



Project 15: DNS Resolver using AF_XDP

CS331 - Computer Networks

TEAM - 13

Guntas Singh Saran (22110089)

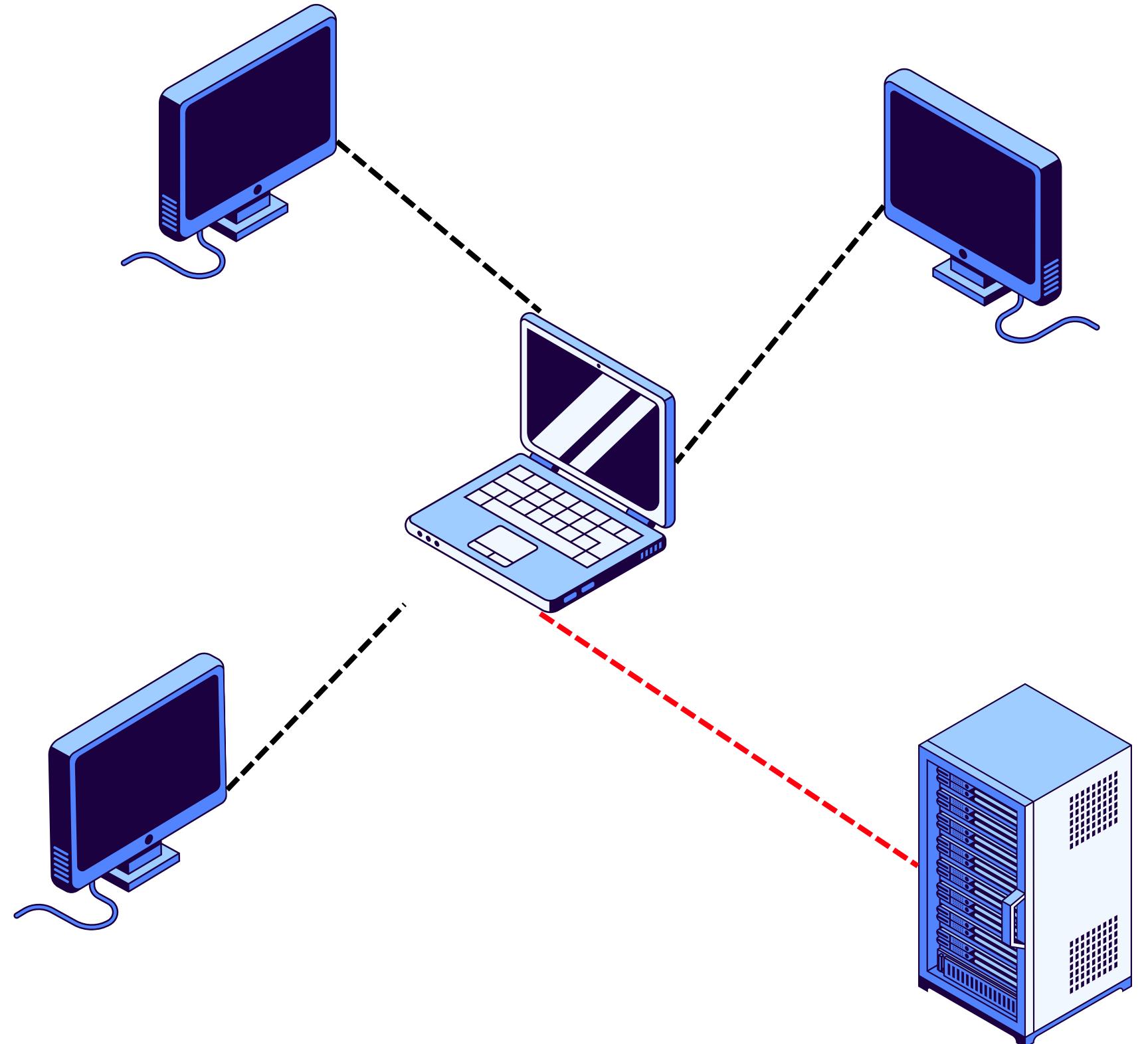
Hrriday V. Ruparel (22110099)

Kishan Ved (22110122)

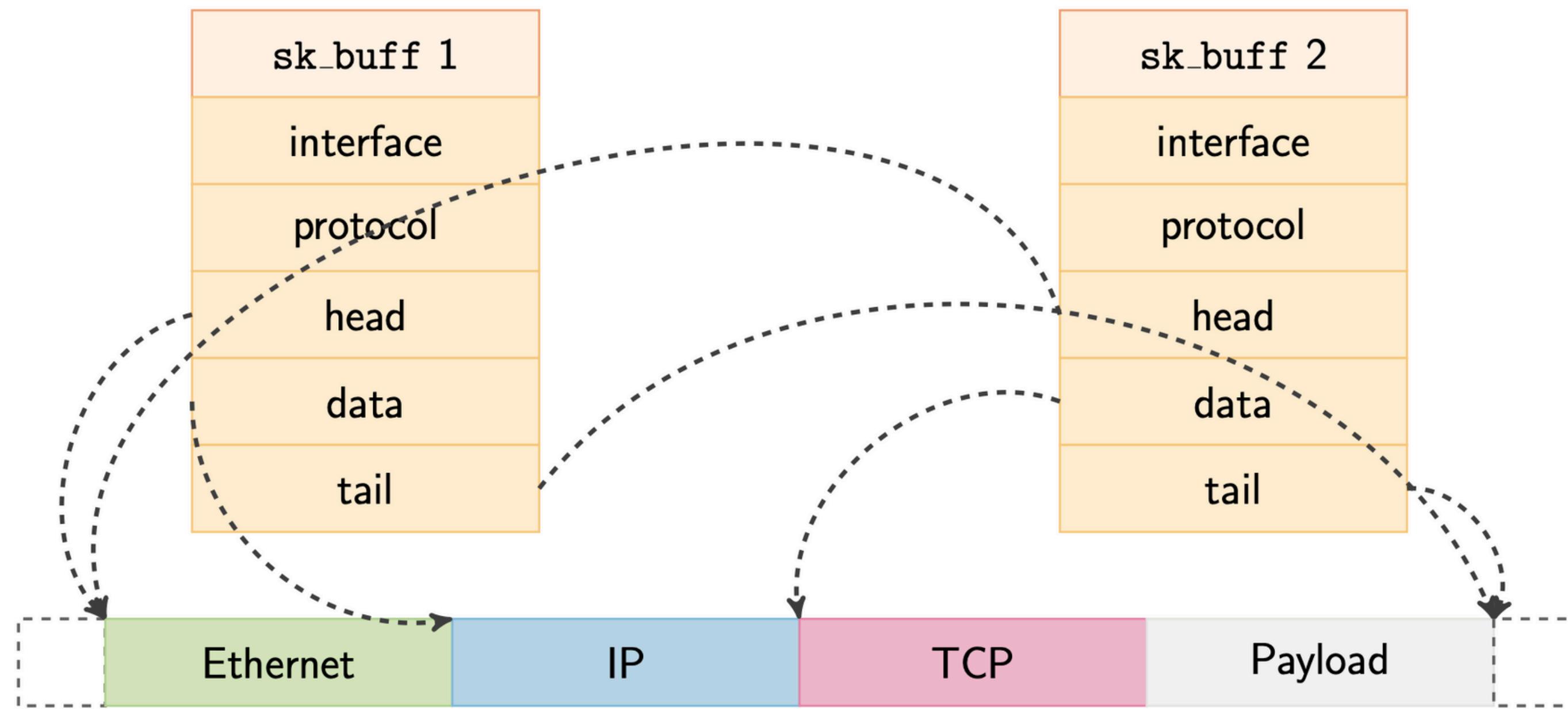
Pranav Patil (22110199)

