network events in a transparent way.
- Reliable and trustworthy information on the status of the system when events take place.
- Low overhead event-based monitoring
- Information on users, network statistics, containers and processes
- Go and C/C++ support
- https://github.com/ntop/libebpfflow (GNU LGPL)

nProbe Mini: A Container-aware Probe

- The original nProbe has been extended with libebpf and Netlink support for exporting network traffic information and statistics.

	eBPF	Netlink
Availability	Modern Linux (Centos 7, Ubuntu 16.04+)	Any Linux
Admin Rights	Root	Any User
Purpose	Provide information about traffic flows (creation, deletion, and updates such as retransmissions).	Periodic network status (e.g. established connections) and traffic statistics (e.g. interface stats)

nProbe Mini: Monitoring Features

- Ability to track (TCP and UDP, IPv4/v6):
  - Flow creation/deletion
  - Periodic events (e.g. in case of TCP retransmission)
  - Periodic flow counter export
  - Container/process/user to traffic
- Minimum CPU/memory requirements

```
top - 13:04:42 up 19 days,  3:05,  4 users,  load average: 0.33, 0.23, 0.25
Tasks:   1 total,   0 running,   1 sleeping,   0 stopped,   0 zombie
%Cpu(s):  1.5 us,  0.8 sy,  0.0 ni, 97.6 id,  0.1 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem : 16366308 total,   179176 free, 10970808 used,  5216324 buff/cache
KiB Swap: 16716796 total, 14830332 free,  1886464 used.  4970688 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
 4760 root      20   0  332132 110604  43372 S   0.7  0.7   1:56.50 nprobe_mini
```

Flow Information: eBPF

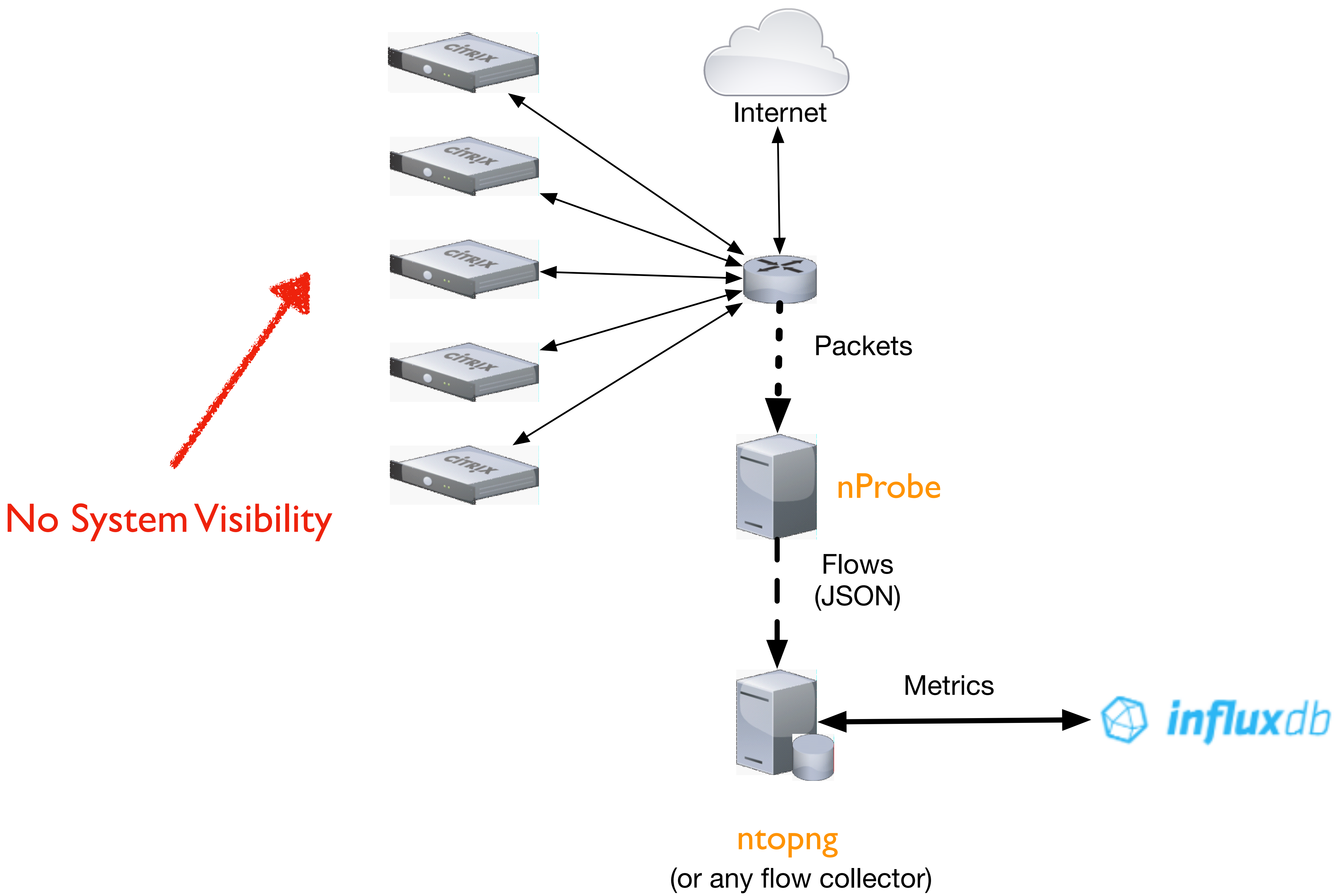

```
{
        "timestamp": "1556532359.110896",
        "LOCAL_PROCESS": {
                "PID": 8950,
                "UID": 100,
                "GID": 65534,
                "PROCESS_PATH": "\/usr\/lib\/apt\/methods\/http"
        },
        "LOCAL_FATHER_PROCESS": {
                "PID": 8947,
                "UID": 0,
                "GID": 0,
                "PROCESS_PATH": "\/usr\/bin\/apt-get"
        },
        "EVENT_TYPE": "SEND",
        "IP_PROTOCOL_VERSION": 4,
        "PROTOCOL": 17,
        "L4_LOCAL_PORT": 57756,
        "L4_REMOTE_PORT": 53,
        "IPV4_LOCAL_ADDR": "192.12.193.11",
        "IPV4_REMOTE_ADDR": "192.12.192.6",
        "LOCAL_CONTAINER": {
                "DOCKER": {
                        "ID": "cf5485c07181",
                        "NAME": "docker_monitor"
                }
        },
        "EXPORTER_IPV4_ADDRESS": "192.12.193.11"
}
```

