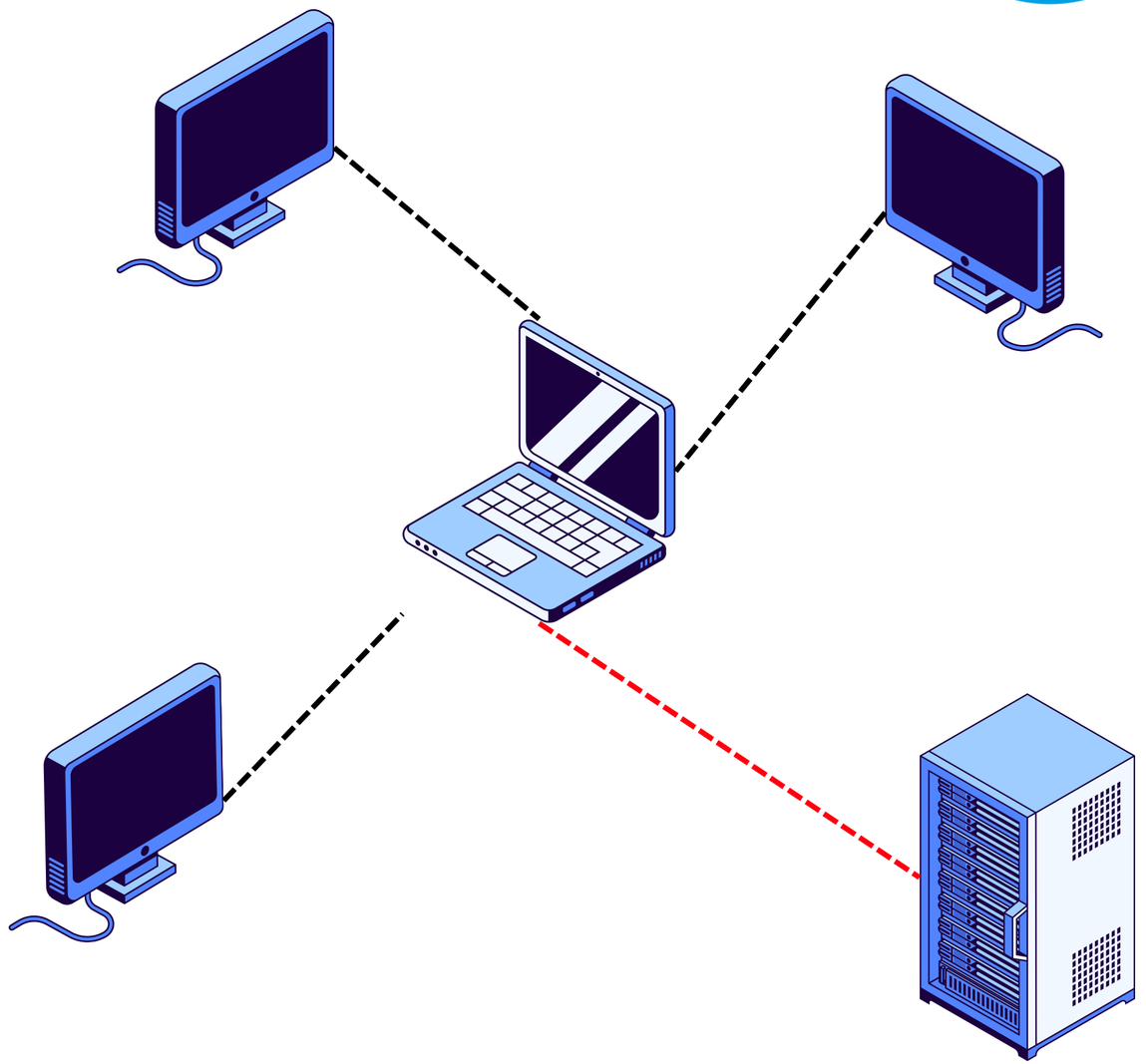
Project 15: DNS Resolver using AF_XDP

CS331 - Computer Networks



TEAM - 13

Guntas Singh Saran (22110089)

Hrriday V. Ruparel (22110099)

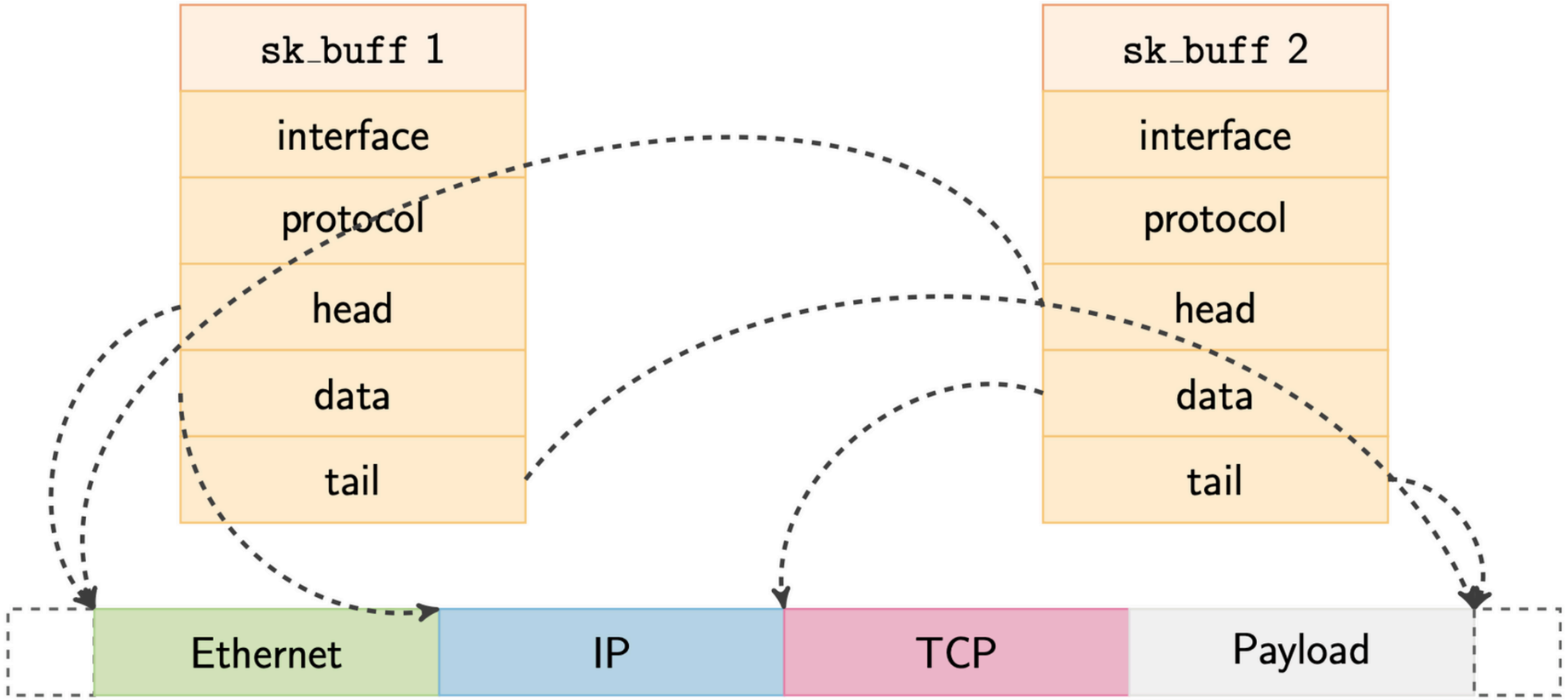
Kishan Ved (22110122)

Pranav Patil (22110199)

Indian Institue of Technology Gandhinagar
Palaj, Gujarat - 382355

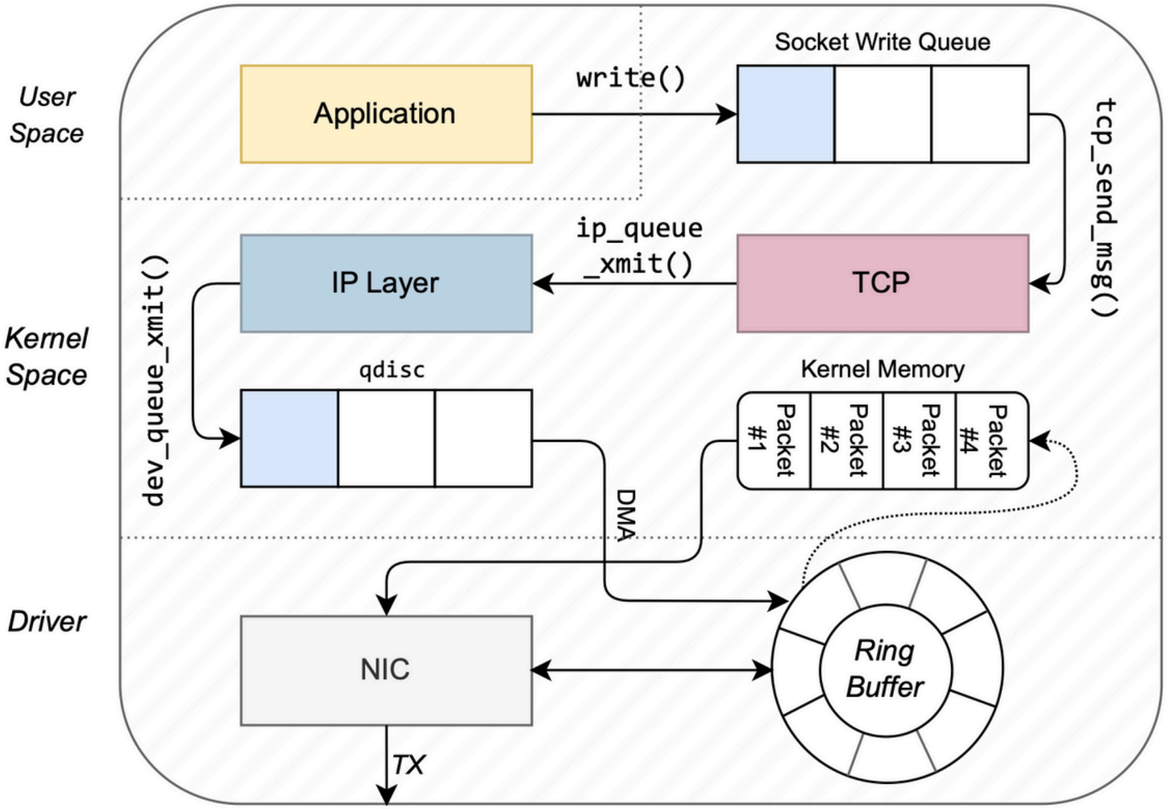
Source: https://github.com/Kishan-Ved/dns-af_xdp

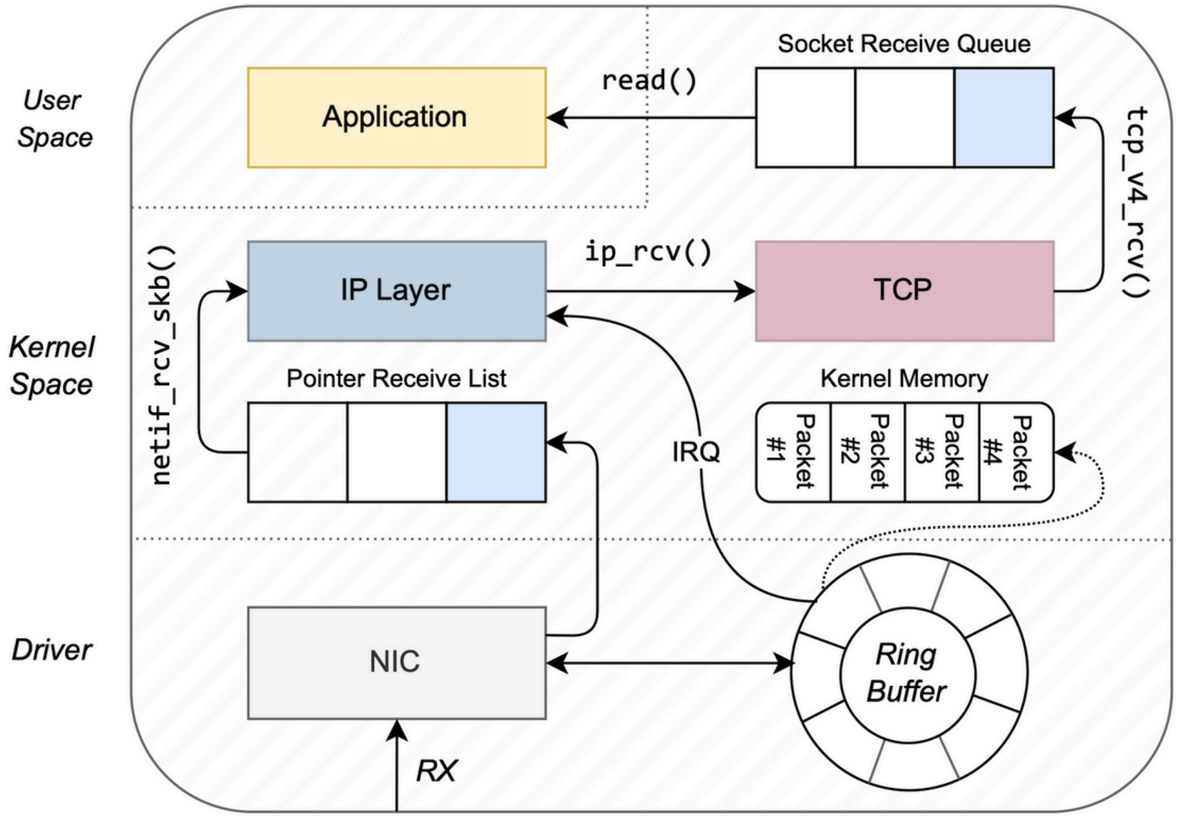
sk_buff in the Kernel



Source: Stephen et al. The Path of a Packet Through the Linux Kernel

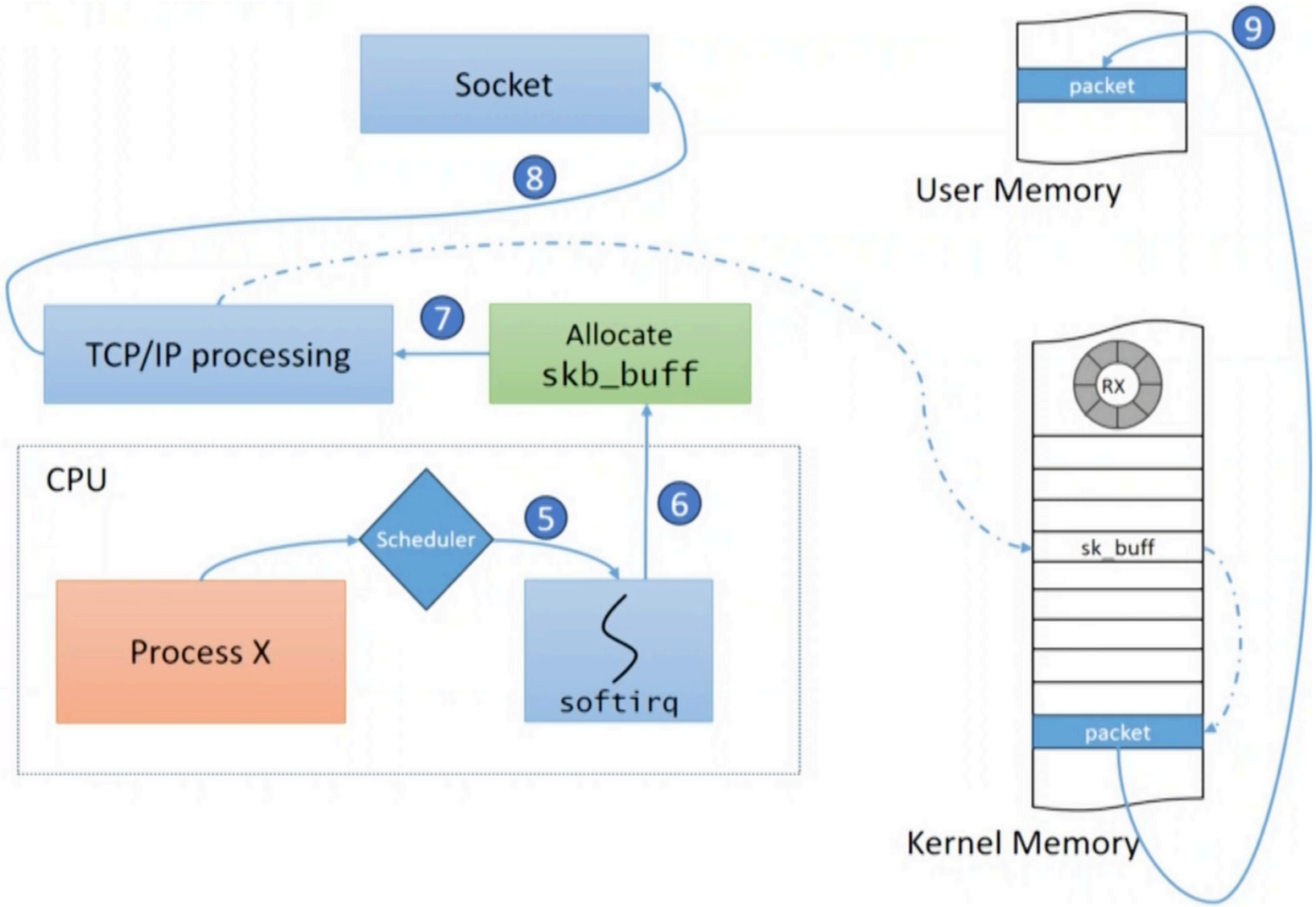
Egress and Ingress Paths through Linux Kernel Network Stack



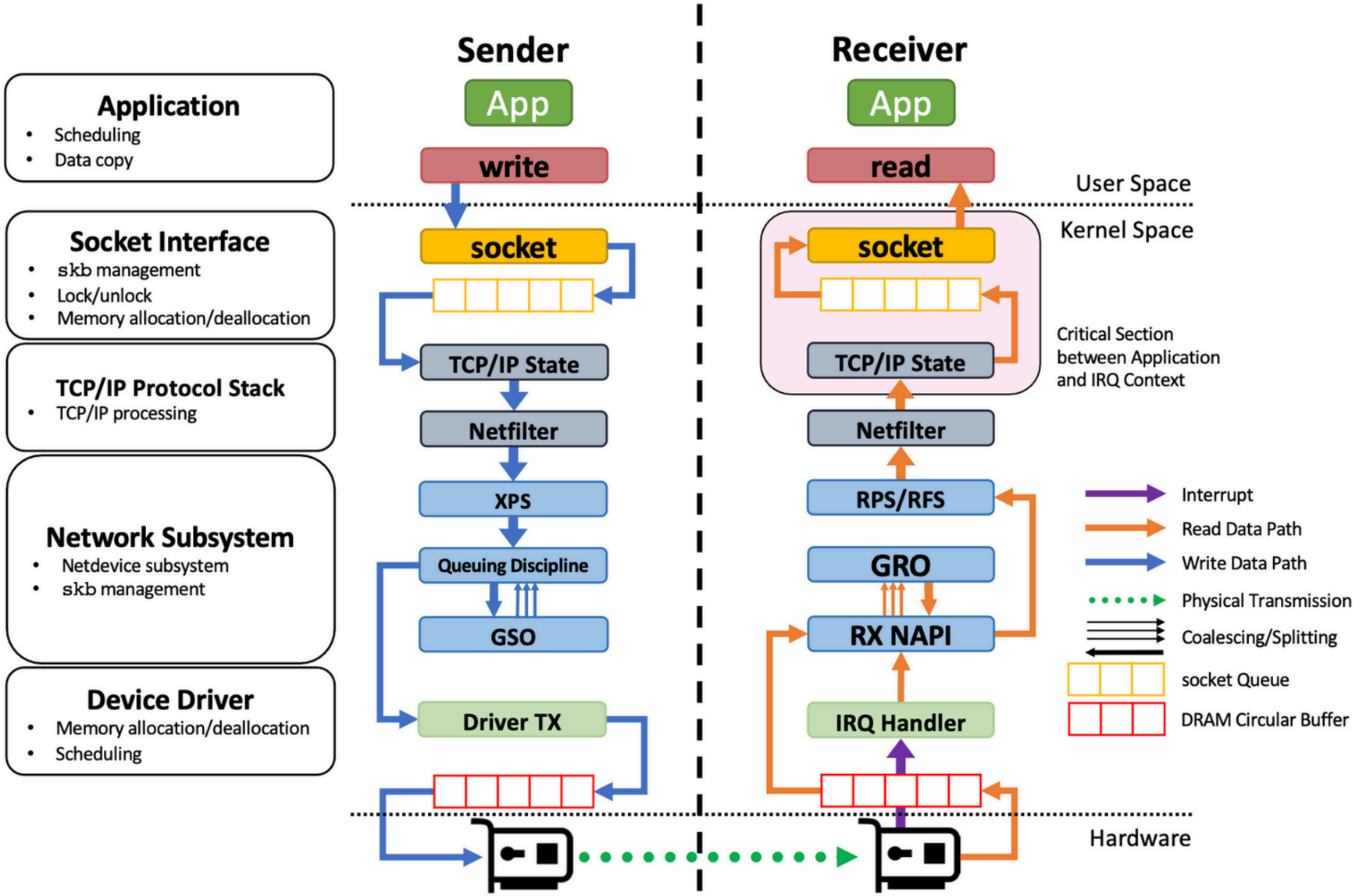


Source: [Stephen et al. The Path of a Packet Through the Linux Kernel]

