ipt_geofence - Performance Evaluation

Francesco Lorenzoni, Yuri Caprini, Salvatore Guastella

June 21, 2022

Contents

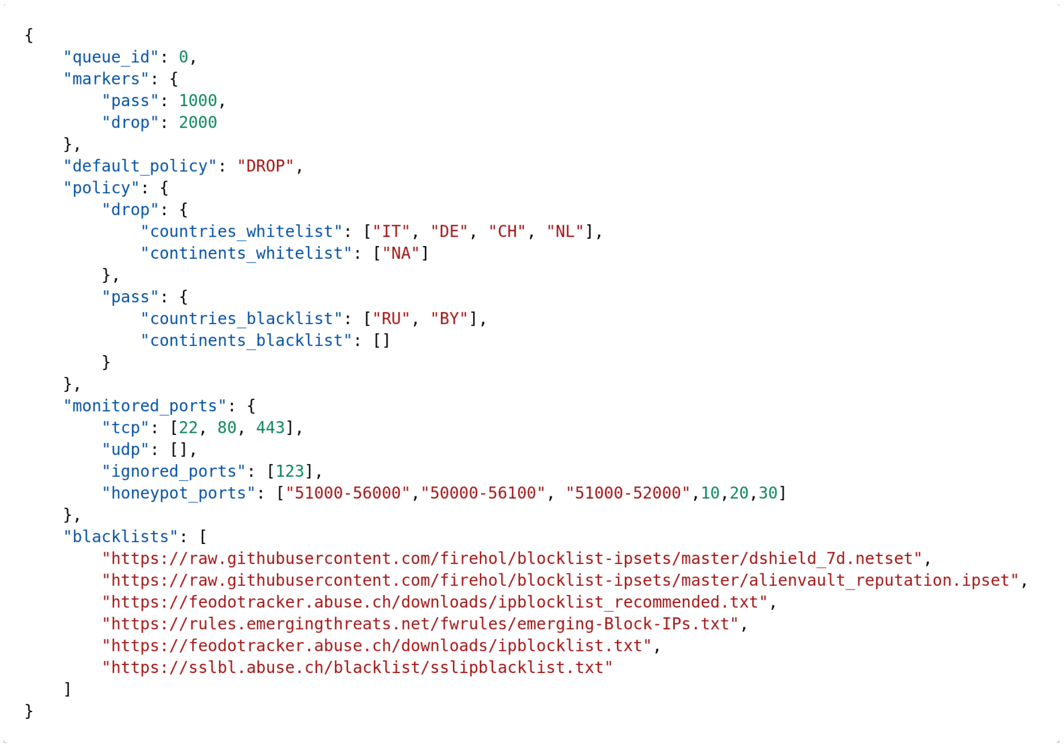
Abstract

ipt_geofence is a tool developed during the 21/22 *Network Management* UniPi course , that allows you to protect a host or a network by blocking incoming and outgoing connections according to some of their properties deduced from the analysis of their related packets.
Currently *ipt_geofence* is able to block connections from/to unwanted countries (*geofencing*), unwanted hosts (*blacklisting*) or hosts connecting on user-defined port ranges (*honeypot*).
Under ordinary network conditions *ipt_geofence* performances should be extremely good: even if packets inspection could result costly, *ipt_geofence* needs to analyze only one packet per connection to decide whether to block it or not, making the overall impact on the network negligible.
This work aims to confirm these expectations measuring the **delay** introduced by *ipt_geofence* on the latency of a single connection in order to make data-supported considerations on the performance degradation of the whole network.

1 Introduction to *ipt_geofence*

Before we deep dive into the analysis itself, let's briefly discuss how to run *ipt_geofence*. Following the steps described at *ipt_geofence* official repo is enough to get started —make sure to execute the firewall setup script—, however, it is relevant to discuss how to correctly use the `.json` configuration file and examine the information *ipt_geofence* prints on `stdout`. Bear in mind that *ipt_geofence* will issue a binary (PASS or DROP) verdict for each analyzed packet, and that the parameters of each of its features (*geofencing*, *blacklisting* and *honeypot*) will affect its evaluation.