Indian Institute of Technology Gandhinagar
Palaj, Gujarat - 382355

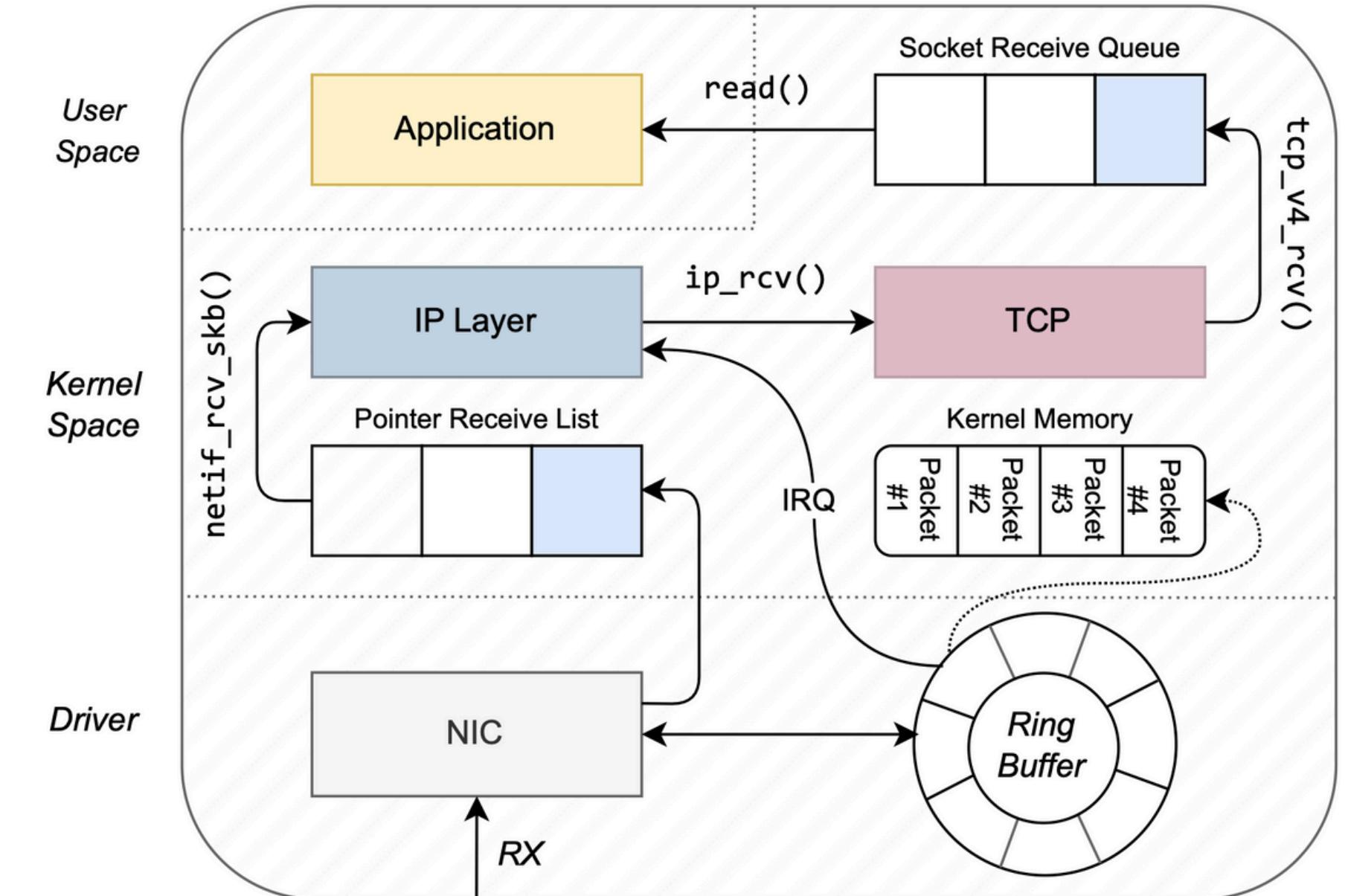
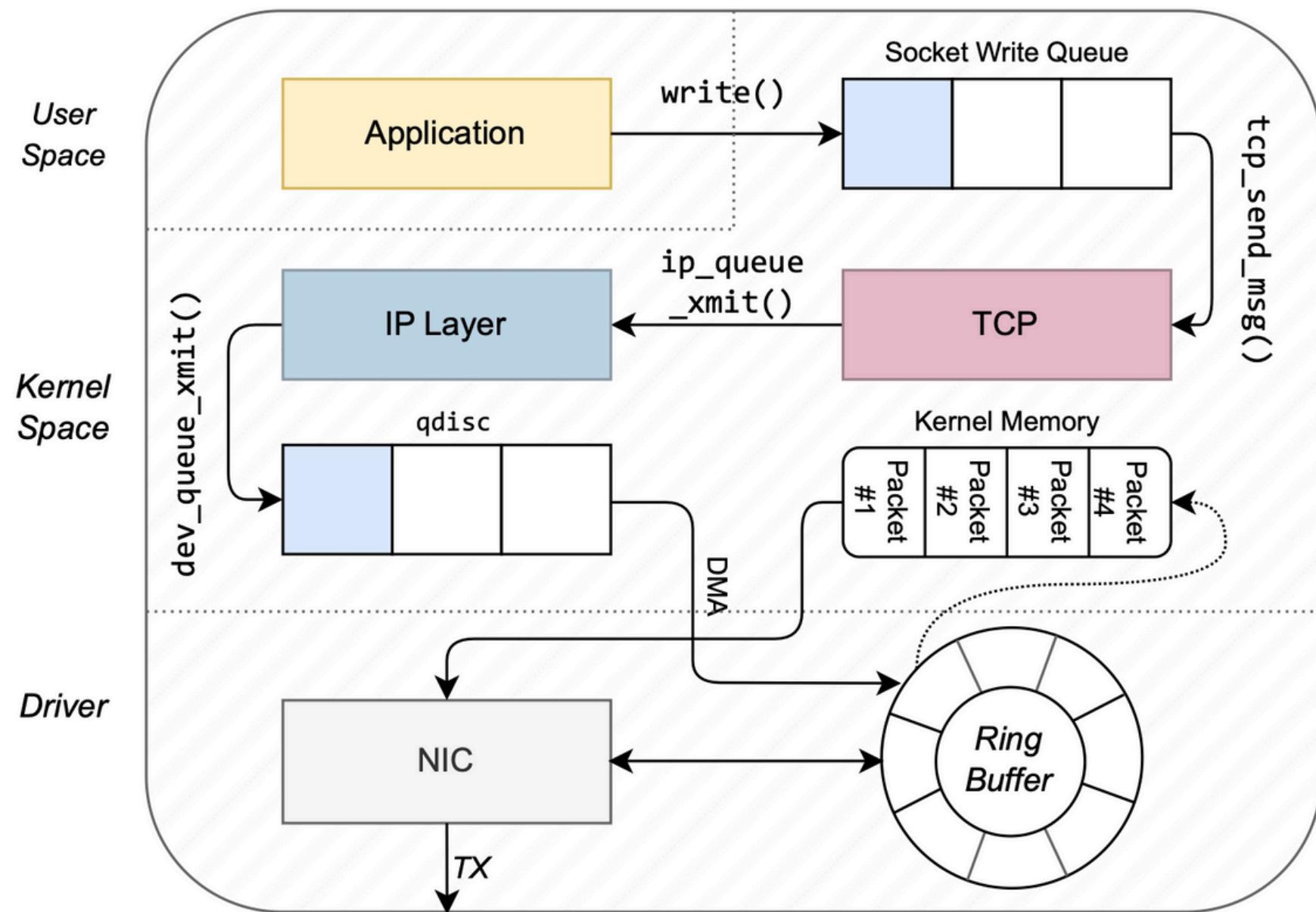
Source: https://github.com/Kishan-Ved/dns-af_xdp