Path of the packet in Kernel Network Stack

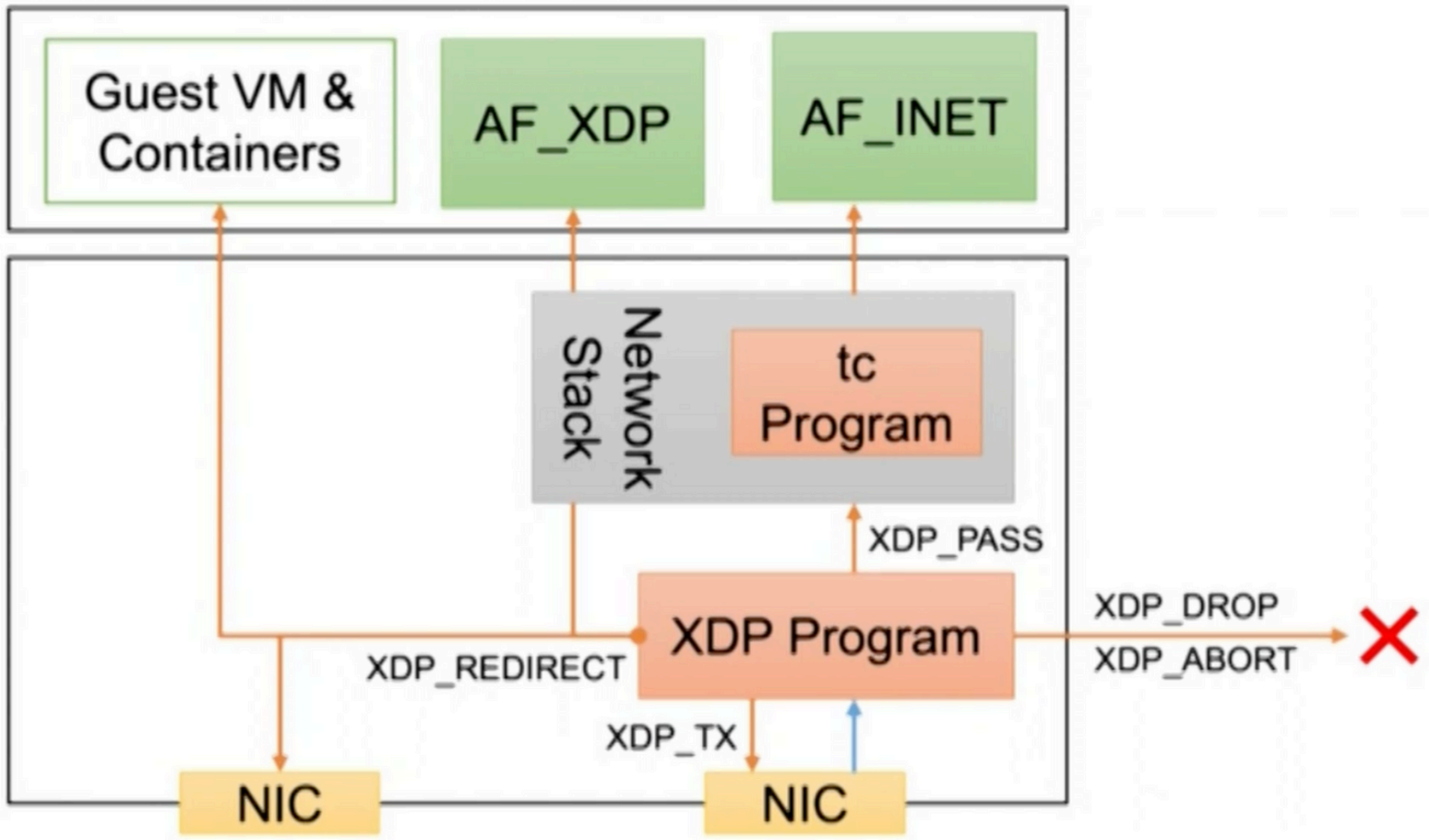


Source: [Prof. Mythili Vutukuru's Talk]



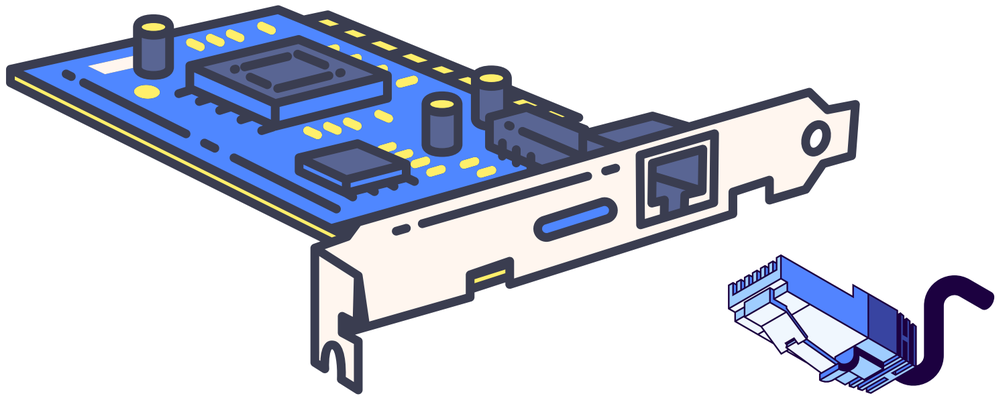
Source: Cai et al. Understanding Network Stack Overheads

BPF Program @ XDP Hook



BPF Program @ XDP Hook

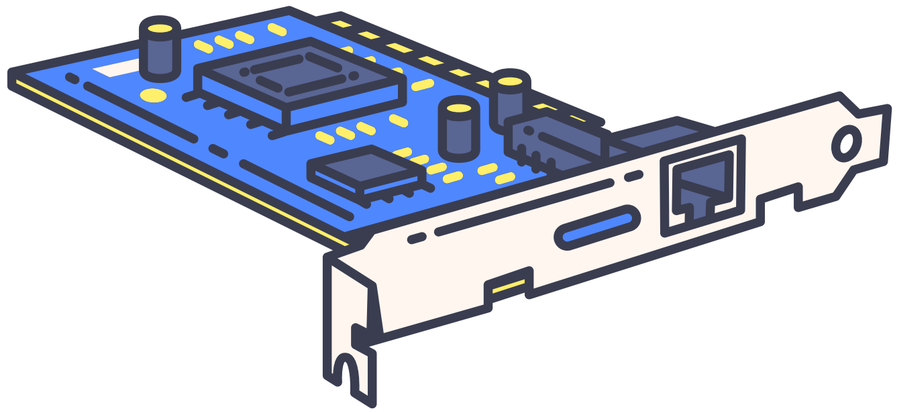
```
struct {
    __uint(type, BPF_MAP_TYPE_XSKMAP);
    __type(key, __u32);
    __type(value, __u32);
    __uint(max_entries, 64);
    __uint(pinning, LIBBPF_PIN_BY_NAME);
} xsks_map SEC(".maps");

struct {
        __uint(type, BPF_MAP_TYPE_PERCPU_ARRAY);
        __type(key, __u32);
        __type(value, __u32);
        __uint(max_entries, 64);
} xdp_stats_map SEC(".maps");

/* Header cursor to keep track of current parsing position */
struct hdr_cursor {
        void *pos;
};
```



BPF Program @ XDP Hook

- eBPF programs can be safely embedded into special hook points to add custom functionality to kernel network stack.

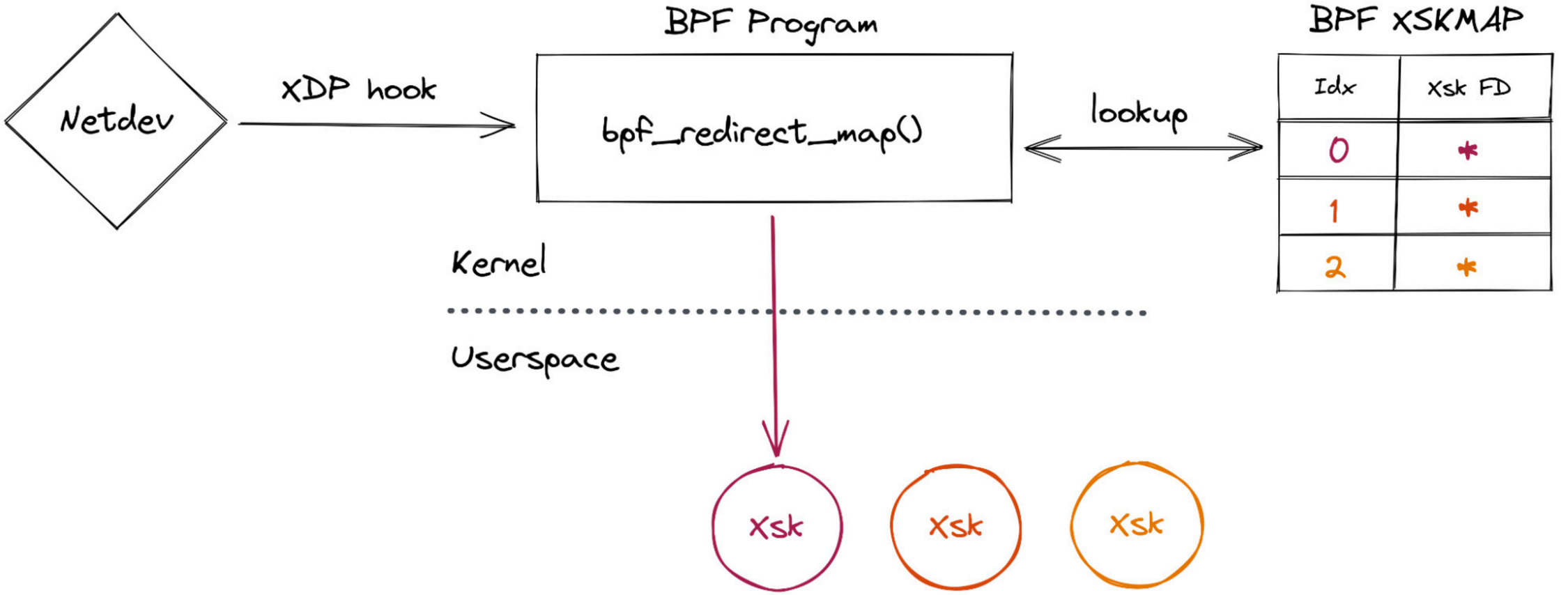
- All ingress traffic is first processes by an XDP program, it can make a decision on which traffic to pass to the stack and which to bypass.



- This allows a user to bypass traffic for very specific applications, ports (53 here) and/or protocols without disrupting the normal packet processing.

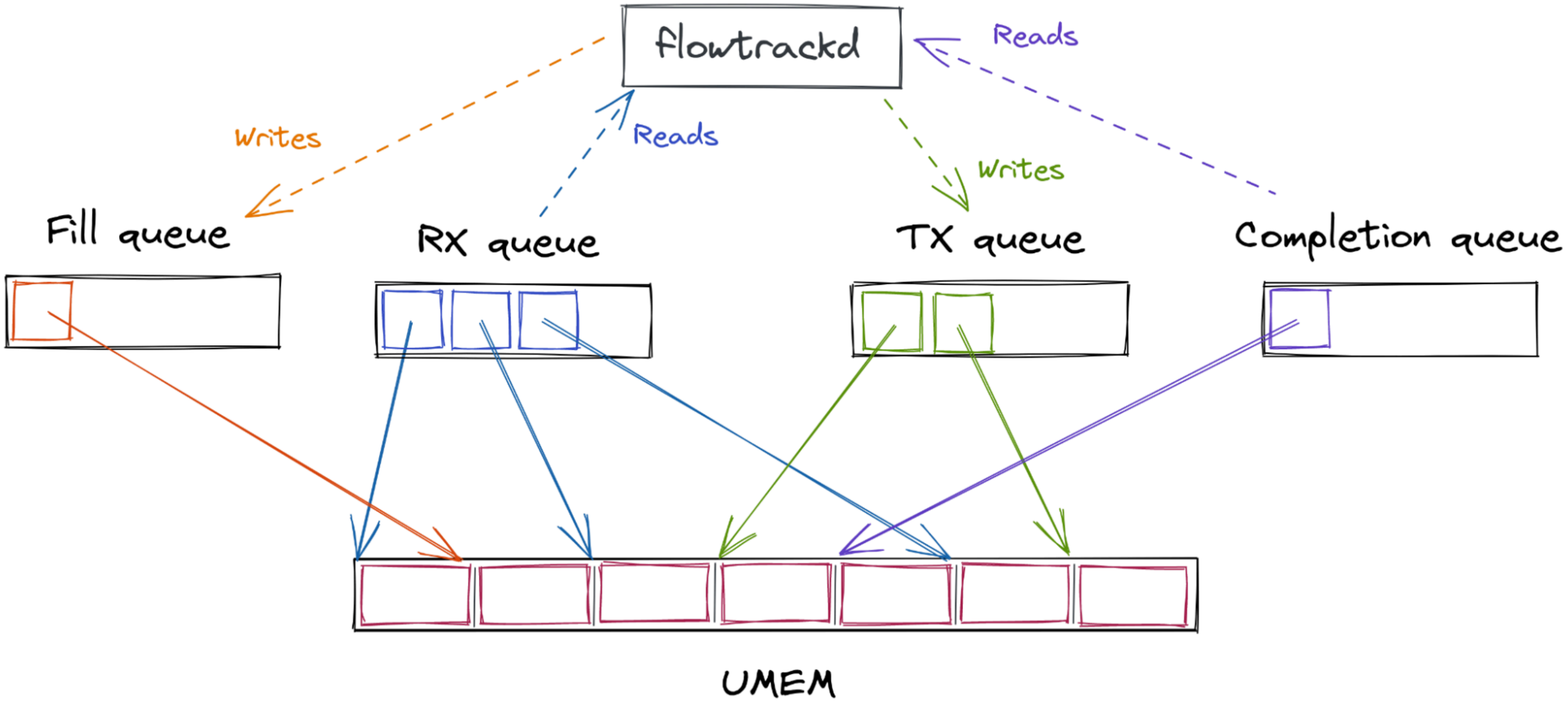
BPF Program @ XDP Hook

```c
SEC("xdp_dns_filter")
int xdp_dns_filter_func(struct xdp_md *ctx)
{
    void *data_end = (void *)(long)ctx->data_end;
    void *data = (void *)(long)ctx->data;
    int index = ctx->rx_queue_index;

    struct ethhdr *eth;
        struct iphdr *iphdr;
        struct ipv6hdr *ipv6hdr;
    struct udphdr *udphdr;

    struct hdr_cursor nh;
    int eth_type;
        int ip_type;
    int udp_len;

    /* Start next header cursor position at data start */
    nh.pos = data;

    /* Parse the Ethernet and IP headers */
    eth_type = parse_ethhdr(&nh, data_end, &eth);
        if (eth_type == bpf_htons(ETH_P_IP)) {
                ip_type = parse_iphdr(&nh, data_end, &iphdr);
                if (ip_type != IPPROTO_UDP)
                        return XDP_PASS;
        } else if (eth_type == bpf_htons(ETH_P_IPV6)) {
                ip_type = parse_ip6hdr(&nh, data_end, &ipv6hdr);
                if (ip_type != IPPROTO_UDP)
                        return XDP_PASS;
        } else {
                return XDP_PASS;
        }

    /* Check if the packet is a DNS query */
    if (ip_type == IPPROTO_UDP) {
        udp_len = parse_udphdr(&nh, data_end, &udphdr);
        if (udp_len < 0)
            return XDP_PASS;

        // Only process DNS queries: UDP dest port must be 53
        if (udphdr->dest != bpf_htons(53))
            return XDP_PASS;

        /* A set entry here means that the correspnding queue_id
            * has an active AF_XDP socket bound to it. */
        if (bpf_map_lookup_elem(&xsks_map, &index))
            return bpf_redirect_map(&xsks_map, index, 0);
    }

    return XDP_PASS; // Pass all other packets
}
```

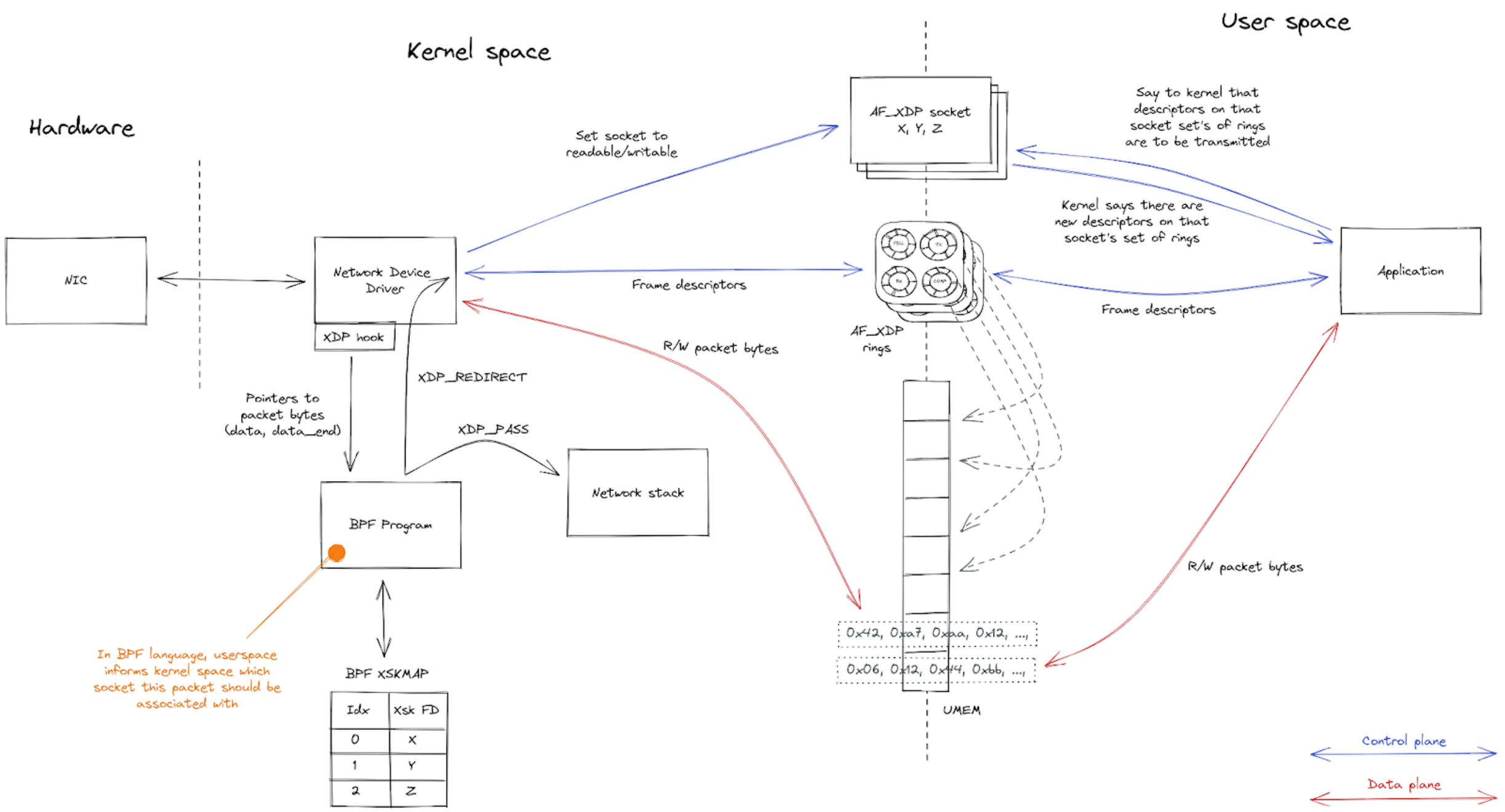


XDP Socket

Source: [Cloudflare Blog: AF_XDP Network Namespaces]



