Monitoring Containerised Application Environments with eBPF

Luca Deri <deri@ntop.org>
@lucaderi
About ntop.org

• ntop develops of open source network traffic monitoring applications.
• ntop (circa 1998) is the first app we released and it is a web-based network monitoring application.
• Today our products range include
  ◦ Traffic monitoring tools
  ◦ High-speed packet processing
  ◦ Deep-packet inspection
  ◦ Cybersecurity applications
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 person with an opinion
(W. Edwards Deming, 1900 – 1993)
What Happens in Our Network?

• Do we have control over our network?
• It’s not possible to imagine a healthy network without a clear understanding of traffic flowing on our network.
• Knowledge is the first step towards evaluation of potential network security issues.
• Event correlation can provide us timely information about our network health.
(We Used to Say) Packets Never Lie

• Packet analysis provide useful information for understanding:
  ◦ Network traffic issues.
  ◦ Network usage not compliant with network policies (note: firewalls cannot help here).
  ◦ Non-optimal performance.
  ◦ Potential security flaws.
  ◦ Ongoing (latent) attacks.
  ◦ Data breach.

• But… packets are too fine grained so we need to aggregate them into flows (5 tuple IP/port src/dst, protocol).
ntop Ecosystem (2009): Packets Everywhere
ntop Ecosystem (2019): Still Packets [1/2]
What’s Wrong with Packets?

Nothing in general but…

• It is a paradigm good for monitoring network traffic from outside of systems on a passive way.

• Encryption is challenging DPI techniques (BTW ntop maintains an open source DPI toolkit called nDPI).

• Developers need to handle fragmentation, flow reconstruction, packet loss/retransmissions… metrics that would be already available inside a system.
What about Containers?

- Make services portable across host platforms.
- Provide an additional layer of isolation over processes.
- Allow each service to use its own dependencies.
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.
Network and System Visibility

• Even on a container-centric sites we still need to:
  ◦ Monitor the infrastructure where containers are deployed: SNMP, NetFlow/IPFIX.
  ◦ Enable system introspection also to (legacy) non-containerised systems so the whole infrastructure is monitored seamlessly.
  ◦ Export visibility metrics to existing monitoring tools on a format they can understand. This to enhance the existing monitoring console and avoid custom solutions.
What about NETLINK? [1/3]

- AF_NETLINK sockets are used for IPC between kernel and userspace processes, heavily used for monitoring and configuration (e.g. iproute2).
- What about using NETLINK_INET_DIAG for monitoring sockets?
- Good idea but events are limited to socket close

```c
enum sknetlink_groups {
    SKNLGRP_NONE,
    SKNLGRP_INET_TCP_DESTROY,
    SKNLGRP_INET_UDP_DESTROY,
    SKNLGRP_INET6_TCP_DESTROY,
    SKNLGRP_INET6_UDP_DESTROY,
    __SKNLGRP_MAX,
};
```

From linux/sock_diag.h
What about NETLINK? [2/3]

for(std::map<u_int32_t,u_int32_t>::iterator it=namespaces.begin(); it!=namespaces.end(); ++it) {
    u_int32_t ns  = it->first;
    u_int32_t pid = it->second;

    switch_namespace(ns, pid);  // Jump on the container namespace

    ...  
    if(send_diag_msg(nl_sock, (j == 0) ? AF_INET : AF_INET6, i) < 0) {  
        perror("sendmsg: ");
        return EXIT_FAILURE;
    }

    while(do_loop) {
        numbytes = recv(nl_sock, recv_buf, sizeof(recv_buf), 0);
        nlh = (struct nlmsghdr*) recv_buf;

        while(NLMSG_OK(nlh, numbytes)) {
            if(nlh->nlmsg_type == NLMSG_DONE) {  do_loop = 0; break;  }        

            diag_msg = (struct inet_diag_msg*) NLMSG_DATA(nlh);
            rtalen = nlh->nlmsg_len - NLMSG_LENGTH(sizeof(*diag_msg));
            parse_diag_msg(&inodes, diag_msg, rtalen, &results, i, ns);

            nlh = NLMSG_NEXT(nlh, numbytes);
        }
    }

    close(netns); /* Close namespace */
}
What about NETLINK? [3/3]