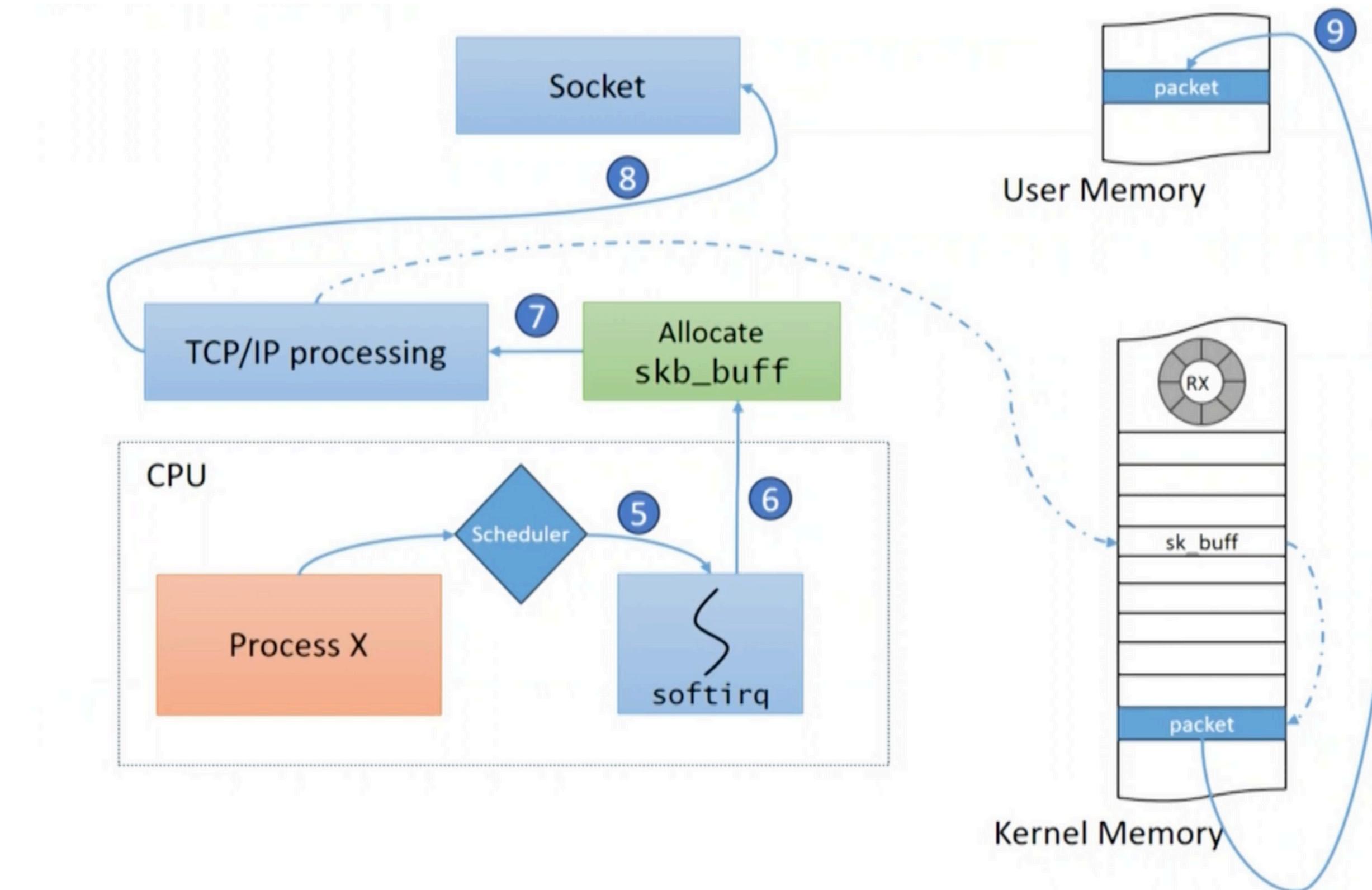
sk_buff in the Kernel

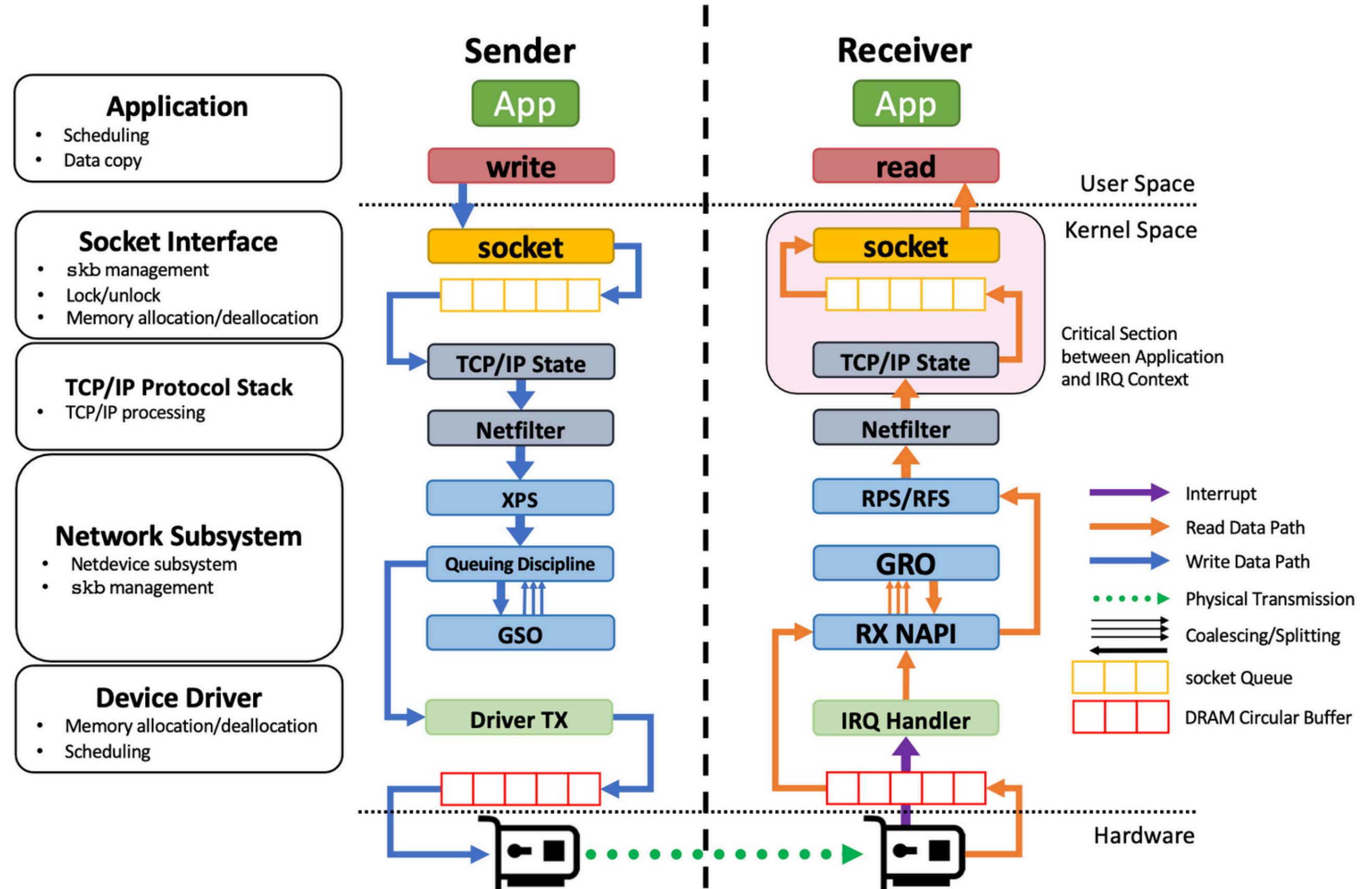


Egress and Ingress Paths through Linux Kernel Network Stack

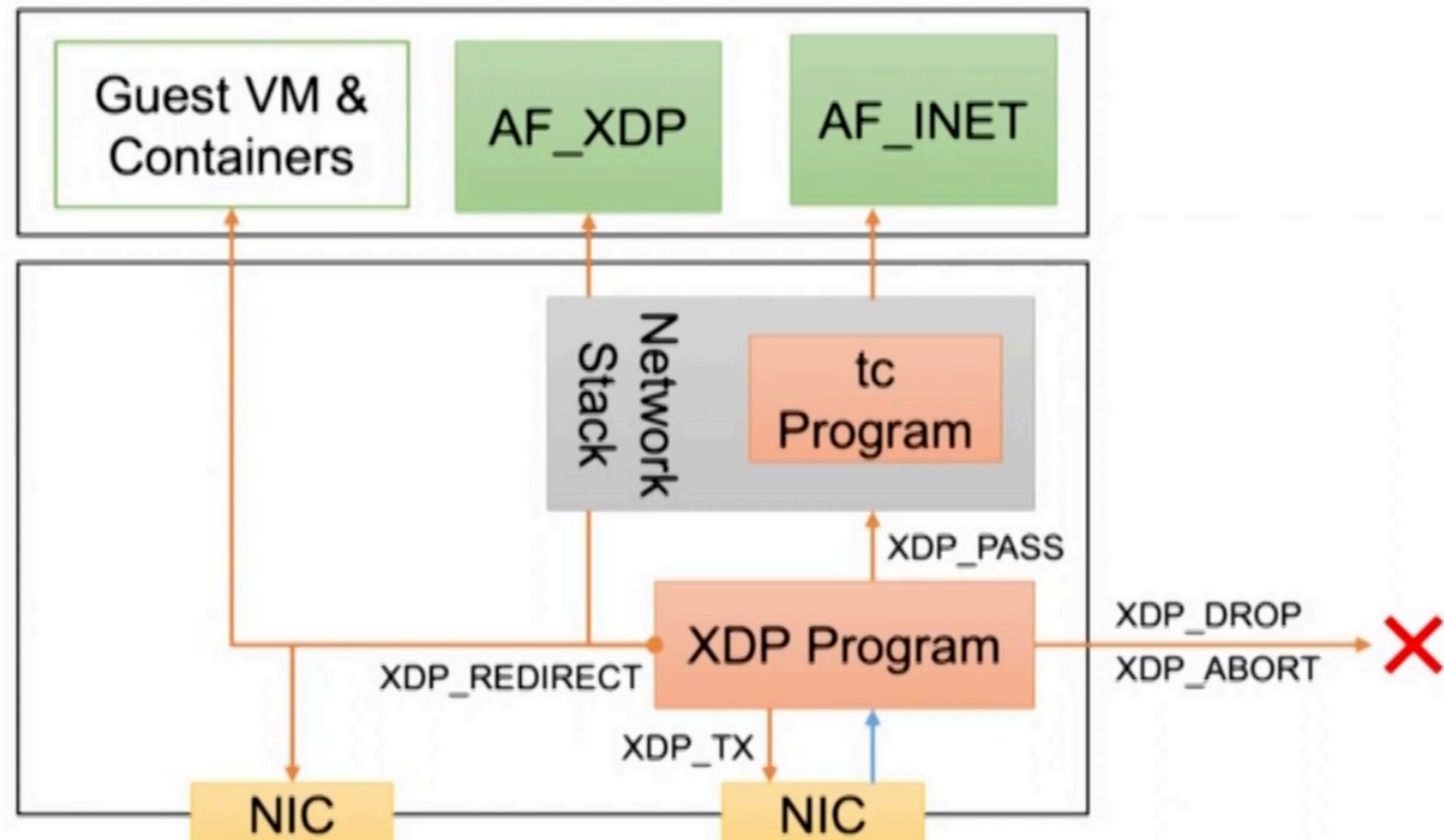


Path of the packet in Kernel Network Stack





BPF Program @ XDP Hook

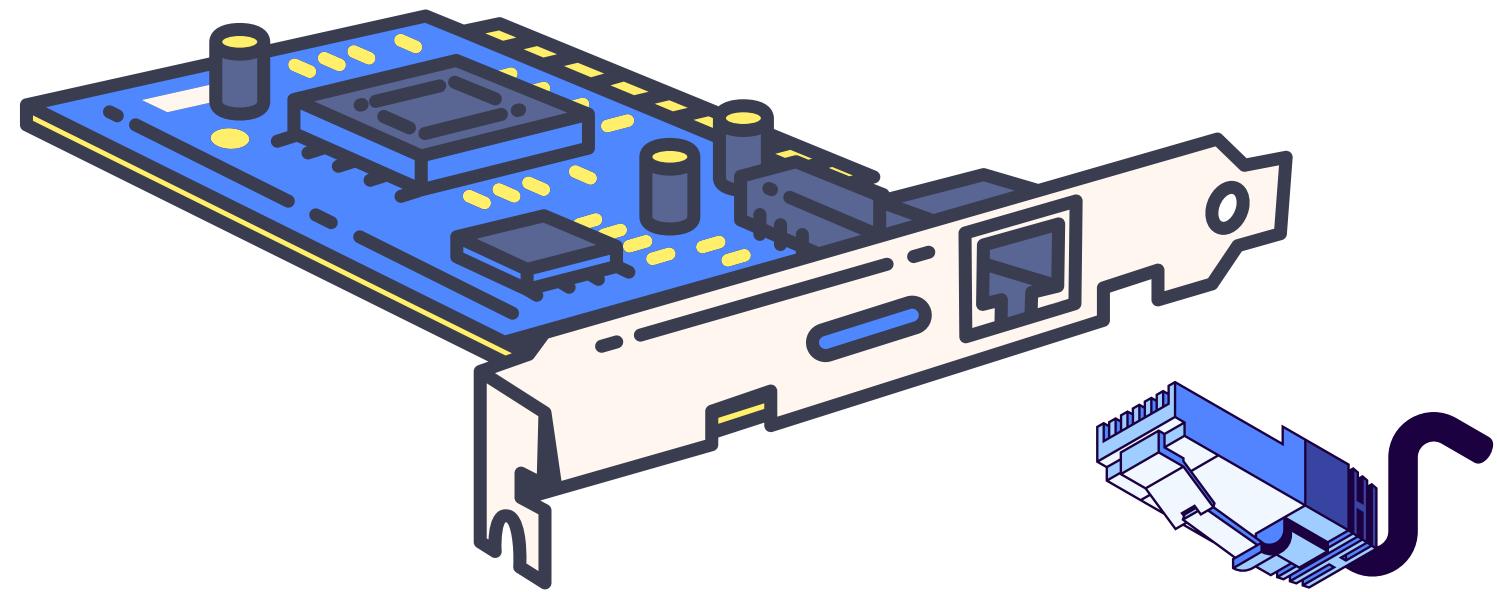


BPF Program @ XDP Hook

```
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");

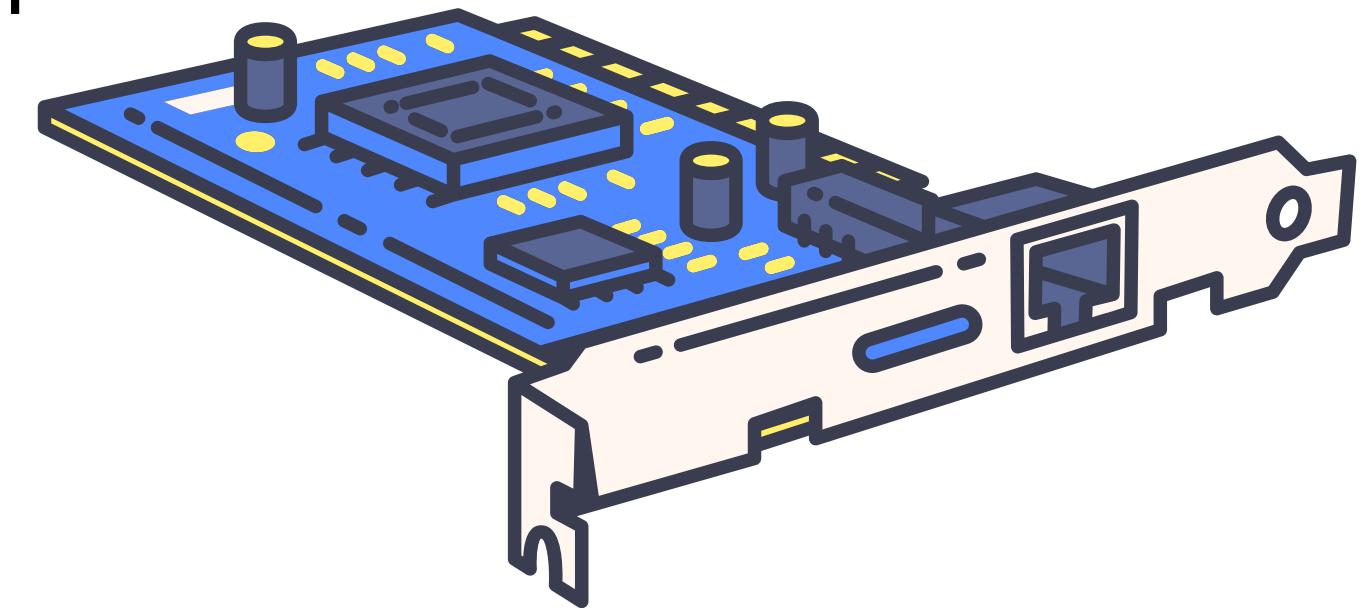
struct {
    __uint(type, BPF_MAP_TYPE_PERCPU_ARRAY);
    __type(key, __u32);
    __type(value, __u32);
    __uint(max_entries, 64);
} xdp_stats_map SEC(".maps");

/* 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

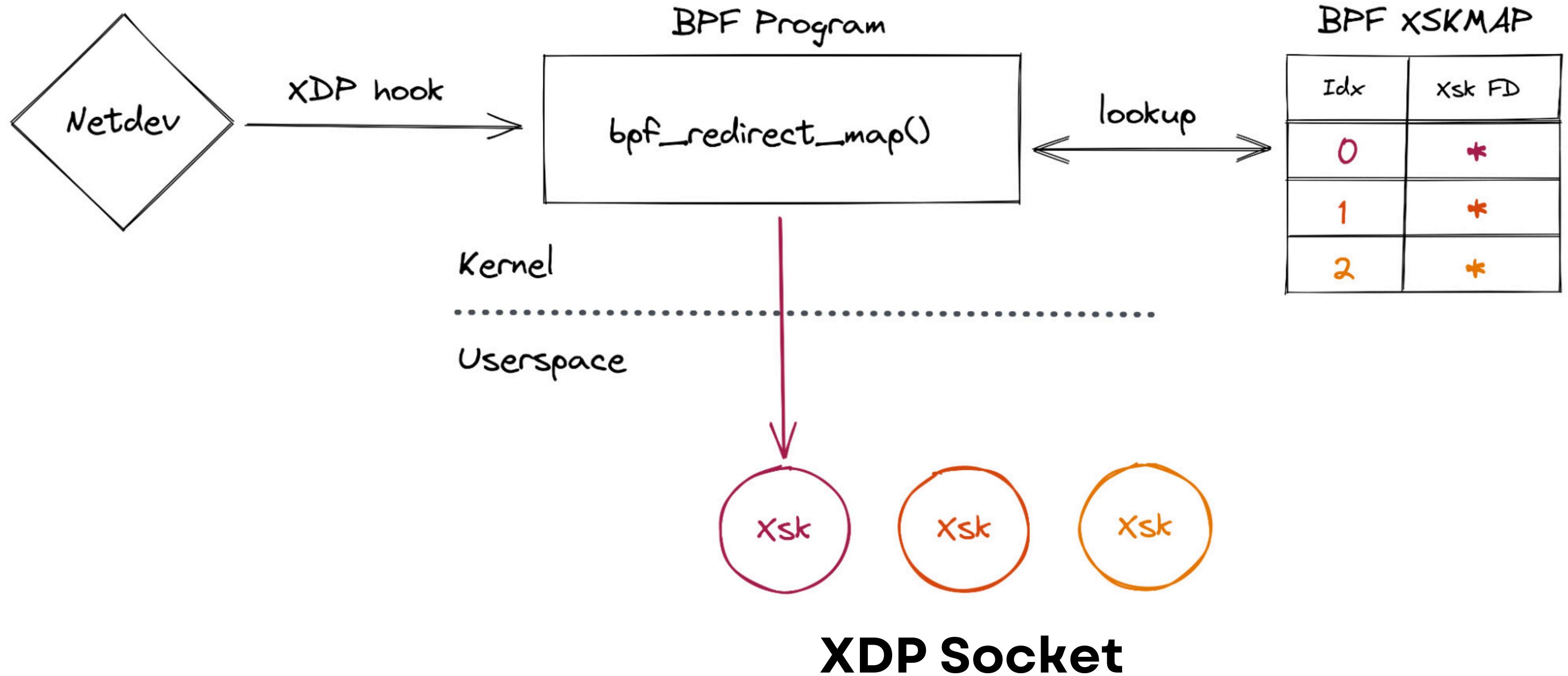
```
/* Parse the Ethernet and IP headers */
eth_type = parse_ethhdr(&nh, data_end, &eth);
    if (eth_type == bpf_htons(ETH_P_IP)) {
        ip_type = parse_iphdr(&nh, data_end, &iphdr);
        if (ip_type != IPPROTO_UDP)
            return XDP_PASS;
    } else if (eth_type == bpf_htons(ETH_P_IPV6))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