Source: [Cloudflare Blog: AF_XDP Network Namespaces]

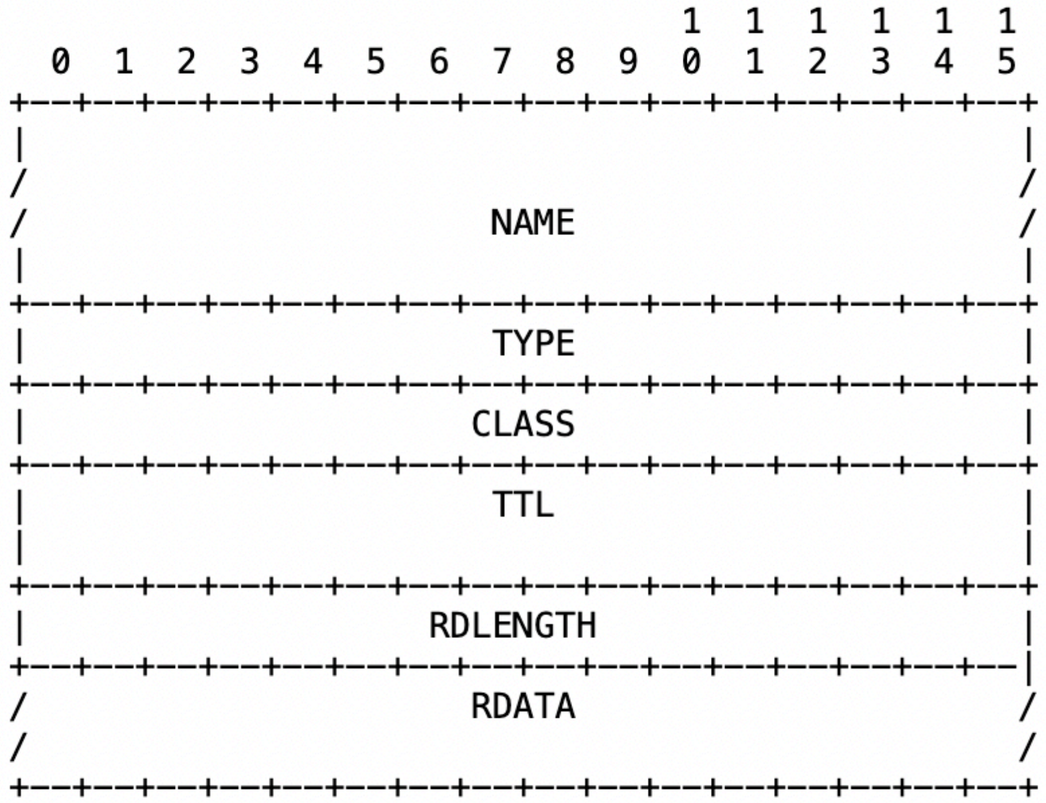


Source: [Cloudflare Blog: AF_XDP Kernel User Bug]

RFC 1035!

3.2.1. Format

All RRs have the same top level format shown below:



where:

NAME — an owner name, i.e., the name of the node to which this resource record pertains.

TYPE — two octets containing one of the RR TYPE codes.

CLASS — two octets containing one of the RR CLASS codes.

TTL — a 32 bit signed integer that specifies the time interval that the resource record may be cached before the source of the information should again be consulted. Zero values are interpreted to mean that the RR can only be used for the transaction in progress, and should not be cached. For example, SOA records are always distributed with a zero TTL to prohibit caching. Zero values can also be used for extremely volatile data.

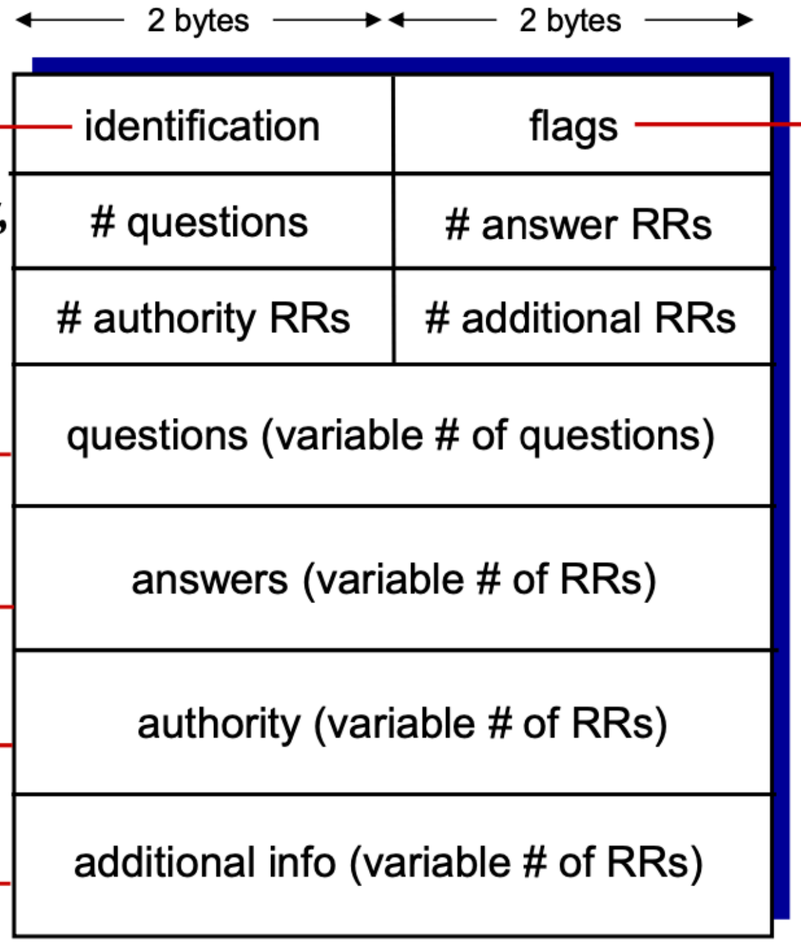
RDLENGTH — an unsigned 16 bit integer that specifies the length in octets of the RDATA field.

Source: RFC 1035: DNS

DNS Protocol Messages

msg header

- identification: 16 bit # for query, reply to query uses same #



- flags:
  - query or reply
  - recursion desired
  - recursion available
  - reply is authoritative

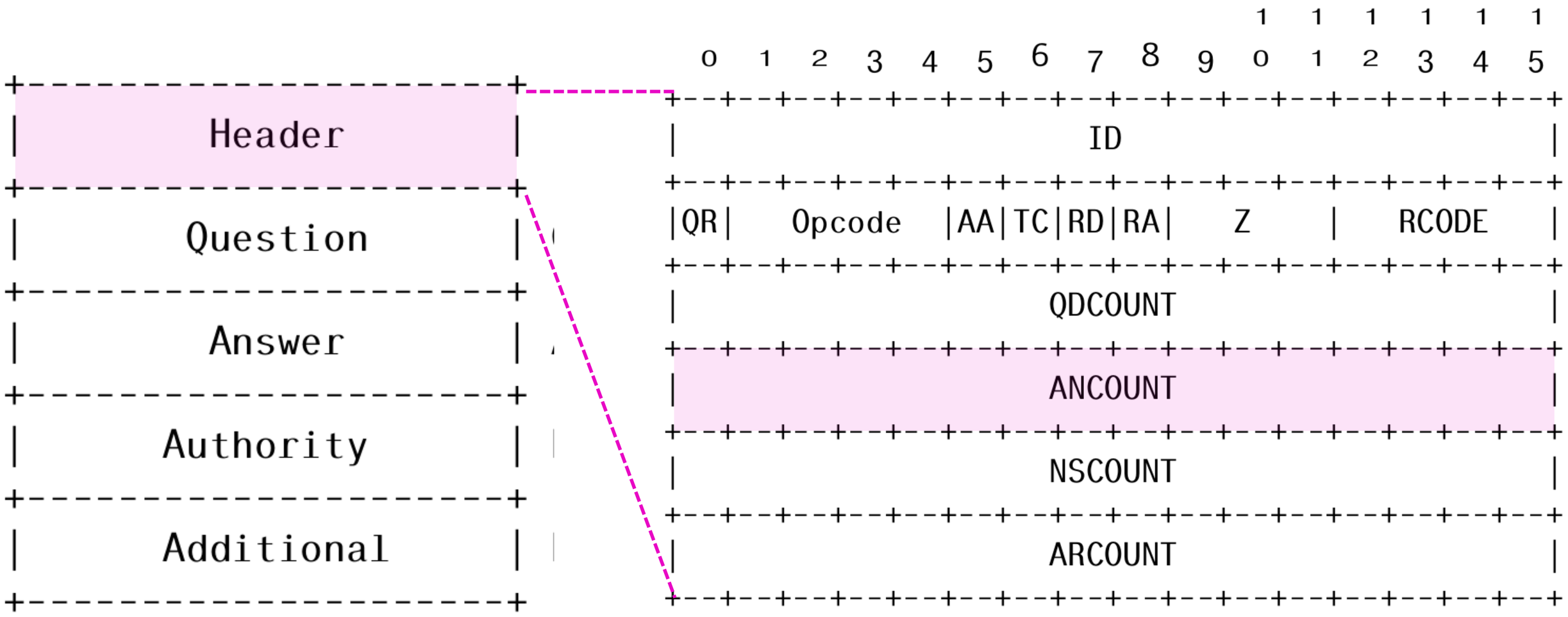
name, type fields for a query

RRs in response to query

records for authoritative servers

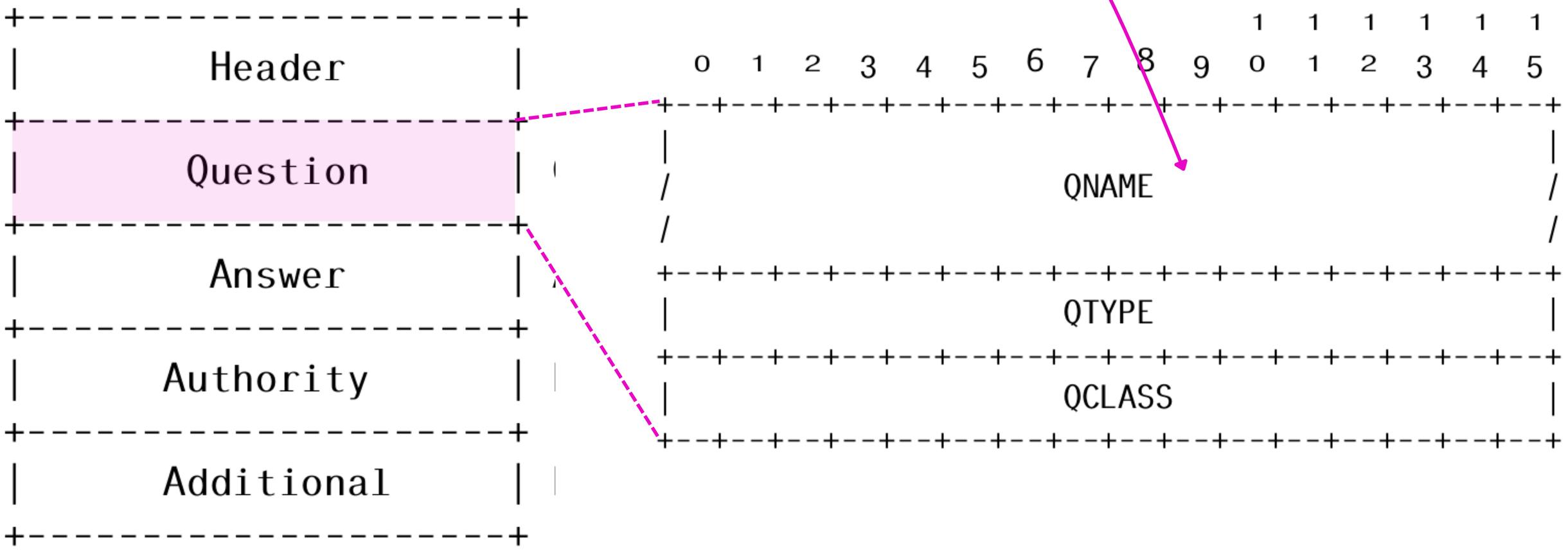
additional "helpful" info that may be used

DNS Packet Structure



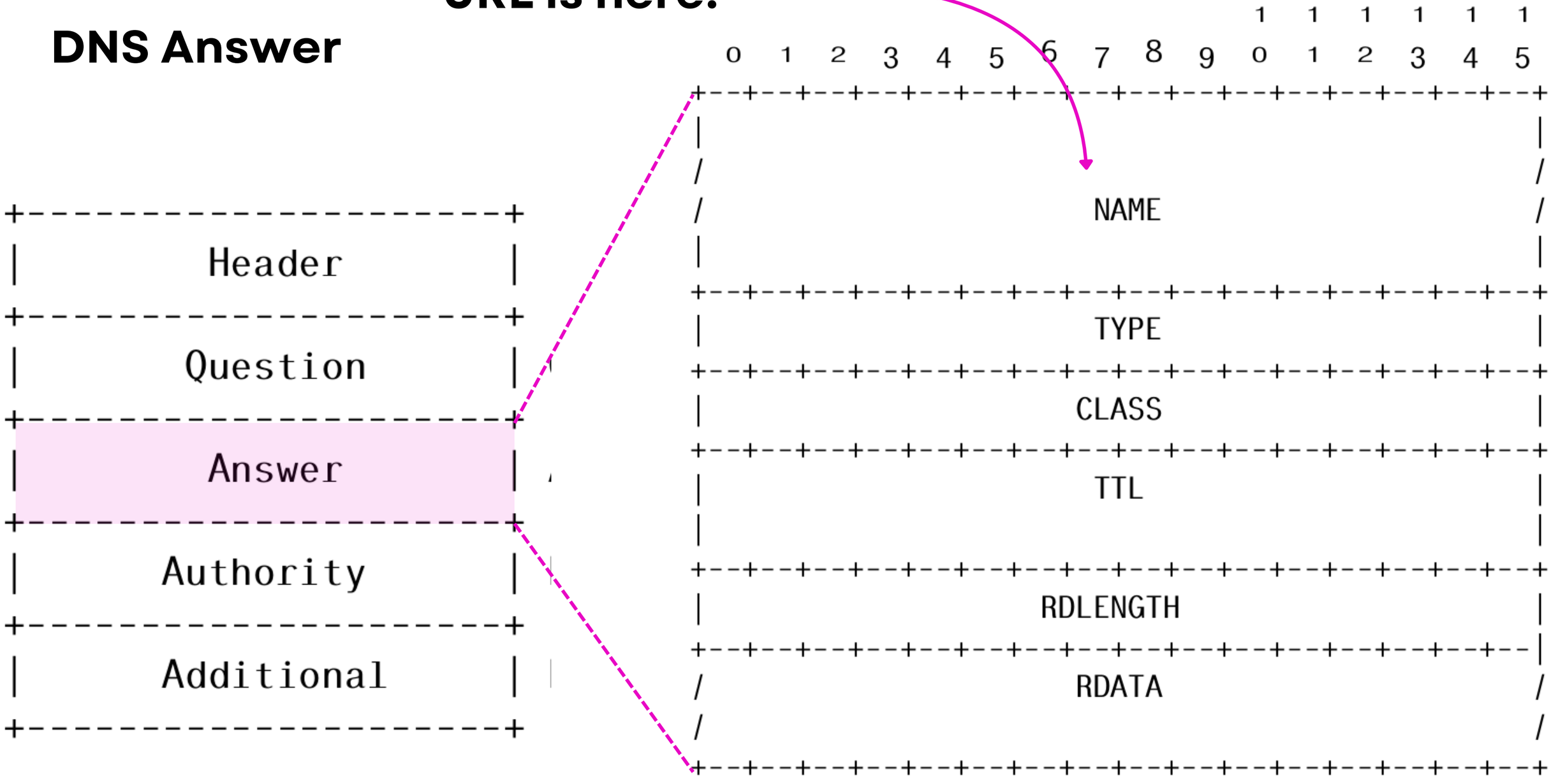
DNS Question

URL is here!

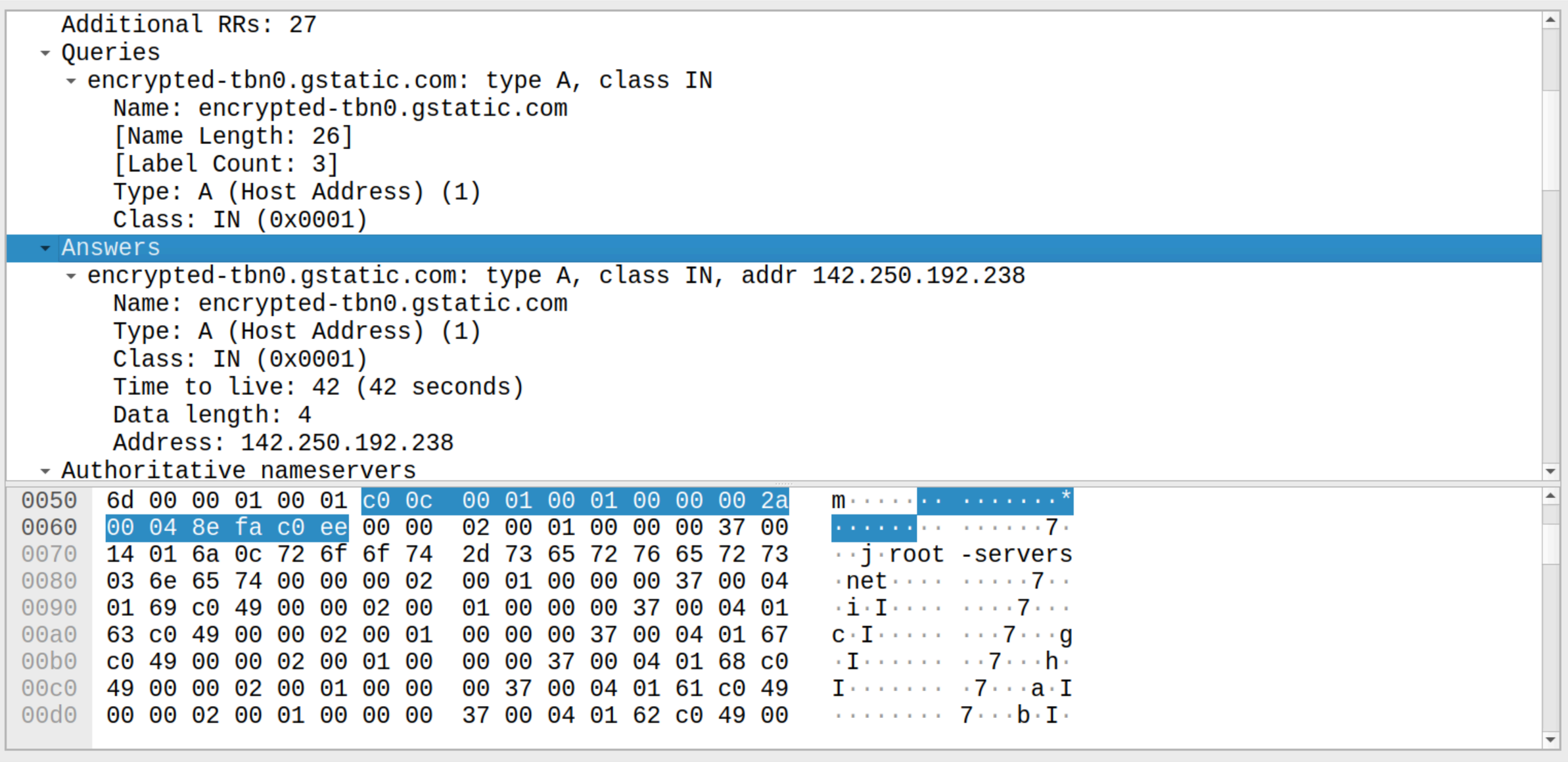


DNS Answer

URL is here!