1.1 Configuration



```
{
    "queue_id": 0,
    "markers": {
        "pass": 1000,
        "drop": 2000
    },
    "default_policy": "DROP",
    "policy": {
        "drop": {
            "countries_whitelist": ["IT", "DE", "CH", "NL"],
            "continents_whitelist": ["NA"]
        },
        "pass": {
            "countries_blacklist": ["RU", "BY"],
            "continents_blacklist": []
        }
    },
    "monitored_ports": {
        "tcp": [22, 80, 443],
        "udp": [],
        "ignored_ports": [123],
        "honeypot_ports": ["51000-56000","50000-56100", "51000-52000",10,20,30]
    },
    "blacklists": [
        "https://raw.githubusercontent.com/firehol/blocklist-ipsets/master/dshield_7d.netset",
        "https://raw.githubusercontent.com/firehol/blocklist-ipsets/master/alienvault_reputation.ipset",
        "https://feodotracker.abuse.ch/downloads/ipblocklist_recommended.txt",
        "https://rules.emergingthreats.net/fwrules/emerging-Block-IPs.txt",
        "https://feodotracker.abuse.ch/downloads/ipblocklist.txt",
        "https://sslbl.abuse.ch/blacklist/sslipblacklist.txt"
    ]
}
```

Figure 1: `.json` configuration file screenshot

The first two fields are most likely of no interest, unless you are currently using *ipt_geofence* along with other tools that are using the same *netfilter* markers or `nfqueue` id. In that case there would be a conflict and it would be necessary to change those values, so that they are different from those already in use.
`"default_policy"="PASS"|"DROP"` allows to choose the default behavior which will determine if *ipt_geofence* will consider *blacklist* over *whitelist* fields (configured below) or vice-versa.

The first three fields in `"monitored_ports"` allow to narrow down the traffic to be eventually blocked by specifying which TCP and UDP ports *ipt_geofence* considers when issuing its verdict. For connections associated to *ignored* or not *monitored* ports, the **default** marker is set.
`"honeypot_ports"` defines the ports (or port ranges) on which *ipt_geofence* acts as a *honeypot*, i.e. a host sending a packet on one of these ports is banned for 15 minutes and any connection to or from it is **blocked**.
An attacker who is trying to detect vulnerabilities on such ports will be blocked, just as a bee, attracted by *honey*, would get stuck in the *pot*.

The last field allows to define a list of *blacklists* to be loaded on *ipt_geofence* startup [1]; any connection to or from a host belonging to one of these blacklists is **blocked**.

1.2 Execution

As mentioned earlier, you must first configure firewall rules by running (as root) the following command:

```
./ipt_config_utils/single_iface.sh
```

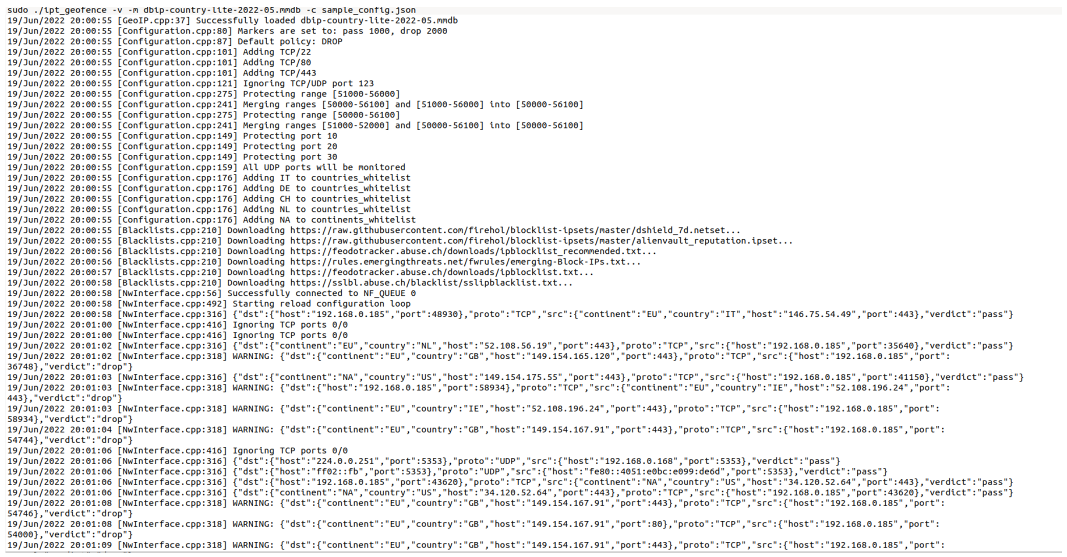
Once this is done you are able to run *ipt_geofence* and filter connections:

```
./ipt_geofence -v -m database.mmdb -c config.json
```

In addition to `-m` and `-c` parameters, path to country database and path config file respectively, you can specify `-v` options, which makes the output more verbose by printing configuration info and connections marked with `PASS` (only `DROP` marked connections are displayed by default).

When *ipt_geofence* boots it prints a summary of its configuration parameters as it reads them from the `.json` file, then it starts to analyze packets, printing some information for each of them.