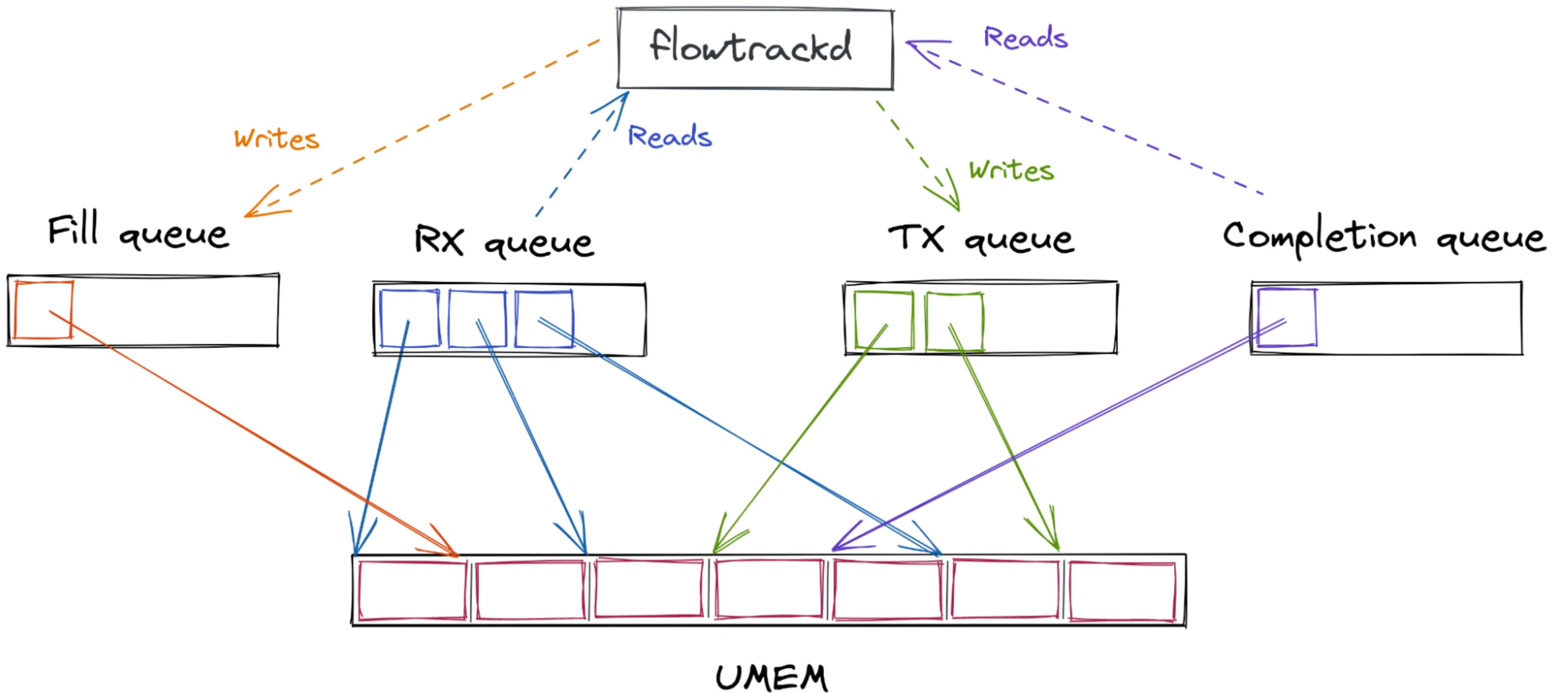
/* Check if the packet is a DNS query */
if (ip_type == IPPROTO_UDP) {
    udp_len = parse_udphdr(&nh, data_end, &udphdr);
    if (udp_len < 0)
        return XDP_PASS;

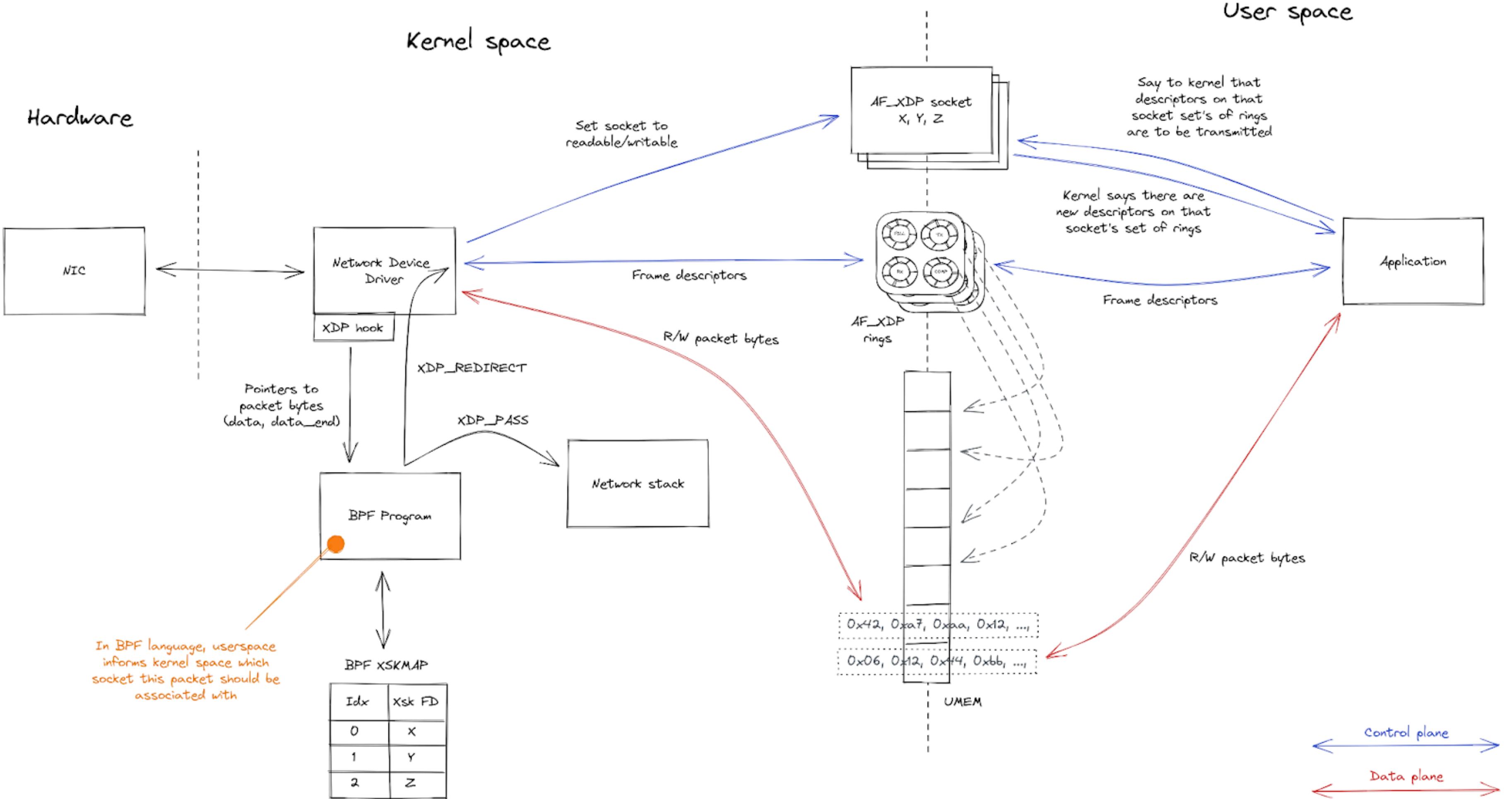
    // Only process DNS queries: UDP dest port must be 53
    if (udphdr->dest != bpf_htons(53))
        return XDP_PASS;