DNS Packet on Wireshark



BUT!

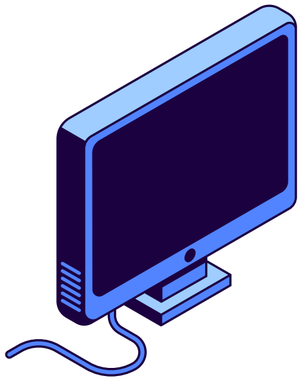
```
▸ Frame 2: 105 bytes on wire (840 bits), 105 bytes captured (840 bits)
▸ Ethernet II, Src: 06:44:b6:11:6d:ef (06:44:b6:11:6d:ef), Dst: fe:59:35:d0:61:aa (fe:59:35:d0:61:aa)
▸ Internet Protocol Version 4, Src: 10.11.1.1, Dst: 10.11.1.2
▸ User Datagram Protocol, Src Port: 39138, Dst Port: 53
▾ Domain Name System (response)
    Transaction ID: 0x1234
  ▸ Flags: 0x8180 Standard query response, No error
    Questions: 1
    Answer RRs: 2
    Authority RRs: 0
    Additional RRs: 0
  ▸ Queries
  ▾ Answers
    ▾ www.iitgn.ac.in: type CNAME, class IN, cname iitgn.ac.in
        Name: www.iitgn.ac.in
        Type: CNAME (Canonical NAME for an alias) (5)
        Class: IN (0x0001)
        Time to live: 300 (5 minutes)
        Data length: 2
        CNAME: iitgn.ac.in
    ▾ iitgn.ac.in: type A, class IN, addr 4.213.170.54
        Name: iitgn.ac.in
        Type: A (Host Address) (1)
        Class: IN (0x0001)
        Time to live: 300 (5 minutes)
        Data length: 4
        Address: 4.213.170.54
    [Unsolicited: True]
```

User Program (XSK - XDP Socket)

stub resolver
(us)

User Program (XSK - XDP Socket)

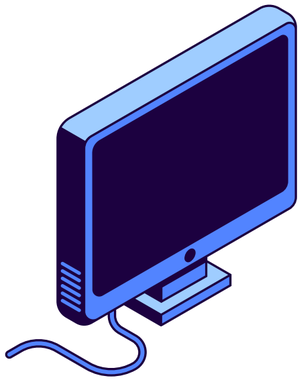
host



stub resolver
(us)

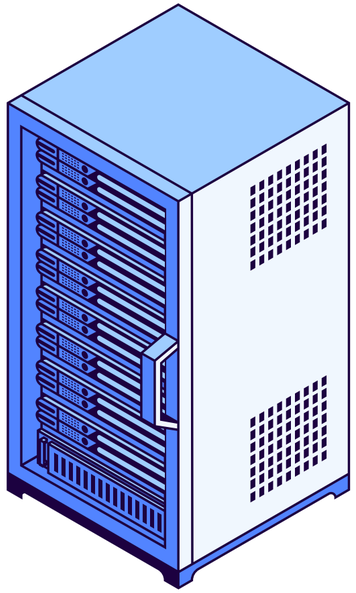
User Program (XSK - XDP Socket)

host



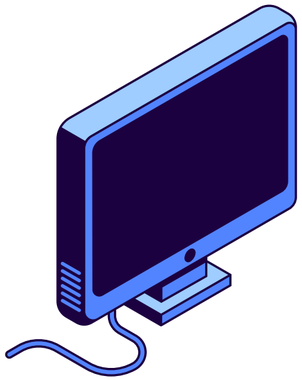
stub resolver
(us)

DNS Server
(8.8.8.8)

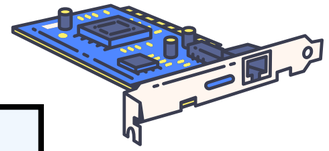


User Program (XSK - XDP Socket)

host



stub resolver
(us)

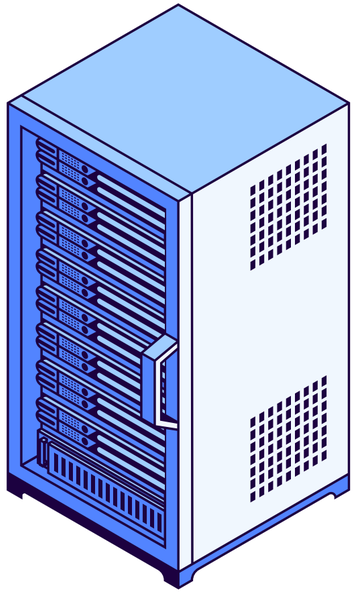


BPF
Program

UDP
dst port = 53

NIC

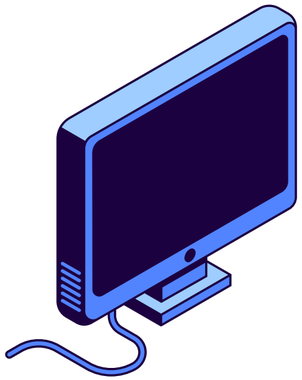
DNS Server
(8.8.8.8)



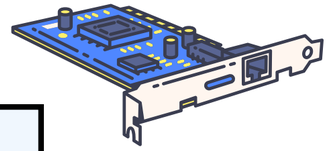
User Program (XSK - XDP Socket)

host

```
dig @stubIP <url>
```



stub resolver
(us)

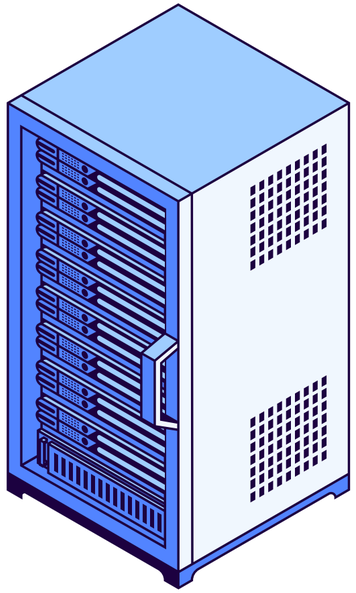


BPF
Program

NIC

UDP
dst port = 53

DNS Server
(8.8.8.8)



User Program (XSK - XDP Socket)

host

`dig @stubIP <url>`

stub resolver
(us)

DNS Server
(8.8.8.8)

AF_XDP

SOCK_DGRAM

BPF
Program

NIC

UDP
dst port = 53

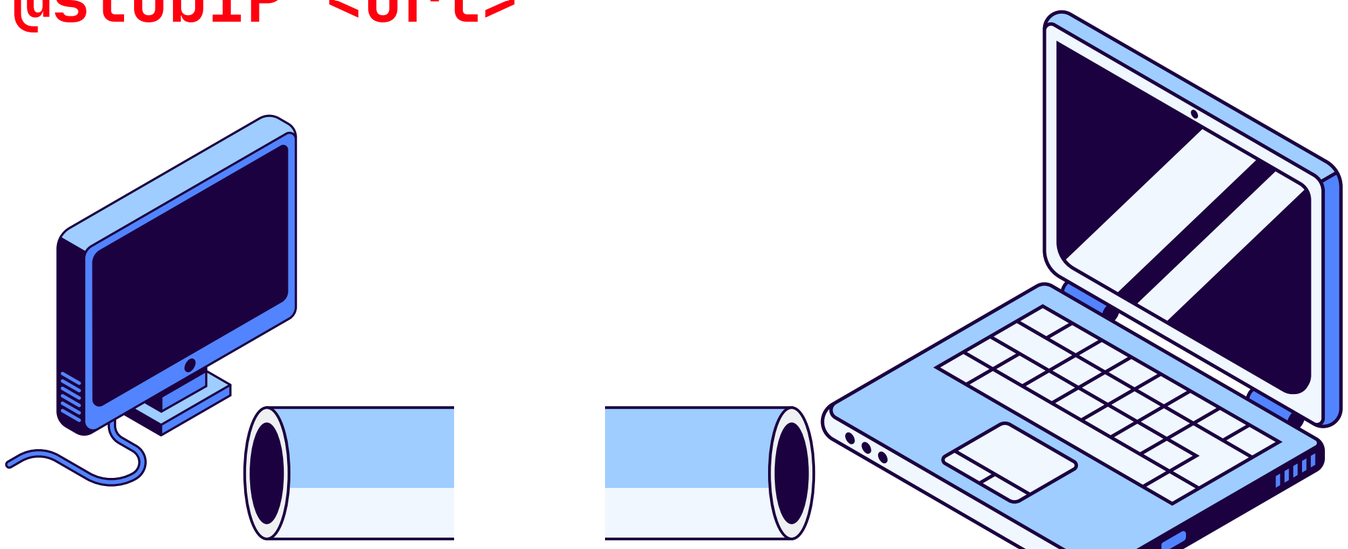
User Program (XSK - XDP Socket)

host

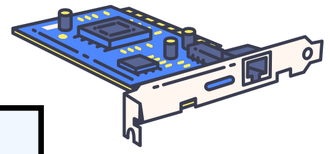
`dig @stubIP <url>`

stub resolver
(us)

DNS Server
(8.8.8.8)



AF_XDP

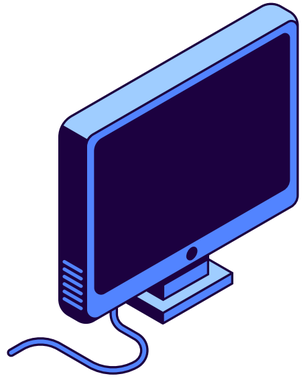


BPF
Program

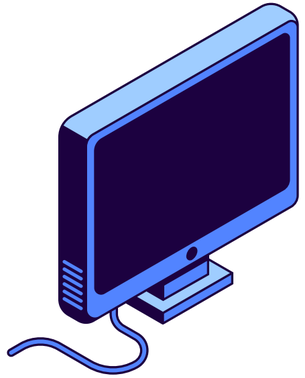
NIC

UDP
dst port = 53

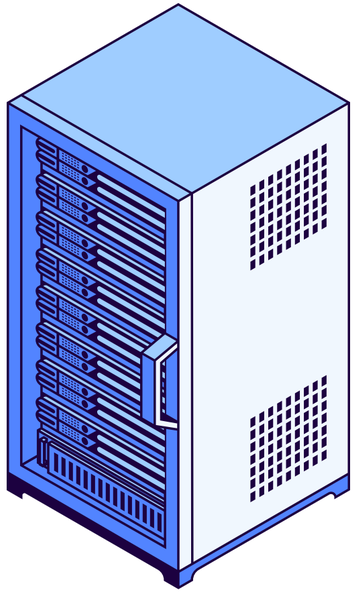
XDP
Redirect

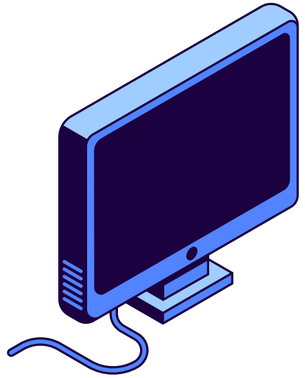


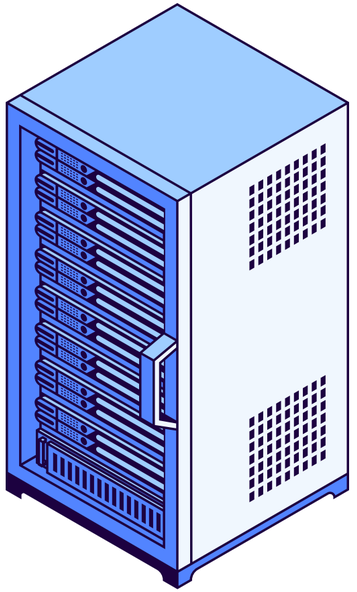
XDP
Redirect



XDP
Redirect







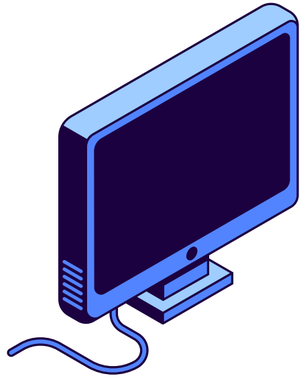
```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|Version|  IHL  |Type of Service|          Total Length         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|         Identification        |Flags|      Fragment Offset    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Time to Live |    Protocol   |         Header Checksum       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       Source Address                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Destination Address                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Options                    |    Padding    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

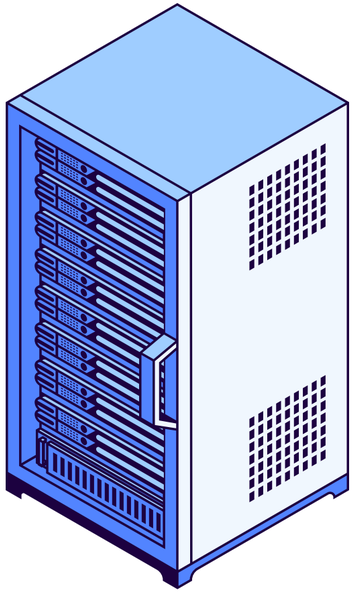
```
                                1  1  1  1  1  1
  0  1  2  3  4  5  6  7  8  9  0  1  2  3  4  5
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                                               |
/                                               /
/                      NAME                     /
|                                               |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                      TYPE                     |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                      CLASS                    |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                      TTL                      |
|                                               |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                    RDLENGTH                   |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--|
/                     RDATA                     /
/                                               /
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```

XDP
Redirect

Query RR

UDP Payload



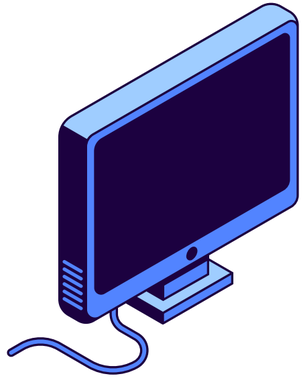


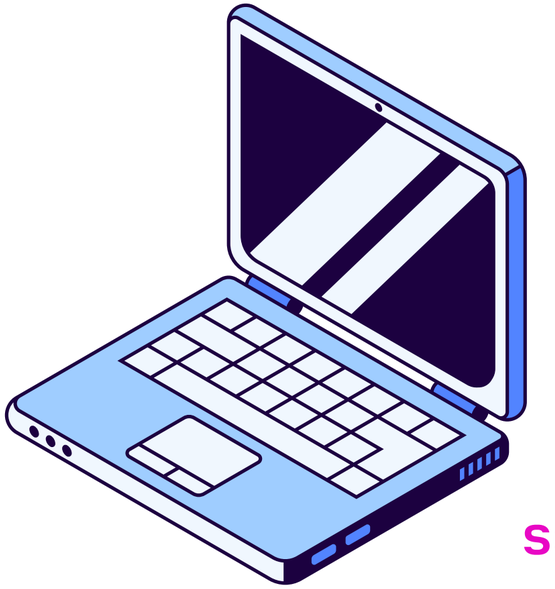
```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|Version|  IHL  |Type of Service|          Total Length         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|         Identification        |Flags|      Fragment Offset    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Time to Live |    Protocol   |         Header Checksum       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       Source Address                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Destination Address                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Options                    |    Padding    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

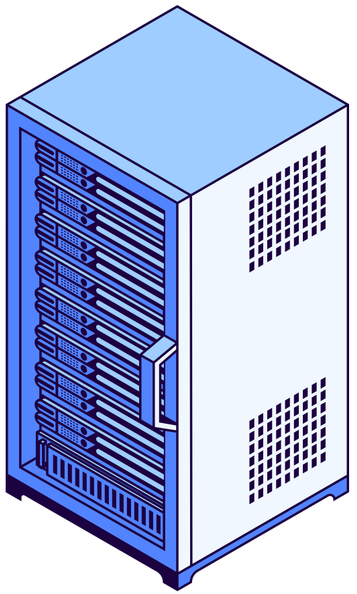
```
                                1  1  1  1  1  1
  0  1  2  3  4  5  6  7  8  9  0  1  2  3  4  5
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                                               |
/                                               /
/                      NAME                     /
|                                               |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                      TYPE                     |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                     CLASS                     |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                      TTL                      |
|                                               |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                   RDLENGTH                    |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--|
/                     RDATA                     /
/                                               /
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```

