More tinkering with DNS and XDP

Tom Carpay, Luuk Hendriks & Willem Toorop

NLNETLABS

UKNOF 47
Motivation & goals

- Programmable networks are hot (see also: P4), and for good reason!
- Flexibility in the data plane without sacrificing performance
- Specifically for XDP: easy way to perform some parts \textit{in kernel} (heavy lifting) but still have traditional userspace software 'above' that.

XDP does not have to replace everything we do in userspace, such as DNS, it can \textit{augment} it.

Featured in this presentation: RRL + some other examples
Response Rate Limiting 101 (of incoming queries)

- **When** Queries per Second > X (from certain source IP or Prefix)
- **Then** Return truncated (or drop)
(e)BPF, XDP, DNS

(Extended) Berkeley Packet Filter (eBPF):

Historically the VM that handles your `tcpdump` filters. Nowadays a much more powerful concept with a slightly deceiving name: run verified code in kernel space without rebooting.

eXpress Data Path (XDP):

Network driver hook to run BPF code. Executed before anything happens in the kernel networking stack. Can be hardware offloaded for even more performance.

DNS:

DNS (RFC1034, RFC1035)
A packet's destiny: XDP return codes

Classic in-kernel stack, no XDP
A packet's destiny: XDP return codes

- **XDP_PASS**: pass on to network stack
- **XDP_TX**: send it out of ingress NIC
- **XDP_DROP**: drop the packet
- **XDP_REDIRECTED**: send out other NIC
A packet's destiny: XDP return codes

- **XDP_PASS**: pass on to network stack
- **XDP_TX**: send it out of ingress NIC
- **XDP_DROP**: drop the packet
- **XDP_REDIRECTED**: send out other NIC
A packet's destiny: XDP return codes

- XDP_PASS: pass on to network stack
- XDP_TX: send it out of ingress NIC
- XDP_DROP: drop the packet
- XDP_REDIRECTED: send out other NIC
Using the special AF_XDP socket type one can reach the application while bypassing the entire network stack. (special case of XDP_REDIRECT)
Towards *augmenting* DNS software

< This work is about:

adding functionality that is agnostic of DNS software running on the OS.

It's not about:

Adapting existing software to use AF_XDP sockets;
Implementing feature complete nameservers/resolvers in XDP
Workflow

- write C code: rrl.c
- compile: rrl.o (NB: successful compilation does not guarantee the next step!)
- attach rrl.o network interface, e.g. using iproute2:

  ```bash
  # ip link set dev enol xdpgeneric obj rrl.o sec xdp
  ```

- verifier checks the program: does it terminate? Is it not too complex? Stays within bounds?
- no objections found? code is now active on the interface, on ingress, processing incoming packets before the OS network stack sees them

- any further interaction (if any) with the running code goes via BPF maps
- no modprobe, no reboot, no reconfiguration of userspace software
Response Rate Limiting

- Check whether incoming packet:
  - is Ethernet/IP/UDP with dst port 53, and,
  - contains a correctly formatted DNS query
    ■ (if not, XDP_PASS the packet upwards to the stack)

- Now we know we are dealing with a DNS query, we:
  - calculate the QPS rate for this src_addr (i.e. keeping state, using maps)
  - based on that rate, return:
    XDP_PASS (no rate limiting applied), or
    XDP_DROP, or XDP_TX with TC=1 (if we want to RRL this query)
On the state of BPF Maps

Datastructures *specific* to BPF, require specific functions to read/write at runtime, e.g.:

- `bpf_map_lookup_elem()`
- `bpf_map_update_elem()`
- `bpf_map_delete_elem()`

NB: Hardware offloading might not support all of these map types
Maps: inter-packet state

Keeping state in-between packets using BPF maps:

- datastructure: hashmap
- key: IPv6/IPv4 src address (of incoming queries)
- value: our own struct `bucket`, enabling rate calculation

```c
1 struct bucket {
2     uint64_t start_time;
3     uint64_t n_packets;
4 }
5
6 struct bpf_map_def SEC("maps") state_map = {
7     .type = BPF_MAP_TYPE_PERCPU_HASH,
8     .key_size = sizeof(uint32_t),
9     .value_size = sizeof(struct bucket),
10    .max_entries = 1000000
11 };
12
13 struct bpf_map_def SEC("maps") state_map_v6 = {
14     .type = BPF_MAP_TYPE_PERCPU_HASH,
15     .key_size = sizeof(struct in6_addr),
16     .value_size = sizeof(struct bucket),
17    .max_entries = 1000000
18 };
```
Maps: configuration from userspace

Operator request: "RRL, but not for $very_important_prefix"

Run-time configuration from userspace using maps:

- datastructure: LPM trie
- key: IPv6/IPv4 src address (of incoming queries)
- value: hit counter
- read/write using `bpftool`, or, your own custom userspace tool.
Demo time 😈

- example of how to compile
- example of how to load it
- screenshot of rrl.o in action
Demo time

- example of how to compile
- example of how to load it
- screenshot of rrl.o in action (flamethrower?)

```
root@ron2021:~ # apt install git build-essential make clang gcc-multilib libelf-dev linux-tools-common
```
Demo time

- example of how to compile
- example of how to load it
- screenshot of rrl.o in action (flamethrower?)

root@ron2021:~$ make
root@ron2021:~$ make

Reading state information... Done
build-essential is already the newest version (12.4ubuntu1).
gcc-multilib is already the newest version (4.7.4.0-1ubuntu2.3).
git is already the newest version (1:2.17.1-1ubuntu0.7).
libelf-dev is already the newest version (0.170-0.4ubuntu0.1).
linux-tools-common is already the newest version (4.15.0-135.139).
clang is already the newest version (1:6.0-41-exp5-ubuntu1).
0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
root@ron2021:~#
root@ron2021:~# git clone https://github.com/NLnetLabs/XDPeriments.git
Cloning into 'XDPeriments'...
remote: Enumerating objects: 107, done.
remote: Counting objects: 100% (107/107), done.
remote: Compressing objects: 100% (71/71), done.
remote: Total 107 (delta 47), reused 87 (delta 33), pack-reused 0
Receiving objects: 100% (107/107), 32.80 KiB | 1.49 MiB/s, done.
Resolving deltas: 100% (47/47), done.
root@ron2021:~#
Demo time

- example of how to compile
- example of how to load it
- screenshot of rrl.o in action (flamethrower?)

```
root@ron2021:~/XDPeriments/libbpf/src
Reading state information... Done
build-essential is already the newest version (12.4ubuntu1).
maker is already the newest version (4.1-9.1ubuntu1).
gcc-multilib is already the newest version (4:7.4.0-1ubuntu2.3).
git is already the newest version (1:2.17.1-1ubuntu0.7).
libelf-dev is already the newest version (0.170-0.4ubuntu0.1).
linux-tools-common is already the newest version (4.15.0-135.139).
clang is already the newest version (1:6.0-41-exp5-ubuntu1).
0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
root@ron2021:~#
root@ron2021:~# git clone https://github.com/NLnetLabs/XDPeriments.git
Cloning into 'XDPeriments'...
remote: Enumerating objects: 107, done.
remote: Counting objects: 100% (107/107), done.
remote: Compressing objects: 100% (71/71), done.
remote: Total 107 (delta 47), reused 87 (delta 33), pack-reused 0
Receiving objects: 100% (107/107), 32.80 KiB | 1.49 MiB/s, done.
Resolving deltas: 100% (47/47), done.
root@ron2021:~#
root@ron2021:~# cd XDPeriments
root@ron2021:~/XDPeriments# git submodule update --init
Submodule 'libbpf' (https://github.com/libbpf/libbpf) registered for path 'libbpf'
Cloning into '/root/XDPeriments/libbpf'...
Submodule path 'libbpf': checked out '1b42b15b5e6dec568e8826ed908a5aced32317c'
root@ron2021:~/XDPeriments#```
Demo time

- example of how to compile
- example of how to load it
- screenshot of rrl.o in action (flamethrower?)