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

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



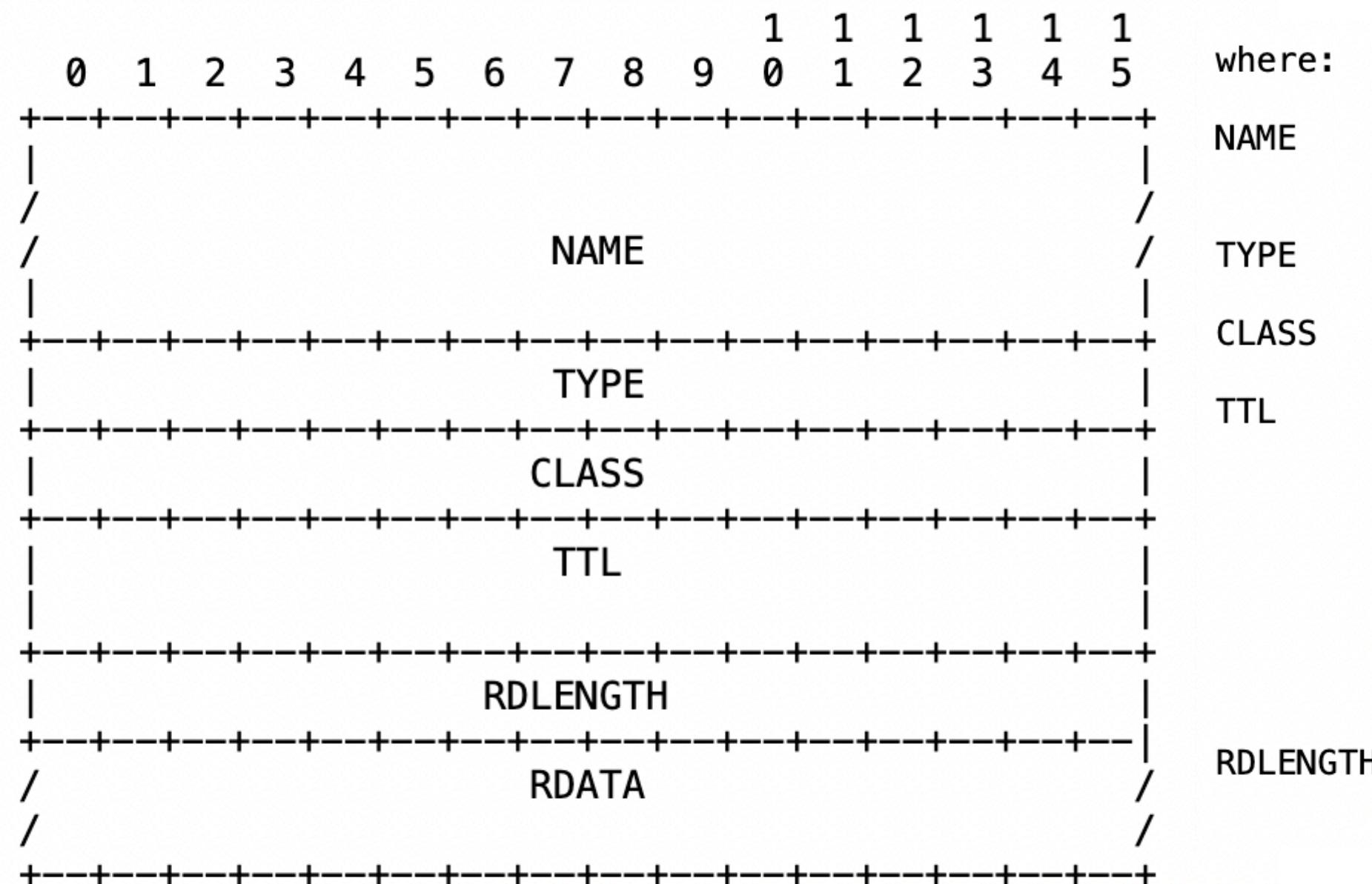




RFC 1035!

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.

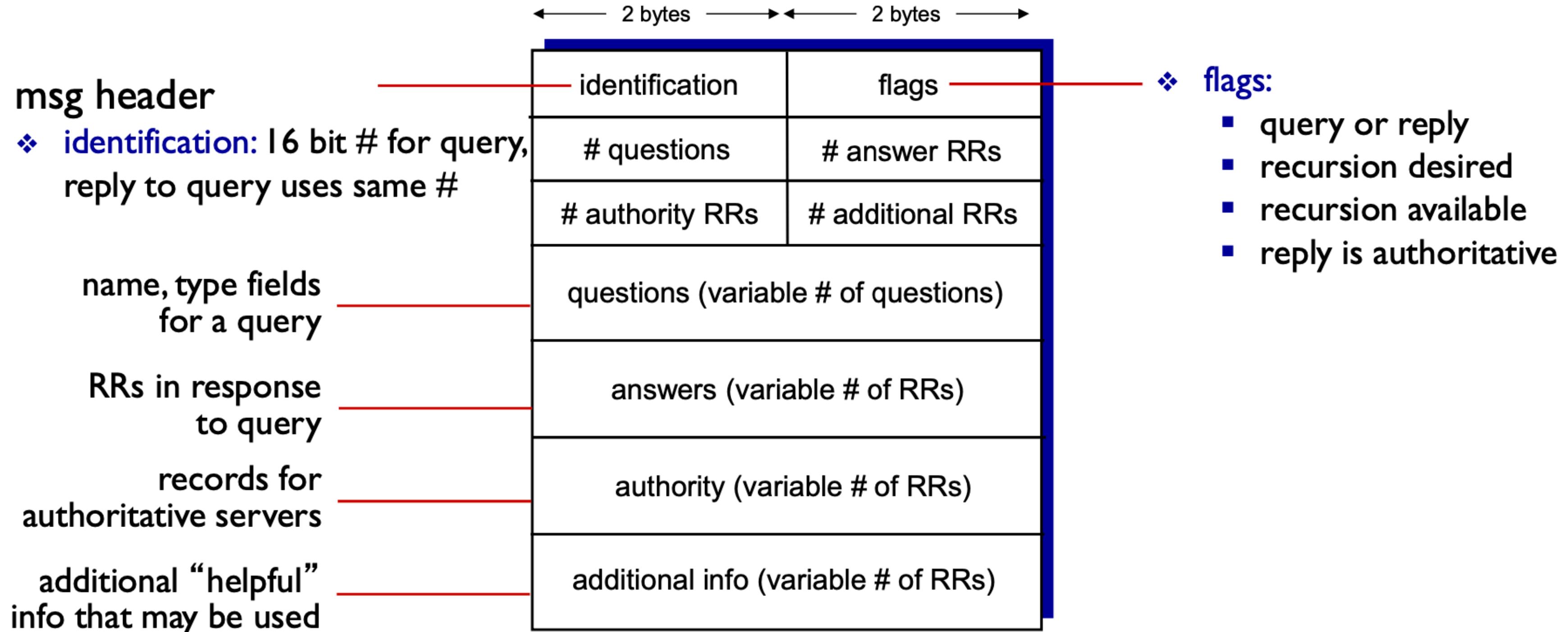
TYPE
two octets containing one of the RR TYPE codes.

CLASS
two octets containing one of the RR CLASS codes.

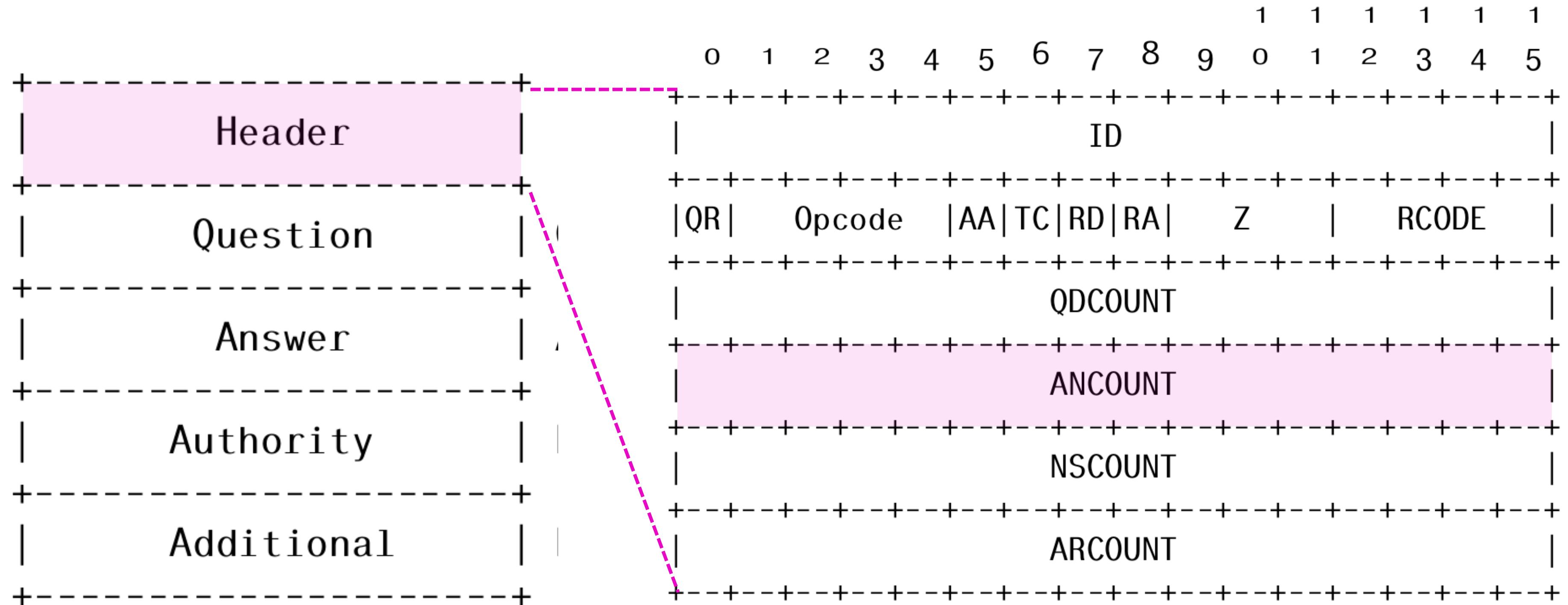
TTL
a 32 bit signed integer that specifies the time interval that the resource record may be cached before the source of the information should again be consulted. Zero values are interpreted to mean that the RR can only be used for the transaction in progress, and should not be cached. For example, 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.