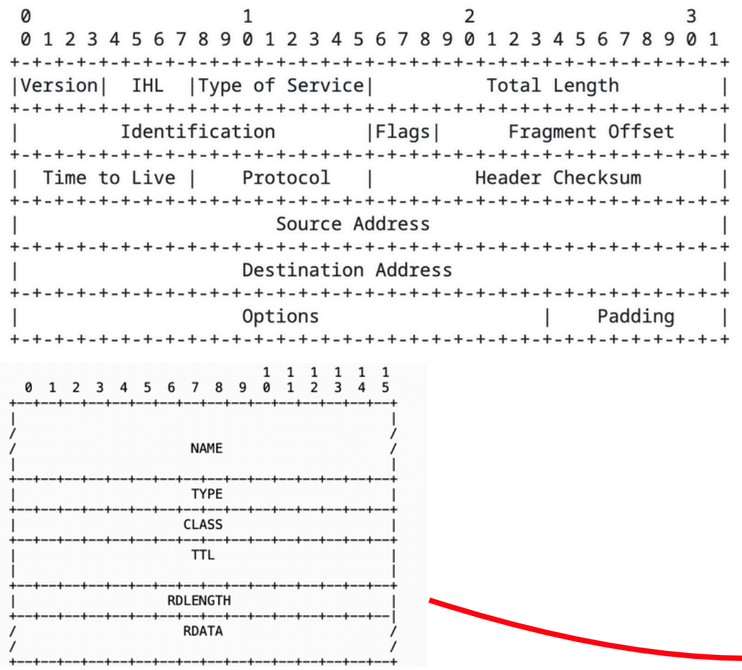
XDP
Redirect

Query RR
UDP Payload





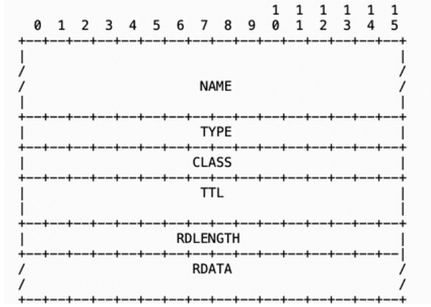




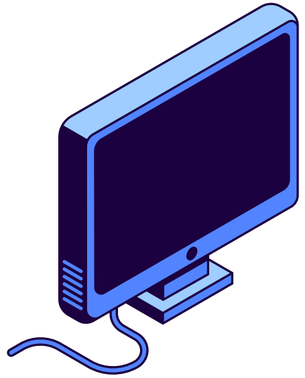
Query RR
UDP Payload

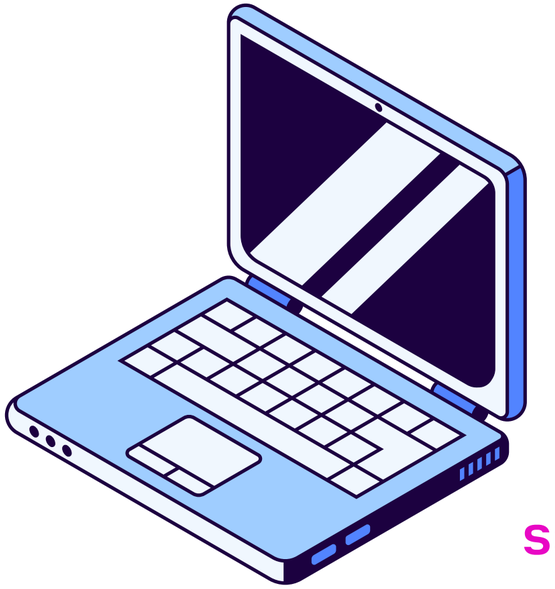
XDP
Redirect

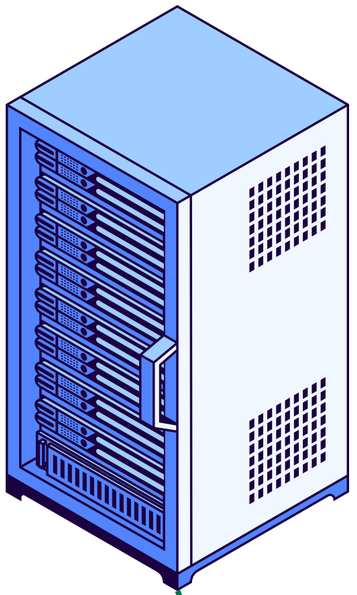
Query RR
UDP Payload

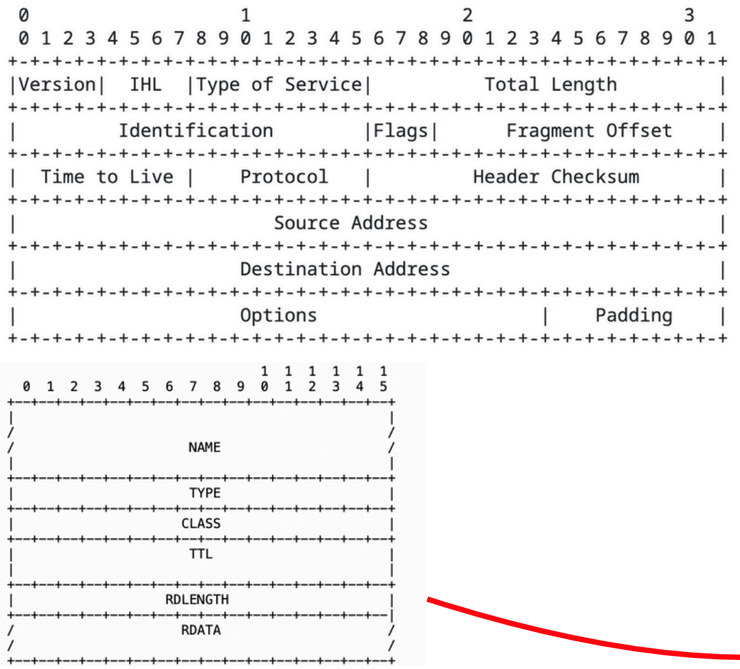


`sendto(UDP_SOCK_FD)`





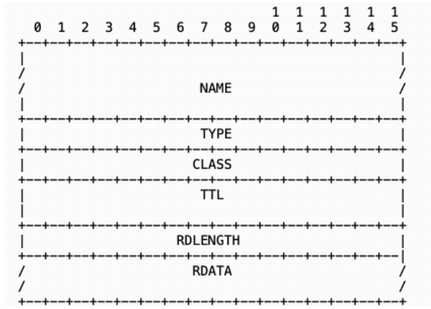




Query RR
UDP Payload

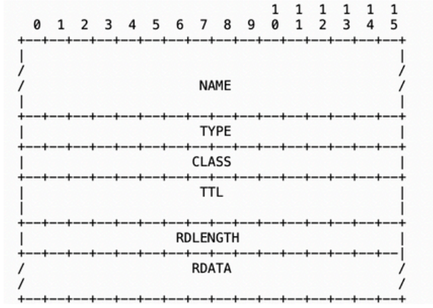
XDP
Redirect

Query RR
UDP Payload

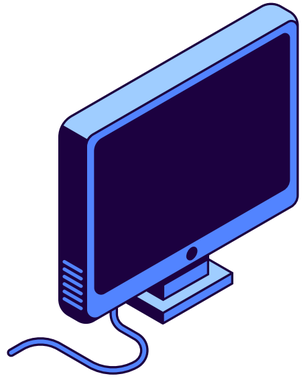


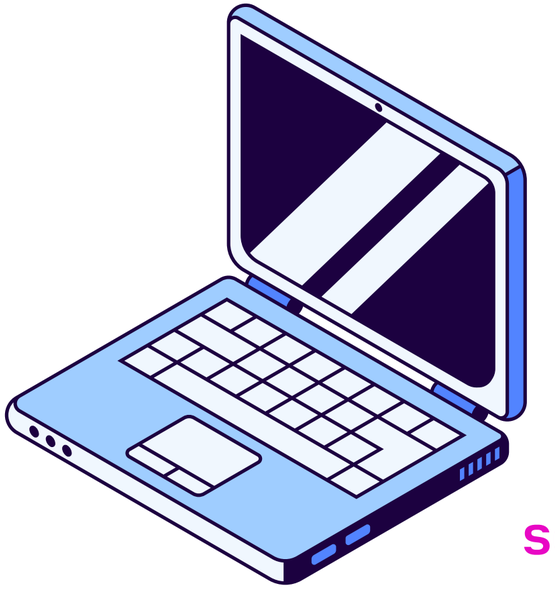
`sendto(UDP_SOCK_FD)`

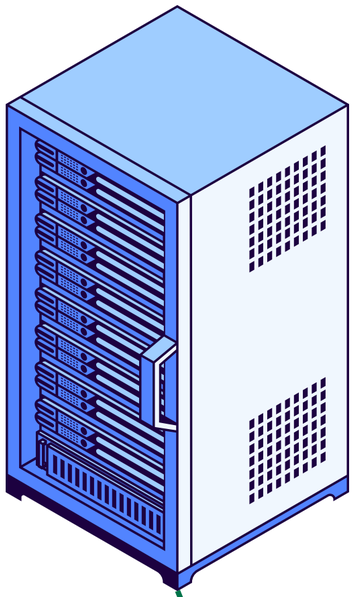
Answer RR
UDP Payload

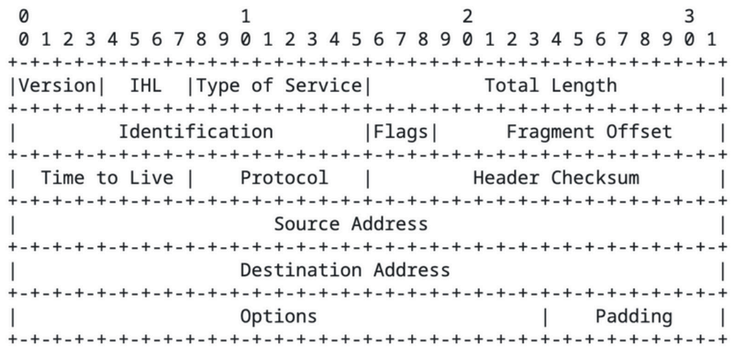


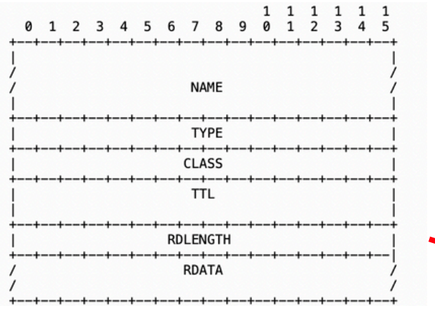
CACHE IT!







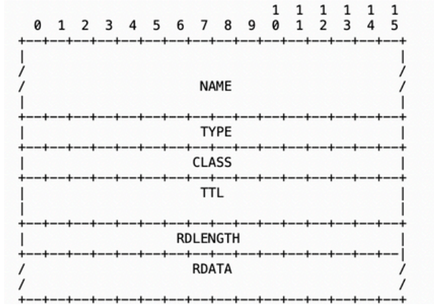




Query RR
UDP Payload

XDP
Redirect

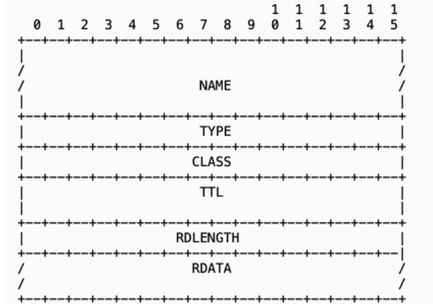
Query RR
UDP Payload



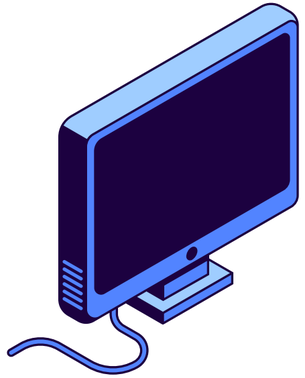
`sendto(UDP_SOCK_FD)`

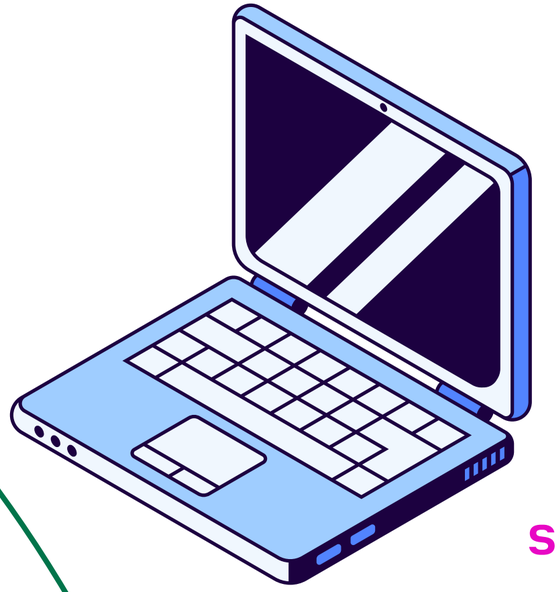
`rcvfrom(UDP_SOCK_FD)`

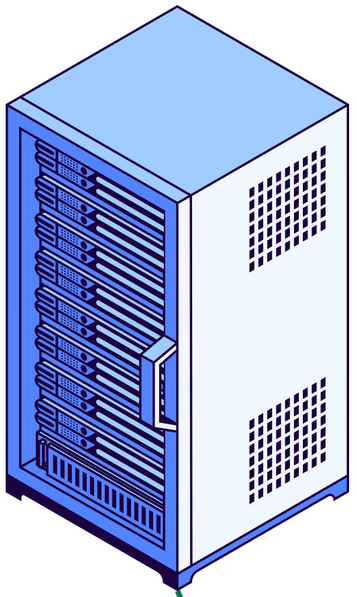
Answer RR
UDP Payload



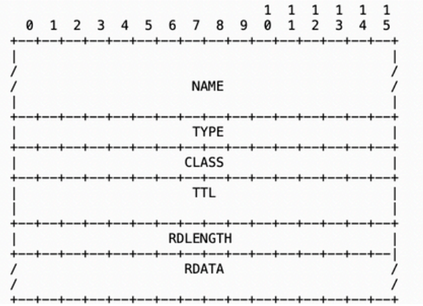
CACHE IT!

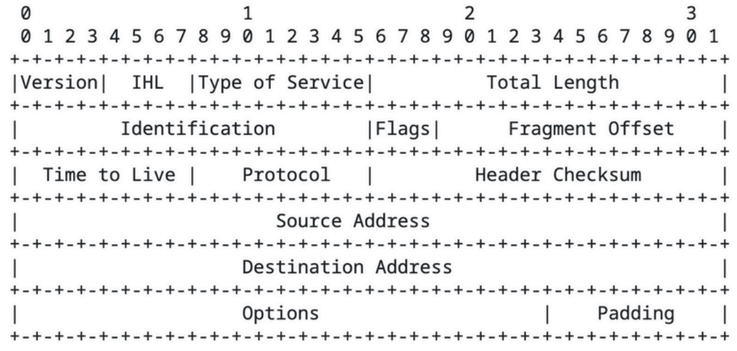






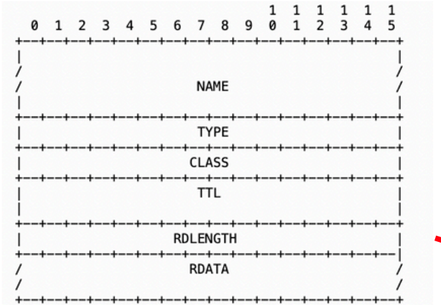
Query RR
UDP Payload





SWAP!

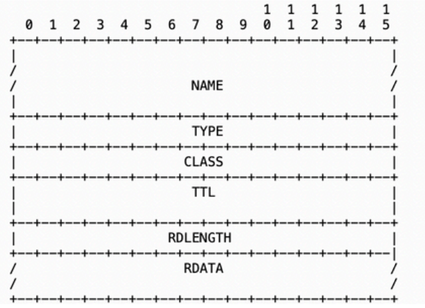
XDP
Redirect



Query RR
UDP Payload

`sendto(UDP_SOCK_FD)`

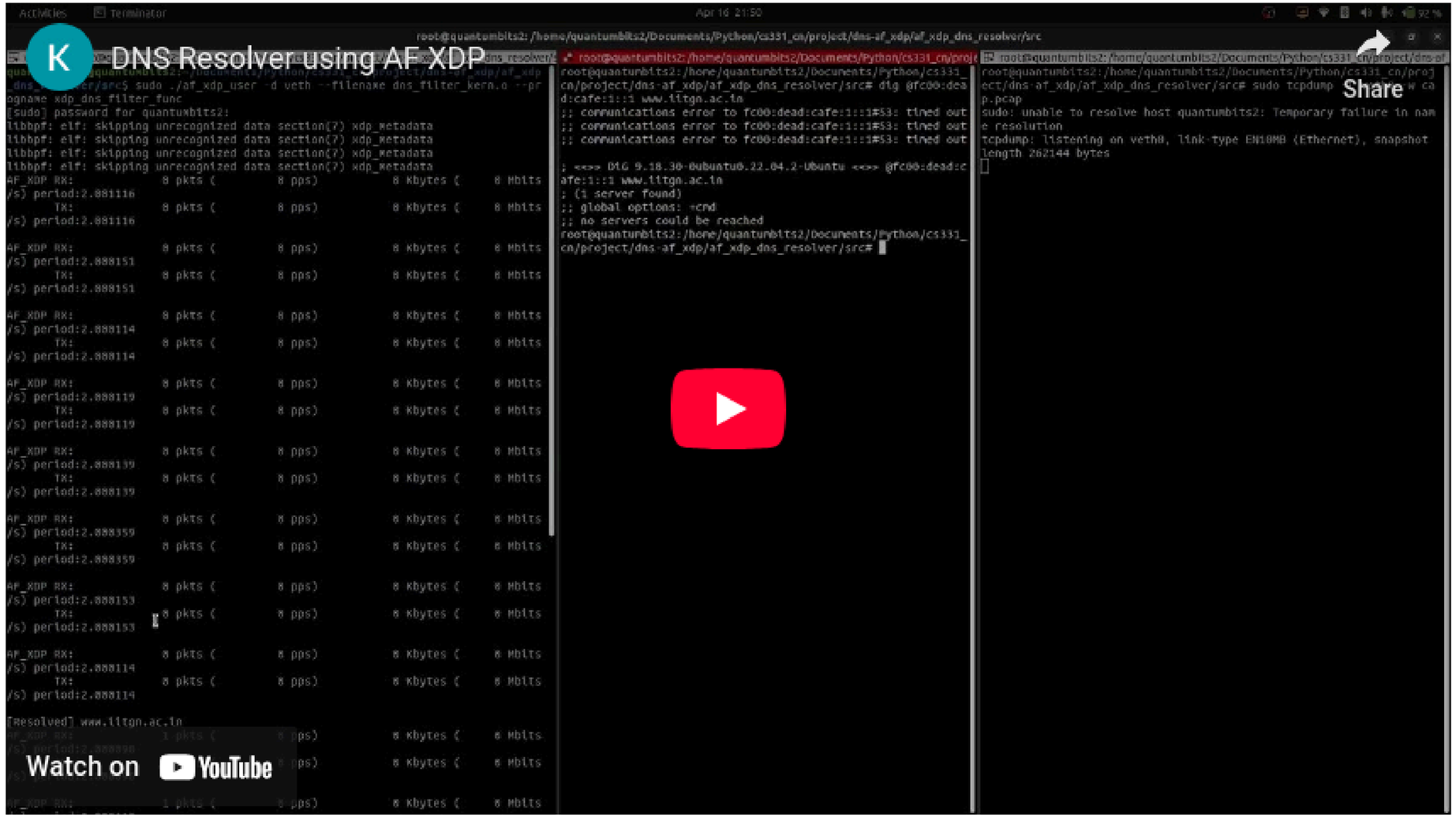
Answer RR
UDP Payload



`rcvfrom(UDP_SOCK_FD)`

CACHE IT!

Demonstration



User Program (XSK - XDP Socket)

```c
#define NUM_FRAMES 4096
#define FRAME_SIZE XSK_UMEM__DEFAULT_FRAME_SIZE
#define RX_BATCH_SIZE 64
#define INVALID_UMEM_FRAME UINT64_MAX
#define DNS_PORT 53
#define MAX_DNS_PACKET 512
```

```c
struct xsk_umem_info
{
        struct xsk_ring_prod fq;
        struct xsk_ring_cons cq;
        struct xsk_umem *umem;
        void *buffer;
};
```

```c
static struct xsk_socket_info *xsk_configure_socket(struct config *cfg, struct xsk_umem_info *umem)

{
        struct xsk_socket_config xsk_cfg;
        struct xsk_socket_info *xsk_info;
        uint32_t idx;
        int i;
        int ret;
        uint32_t prog_id;

        xsk_info = calloc(1, sizeof(*xsk_info));
        if (!xsk_info)
                return NULL;

        xsk_info->umem = umem;
        xsk_cfg.rx_size = XSK_RING_CONS__DEFAULT_NUM_DESCS;
        xsk_cfg.tx_size = XSK_RING_PROD__DEFAULT_NUM_DESCS;
        xsk_cfg.xdp_flags = cfg->xdp_flags;
        xsk_cfg.bind_flags = cfg->xsk_bind_flags;
        xsk_cfg.libbpf_flags = (custom_xsk) ? XSK_LIBBPF_FLAGS__INHIBIT_PROG_LOAD : 0;
        ret = xsk_socket__create(&xsk_info->xsk, cfg->ifname,
                                 cfg->xsk_if_queue, umem->umem, &xsk_info->rx,
                                 &xsk_info->tx, &xsk_cfg);
```

```c
if (custom_xsk)
{
        ret = xsk_socket__update_xskmap(xsk_info->xsk, xsk_map_fd);
        if (ret)
                goto error_exit;
}
else
{
        /* Getting the program ID must be after the xdp_socket__create() call */
        if (bpf_xdp_query_id(cfg->ifindex, cfg->xdp_flags, &prog_id))
                goto error_exit;
}

/* Initialize umem frame allocation */
for (i = 0; i < NUM_FRAMES; i++)
        xsk_info->umem_frame_addr[i] = i * FRAME_SIZE;

xsk_info->umem_frame_free = NUM_FRAMES;

/* Stuff the receive path with buffers, we assume we have enough */
ret = xsk_ring_prod__reserve(&xsk_info->umem->fq,
                             XSK_RING_PROD__DEFAULT_NUM_DESCS,
                             &idx);

if (ret != XSK_RING_PROD__DEFAULT_NUM_DESCS)
        goto error_exit;

for (i = 0; i < XSK_RING_PROD__DEFAULT_NUM_DESCS; i++)
        *xsk_ring_prod__fill_addr(&xsk_info->umem->fq, idx++) =
                xsk_alloc_umem_frame(xsk_info);

xsk_ring_prod__submit(&xsk_info->umem->fq,
                      XSK_RING_PROD__DEFAULT_NUM_DESCS);

return xsk_info;
```