[TCP4][deri/1000][/home/deri/.dropbox-dist/dropbox-lnx.x86_64-70.4.93/dropbox][PID: 15972][192.168.1.23:48050 <-> 34.228.137.164:443][CLOSE-WAIT][Retrans 0][UnackSegments 0][LostPkts 0][RTT 103.67 ms (variance 5.712 ms)]

[TCP4][deri/1000][/home/deri/.dropbox-dist/dropbox-lnx.x86_64-70.4.93/dropbox][PID: 15972][192.168.1.23:47178 <-> 162.125.18.133:443][ESTABLISHED][Retrans 0][UnackSegments 0][LostPkts 0][RTT 103.57 ms (variance 15.179 ms)]

[TCP4][root/0][/home/deri/ntopng/ntopng][PID: 11346][127.0.0.1:39156 <-> 127.0.0.1:6379][ESTABLISHED][Retrans 0][UnackSegments 0][LostPkts 0][RTT 0.031 ms (variance 0.007 ms)]

[TCP4][root/0][/home/deri/ntopng/ntopng][PID: 11346][127.0.0.1:39154 <-> 127.0.0.1:6379][ESTABLISHED][Retrans 0][UnackSegments 0][LostPkts 0][RTT 0.101 ms (variance 0.074 ms)]

[TCP4][deri/1000][/usr/lib/thunderbird/thunderbird][PID: 8177][192.168.1.23:51580 <-> 93.62.150.157:993][ESTABLISHED][Retrans 0][UnackSegments 0][LostPkts 0][RTT 7.675 ms (variance 0.229 ms)]

[TCP4][redis/118][/usr/bin/redis-check-rdb][PID: 1960][127.0.0.1:6379 <-> 127.0.0.1:39154][ESTABLISHED][Retrans 0][UnackSegments 0][LostPkts 0][RTT 13.701 ms (variance 20.28 ms)]

[TCP4][deri/1000][/home/deri/.dropbox-dist/dropbox-lnx.x86_64-70.4.93/dropbox][PID: 15972][192.168.1.23:46300 <-> 162.125.18.133:443][ESTABLISHED][Retrans 0][UnackSegments 0][LostPkts 0][RTT 100.092 ms (variance 0.043 ms)]
Welcome to eBPF

• In 1997, it was introduced in Linux kernel as a technology for in-kernel packet filtering. The authors are Steven McCanne and Van Jacobson from Lawrence Berkeley Laboratory.

• eBPF extended the original BPF virtual machine, allowing it to process other kind of events execute various actions other than packet filtering.
How eBPF Works

USER

KERNEL

event

bpf code

native code

LLVM / clang

bytecode

JIT + Verifier
eBPF and Containers

• Container can be found in proc/cgroup, however retrieving information from there is a too slow operation.

• Because containers are processes, we can navigate through kernel data structures and read information from inside the kernel where the container identifier can be collected.

• Further interaction with the container runtimes (e.g. containerd or dockerd) in use is required to collect detailed information.
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: A Container-aware Probe [1/2]

- nProbe is a home-grown packet-based NetFlow/IPFIX probe and collector that can generate flow information (i.e. 5-tuple key with traffic counters).
nProbe: A Container-aware Probe [2/2]

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

<table>
<thead>
<tr>
<th>Availability</th>
<th>eBPF</th>
<th>Netlink</th>
</tr>
</thead>
<tbody>
<tr>
<td>Modern Linux (Centos 7, Ubuntu 16.04+)</td>
<td>Any Linux</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Admin Rights</th>
<th>eBPF</th>
<th>Netlink</th>
</tr>
</thead>
<tbody>
<tr>
<td>Root</td>
<td>Any User</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Purpose</th>
<th>eBPF</th>
<th>Netlink</th>
</tr>
</thead>
<tbody>
<tr>
<td>Provide information about traffic flows (creation, deletion, and updates such as retransmissions).</td>
<td>Periodic network status (e.g. established connections) and traffic statistics (e.g. interface stats).</td>
<td></td>
</tr>
</tbody>
</table>
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
Flow Creation / Termination / Status