Flow Information: Netlink

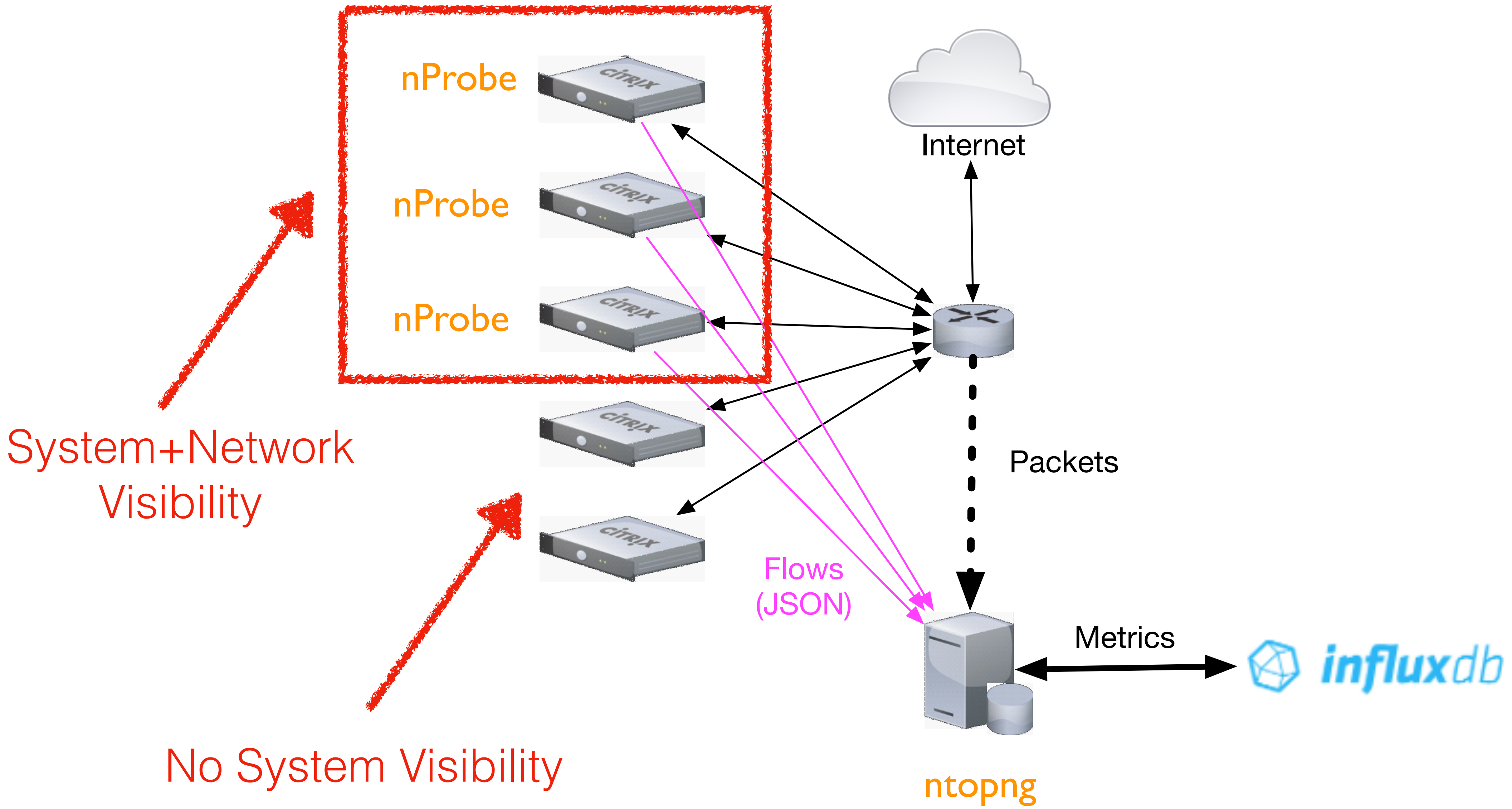

```
{
    "timestamp": "1556532168.971859",
    "PROTOCOL": 6,
    "IP_PROTOCOL_VERSION": 6,
    "USER_NAME": "deri",
    "IPV6_LOCAL_ADDR": "2a00:d40:1:3:x:x:x:x",
    "IPV6_REMOTE_ADDR": "2a00:d40:1:1:x:x:x:x",
    "L4_LOCAL_PORT": 41234,
    "L4_REMOTE_PORT": 22,
    "TCP": {
        "CONN_STATE": "ESTABLISHED",
        "RETRAN_PKTS": 0,
        "UNACK_SEGMENTS": 0,
        "LOST_PKTS": 0,
        "SEGS_IN": 3786,
        "SEGS_OUT": 5426,
        "BYTES_RCVD": 378173,
        "RTT": 4.1440,
        "RTT_VARIANCE": 5.3220,
        "CURRENT_RATE": 55125152.0,
        "DELIVERY_RATE": 6720000.0
    },
    "LOCAL_PROCESS": {
        "PID": 22581,
        "UID": 1000,
        "UID_NAME": "deri",
        "GID": 1000,
        "GID_NAME": "deri",
        "VM_SIZE": 53704,
        "VM_PEAK": 53876,
        "PROCESS_PATH": "\/usr\/bin\/ssh"
    },
    "LOCAL_FATHER_PROCESS": {
        "PID": 8562,
        "UID": 1000,
        "UID_NAME": "deri",
        "GID": 1000,
        "GID_NAME": "deri",
        "VM_SIZE": 21468,
        "VM_PEAK": 21468,
        "PROCESS_PATH": "\/bin\/tcsh"
    },
    "LOCAL_CONTAINER": {
        "DOCKER": {
            "NAME": "docker_monitor"
        }
    },
    "EXPORTER_IPV4_ADDRESS": “192.12.193.11"
}
```