[1] they are periodically reloaded once a day



```
sudo ./ipt_geofence -v -m dbip-country-lite-2022-05.mmdb -c sample_config.json
19/Jun/2022 20:00:55 [GeoIP.cpp:37] Successfully loaded dbip-country-lite-2022-05.mmdb
19/Jun/2022 20:00:55 [Configuration.cpp:80] Markers are set to: pass 1000, drop 2000
19/Jun/2022 20:00:55 [Configuration.cpp:87] Default policy: DROP
19/Jun/2022 20:00:55 [Configuration.cpp:101] Adding TCP/22
19/Jun/2022 20:00:55 [Configuration.cpp:101] Adding TCP/80
19/Jun/2022 20:00:55 [Configuration.cpp:101] Adding TCP/443
19/Jun/2022 20:00:55 [Configuration.cpp:121] Ignoring TCP/UDP port 123
19/Jun/2022 20:00:55 [Configuration.cpp:275] Protecting range [51000-56000]
19/Jun/2022 20:00:55 [Configuration.cpp:241] Merging ranges [50000-56100] and [51000-56000] into [50000-56100]
19/Jun/2022 20:00:55 [Configuration.cpp:275] Protecting range [50000-56100]
19/Jun/2022 20:00:55 [Configuration.cpp:241] Merging ranges [51000-52000] and [50000-56100] into [50000-56100]
19/Jun/2022 20:00:55 [Configuration.cpp:149] Protecting port 10
19/Jun/2022 20:00:55 [Configuration.cpp:149] Protecting port 20
19/Jun/2022 20:00:55 [Configuration.cpp:149] Protecting port 30
19/Jun/2022 20:00:55 [Configuration.cpp:159] All UDP ports will be monitored
19/Jun/2022 20:00:55 [Configuration.cpp:176] Adding IT to countries_whitelist
19/Jun/2022 20:00:55 [Configuration.cpp:176] Adding DE to countries_whitelist
19/Jun/2022 20:00:55 [Configuration.cpp:176] Adding CH to countries_whitelist
19/Jun/2022 20:00:55 [Configuration.cpp:176] Adding NL to countries_whitelist
19/Jun/2022 20:00:55 [Configuration.cpp:176] Adding NA to continents_whitelist
19/Jun/2022 20:00:55 [Blacklists.cpp:210] Downloading https://raw.githubusercontent.com/firehol/blocklist-ipsets/master/dshield_7d.netset...
19/Jun/2022 20:00:55 [Blacklists.cpp:210] Downloading https://raw.githubusercontent.com/firehol/blocklist-ipsets/master/alienvault_reputation.ipset...
19/Jun/2022 20:00:56 [Blacklists.cpp:210] Downloading https://feodotracker.abuse.ch/downloads/ipblocklist_recommended.txt...
19/Jun/2022 20:00:56 [Blacklists.cpp:210] Downloading https://rules.emergingthreats.net/fwrules/emerging-Block-IPs.txt...
19/Jun/2022 20:00:57 [Blacklists.cpp:210] Downloading https://feodotracker.abuse.ch/downloads/ipblocklist.txt...
19/Jun/2022 20:00:58 [Blacklists.cpp:210] Downloading https://sslbl.abuse.ch/blacklist/sslipblacklist.txt...
19/Jun/2022 20:00:58 [NwInterface.cpp:56] Successfully connected to NF_QUEUE 0
19/Jun/2022 20:00:58 [NwInterface.cpp:492] Starting reload configuration loop
19/Jun/2022 20:00:58 [NwInterface.cpp:316] {"dst":{"host":"192.168.0.185","port":48930},"proto":"TCP","src":{"continent":"EU","country":"IT","host":"146.75.54.49","port":443},"verdict":"pass"}