User Program (XSK - XDP Socket)

```c
static bool process_packet(struct xsk_socket_info *xsk, uint64_t addr, uint32_t len, int sock, struct sockaddr_in *resolver_addr)
{
    uint8_t *pkt = xsk_umem__get_data(xsk->umem->buffer, addr);
    struct ethhdr *eth = (struct ethhdr *)pkt;
    struct dns_response *dns_res = NULL;
    bool is_ipv4 = (ntohs(eth->h_proto) == ETH_P_IP);
    bool is_ipv6 = (ntohs(eth->h_proto) == ETH_P_IPV6);
    uint32_t tx_idx = 0;
    int ret;

    if (is_ipv4) {
        struct iphdr *ip = (struct iphdr *)(pkt + sizeof(struct ethhdr));
        if (ip->protocol == IPPROTO_UDP) {
            struct udphdr *udp = (struct udphdr *)((uint8_t *)ip + ip->ihl * 4);
            uint8_t *dns_payload = (uint8_t *)udp + sizeof(struct udphdr);
                        int dns_payload_len = ntohs(udp->len) - sizeof(struct udphdr);
            char hostname[256];
            parse_dns_query_name(dns_payload + 12, hostname);
            dns_res = resolve_and_log_ip(hostname, dns_payload, dns_payload_len, sock, resolver_addr);
```

User Program (XSK - XDP Socket)

```
// Parse question length
while (dns_payload[12 + question_len] != 0x00 && (12 + question_len) < dns_payload_len) {
    question_len++;
}
question_len += 5; // Null byte + QTYPE + QCLASS
if (question_len + 12 > dns_payload_len) {
    fprintf(stderr, "Invalid question length for %s\n", hostname);
    free(dns_res);
    return false;
}
```

```
            memcpy(dns_payload, dns_res->payload, dns_res->len);
            udp->len = htons(dns_res->len + sizeof(struct udphdr));
            ip->tot_len = htons(dns_res->len + sizeof(struct udphdr) + ip->ihl * 4);
            len = sizeof(struct ethhdr) + ip->ihl * 4 + sizeof(struct udphdr) + dns_res->len;
            udp->check = 0;
            // udp->check = csum16_add(csum16_sub(~ip->check, htons(udp->len)), udp->len);

// Swap addresses
uint8_t tmp_mac[ETH_ALEN];
memcpy(tmp_mac, eth->h_dest, ETH_ALEN);
memcpy(eth->h_dest, eth->h_source, ETH_ALEN);
memcpy(eth->h_source, tmp_mac, ETH_ALEN);
uint32_t tmp_ip = ip->saddr;
ip->saddr = ip->daddr;
ip->daddr = tmp_ip;
```

User Program (XSK - XDP Socket)

```
ret = xsk_ring_prod__reserve(&xsk->tx, 1, &tx_idx);
if (ret != 1) {
    fprintf(stderr, "Failed to reserve TX slot\n");
    return false;
}

xsk_ring_prod__tx_desc(&xsk->tx, tx_idx)->addr = addr;
xsk_ring_prod__tx_desc(&xsk->tx, tx_idx)->len = len;
xsk_ring_prod__submit(&xsk->tx, 1);
xsk->outstanding_tx++;

xsk->stats.tx_bytes += len;
xsk->stats.tx_packets++;
pkt_counter++;
// printf("Packet %d sent: %s\n", pkt_counter, dns_res && dns_res->ipstr[0] ? dns_res->ipstr : "unknown");
return true;
```

User Program (XSK - XDP Socket)

```
// Check cache
json_t *cached_bin_path = json_object_get(cache, hostname);
if (cached_bin_path)
{
    const char *cached_bin_filepath = json_string_value(cached_bin_path);
    printf("[Cache Hit] %s\n", hostname);
            FILE *bin_fp = fopen(cached_bin_filepath, "rb");

            fseek(bin_fp, 0, SEEK_END);
            size_t file_size = ftell(bin_fp);
            fseek(bin_fp, 0, SEEK_SET);

            result->len = file_size;
            fread(result->payload, 1, file_size, bin_fp);
            fclose(bin_fp);
            json_decref(cache);
            return result;
}
```

User Program (XSK - XDP Socket)

```c
int main(int argc, char **argv)
{
        /* CREATE ONE DGRAM SOCKET HERE ITSELF! */
        int sock = socket(AF_INET, SOCK_DGRAM, 0);
    if (sock < 0) fprintf(stderr, "Failed to create socket: %s\n", strerror(errno));

    // Set timeout
    struct timeval timeout;
    timeout.tv_sec = 5;
    timeout.tv_usec = 0;
    setsockopt(sock, SOL_SOCKET, SO_RCVTIMEO, &timeout, sizeof(timeout));

    // Upstream resolver
        struct sockaddr_in *resolver_addr = malloc(sizeof(struct sockaddr_in));
        if (!resolver_addr) {
                fprintf(stderr, "Failed to allocate memory for resolver_addr\n");
                exit(EXIT_FAILURE);
        }
        memset(resolver_addr, 0, sizeof(struct sockaddr_in));
        resolver_addr->sin_family = AF_INET;
        resolver_addr->sin_port = htons(DNS_PORT);
        inet_pton(AF_INET, "8.8.8.8", &resolver_addr->sin_addr);

        /* END OF SOCKET CREATION */
```

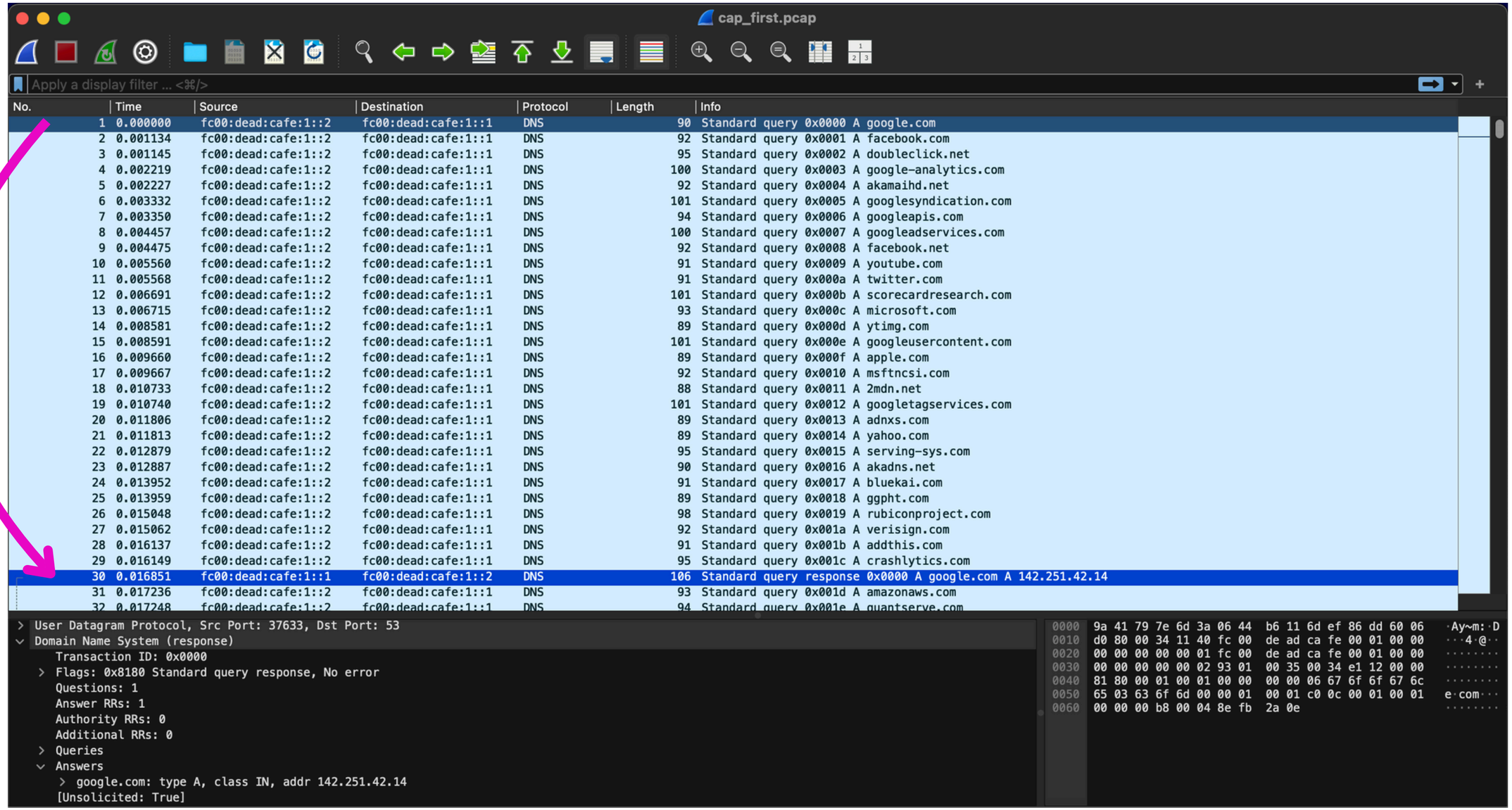
User Program (XSK - XDP Socket)

```
/*
    MAIN CRUX - CREATE A UDP SOCKET AND SEND A DNS QUERY
    RECEIVE THE RESPONSE BACK AND LOG INTO CACHE
    */

// Send query
if (sendto(sock, dns_payload, dns_payload_len, 0, (struct sockaddr *)resolver_addr, sizeof(*resolver_addr)) < 0) {
    fprintf(stderr, "Failed to send query: %s\n", strerror(errno));
    free(result);
    json_decref(cache);
    return NULL;
}

// Receive response
uint8_t response[MAX_DNS_PACKET];
ssize_t response_len = recvfrom(sock, response, MAX_DNS_PACKET, 0, NULL, NULL);
if (response_len < 0) {
    fprintf(stderr, "Failed to receive response: %s\n", strerror(errno));
    close(sock);
    free(result);
    json_decref(cache);
    return NULL;
}

// Copy response to result
memcpy(result->payload, response, response_len);
result->len = response_len;
```

User Program (XSK - XDP Socket)

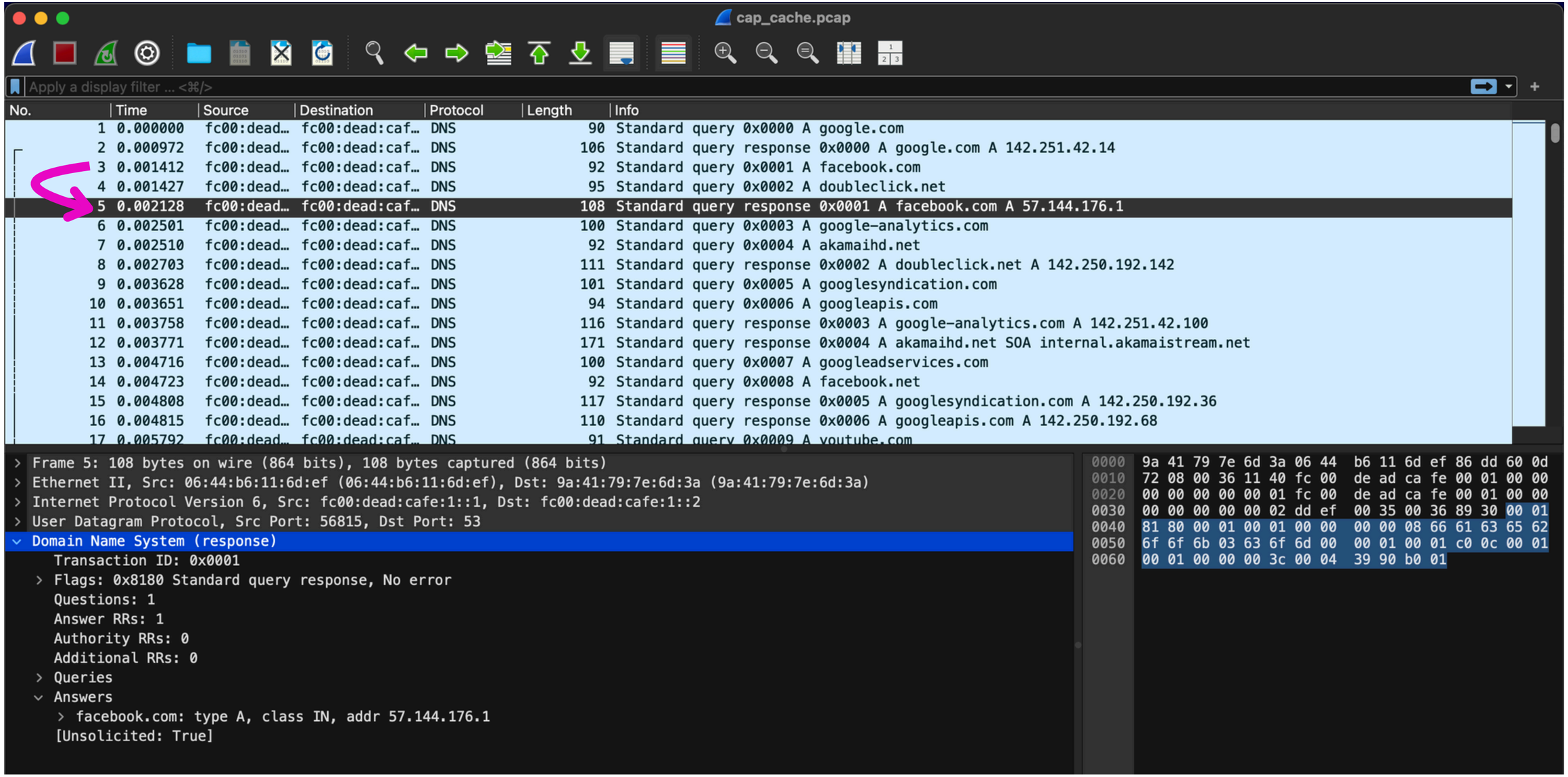
```c
static void parse_dns_query_name(uint8_t *dns_payload, char *hostname)
{
    int pos = 0, hostname_pos = 0;
    uint8_t len;

    while ((len = dns_payload[pos]) != 0)
    {
        if (pos + len + 1 > 255)
            break; // Prevent buffer overflow
        if (hostname_pos > 0)
            hostname[hostname_pos++] = '.';
        memcpy(hostname + hostname_pos, dns_payload + pos + 1, len);
        hostname_pos += len;
        pos += len + 1;
    }
    hostname[hostname_pos] = '\0';
}
```

Metrics: First Hits (dnsperf)



Metrics: Cache Hits (dnsperf)

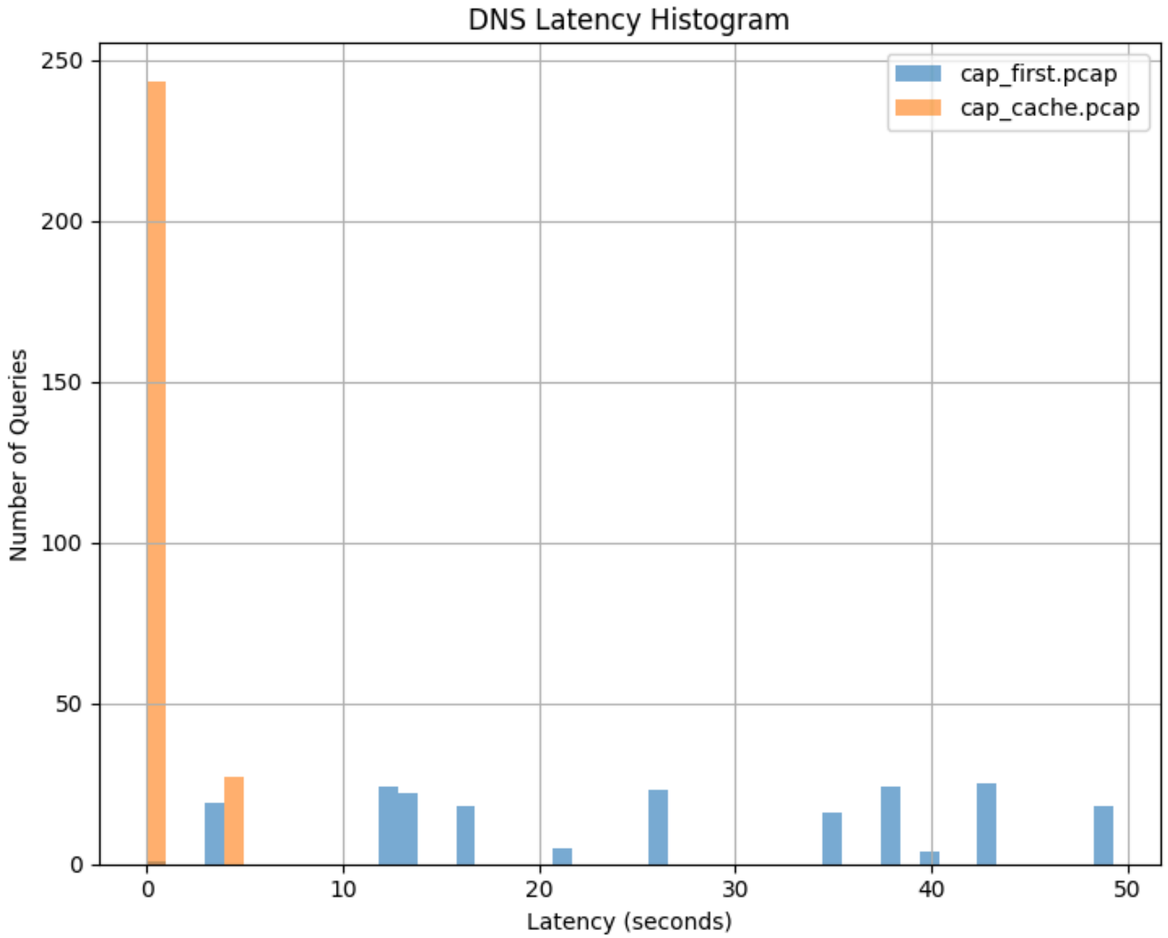
x10 Speedup!