9/Apr/2019 12:09:54 [EBPF.cpp:178] [eBPF] { "timestamp": "1556532594.175074", "LOCAL_PROCESS": { "PID": 17932, "UID": 135, "GID": 145, "PROCESS_PATH": "/usr/bin/influxd" }, "LOCAL_FATHER_PROCESS": { "PID": 1, "UID": 0, "GID": 0, "PROCESS_PATH": "/lib/systemd/systemd" }, "EVENT_TYPE": "ACCEPT", "IP_PROTOCOL_VERSION": 4, "PROTOCOL": 6, "L4_LOCAL_PORT": 51176, "L4_REMOTE_PORT": 8086, "IPV4_LOCAL_ADDR": "127.0.0.1", "IPV4_REMOTE_ADDR": "127.0.0.1", "EXPORTER_IPV4_ADDRESS": "x.x.x.x" }

29/Apr/2019 12:09:54 [EBPF.cpp:178] [eBPF] { "timestamp": "1556532594.187459", "LOCAL_PROCESS": { "PID": 31141, "UID": 135, "GID": 145, "PROCESS_PATH": "/usr/bin/influxd" }, "LOCAL_FATHER_PROCESS": { "PID": 1, "UID": 0, "GID": 0, "PROCESS_PATH": "/lib/systemd/systemd" }, "EVENT_TYPE": "CLOSE", "IP_PROTOCOL_VERSION": 4, "PROTOCOL": 6, "L4_LOCAL_PORT": 51176, "L4_REMOTE_PORT": 8086, "IPV4_LOCAL_ADDR": "127.0.0.1", "IPV4_REMOTE_ADDR": "127.0.0.1", "EXPORTER_IPV4_ADDRESS": "x.x.x.x" }

28/Apr/2019 23:48:51 [Netlink.cpp:830] [Netlink] [flow] { "timestamp": "1556488131.124793", "PROTOCOL": 6, "IP_PROTOCOL_VERSION": 4, "USER_NAME": "influxdb", "IPV4_LOCAL_ADDR": "127.0.0.1", "IPV4_REMOTE_ADDR": "0.0.0.0", "L4_LOCAL_PORT": 8088, "L4_REMOTE_PORT": 0, "TCP": { "CONN_STATE": "LISTEN" }, "LOCAL_PROCESS": { "PROCESS_ID": 31127, "USER_ID": 135, "PROCESS_PATH": "/usr/bin/influxd" }, "EXPORTER_IPV4_ADDRESS": "x.x.x.x" }
sFlow/SNMP-like Interface Stats

28/Apr/2019 23:46:29 [Netlink.cpp:1159] [Netlink] [counters] {
    "timestamp": "1555983411.568049",
    "ifName": "veth70b1674b", "ifIndex": 12,
    "LOCAL_CONTAINER": {
        "KUBE": {
            "NAME": "dnsmasq",
            "POD": "kube-dns-6bfbd666c-2wflq",
            "NS": "kube-system"
        }
    },
    "ifInOctets": 17973342,
    "ifInPackets": 75520,
    "ifInErrors": 0,
    "ifInDrops": 0,
    "ifOutOctets": 17525175,
    "ifOutPackets": 77059,
    "EXPORTER_IPV4_ADDRESS": "x.x.x.x"
}

28/Apr/2019 23:46:29 [Netlink.cpp:1159] [Netlink] [counters] {
    "timestamp": "1555983411.568269",
    "ifName": "veth9999d981", "ifIndex": 14,
    "LOCAL_CONTAINER": {
        "KUBE": {
            "NAME": "heapster",
            "POD": "heapster-v1.5.2-6b5d7b57f9-qx6kz",
            "NS": "kube-system"
        }
    },
    "ifInOctets": 49552061,
    "ifInPackets": 50511,
    "ifInErrors": 0,
    "ifInDrops": 0,
    "ifOutOctets": 55473238,
    "ifOutPackets": 57081,
    "EXPORTER_IPV4_ADDRESS": "x.x.x.x"
}