19/Jun/2022 20:01:00 [NwInterface.cpp:416] Ignoring TCP ports 0/0
19/Jun/2022 20:01:00 [NwInterface.cpp:416] Ignoring TCP ports 0/0
19/Jun/2022 20:01:02 [NwInterface.cpp:316] {"dst":{"continent":"EU","country":"NL","host":"52.108.56.19","port":443},"proto":"TCP","src":{"host":"192.168.0.185","port":35640},"verdict":"pass"}
19/Jun/2022 20:01:02 [NwInterface.cpp:318] WARNING: {"dst":{"continent":"EU","country":"GB","host":"149.154.165.120","port":443},"proto":"TCP","src":{"host":"192.168.0.185","port":
36748},"verdict":"drop"}
19/Jun/2022 20:01:03 [NwInterface.cpp:316] {"dst":{"continent":"NA","country":"US","host":"149.154.175.55","port":443},"proto":"TCP","src":{"host":"192.168.0.185","port":41150},"verdict":"pass"}
19/Jun/2022 20:01:03 [NwInterface.cpp:318] WARNING: {"dst":{"host":"192.168.0.185","port":58934},"proto":"TCP","src":{"continent":"EU","country":"IE","host":"52.108.196.24","port":
443},"verdict":"drop"}
19/Jun/2022 20:01:03 [NwInterface.cpp:318] WARNING: {"dst":{"continent":"EU","country":"IE","host":"52.108.196.24","port":443},"proto":"TCP","src":{"host":"192.168.0.185","port":
58934},"verdict":"drop"}
19/Jun/2022 20:01:04 [NwInterface.cpp:318] WARNING: {"dst":{"continent":"EU","country":"GB","host":"149.154.167.91","port":443},"proto":"TCP","src":{"host":"192.168.0.185","port":
54744},"verdict":"drop"}
19/Jun/2022 20:01:06 [NwInterface.cpp:416] Ignoring TCP ports 0/0
19/Jun/2022 20:01:06 [NwInterface.cpp:316] {"dst":{"host":"224.0.0.251","port":5353},"proto":"UDP","src":{"host":"192.168.0.168","port":5353},"verdict":"pass"}
19/Jun/2022 20:01:06 [NwInterface.cpp:316] {"dst":{"host":"ff02::fb","port":5353},"proto":"UDP","src":{"host":"fe80::4051:e0bc:e099:de6d","port":5353},"verdict":"pass"}
19/Jun/2022 20:01:06 [NwInterface.cpp:316] {"dst":{"host":"192.168.0.185","port":43620},"proto":"TCP","src":{"continent":"NA","country":"US","host":"34.120.52.64","port":443},"verdict":"pass"}
19/Jun/2022 20:01:06 [NwInterface.cpp:316] {"dst":{"continent":"NA","country":"US","host":"34.120.52.64","port":443},"proto":"TCP","src":{"host":"192.168.0.185","port":43620},"verdict":"pass"}
19/Jun/2022 20:01:08 [NwInterface.cpp:318] WARNING: {"dst":{"continent":"EU","country":"GB","host":"149.154.167.91","port":443},"proto":"TCP","src":{"host":"192.168.0.185","port":
54746},"verdict":"drop"}
19/Jun/2022 20:01:08 [NwInterface.cpp:318] WARNING: {"dst":{"continent":"EU","country":"GB","host":"149.154.167.91","port":80},"proto":"TCP","src":{"host":"192.168.0.185","port":
54000},"verdict":"drop"}
19/Jun/2022 20:01:09 [NwInterface.cpp:318] WARNING: {"dst":{"continent":"EU","country":"GB","host":"149.154.167.91","port":443},"proto":"TCP","src":{"host":"192.168.0.185","port":
```