```
root@ron2021:~ /XDPeriments/libbpf/src
reading state information... done
build-essential is already the newest version (12.4ubuntu1).
makes is already the newest version (4.1-9.1ubuntu1).
gcc-multilib is already the newest version (4:7.4.0-1ubuntu2.3).
git is already the newest version (1:2.17.1-1ubuntu0.7).
libelf-dev is already the newest version (0.170-0.4ubuntu9.1).
linux-tools-common is already the newest version (4.15.0-135.139).
clang is already the newest version (1:6.0-41-exp5-ubuntu1).
0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
root@ron2021:~ #
root@ron2021:~ # git clone https://github.com/NLnetLabs/XDPeriments.git
cloning into 'XDPeriments'...
remote: Enumerating objects: 107, done.
remote: Counting objects: 100% (107/107), done.
remote: Compressing objects: 100% (71/71), done.
remote: Total 107 (delta 47), reused 87 (delta 33), pack-reused 0
Receiving objects: 100% (107/107), 32.80 KiB | 1.49 MiB/s, done.
Resolving deltas: 100% (47/47), done.
root@ron2021:~ #
root@ron2021:~ # cd XDPeriments
root@ron2021:~ /XDPeriments# git submodule update --init
Submodule 'libbpf' (https://github.com/libbpf/libbpf) registered for path 'libbpf'
cloning into '/root/XDPeriments/libbpf'...
Submodule path 'libbpf': checked out '1b42b15b5e6dec568e8826ed908a5acedd32317c'
root@ron2021:~ /XDPeriments#
root@ron2021:~ /XDPeriments# cd libbpf/src/
root@ron2021:~ /XDPeriments/libbpf/src# make
```
Demo time

- example of how to compile
- example of how to load it
- screenshot of rrl.o in action (flamethrower?)
Demonstration:

- example of how to compile
- example of how to load it
- screenshot of rrl.o in action (flamethrower?)

```
root@ron2021:~/.XDPeriments/RRL/Round3

sed -e "s|@PREFIX@|/usr|" \
    -e "s|@LIBDIR@|/usr/lib64|" \
    -e "s|@VERSION@|0.1.0|"
< libbpf.pc.template > libbpf.pc
cd ..../../.XDPeriments/RRL/Round3
make

clang -target bpf -O2 -Wall -Werror -I ..../libbpf/src -c -o xdp_rrl.o xdp_rrl.c
c clang -static -O2 -Wall -Werror -I ..../libbpf/src -o xdp_rrl_vipct1 xdp_rrl_vipct1.c -L..../libbpf/src -L..../libbpf -lelf -lz
make

make vip_maps

sudo mount -t bpf none /sys/fs/bpf
sudo bpftool map create /sys/fs/bpf/rrl_exclude_v4_prefixes flags 1 \  
    name exclude_v4_prefixes type lpm_trie key 8 value 8 entries 10000
sudo bpftool map create /sys/fs/bpf/rrl_exclude_v6_prefixes flags 1 \  
    name exclude_v6_prefixes type lpm_trie key 12 value 8 entries 10000
```
Demo time

- example of how to compile
- example of how to load it
- screenshot of rrl.o in action (flamethrower?)

root@ron2021:~/XDPeriments/RRL/Round3

sed -e "s@PREFIX@/usr/" \n    -e "s@LIBDIR@/usr/lib64@" \n    -e "s@VERSION@0.1.0@" \n< libbpf.pc.template > libbpf.pc

root@ron2021:~/XDPeriments/libbpf/src#
root@ron2021:~/XDPeriments/libbpf/src# cd ..../..../RRL/Round3
root@ron2021:~/XDPeriments/RRL/Round3# make

clang -target bpf -O2 -Wall -Werror -I ../../libbpf/src -c -o xdp_rrl.o xdp_rrl.c
clang -static -O2 -Wall -Werror -I ../../libbpf/src -o xdp_rrl_vipctl xdp_rrl_vipctl.c -L../../libbpf/src -lbpf -elf -lz

root@ron2021:~/XDPeriments/RRL/Round3#

root@ron2021:~/XDPeriments/RRL/Round3# make vip_maps

sudo mount -t bpf none /sys/fs/bpf

sudo bpftool map create /sys/fs/bpf/rrl_exclude_v4_prefixes flags 1 \n    name exclude_v4_prefixes type lpm_trie key 8 value 8 entries 10000

sudo bpftool map create /sys/fs/bpf/rrl_exclude_v6_prefixes flags 1 \n    name exclude_v6_prefixes type lpm_trie key 12 value 8 entries 10000

root@ron2021:~/XDPeriments/RRL/Round3#
root@ron2021:~/XDPeriments/RRL/Round3# make load

sudo bpftool prog load xdp_rrl.o /sys/fs/bpf/rrl type xdp \n    map name exclude_v4_prefixes \n    pinned /sys/fs/bpf/rrl_exclude_v4_prefixes \n    map name exclude_v6_prefixes \n    pinned /sys/fs/bpf/rrl_exclude_v6_prefixes

sudo ip --force link set dev eth0 xdpgeneric \n    pinned /sys/fs/bpf/rrl

root@ron2021:~/XDPeriments/RRL/Round3#
Demo time

- example of how to compile
- example of how to load it
- screenshot of rrl.o in action (flamethrower?)