DNS Protocol Messages

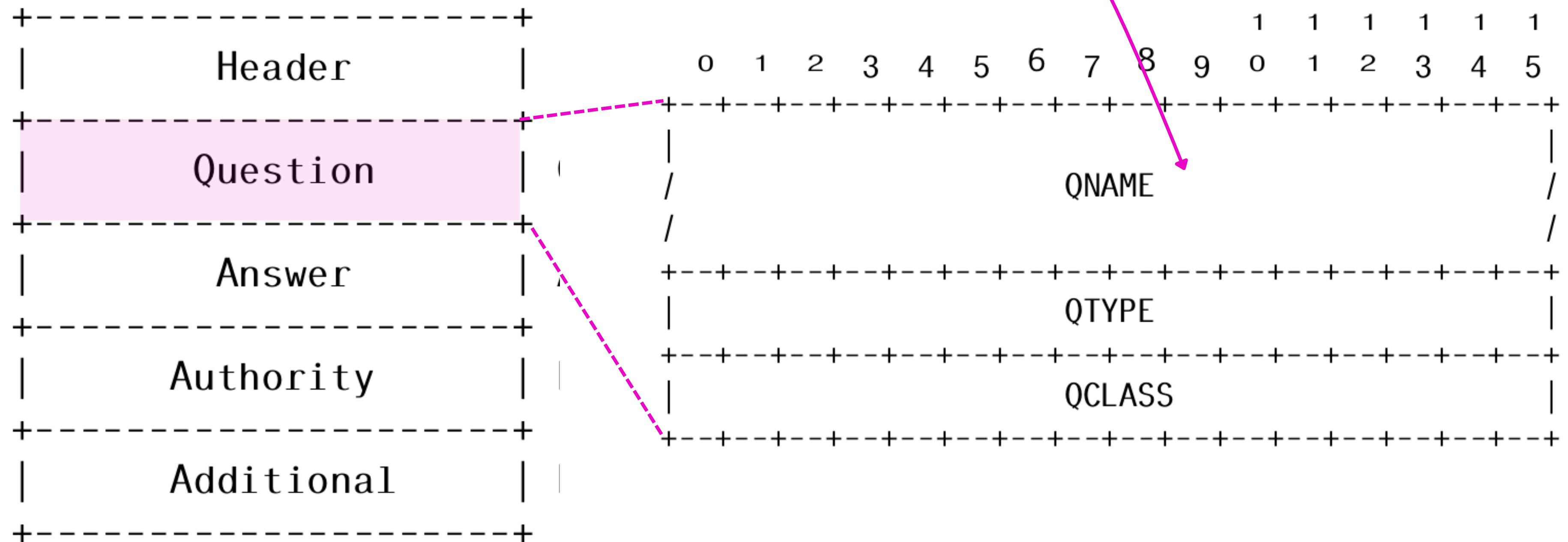


DNS Packet Structure



DNS Question

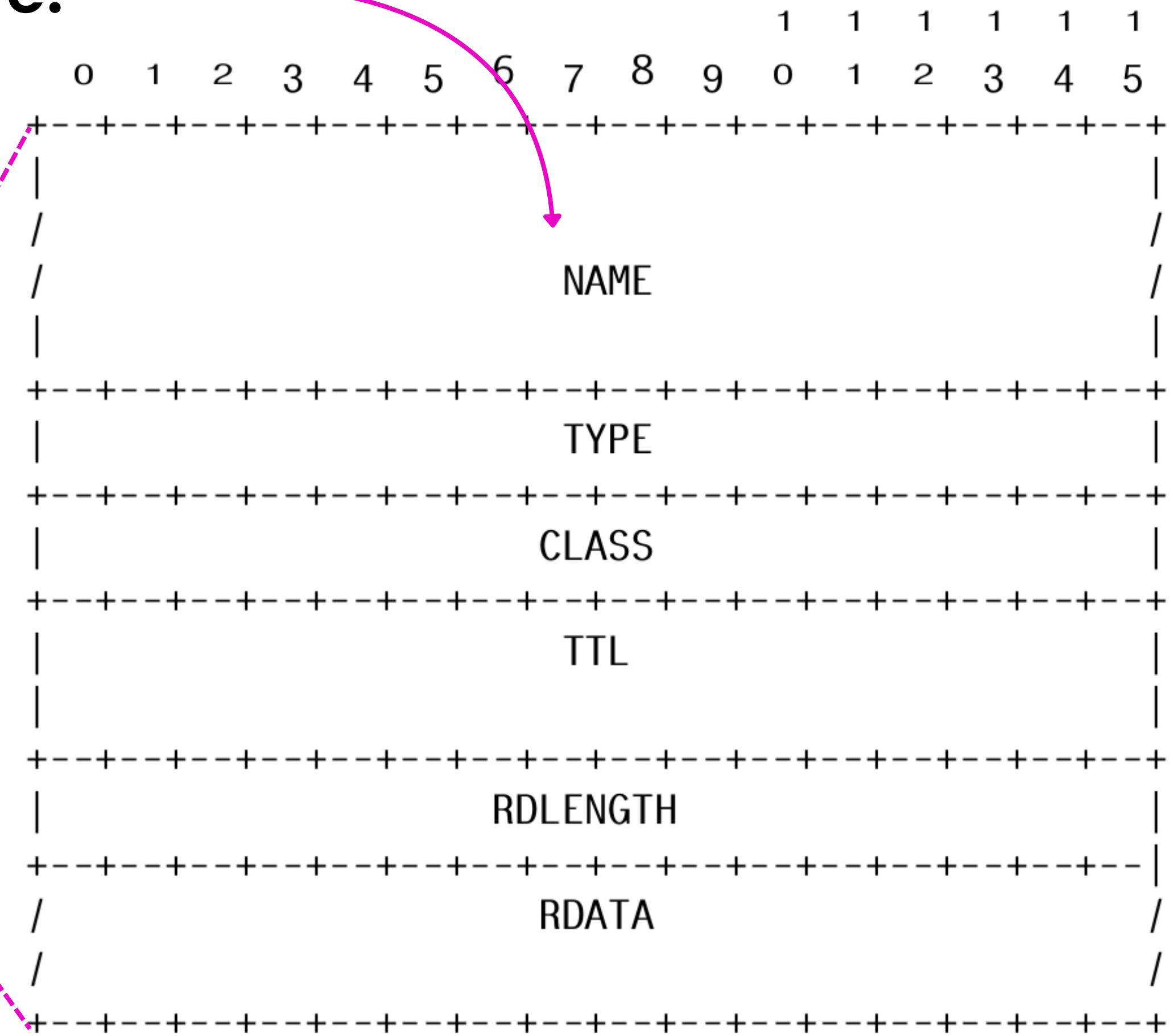
URL is here!



URL is here!

DNS Answer

+-----+ <td>Header</td>	Header
+-----+ <td>Question</td>	Question
+-----+ <td>Answer</td>	Answer
+-----+ <td>Authority</td>	Authority
+-----+ <td>Additional</td>	Additional



DNS Packet on Wireshark

Additional RRs: 27

Queries

- encrypted-tbn0.gstatic.com: type A, class IN
 - Name: encrypted-tbn0.gstatic.com
 - [Name Length: 26]
 - [Label Count: 3]
 - Type: A (Host Address) (1)
 - Class: IN (0x0001)

Answers

- encrypted-tbn0.gstatic.com: type A, class IN, addr 142.250.192.238
 - Name: encrypted-tbn0.gstatic.com
 - Type: A (Host Address) (1)
 - Class: IN (0x0001)
 - Time to live: 42 (42 seconds)
 - Data length: 4
 - Address: 142.250.192.238

Authoritative nameservers

Offset	Hex	Dec	Text
0050	6d 00 00 01 00 01 c0 0c 00 01 00 01 00 00 00 00 2am.....*	
0060	00 04 8e fa c0 ee 00 00 02 00 01 00 00 00 00 37 007.....	
0070	14 01 6a 0c 72 6f 6f 74 2d 73 65 72 76 65 72 73	..j.root-servers	
0080	03 6e 65 74 00 00 00 02 00 00 01 00 00 00 37 00 04	.net.....7...	
0090	01 69 c0 49 00 00 02 00 01 00 00 00 37 00 04 01	.i.I.....7...	
00a0	63 c0 49 00 00 02 00 01 00 00 00 37 00 04 01 67	c.I.....7...g	
00b0	c0 49 00 00 02 00 01 00 00 00 37 00 04 01 68 c0	.I.....7...h.	
00c0	49 00 00 02 00 01 00 00 00 37 00 04 01 61 c0 49	I.....7...a.I	
00d0	00 00 02 00 01 00 00 00 37 00 04 01 62 c0 49 007...b.I.	