Figure 2: *ipt_geofence* output screenshot

Let's examine some examples:

```
{"dst":{"host":"192.168.0.185","port":45070},
"proto":"TCP",
"src":{"continent":"EU","country":"IT","host":"146.75.54.49","port
    ":443},
"verdict":"pass"}
```

As you can see, there is no `continent` or `country` specified for `dst`, since it is a private IP address, but there is for `src`, which isn't.

```
WARNING:
{"dst":{"host":"192.168.0.185","port":50740},
"proto":"TCP",
"src":{"blacklisted":true,"host":"149.154.167.91","port":443},
"verdict":"drop"}
```

In this case, the record begins with `WARNING` since the verdict issued for this connection is `"drop"`. The reason for `"drop"` is that, as you can see, `src` host was found in a *blacklist*, which may be one of the blacklists loaded from the URLs specified in the configuration file, or the (black)list of *honey-banned* hosts.

2 Architecture overview

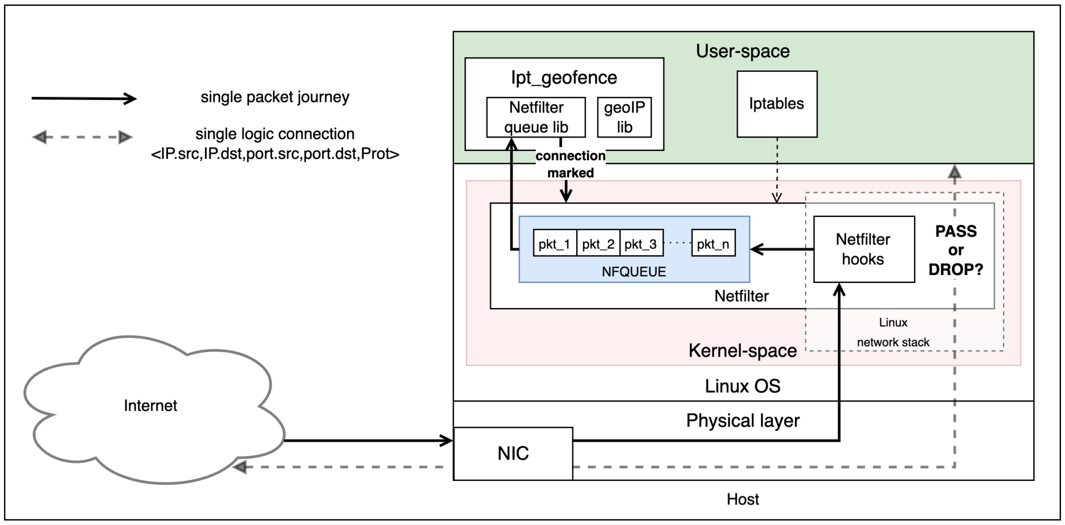


Figure 3: How *ipt_geofence* works under the hood.

Before conducting any experiment, we have to deeply understand how *ipt_geofence* works under the hood, in order to define which software behaviors need investigation, which measurements should be made and how to make them.
We know that a packet sent from or received by a host has to go through several stages of the Linux network stack. *netfilter* is an open source project which provides a one to one mapping of these stages with so called *netfilter* hooks. Before *ipt_geofence* execution, *iptables*, another popular firewalling software that allows you to define rulesets, is used to configure these hooks to perform some actions on the packets that arrive at certain stages of the network stack.

When a packet comes from an interface or user space it reaches the PREROUTING and OUTPUT phase of the network stack respectively. In these stages *netfilter* hooks are set up to queue these packets into a fixed-size NFQUEUE which resides in kernel space.

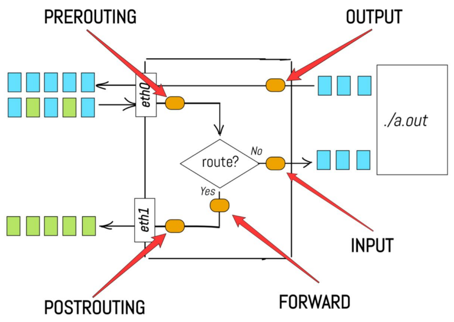


Figure 4: netfilter hooks diagram

ipt_geofence performs an infinite loop, popping out packets from this queue as fast as it can, analyzing them and marking the related connection (identified by a quintuple composed

by source IP, destination IP, source port, destination port and protocol id) with a value[2]. All packets belonging to a *marked* connection are **automatically** dropped or passed depending on the mark value. This means that they are not even queued into NFQUEUE and consequently *ipt_geofence* does not have to analyze them.
This is great: **one** packet per connection needs to be analyzed and even if can be costly, we can guess that the overall performance degradation introduced by *ipt_geofence* on a network under ordinary operating conditions is reduced to minimum.

3 Performance evaluation

We would like to confirm the speculations made at the end of the previous section with some measurements. In short, we'd like to answer this question:

What is the difference between having ipt_geofence *on our network and not having it?*

To do so, we have to first measure the average delay introduced by *ipt_geofence* (D_g) on the latency of a single packet belonging to an unmarked incoming or outgoing connection and then make some considerations based on it.

3.1 Simulation setup

Our approach to D_g calculation starts from setting up a simulation environment composed by **two** machines: $host_A$ and $host_B$. A single network interface (NIC_A) on $host_A$ is physically connected to another single one (NIC_B) on $host_B$, creating a private network of two nodes, isolated from the rest of the world.
On $host_A$ resides a running instance of *wireshark/tcpdump* capturing packets passing through NIC_A; $host_B$ is configured to route all packets received on NIC_B to $host_A$.
We have to run two simulations: *simulation I* and *simulation II*.
For each one we use *tcpreplay* to send from $host_A$ to $host_B$ at a chosen send rate a sample of n pre-forged packets [3] each belonging to a different connection. In *simulation I* on $host_B$ resides a running instance of *ipt_geofence* which is not present in *simulation II*. When a simulation ends, $host_A$ has captured all packets sent to and received from $host_B$ and has stored them in a single `.pcap` file.
This means that at the end of these two simulations we have two `.pcap` files: $pcap_1$ from *simulation I* and $pcap_2$ from *simulation II*. Now we perform the following procedure: for each packet i in $pcap_1$ and $pcap_2$ we calculate the round trip time ($RTT_I(i)$) for packet i in $pcap_1$ and $RTT_{II}(i)$ for packet i in $pcap_2$) and then we can obtain D_g[4] as follows:

$$D_g = \sum_{i=1}^{n} \frac{RTT_I(i) - RTT_{II}(i)}{n} \tag{1}$$

Note: We excluded dirty data from this evaluation by not counting negative and *min/max* addends.