```bash
sudo bpftool map create /sys/fs/bpf/rrl_exclude_v4_prefixes flags 1 \
    name exclude_v4_prefixes type lpm_trie key 8 value 8 entries 10000
sudo bpftool map create /sys/fs/bpf/rrl_exclude_v6_prefixes flags 1 \
    name exclude_v6_prefixes type lpm_trie key 12 value 8 entries 10000

root@ron2021:~/XDPeriments/RRL/Round3#
root@ron2021:~/XDPeriments/RRL/Round3# make load
sudo bpftool prog load xdp_rrl.o /sys/fs/bpf/rrl type xdp \
    map name exclude_v4_prefixes \
    pinned /sys/fs/bpf/rrl_exclude_v4_prefixes \
    map name exclude_v6_prefixes \
    pinned /sys/fs/bpf/rrl_exclude_v6_prefixes

sudo ip --force link set dev eth0 xdpgeneric \
    pinned /sys/fs/bpf/rrl

root@ron2021:~/XDPeriments/RRL/Round3#
root@ron2021:~/XDPeriments/RRL/Round3# bpftool map | tail -8
20: lpm_trie name exclude_v4_pref flags 0x1
    key 8B value 8B max_entries 100000 memlock 524288B
21: lpm_trie name exclude_v6_pref flags 0x1
    key 12B value 8B max_entries 100000 memlock 561152B
23: percpu_hash name state_map flags 0x0
    key 4B value 16B max_entries 100000 memlock 320778240B
24: percpu_hash name state_map_v6 flags 0x0
    key 16B value 16B max_entries 100000 memlock 328777728B

root@ron2021:~/XDPeriments/RRL/Round3#
root@ron2021:~/XDPeriments/RRL/Round3# bpftool map dump id 24
Found 0 elements
```

root@ron2021:~/XDPeriments/RRL/Round3#
Demo time

- example of how to compile
- example of how to load it
- screenshot of rrl.o in action (flamethrower?)

```bash
root@ron2021:~/XDPeriments/RRL/Round3# bpftool map dump id 23
key:
  2d 5f 40 00
value (CPU 00): 40 e0 5d 75 81 03 00 00 01 00 00 00 00 00 00
value (CPU 01): 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
value (CPU 02): 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
value (CPU 03): 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
value (CPU 04): 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
value (CPU 05): 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
value (CPU 06): 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
value (CPU 07): 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
value (CPU 08): 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
value (CPU 09): 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
value (CPU 10): 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
value (CPU 11): 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
value (CPU 12): 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
value (CPU 13): 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
value (CPU 14): 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
Found 1 element
```

```
```
```bash
willem@makaak:~ 103x5
willem@makaak:~$ dig -4 @ron2021.nlnetlabs.nl nlnetlabs.nl A +short
185.49.140.10
willem@makaak:~$ ```
Demo time