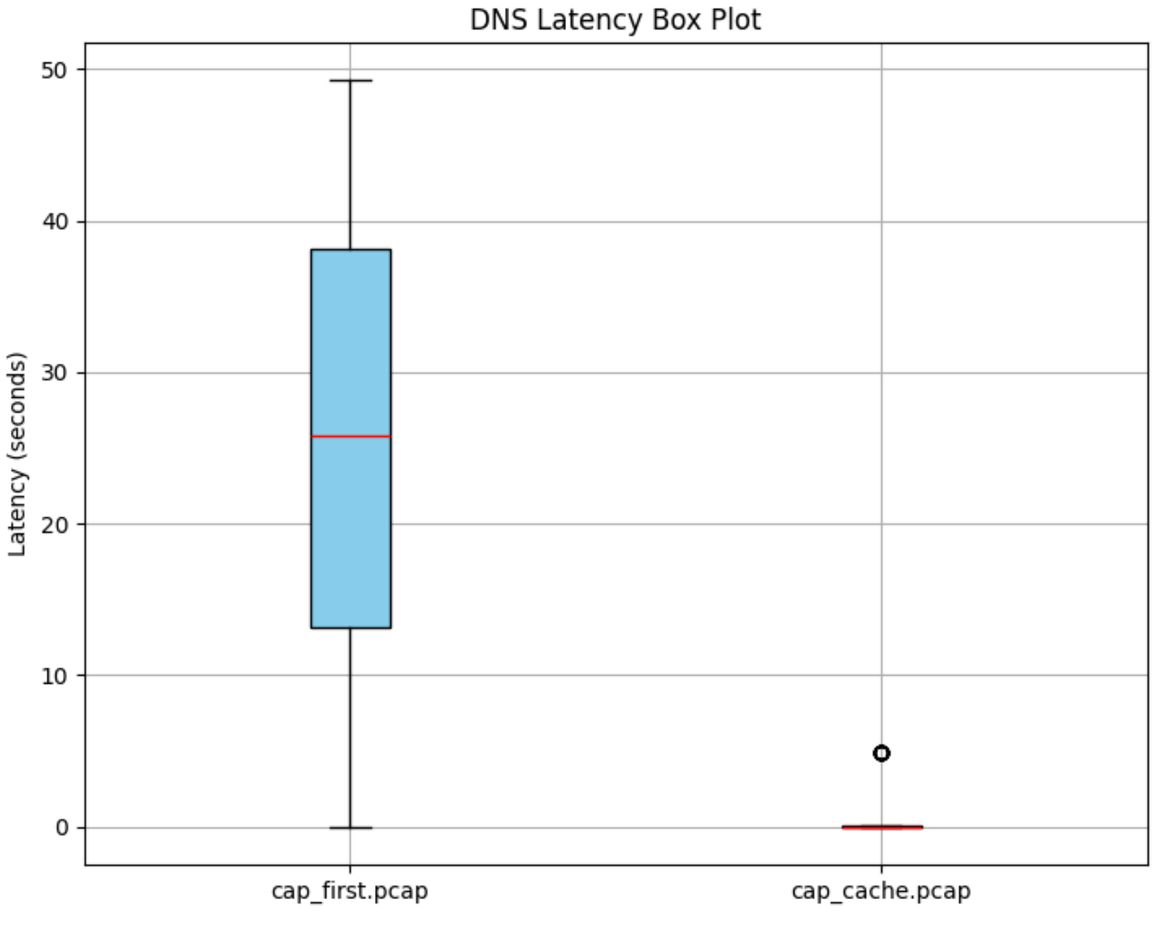


Comparison: First Hits vs. Cache Hits

```
cap_first.pcap:
  Average Latency = 26.322772 seconds
  Median Latency  = 25.766722 seconds

cap_cache.pcap:
  Average Latency = 0.512208 seconds
  Median Latency  = 0.022440 seconds
```

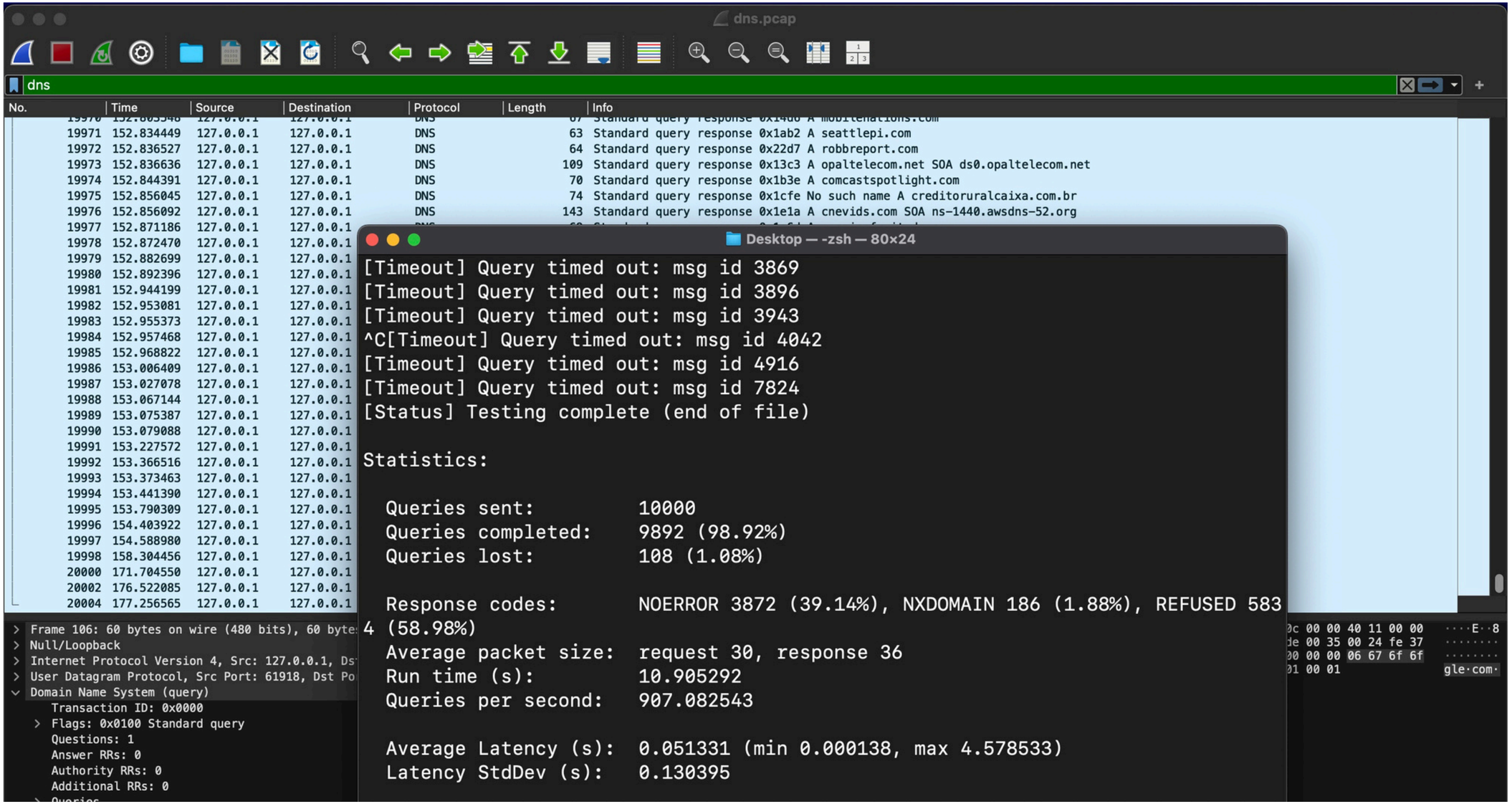
Comparison: First Hits vs. Cache Hits





Direct to DNS Resolver

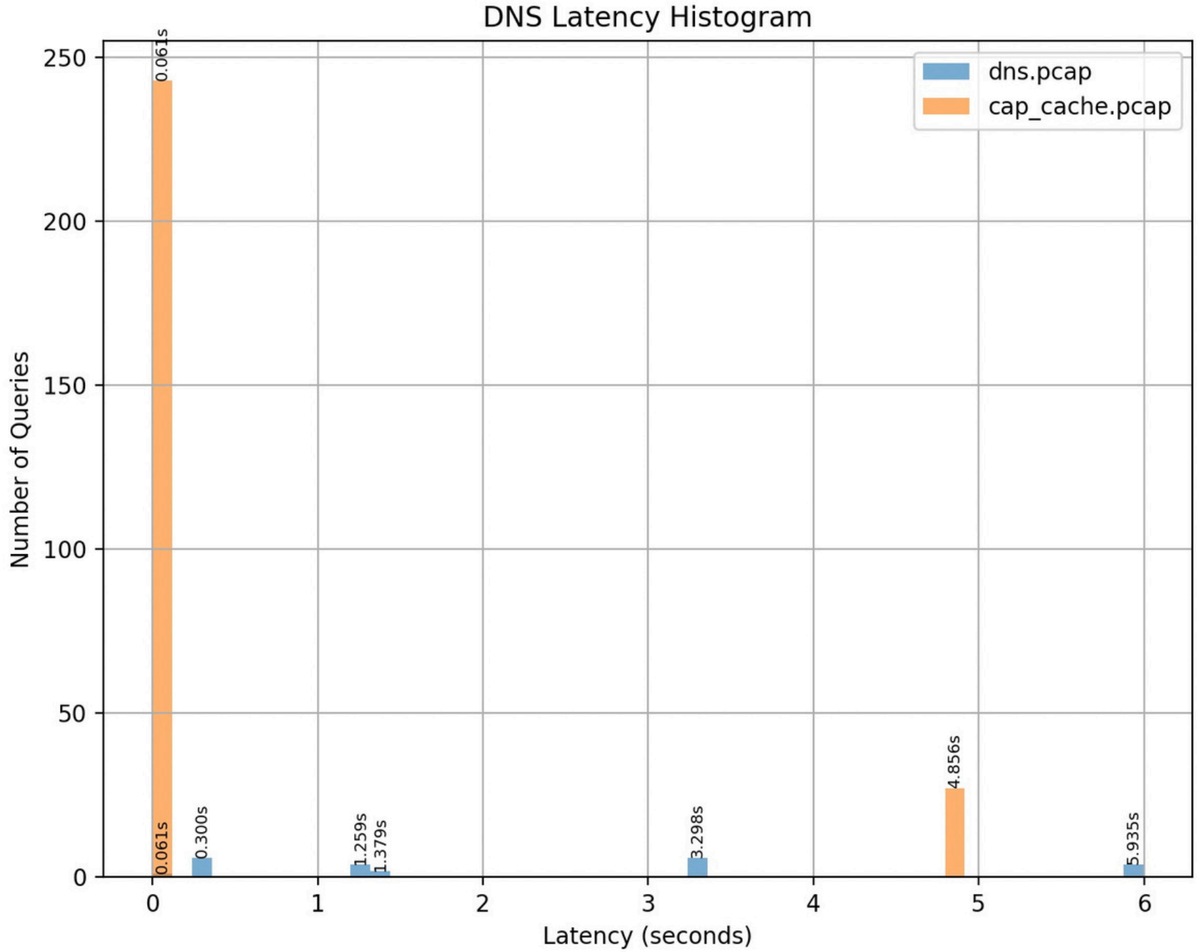
Running on loopback using dnsmasq

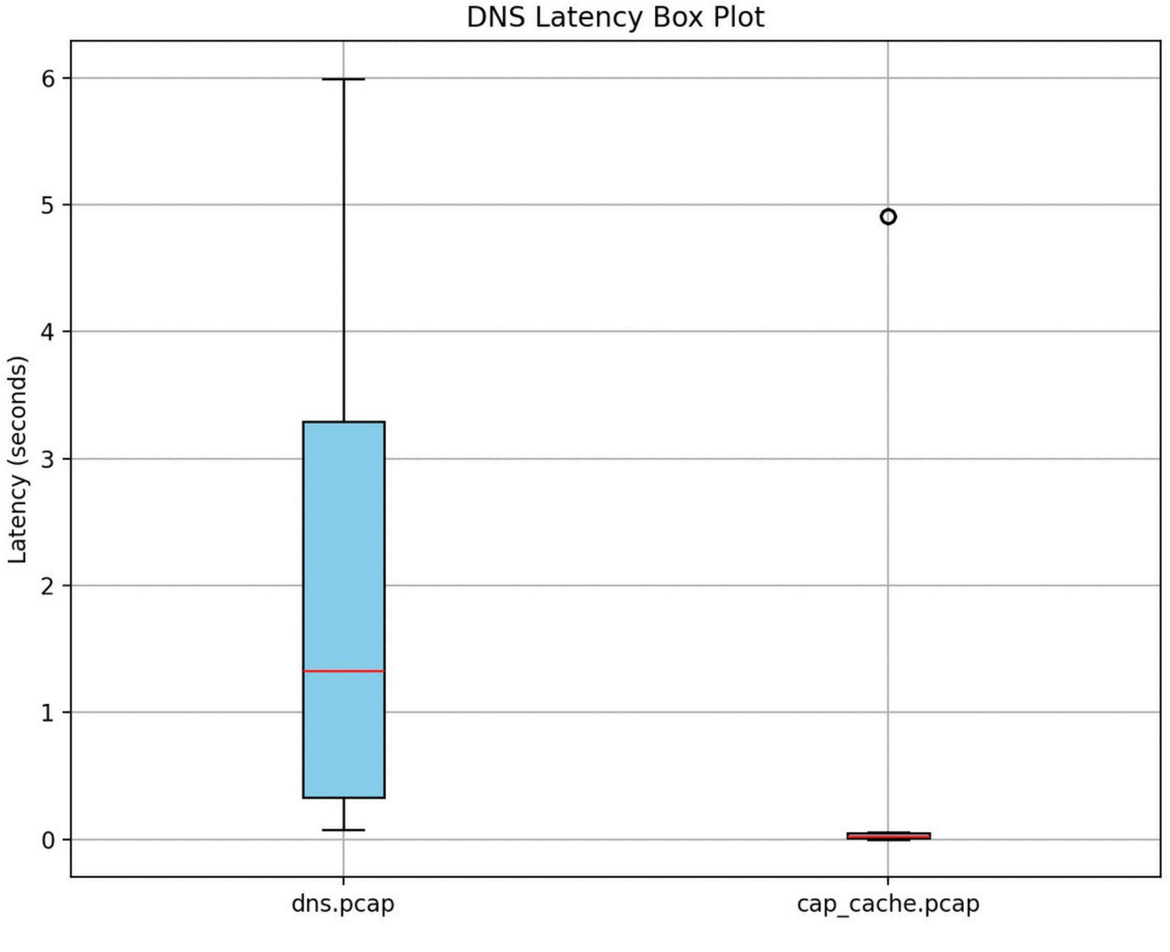


Comparison: Direct DNS vs. Cache Hits

```
dns.pcap:
   Average Latency = 2.305389 seconds
   Median Latency  = 1.328159 seconds

cap_cache.pcap:
   Average Latency = 0.512208 seconds
   Median Latency  = 0.022440 seconds
```