[2]It's either a PASS or DROP mark, but the value itself is configurable
[3]Both the sample and the `scapy` based script used to generate it are available in the Appendix
[4]The python script which calculates D_g and its related stats is available in the Appendix.

3.2 First Attempt

In our first attempt we have sent packets from $host_A$ to $host_B$ executing the following command:

```
sudo tcpreplay -i enp2s0 -t 100K_pkts_sample.pcap
```

Which is interpreted as: *read* (10^5) *packets from* `100k_pkts_sample.pcap` *and send them as fast as possible (*`-t` *stands for* `top speed`*) on* `enp2s0` *interface.*
On $host_A$ we ran also `tcpdump`, to capture packets passing through NIC_A[5].
Then we applied the procedure described previously and we have obtained these results:

Table 1: 100K pkts $host_A \to host_B$

sent-rate [pps]	D_g [s]	$\sigma^2_{D_g}$ [s^2]	σ_{D_g} [s]
267597	0.64157501505	0.11095196949905312	0.33309453537855155

Since each simulation lasted ~2 seconds, the first thing that caught our eye is D_g itself: *How can a single packet need over half a second, if* 10^5 *packets went back and forth in less than 2 seconds?*
There was clearly something wrong in our measurement method and to find out what, we need to think about *ipt_geofence* underlying architecture.

3.3 The queueing problem

Let's focus on the role that NFQUEUE plays in the functioning of the whole module.
We know from queue theory that when the push-rate (ν_{push}) of the elements pushed into a fixed-length queue is greater than the pop-rate (ν_{pop}), —*assuming they do **not** vary over time*— after a certain interval of time that queue will become **full** (discarding subsequent pushed packets in our case study) and it will remain in that state as long as the initial conditions persist. On the contrary, if the ν_{push} is less than or equal to the ν_{pop}, there is no queuing time for elements pushed to the queue.
A queue acts like a *"sponge"*: when $\nu_{push} > \nu_{pop}$ it can absorb the elements for a certain period of time, but when this one has elapsed it will become full and begin to drip.
When $\nu_{push} \leq \nu_{pop}$ the queue is basically useless as the entity that pops the element from the queue is always ready to take next one as soon as it is pushed to the queue. In our case study "the entity" is obviously *ipt_geofence*, the queue is NFQUEUE, ν_{push} is equal to the rate of packets belonging to unmarked connections that are pushed to NFQUEUE, and $\nu_{pop} = \frac{1}{T_e}$, where T_e is the average time that *ipt_geofence* takes to pop a packet from NFQUEUE and elaborate it. Now let's make some observations helping us with some figures.

[5]the two resulting capture files from simulations are available in the Appendix

We note from Fig.4, that $D_g = T_{H2Q} + T_e + T_{G2K}$, where:

- T_{H2Q} it's the average time that a packet takes to go from a *netfilter* hook to NFQUEUE.
- T_{G2K} it's the average time that a packet takes to go from *ipt_geofence* to next phase of the network stack inside the kernel.

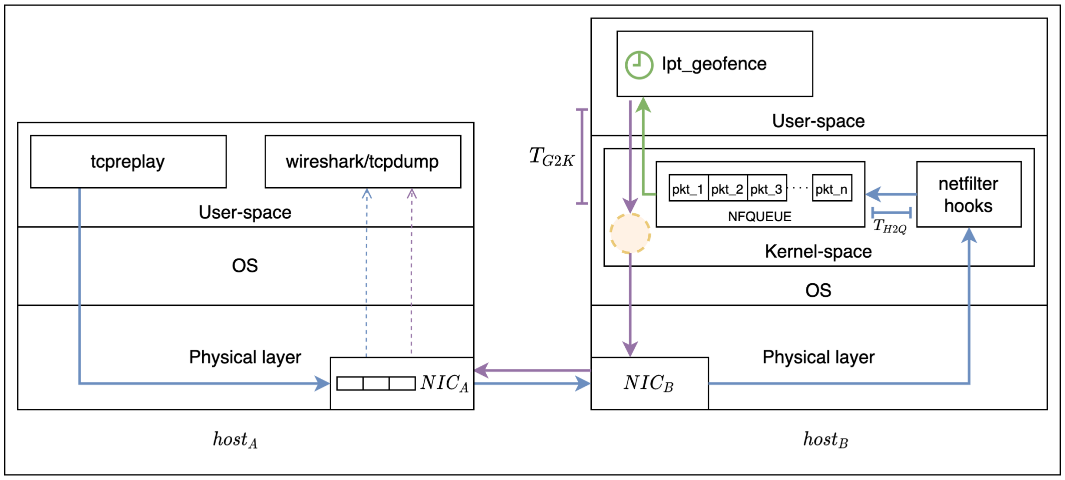


Figure 4: the travel of a packet for *simulation I* in our simulation environment.