28/Apr/2019 23:46:29 [Netlink.cpp:1159] [Netlink] [counters] {
    "timestamp": "1555983411.568510",
    "ifName": "vethd0da6da5", "ifIndex": 15,
    "LOCAL_CONTAINER": {
        "KUBE": {
            "NAME": "microbot",
            "POD": "microbot-7cc7d85487-7mmkg",
            "NS": "default"
        }
    },
    "ifInOctets": 1538,
    "ifInPackets": 21,
    "ifInErrors": 0,
    "ifInDrops": 0,
    "ifOutOctets": 349756,
    "ifOutPackets": 1100,
    "EXPORTER_IPV4_ADDRESS": "x.x.x.x"
}

28/Apr/2019 23:46:29 [Netlink.cpp:1159] [Netlink] [counters] {
    "timestamp": "1556487989.626129",
    "ifName": "veth973a1f7", "ifIndex": 23,
    "LOCAL_CONTAINER": {
        "DOCKER": {
            "NAME": "ubuntu_test"
        }
    },
    "ifInOctets": 220771,
    "ifInPackets": 3066,
    "ifInErrors": 0,
    "ifInDrops": 0,
    "ifOutOctets": 70740354,
    "ifOutPackets": 78749,
    "EXPORTER_IPV4_ADDRESS": "x.x.x.x"
}

28/Apr/2019 23:46:29 [Netlink.cpp:1159] [Netlink] [counters] {
    "timestamp": "1556487989.626174",
    "ifName": "veth40297a6", "ifIndex": 21,
    "LOCAL_CONTAINER": {
        "DOCKER": {
            "NAME": "tecmint-web3"
        }
    },
    "ifInOctets": 32477,
    "ifInPackets": 328,
    "ifInErrors": 0,
    "ifInDrops": 0,
    "ifOutOctets": 13110951,
    "ifOutPackets": 40902,
    "EXPORTER_IPV4_ADDRESS": "x.x.x.x"}
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",
    "IPV6_REMOTE_ADDR": "2a00:d40:1:1: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"
}
```
Data Collection Architecture

- **Telegraf**: Agent for Collecting and Reporting Metrics and Events
- **InfluxDB**: Purpose Built Time Series Database
- **Kapacitor**: Real-time Streaming Data Processing Engine
- **Chronograf**: Complete Interface for the InfluxData Platform

**ntopng**

**nProbe**

**Flows (JSON)**

**Alerting Frameworks**
Packet-only Deployment

No System Visibility

Internet

Packets

nProbe

Flows (JSON)

Metrics

ntopng
(or any flow collector)

© 2019 - ntop.org
Packets + Metadata Deployment

- nProbe
- nProbe
- nProbe

System+Network Visibility

No System Visibility

Internet

Packets

Flows (JSON)

Metrics

InfluxDB

ntopng
Packetless Deployment

Full Visibility
(need 1 nProbe container per system)
ntopng: Process Hierarchy

<table>
<thead>
<tr>
<th>Flow Peers [Client / Server]</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Protocol / Application</td>
<td>UDP / SNMP (Network)</td>
</tr>
</tbody>
</table>