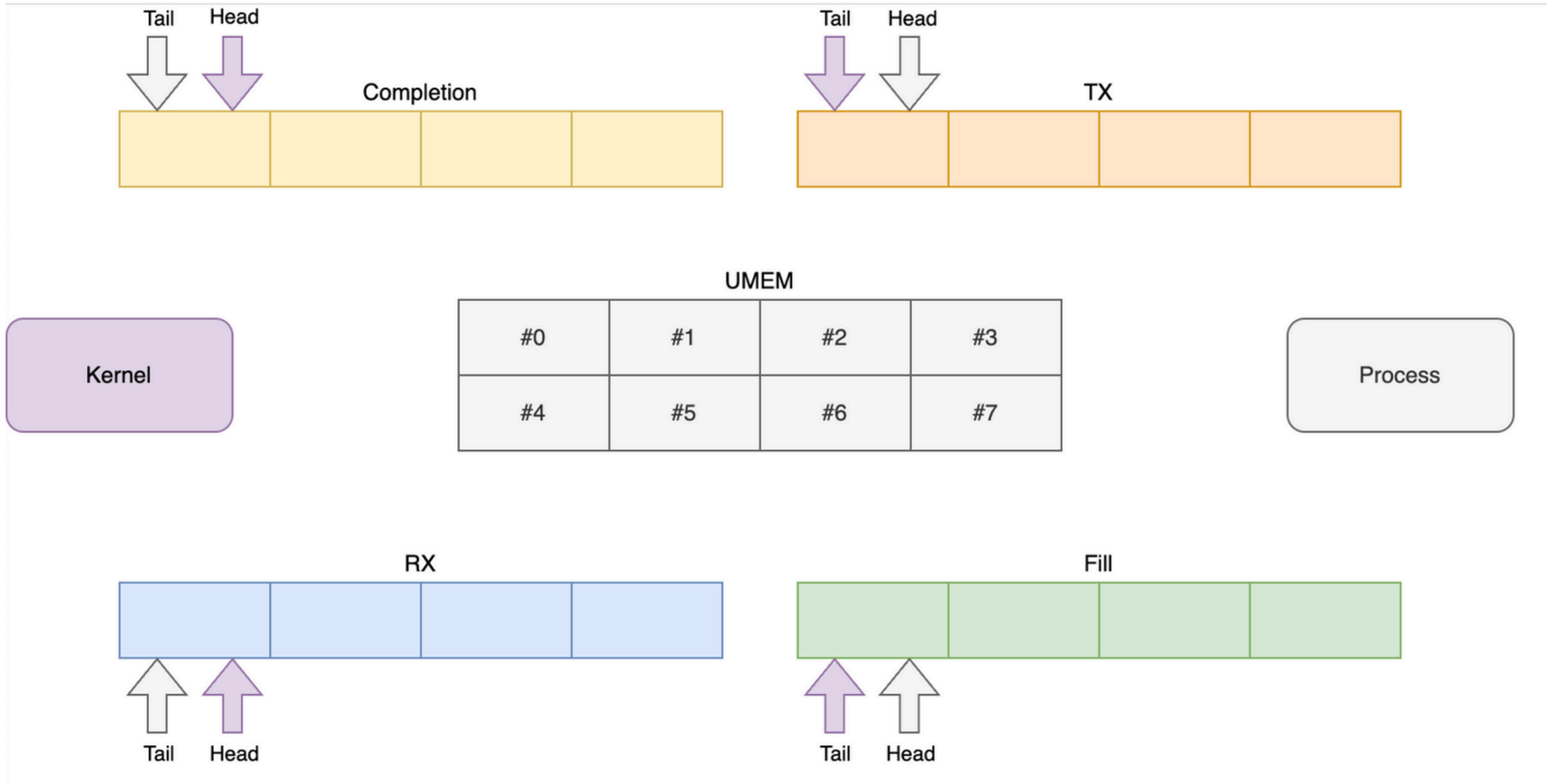
Comparison: Direct DNS vs. Cache Hits



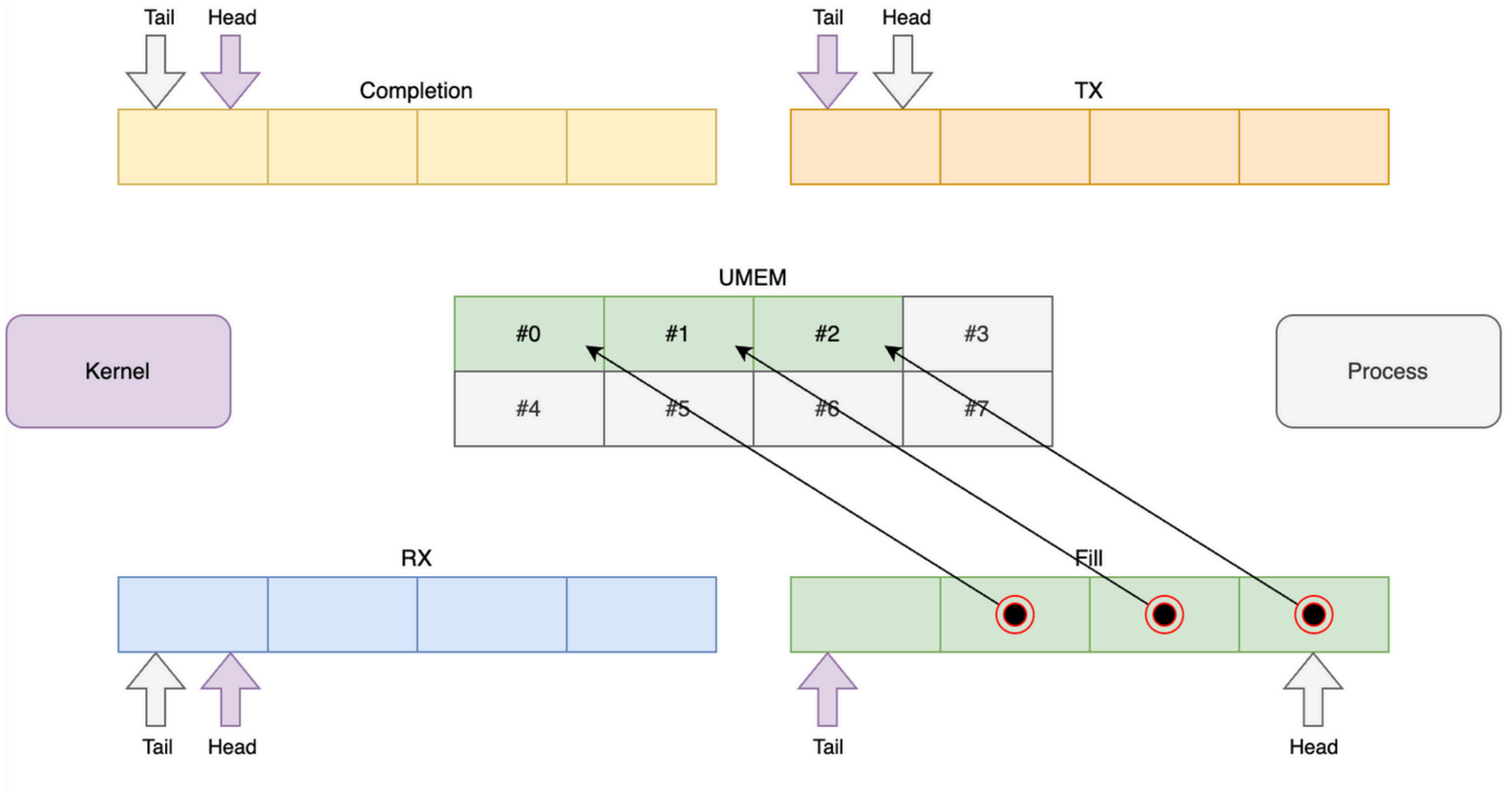


Appendix
XDP Socket Working

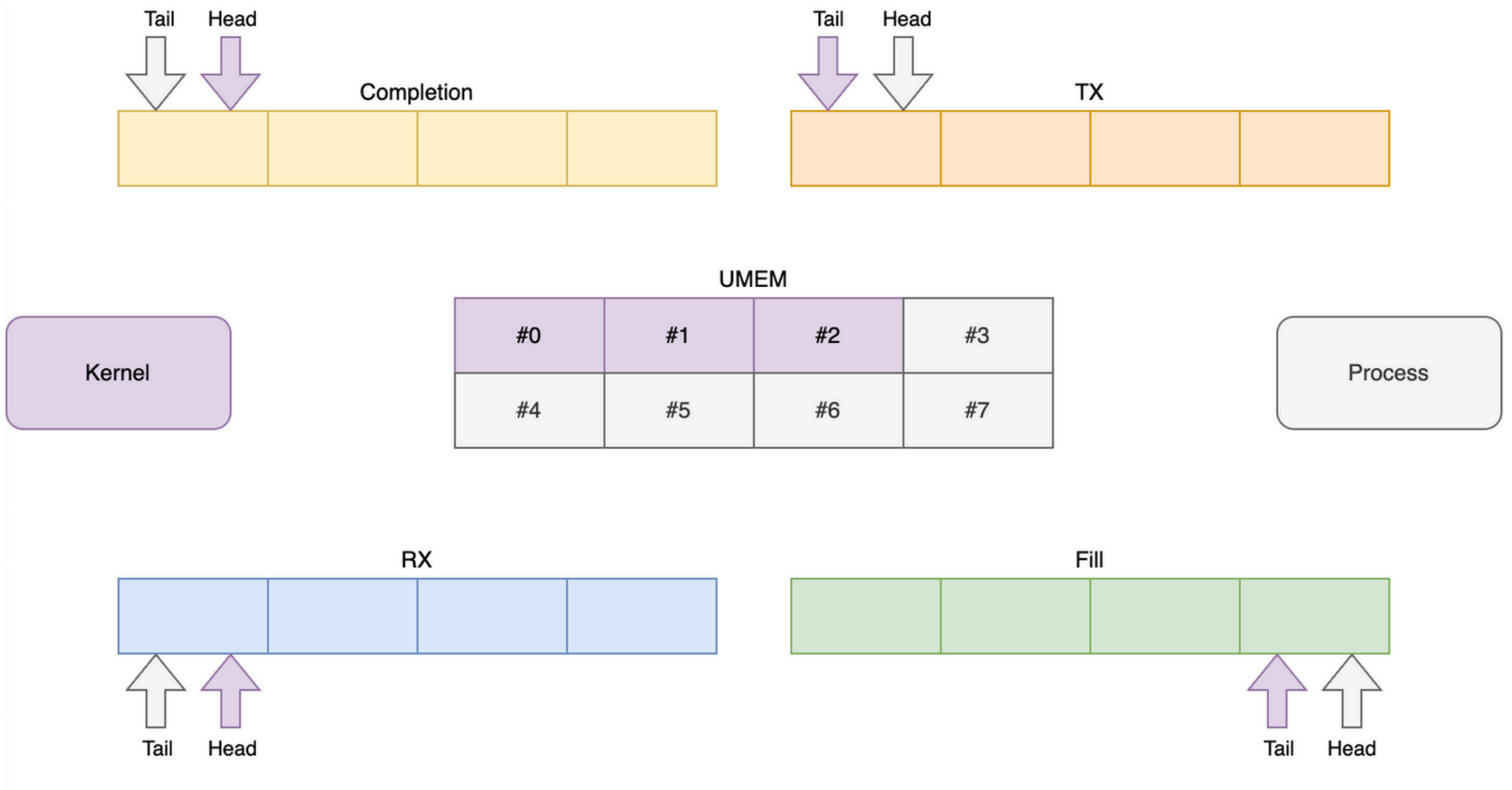
XSK - XDP Socket



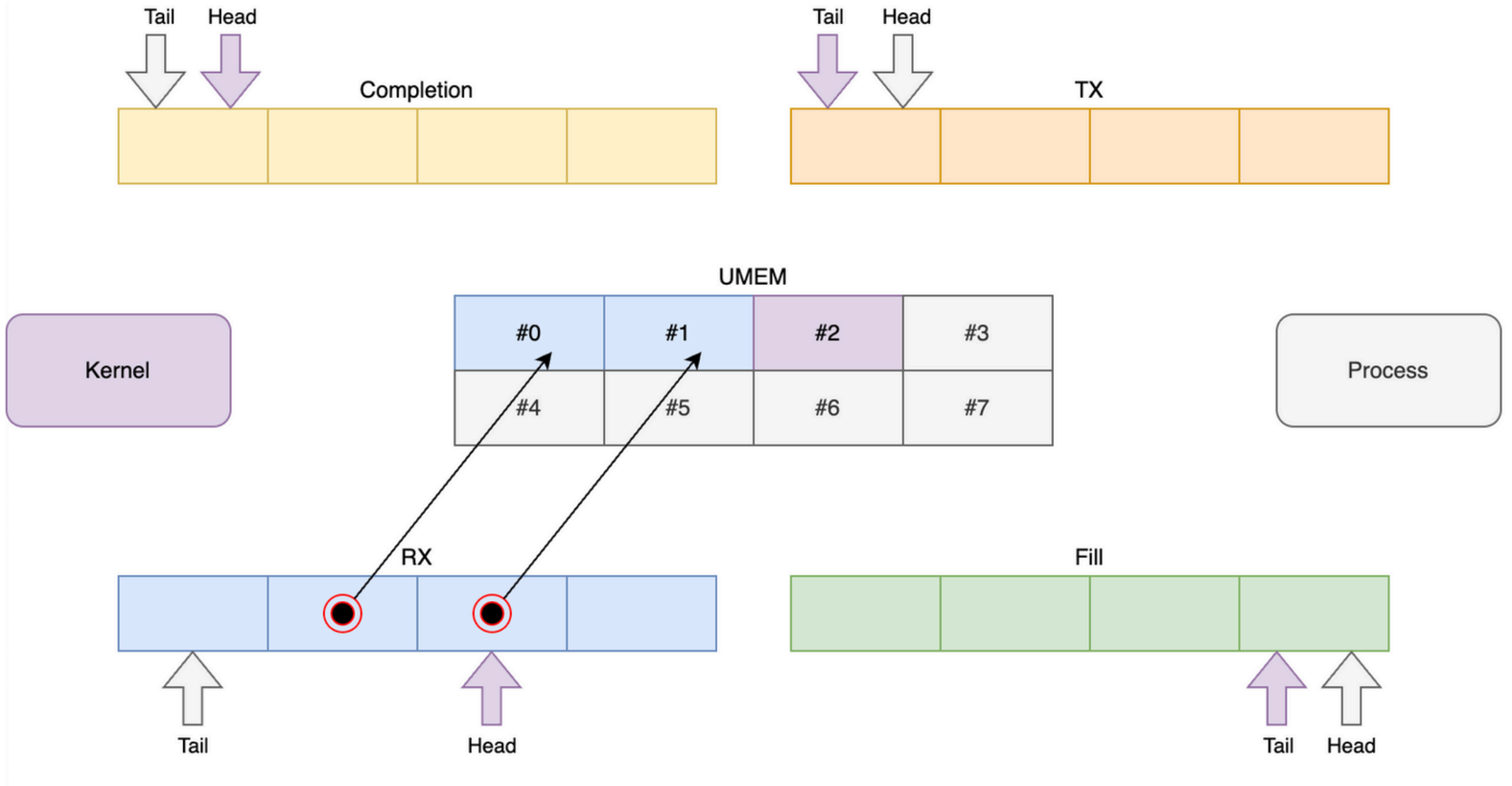
XSK - XDP Socket



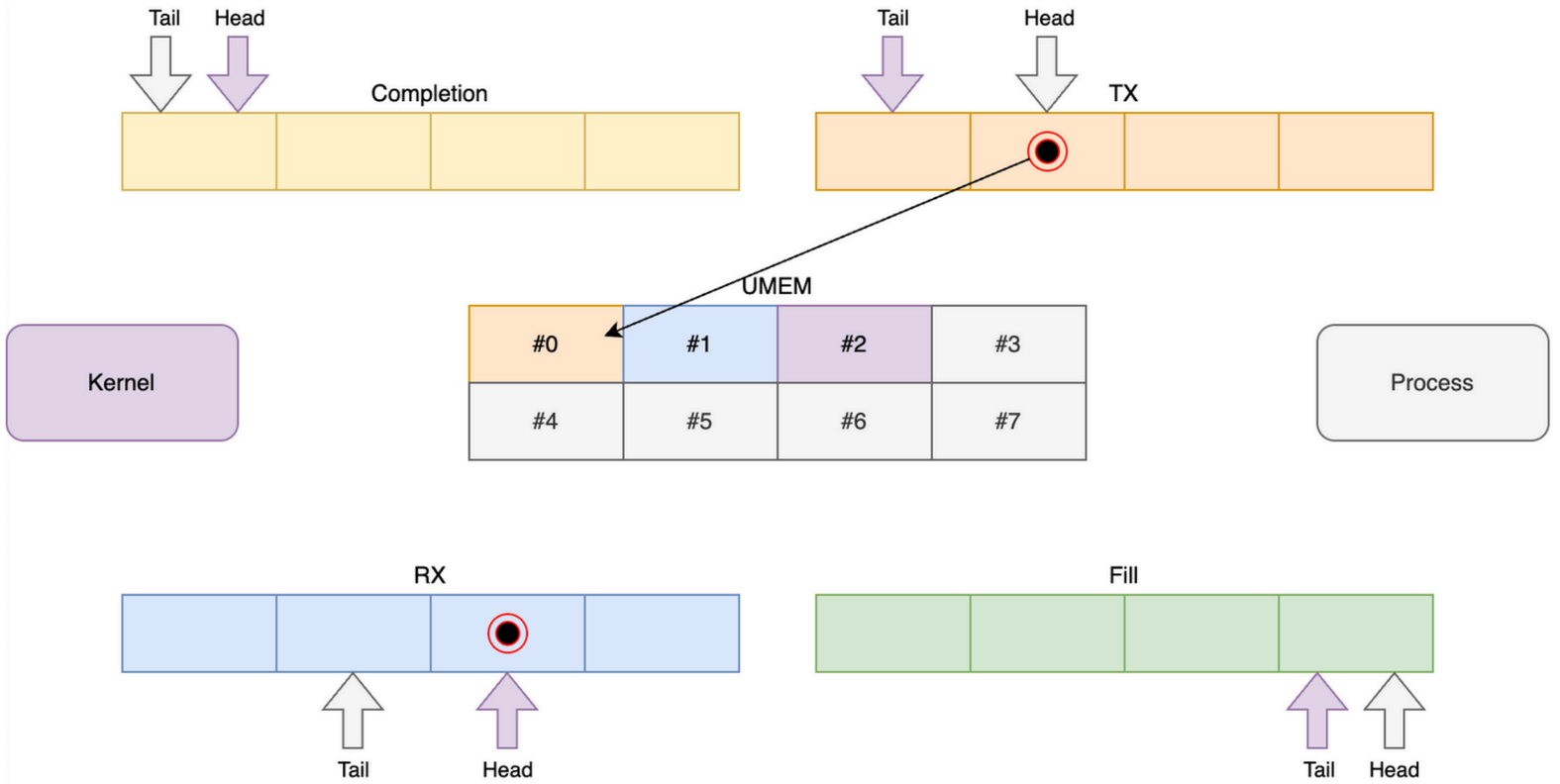
XSK - XDP Socket



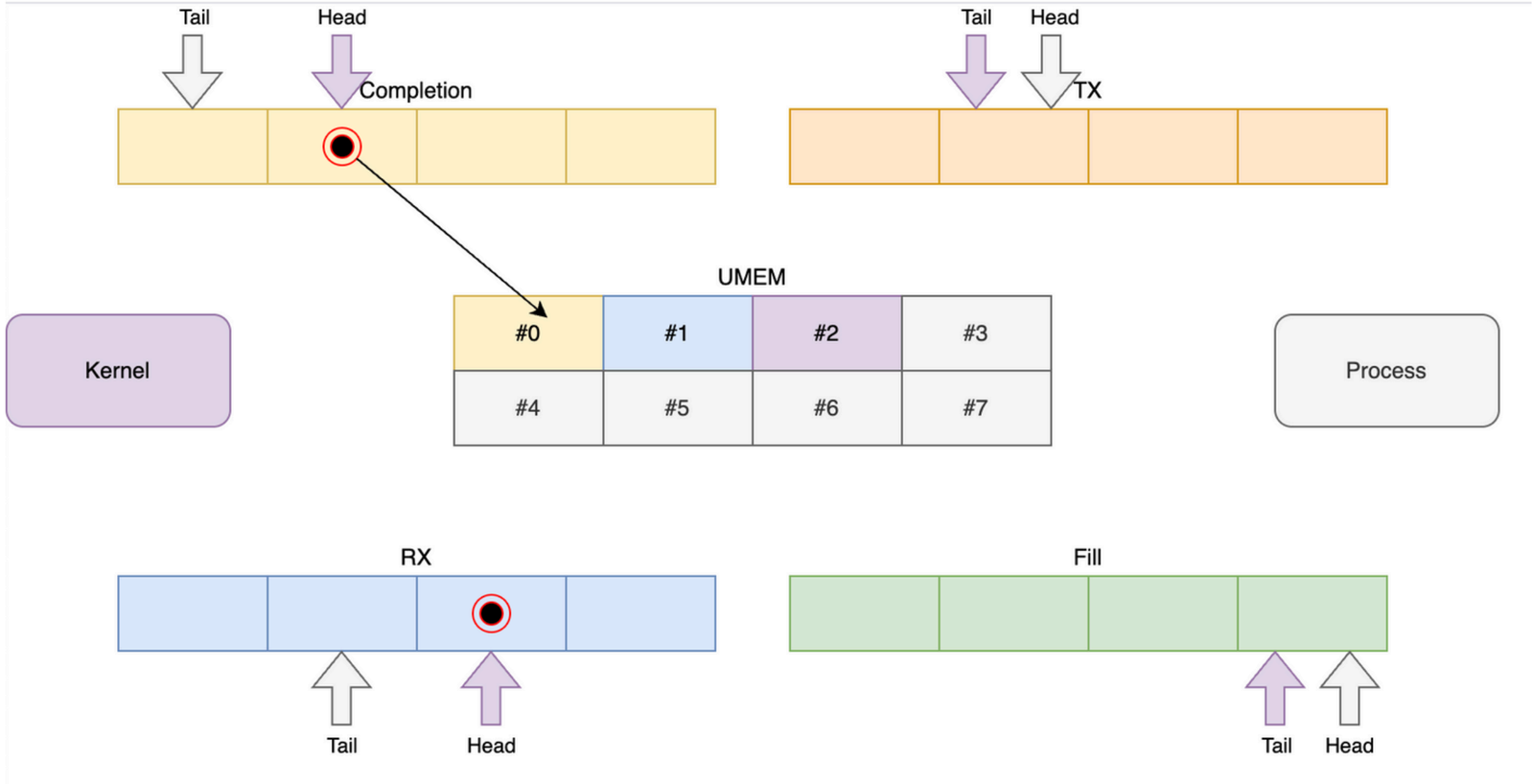
XSK - XDP Socket



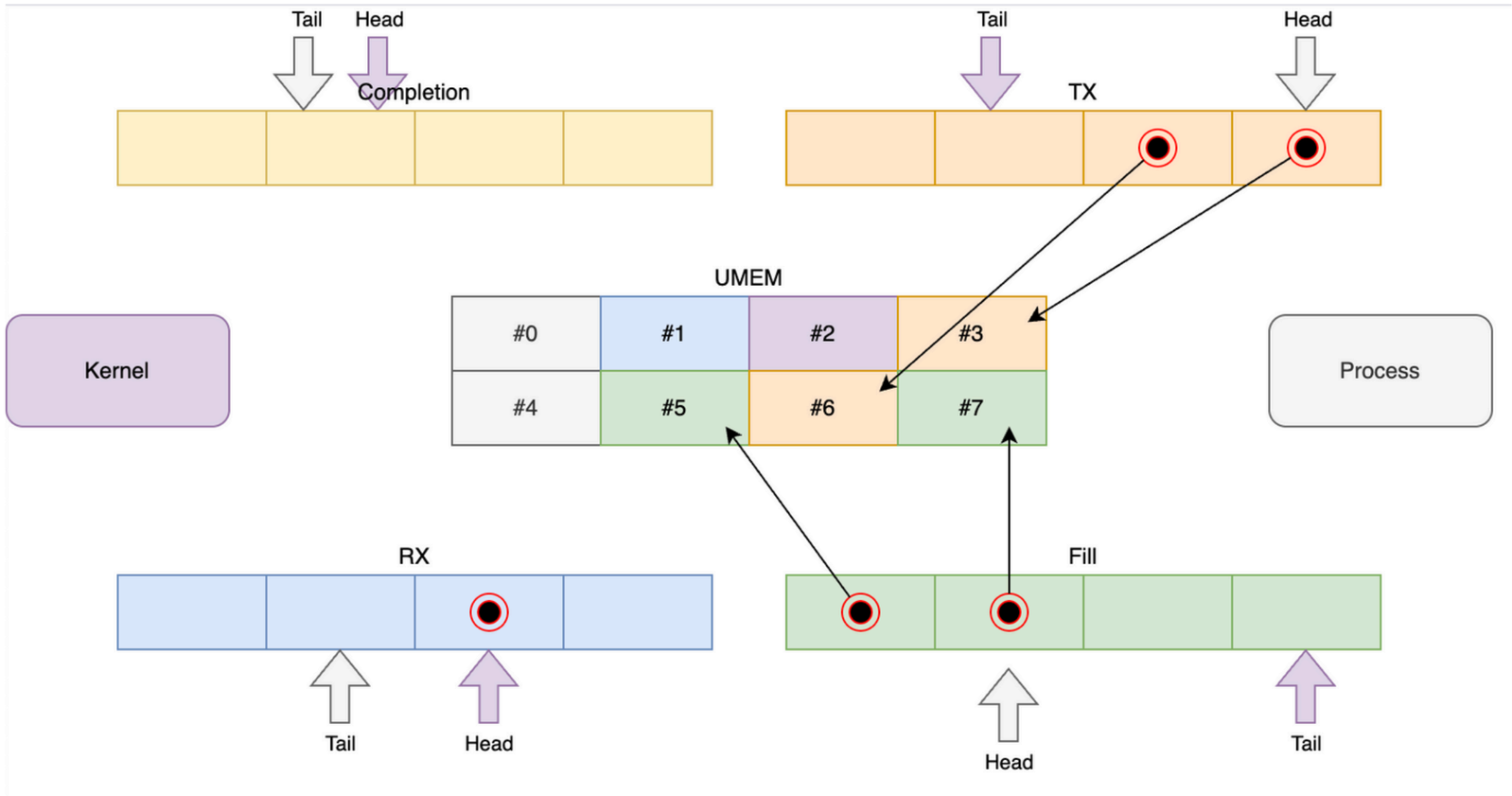
XSK - XDP Socket



XSK - XDP Socket



XSK - XDP Socket



THANK YOU

TEAM - 13

Guntas Singh Saran (guntassingh.saran@iitgn.ac.in)

Hrriday V. Ruparel (hrriday.ruparel@iitgn.ac.in)

Kishan Ved (kishan.ved@iitgn.ac.in)

Pravav Patil (pranav.patil@iitgn.ac.in)

Indian Institute of Technology Gandhinagar
Palaj, Gujarat - 382355