- example of how to compile
- example of how to load it
- screenshot of rrl.o in action (flamethrower?)

```c
#define RRL_N_CPUS 2
/* This should be the number of CPUs on your system. Get it by running:
 */
   echo "CPUs: $(grep -c processor /proc/cpuinfo)"
*/

#define RRL_SIZE 1000000
/* This option gives the size of the hashtable. More buckets
 * use more memory, and reduce the chance of hash collisions.
 */

#define RRL_RATELIMIT 200
/* The max qps allowed (from one query source). If set to 0 then it is disabled
 * (unlimited rate). Once the rate limit is reached, responses will be dropped.
 * However, one in every RRL_SLIP number of responses is allowed, with the TC
 * bit set. If slip is set to 2, the outgoing response rate will be halved. If
 * it's set to 3, the outgoing response rate will be one-third, and so on. If
 * you set RRL_SLIP to 10, traffic is reduced to 1/10th.
 */
```

"xdp_rrl.c" 625L, 18102C
#define RRL_RATELIMIT 200
/* The max qps allowed (from one query source). If set to 0 then it is disabled
 *(unlimited rate). Once the rate limit is reached, responses will be dropped.
 * However, one in every RRL_SLIP number of responses is allowed, with the TC
 * bit set. If slip is set to 2, the outgoing response rate will be halved. If
 * it's set to 3, the outgoing response rate will be one-third, and so on. If
 * you set RRL_SLIP to 10, traffic is reduced to 1/10th.
 */

#define RRL_SLIP 2
/* This option controls the number of packets discarded before we send back a
 * SLIP response (a response with "truncated" bit set to one). 0 disables the
 * sending of SLIP packets, 1 means every query will get a SLIP response.
 * Default is 2, cuts traffic in half and legit users have a fair chance to get
 * a +TC response.
 */

#define RRL_IPV4_PREFIX_LEN 24
/* IPv4 prefix length. Addresses are grouped by netblock.
 */

#define RRL_IPV6_PREFIX_LEN 48
/* IPv6 prefix length. Addresses are grouped by netblock.
 */
Demo time - example of how to compile - example of how to load it - screenshot of rrl.o in action (flamethrower?)

#define RRL_SIZE 1000000
/* This option gives the size of the hashtable. More buckets
 * use more memory, and reduce the chance of hash collisions.
 */

#define RRL_RATELIMIT 5
/* The max qps allowed (from one query source). If set to 0 then it is disabled
 * (unlimited rate). Once the rate limit is reached, responses will be dropped.
 * However, one in every RRL_SLIP number of responses is allowed, with the TC
 * bit set. If slip is set to 2, the outgoing response rate will be halved. If
 * it's set to 3, the outgoing response rate will be one-third, and so on. If
 * you set RRL_SLIP to 10, traffic is reduced to 1/10th.
 */

#define RRL_SLIP 1
/* This option controls the number of packets discarded before we send back a
 * SLIP response (a response with "truncated" bit set to one). 0 disables the
 * sending of SLIP packets, 1 means every query will get a SLIP response.
 * Default is 2, cuts traffic in half and legit users have a fair chance to get
 * a +TC response.
 */

#define RRL_IPV4_PREFIX_LEN 24
/* IPv4 prefix length. Addresses are grouped by netblock.
 */
Demo time

- example of how to compile
- example of how to load it
- screenshot of rrl.o in action (flamethrower?)

```
willem@makaak:~$ while test 1
> do
> echo `date` `dig @ron2021.nlnetlabs.nl uknof.org.uk A +short +ignore`
> sleep .5
> done
```
Demo:
- example of how to compile
- example of how to load it
- screenshot of rrl.o in action (flamethrower?)
Demo time

- example of how to compile
- example of how to load it
- screenshot of rrl.o in action (flamethrower?)
Demo time

- example of how to compile
- example of how to load it
- screenshot of rrl.o in action (flamethrower?)
Demo time

- example of how to compile
- example of how to load it
- screenshot of rrl.o in action (flamethrower?)
Rate Limiting - lessons learned

We can leverage XDP to *augment* DNS services:

- to deal with *incoming* packets
- handle the packet in XDP, or,
- decide to point it upwards to a userspace nameserver

Maps enable keeping state,
not only for e.g. statistics and rates calculations,
but moreover for *configuration from userspace* at runtime