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]
```

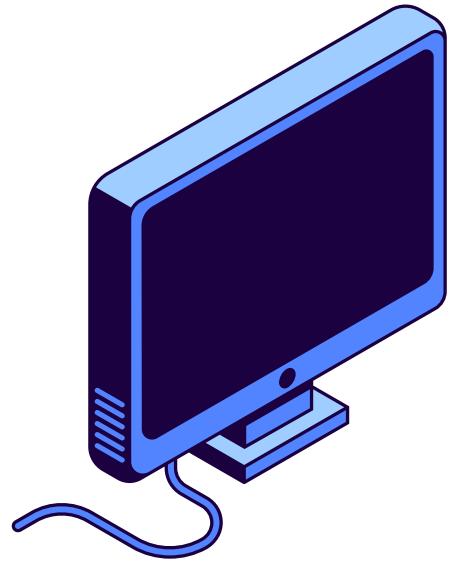
User Program (XSK - XDP Socket)

stub resolver
(us)



User Program (XSK - XDP Socket)

host

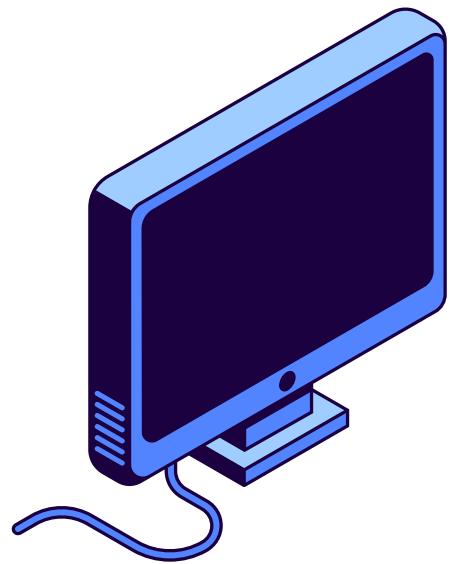


stub resolver
(us)



User Program (XSK - XDP Socket)

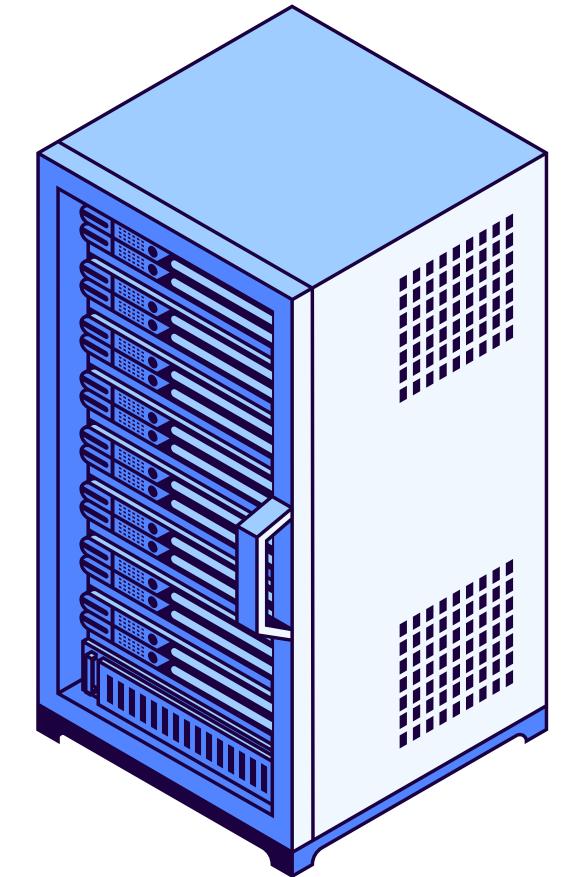
host



stub resolver
(us)

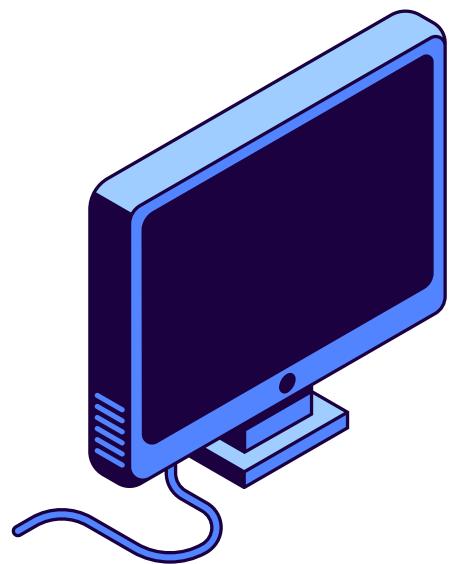


DNS Server
(8.8.8.8)



User Program (XSK - XDP Socket)

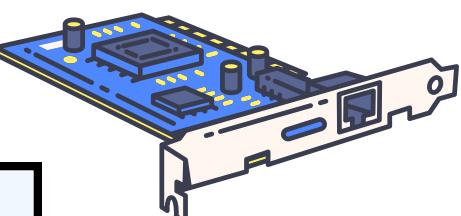
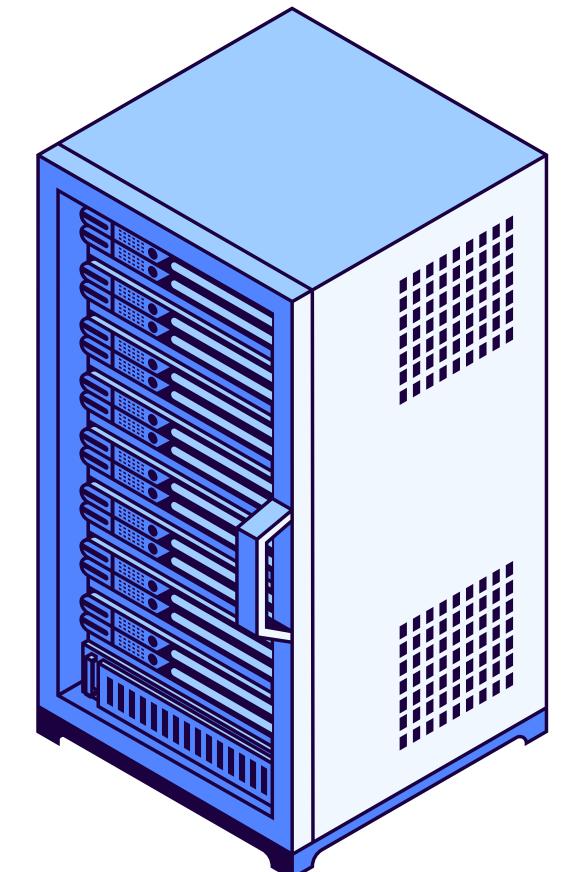
host



stub resolver
(us)



DNS Server
(8.8.8.8)



BPF
Program

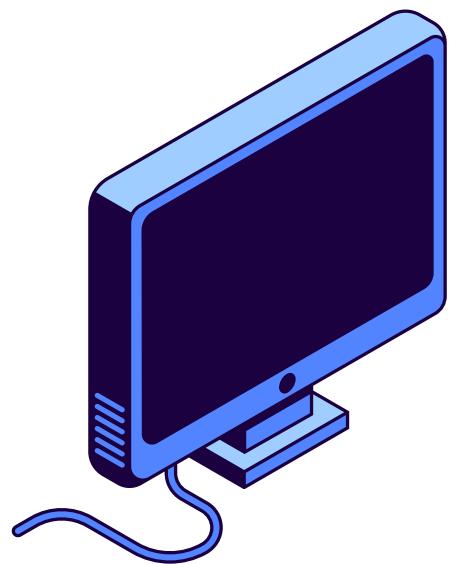
NIC

UDP
dst port = 53

User Program (XSK - XDP Socket)

host

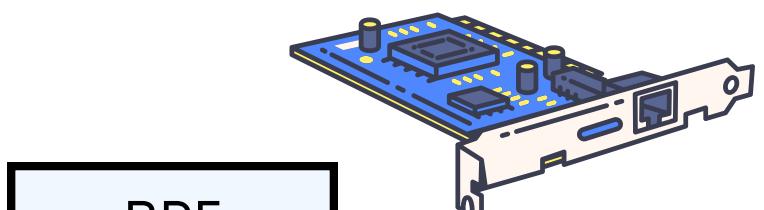
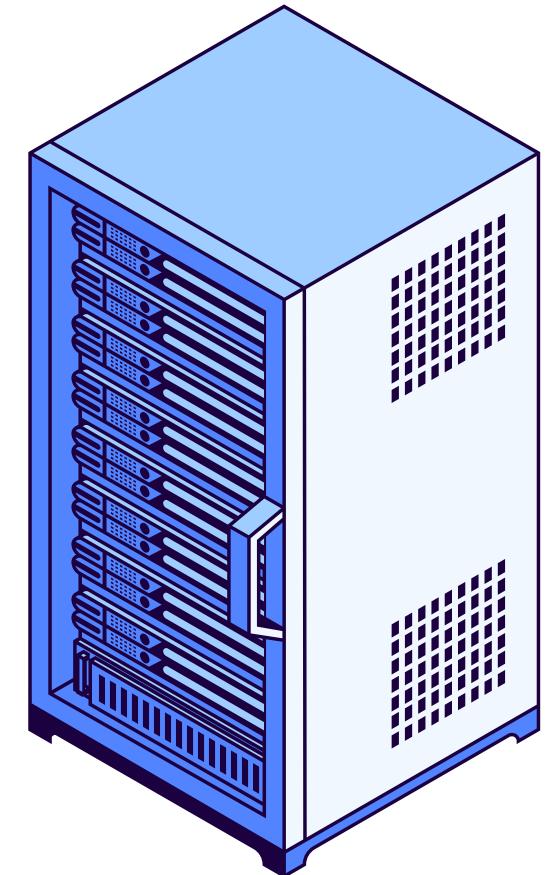
`dig @stubIP <url>`



stub resolver
(us)



DNS Server
(8.8.8.8)



BPF
Program

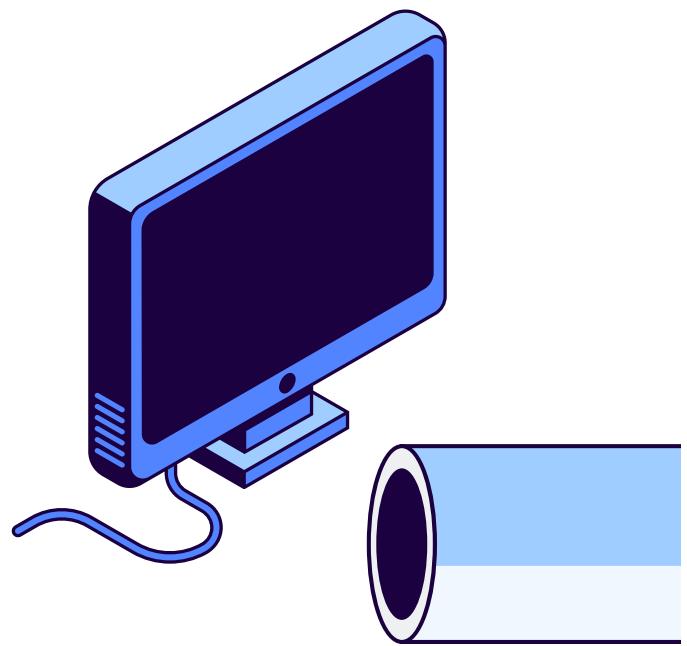
NIC

UDP
dst port = 53

User Program (XSK - XDP Socket)

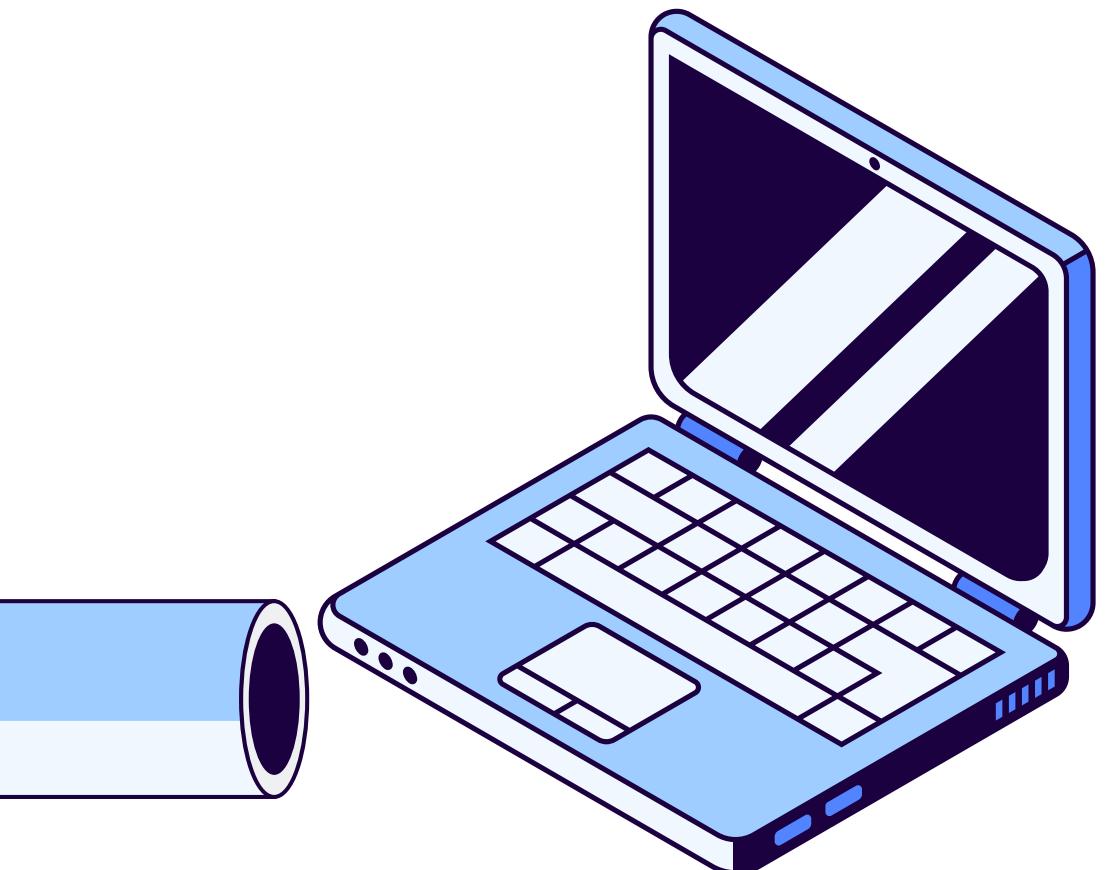
host

dig @stubIP <url>