Read from Kernel

Packet-only Deployment

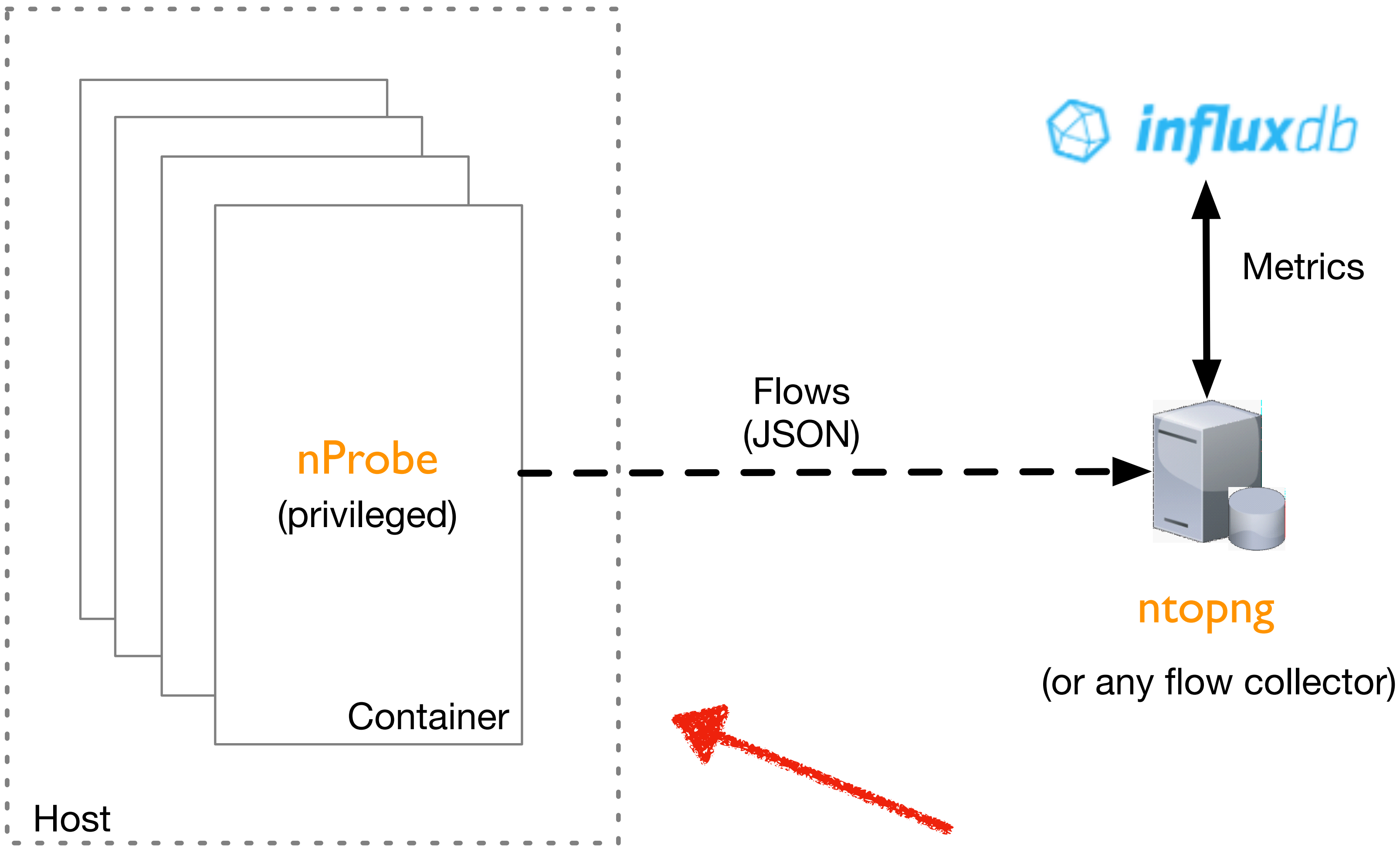


Packets + Metadata Deployment



Remember this acronym

Packetless Deployment

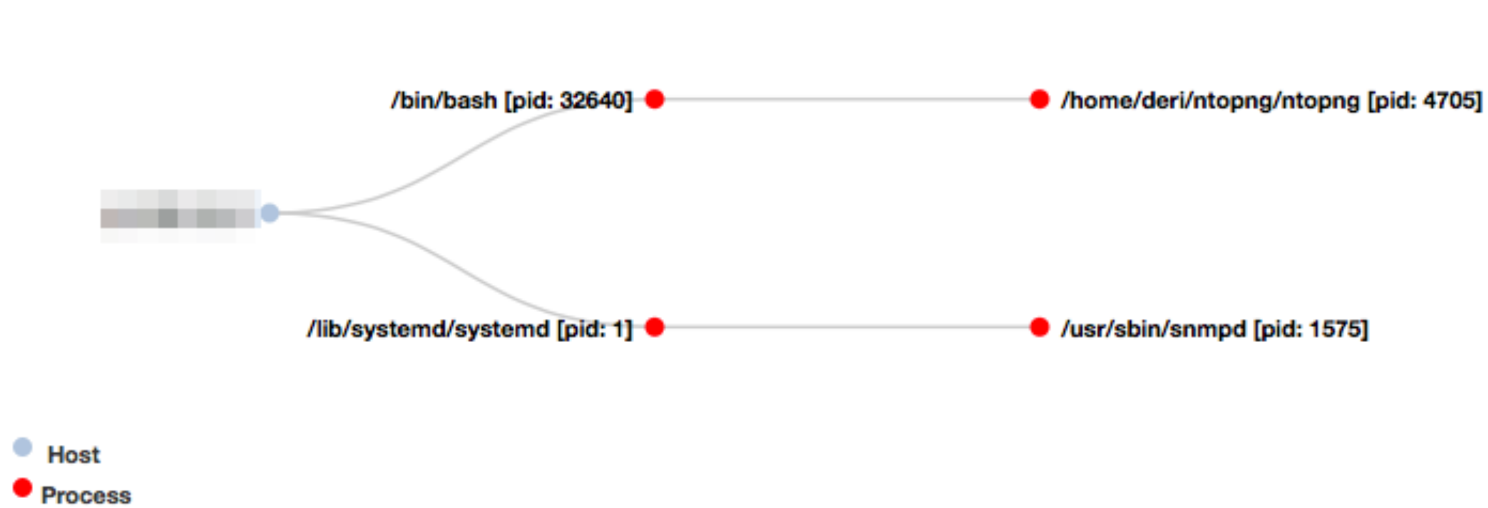


Full Visibility
(need 1 nProbe container per system)

ntopng: Process Hierarchy

Flow: [illegible]:54748 ⇄ [illegible]:161 | Overview | ↩

Flow Peers [Client / Server]	[illegible]:54748 ⇄ [illegible]:161	
Protocol / Application	UDP / SNMP (Network)	
First / Last Seen	29/04/2019 12:48:21 [< 1 sec ago]	29/04/2019 12:48:21 [< 1 sec ago]



Client Process Information	
User Name	nobody
Process PID/Name	/home/deri/ntopng/ntopng [pid: 4705] son of /bin/bash [pid: 32640]
Server Process Information	
User Name	Debian-snmp
Process PID/Name	/usr/sbin/snmpd [pid: 1575] son of /lib/systemd/systemd [pid: 1]

ntopng: Containers/Pod Overview

Containers List

10

	Container	Flows as Client	Flows as Server	Avg RTT as Client	Avg RTT as Server	Avg RTT Variance as Client	Avg RTT Variance as Server
Flows	23f492221784	4		< 0.1 ms		< 0.1 ms	
Flows	3dd3f17ad9ba	1					

Showing 1 to 2 of 2 rows

Pods List

10

Pod	Containers	Flows as Client	Flows as Server	Avg RTT as Client	Avg RTT as Server	Avg RTT Variance as Client	Avg RTT Variance as Server
heapster-v1.5.2-6b5d7b57f9-g2cjz	4	12	2	3.6 ms	6.6 ms	5.0 ms	8.0 ms
kube-dns-6bfbdd666c-jjt75	3	136	110				
kubernetes-dashboard-6fd7f9c494-hpcx7	1	8	3	0.8 ms		0.8 ms	
monitoring-influxdb-grafana-v4-78777c64c8-kmjr4	2	13	2				