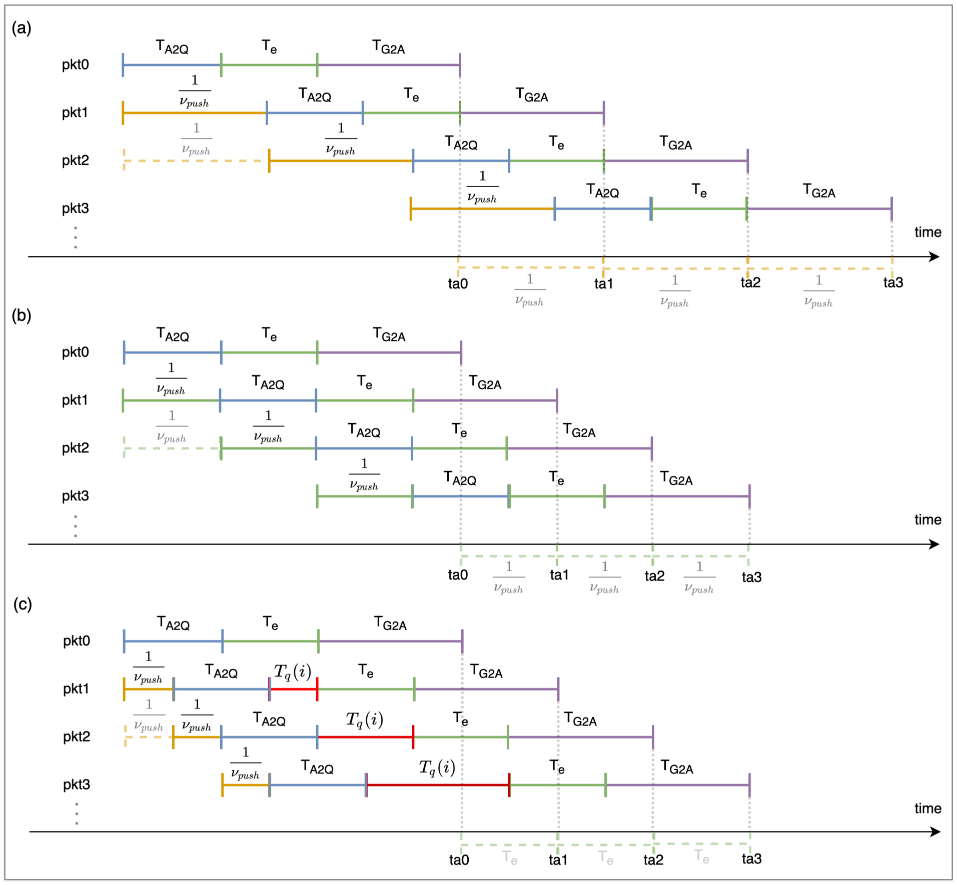


Figure 5: how the round trip time of sent packets varies on the choice of ν_{push} T_{A2Q} and T_{G2A} are colored as in Fig. 4

From Fig.5 we can make a first consideration on how the choice of ν_{push} affects $RTT_I(i)$ and how it relates to T_e:

(a) $\frac{1}{\nu_{push}} > T_e \Rightarrow \nu_{push} < \nu_{pop}$ and the queuing time for each sent packet i is $T_q(i) = 0$

(b) $\frac{1}{\nu_{push}} = T_e \Rightarrow \nu_{push} = \nu_{pop}$ and $T_q(i) = 0$

(c) $\frac{1}{\nu_{push}} < T_e \Rightarrow \nu_{push} > \nu_{pop}$ and $T_q(i) > 0$

From the same figure we can make also a second consideration. Defining ta_i as the instant of arrival of a packet i to $host_A$ after being sent by $host_B$, we get the following cases:

(a) $ta_i - ta_{i-1} = \frac{1}{\nu_{push}} > T_e$

(b) $ta_i - ta_{i-1} = \frac{1}{\nu_{push}} = T_e$

(c) $ta_i - ta_{i-1} = T_e > \frac{1}{\nu_{push}}$

With some approximation we can assert that ν_{push} is equal to the packets sent rate.
This means that in *simulation I* of our first attempt, ν_{push} was extremely high and maybe $\nu_{push} > \nu_{pop}$. We can check this by calculating from $pcap_1$ of *simulation I*:

$$\sum_{i=1}^{n} \frac{|ta_i - ta_{i-1}|}{n}$$

If our guess is right, the quantity just calculated should correspond to T_e (as stated in case (c) of our second consideration), and should result that $\nu_{push} \ > \ \nu_{pop} = \frac{1}{T_e}$. Results in Table 2 match our expectations. This result also tells us that the maximum ν_{pop} that *ipt_geofence*