```
/bin/bash [pid: 32640] ➔ /home/deri/ntopng/ntopng [pid: 4705]

```

**Client Process Information**

<table>
<thead>
<tr>
<th>User Name</th>
<th>nobody</th>
</tr>
</thead>
<tbody>
<tr>
<td>Process PID/Name</td>
<td>/home/deri/ntopng/ntopng [pid: 4705] son of /bin/bash [pid: 32640]</td>
</tr>
</tbody>
</table>

**Server Process Information**

<table>
<thead>
<tr>
<th>User Name</th>
<th>Debian-snmp</th>
</tr>
</thead>
</table>
ntopng: Containers/Pod Overview

### Containers List

<table>
<thead>
<tr>
<th>Container</th>
<th>Flows as Client</th>
<th>Flows as Server</th>
<th>Avg RTT as Client</th>
<th>Avg RTT as Server</th>
<th>Avg RTT Variance as Client</th>
<th>Avg RTT Variance as Server</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flows 23f492221784</td>
<td>4</td>
<td></td>
<td>&lt; 0.1 ms</td>
<td></td>
<td>&lt; 0.1 ms</td>
<td></td>
</tr>
<tr>
<td>Flows 3dd3f17ad9ba</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Showing 1 to 2 of 2 rows

### Pods List

<table>
<thead>
<tr>
<th>Pod</th>
<th>Containers</th>
<th>Flows as Client</th>
<th>Flows as Server</th>
<th>Avg RTT as Client</th>
<th>Avg RTT as Server</th>
<th>Avg RTT Variance as Client</th>
<th>Avg RTT Variance as Server</th>
</tr>
</thead>
<tbody>
<tr>
<td>heapster-v1.5.2-6b5d7b67f9-g2cjz</td>
<td>4</td>
<td>12</td>
<td>2</td>
<td>3.6 ms</td>
<td>6.6 ms</td>
<td>5.0 ms</td>
<td>8.0 ms</td>
</tr>
<tr>
<td>kube-dns-6bfbd9666c-jit75</td>
<td>3</td>
<td>136</td>
<td>110</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>kubernetes-dashboad-6fd7f9c464-hpcx7</td>
<td>1</td>
<td>8</td>
<td>3</td>
<td>0.8 ms</td>
<td></td>
<td></td>
<td>0.8 ms</td>
</tr>
<tr>
<td>monitoring-influxdb-grafana-v4-78777c64c8-kmjr4</td>
<td>2</td>
<td>13</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
ntopng: Container Flows

Recently Active Flows [Container ubuntu_test]

<table>
<thead>
<tr>
<th>Application</th>
<th>Protocol</th>
<th>Client</th>
<th>Server</th>
<th>Client Container</th>
<th>Server Container</th>
<th>Client RTT</th>
<th>Server RTT</th>
<th>Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>Info</td>
<td>SSH</td>
<td>TCP</td>
<td>localhost [root -&gt; telnet.netkit]</td>
<td>localhost [root -&gt; ssh]</td>
<td>ubuntu_test</td>
<td>0.1 ms</td>
<td></td>
<td></td>
</tr>
<tr>
<td>? Unknown</td>
<td>TCP</td>
<td>NoIP:ssh [root -&gt; sshd]</td>
<td>NoIP</td>
<td></td>
<td>ubuntu_test</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>? Unknown</td>
<td>TCP</td>
<td>:: ssh [root -&gt; sshd]</td>
<td>::</td>
<td></td>
<td>ubuntu_test</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Showing 1 to 3 of 3 rows. Idle flows not listed.
ntopng: Traffic Report

Active Flows

Begin Date/Time: 29/04/2019 11:00
End Date/Time: 29/04/2019 11:00

<table>
<thead>
<tr>
<th>Category</th>
<th>Total Bytes</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unspecified</td>
<td>5.21 GB</td>
<td>70.2 %</td>
</tr>
<tr>
<td>Email</td>
<td>1.92 GB</td>
<td>25.9 %</td>
</tr>
<tr>
<td>Database</td>
<td>193.91 MB</td>
<td>2.6 %</td>
</tr>
<tr>
<td>Web</td>
<td>85.11 MB</td>
<td>1.1 %</td>
</tr>
<tr>
<td>RemoteAccess</td>
<td>14.65 MB</td>
<td>0.2 %</td>
</tr>
</tbody>
</table>

65th Percentile: 48,375 Flows
Average: 45,562 Flows
Max: 48,378 Flows @ 29/04/2019 12:38:00
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.