PERCPU BPF map type make processing lock free and blazingly fast
DNS Cookies

A **in-protocol** way to learn prefixes to exclude from rate-limiting automatically
DNS Cookies

A **in-protocol** way to learn prefixes to exclude from rate-limiting automatically
DNS Cookies 101 - DNS Cookies Operation

Server Cookie

```
+-----------------+-------------------+
| Version          | Reserved           |
+-----------------+-------------------+
| Timestamp        |                   |
+-----------------+-------------------+
| Hash             |                   |
```

Hash = SipHash2.4(
Client Cookie
| Version
| Reserved
| Timestamp
| Client-IP
| Server Secret)

37
DNS Cookies 101 - DNS Cookies Operation

- Valid Server Cookie? Large answers
- Valid Server Cookie? Rate-limiting disabled
verify Server Cookies
create Server Cookies
answer queries
Transport Layer (UDP & TCP)
Internet Layer (IPv4 & IPv6)
rate-limiting
invalid or no cookie
verify Server Cookies
UDP
IPv4 / IPv6
Link Layer

User space (DNS service)
Kernel space (sockets)
XDP
Transport Layer (UDP & TCP)

Internet Layer (IPv4 & IPv6)

User space (DNS service)

Kernel space (sockets)

TC (Traffic Control)

XDP

Fully fledged augmentation

answer queries

rate-limiting

verify Server Cookies

link Layer
Fully fledged augmentation

DNS
answer queries

Transport Layer (UDP & TCP)

Internet Layer (IPv4 & IPv6)

IPv4 / IPv6
UDP

add padding

IPv4 / IPv6
UDP

BPF_MAP_TYPE_LRU_HASH

No
padding option present?

Yes

User space (DNS service)

Kernel space (sockets)

TC

XDP
BPF_MAP_TYPE_LRU_HASH

```c
struct query_v6 {
    struct in6_addr addr;
    uint16_t port;
    uint16_t qid;
};

struct bpf_elf_map SEC("maps") queries_v6 = {
    .type = BPF_MAP_TYPE_LRU_HASH,
    .size_key = sizeof(struct query_v6),
    .size_value = sizeof(uint8_t),
    .max_elem = 10000,
    .pinning = PIN_GLOBAL_NS
};

struct query_v4 {
    uint32_t addr;
    uint16_t port;
    uint16_t qid;
};

struct bpf_elf_map SEC("maps") queries_v4 = {
    .type = BPF_MAP_TYPE_LRU_HASH,
    .size_key = sizeof(struct query_v4),
    .size_value = sizeof(uint8_t),
    .max_elem = 10000,
    .pinning = PIN_GLOBAL_NS
};
```
Fully fledged augmentation

stub resolver -> recursive resolver

. -> uk. -> uknof.org.uk

unbound

NSD
4.1.1. Header section format

The header contains the following fields:

```
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                      ID                       |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|QR|   Opcode  |AA|TC|RD|RA|   Z    |   RCODE   |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                    QDCOUNT                    |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                    ANCOUNT                    |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                    NSCOUNT                    |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                    ARCOUNT                    |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```

Where:

- **RD** (Recursion Desired) - this bit may be set in a query and is copied into the response. If RD is set, it directs the name server to pursue the query recursively. Recursive query support is optional.
Concluding ...

A lot is possible!

XDP and eBPF is a very good fit for plain old UDP based DNS. because per packet processing.

Less suitable for TCP based DNS, and probably impossible for DoT and DoH

We think using XDP to augment an existing DNS service is an exciting new idea, and a great new tool in the DNS operator’s toolbox
Looking ahead

- Offloading to actual hardware
- Statistics & logging from XDP
- **AF_XDP support for NSD**
- **Hot self-managing cache**
  Write outgoing answers in a LRU hashmap, answer queries directly from XDP
- **Zone sharding / load balancing**
- **root zone from XDP?**
More tinkering with DNS and XDP

{willem,luuk,tom}@nlnetlabs.nl
https://github.com/NLnetLabs/XDPeriments
https://blog.nlnetlabs.nl/tag/research/

Thank U
Ronald van der Pol