Table 2: T_e calculated on data from $pcap_1$ of our first attempt

ν_{push} [pps]	T_e [s]	$\sigma^2_{T_e}$ [s^2]	$\nu_{pop} = 1/T_e$ [pps]
267597	0.00001596496	1.500452638515512e-10	62637

can guarantee on this machine is:

$$max\{\nu_{pop}\} \approx 60000$$

or equivalently that the maximum ν_{push} of unmarked connection that *ipt_geofence* can handle without eventually dropping anyone on this machine is:

$$max\{\nu_{push}\} \approx 60000$$

Since in our first attempt we have:

$$\nu_{push} = 267597 > max\{\nu_{push}\}$$

we are in the case (c) of Fig. 5 where the queue gets filled up, thus the i-th packet pushed into NFQUEUE has to wait for its **queuing time** $T_q(i)$ to elapse, leading to:

$$RTT_I(i) - RTT_{II}(i) = D_g(i) + \boldsymbol{T_q(i)} \qquad T_q(i) > 0$$

This makes us understand that to get a good estimation of D_g we must choose $\nu_{push} < max\{\nu_{push}\}$ for our simulations.

3.4 Observations on older hardware

We have conducted the above mentioned simulations using a machine equipped with `intel i7-8750H/16GB RAM` as $host_B$. However, we've replicated some of such tests on a much older PC (`i5-4210U`, launched in Q2'14).
We haven't displayed all the data gathered on this machine to avoid redundancy (since the considerations made are almost identical), but it might be of interest to note how T_e and $max\{\nu_{push}\}$ change along with the hardware:

$$T_e = 0.00008903 \qquad max\{\nu_{push}\} \approx 10k$$

As you can see, both of the above values differ from newer hardware of about a x6 factor.

3.5 Second Attempt

Since we have acknowledged why our first attempt provided misleading results, we made a second attempt choosing $\nu_{push} = 30000 < max\{\nu_{push}\} \approx 60000$, and calculating D_g as we did in our first attempt , getting the following results[6]:

Table 3: 100K pkts $host_A \rightarrow host_B$

ν_{push} [pps]	D_g [s]	$\sigma^2_{D_g}$ [s^2]	σ_{D_g} [s]
30000	0.000024608	9.04213e-09	9.50901e-05

$$95\% \text{ confidence interval} = [0.00002400319, 0.0000252133]$$

This time D_g seems to have pretty reasonables values, as also suggested by its 95% confidence interval.

4 Conclusions

From the results of Table 3, the average delay *ipt_geofence* introduces on the latency of a single unmarked connection is $D_g = 0.024608ms$. This value confirms that *ipt_geofence* impact on the network is negligible if compared to the latency of common network services:

- ping request to DNS takes $\approx 20ms \rightarrow 0,12\%$ delay introduced by *ipt_geofence*.
- web page request takes $\approx 377ms \rightarrow 0,013\%$ delay introduced by *ipt_geofence*.
- google search takes $\approx 1474ms \rightarrow 0,0033\%$ delay introduced by *ipt_geofence*.

In addition, it takes at least $60K$ new connections *per second* to obtain noticeable performance degradation (aka *ipt_geofence* module can start dropping unmarked connections), which falls into most unlikely scenarios.

[6]the `.pcap` files used and the python script that calculated these stats are both linked in the Appendix.

Appendix

- *ipt_geofence* official repo: github.com/ntop/ipt_geofence
- python script to calculate T_e on `.pcap` files: github.com/frenzis01/pcap_stats_module
- python script to calculate D_g on `.pcap` files: github.com/yuricaprini/ipt_geofence_performance_evaluation
- scapy-based python script to forge packets: github.com/yuricaprini/ntgenpy
- scapy-generated packets sample:
  github.com/frenzis01/sgr/blob/master/2022/CapriniGuastellaLorenzoni/pcap/sample
- *"First Attempt"* pcap files
  github.com/frenzis01/sgr/tree/master/2022/CapriniGuastellaLorenzoni/pcap/firstAttempt
- *"Second Attempt"* pcap files
  github.com/frenzis01/sgr/tree/master/2022/CapriniGuastellaLorenzoni/pcap/secondAttempt

Authors

Yuri Caprini

- y.caprini@studenti.unipi.it
- https://github.com/yuricaprini

Salvatore Guastella

- s.guastella2@studenti.unipi.it
- https://github.com/salvogs

Francesco Lorenzoni

- f.lorenzoni4@studenti.unipi.it
- https://github.com/frenzis01