ntopng: Container Flows

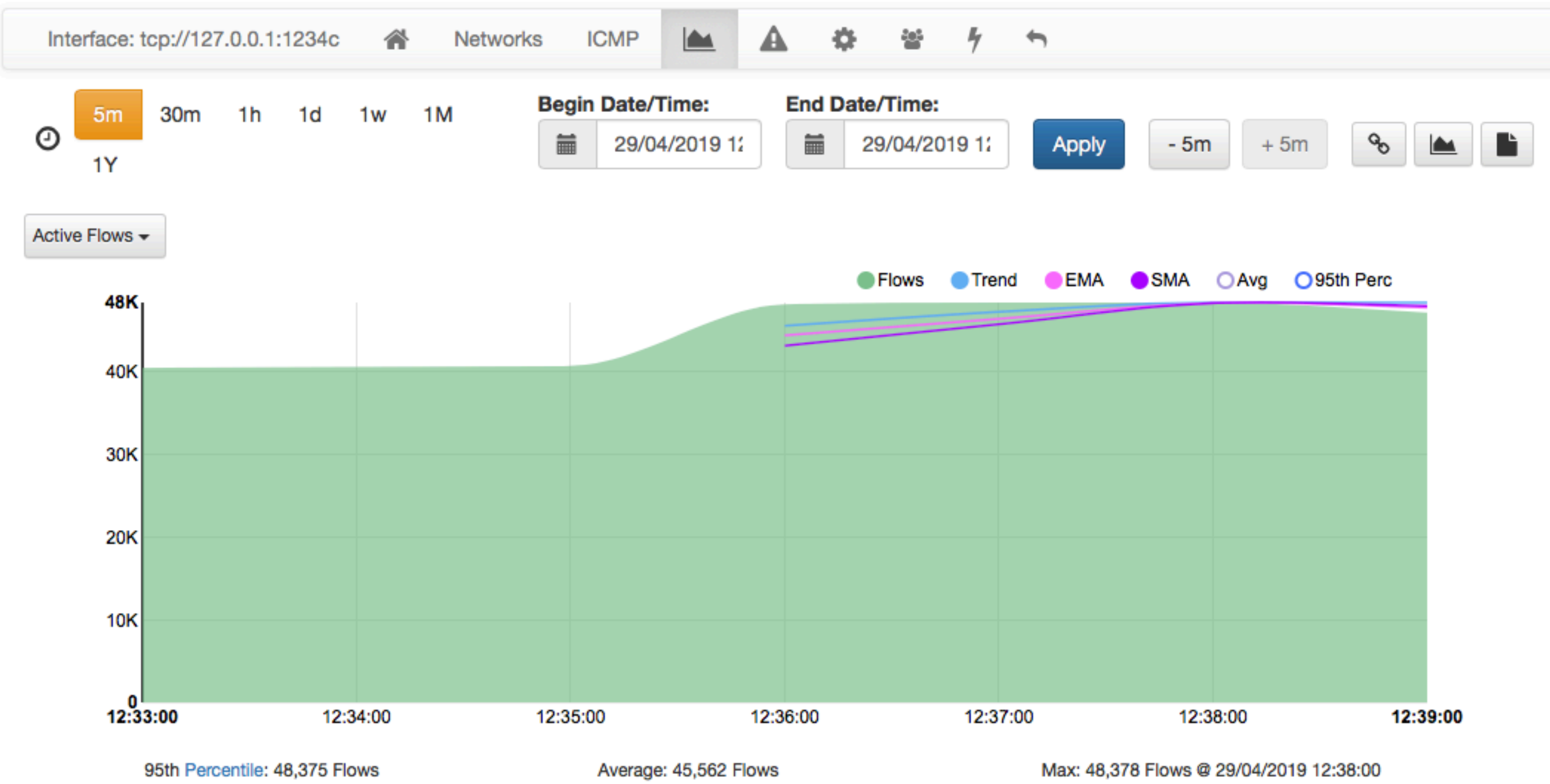
Recently Active Flows [Container ubuntu_test]

10 ▾ Hosts ▾ Status ▾ Container ▾ Applications ▾ Categories ▾ Flow Exporter ▾ IP Version ▾

	Application	Protocol	Client	Server	Client Container	Server Container	Client RTT	Server RTT	Info
Info	SSH	TCP	localhost :53216 [root >_telnet.netkit]	localhost :ssh	ubuntu_test		0.1 ms		
Info	? Unknown	TCP	NoIP:ssh [root >_sshd]	NoIP	ubuntu_test				
Info	? Unknown	TCP	:: :ssh [root >_sshd]	::	ubuntu_test				

Showing 1 to 3 of 3 rows. Idle flows not listed.

ntopng: Traffic Report



Top Categories

10

Category	Total Bytes	Percentage
Unspecified	5.21 GB	70.2 %
Email	1.92 GB	25.9 %
Database	193.91 MB	2.6 %
Web	85.11 MB	1.1 %
RemoteAccess	14.65 MB	0.2 %

Final Remarks

- It is now possible to complement network visibility with system/container information.
- Devops can deploy a resource-savvy libebpf-based container able to monitor all the containers running on a host with limited resources.
- InfluxDB is used to collect system and network metrics using ntopng as data feed.
- Users can choose ntopng or Chronograf/Grafana to implement powerful monitoring dashboards.