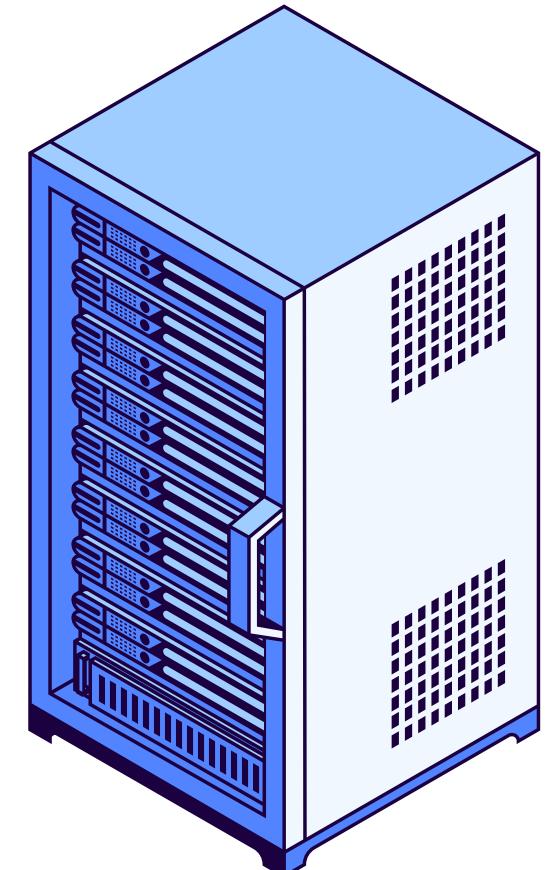


AF_XDP

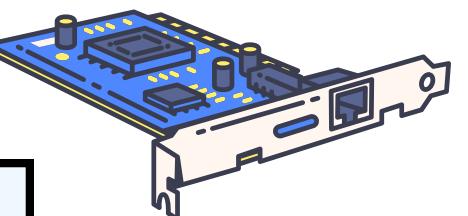
stub resolver
(us)



DNS Server
(8.8.8.8)



BPF
Program



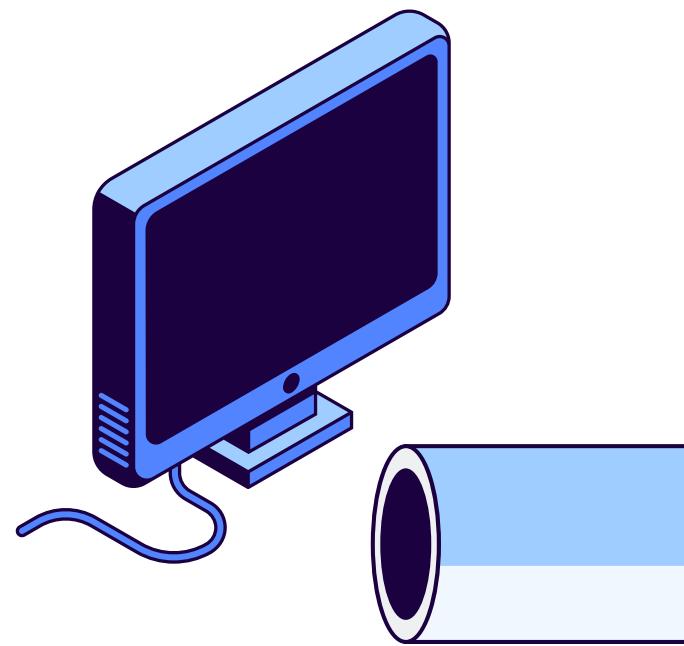
UDP
dst port = 53

NIC

User Program (XSK - XDP Socket)

host

`dig @stubIP <url>`



AF_XDP

stub resolver
(us)

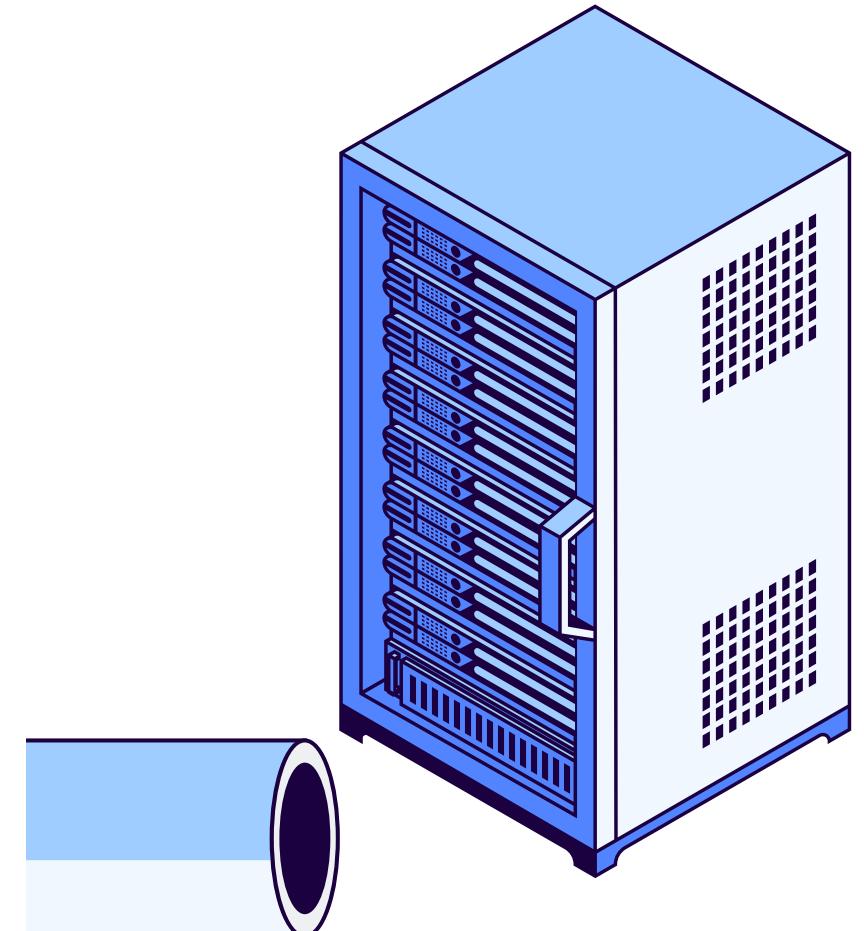


NIC

BPF
Program

UDP
dst port = 53

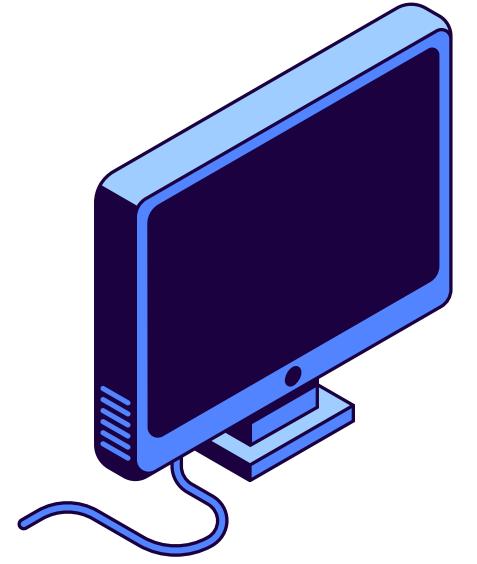
DNS Server
(8.8.8.8)



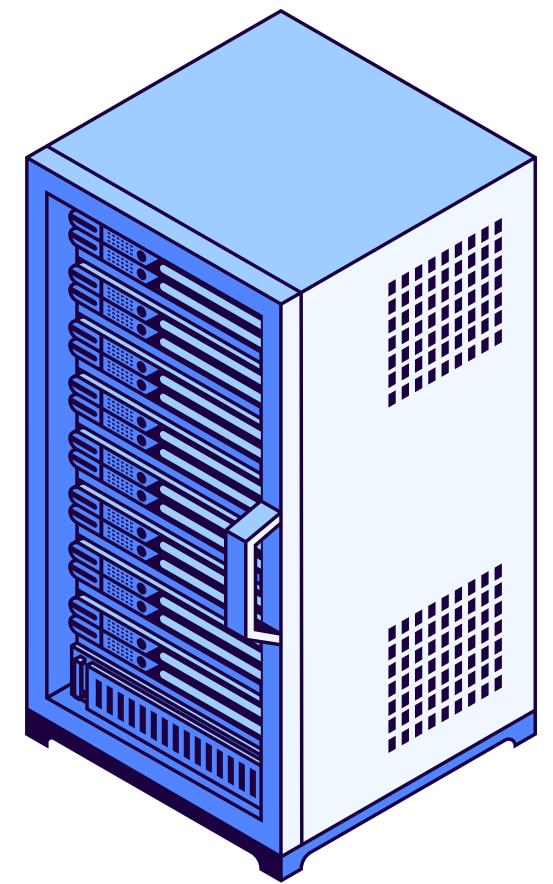
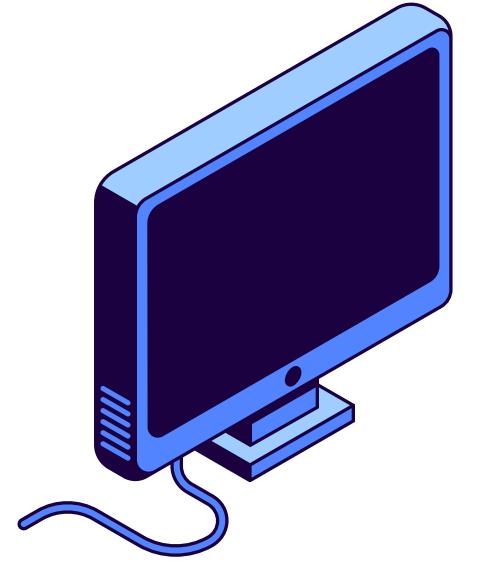
SOCK_DGRAM



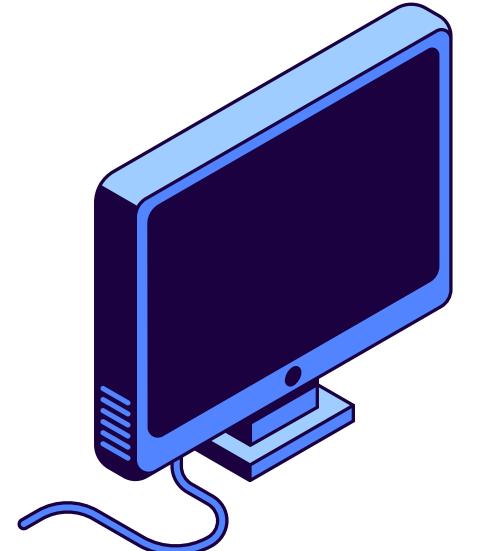
XDP
Redirect



XDP
Redirect



XDP
Redirect

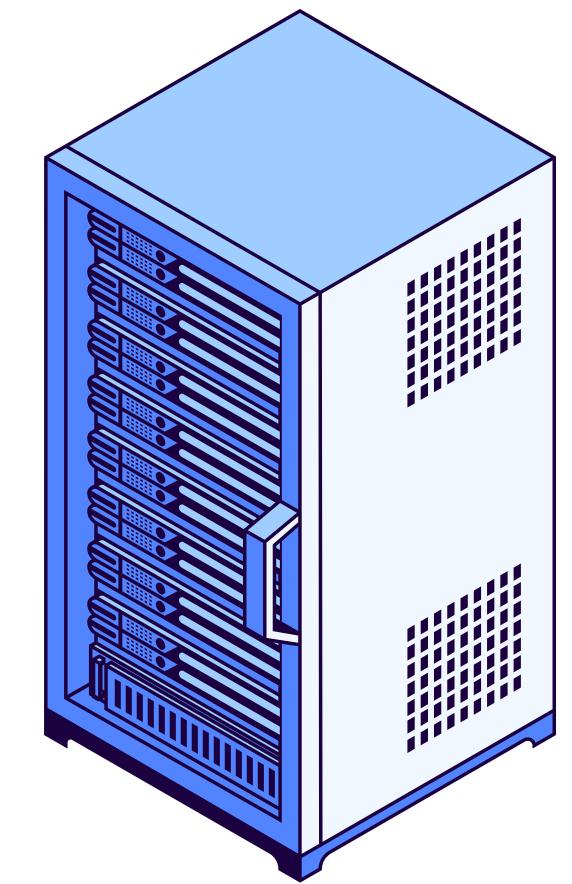


0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1				
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1				
Version		IHL		Type of Service				Total Length														3			
Identification				Flags		Fragment Offset																			
Time to Live		Protocol		Header Checksum																					
Source Address					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

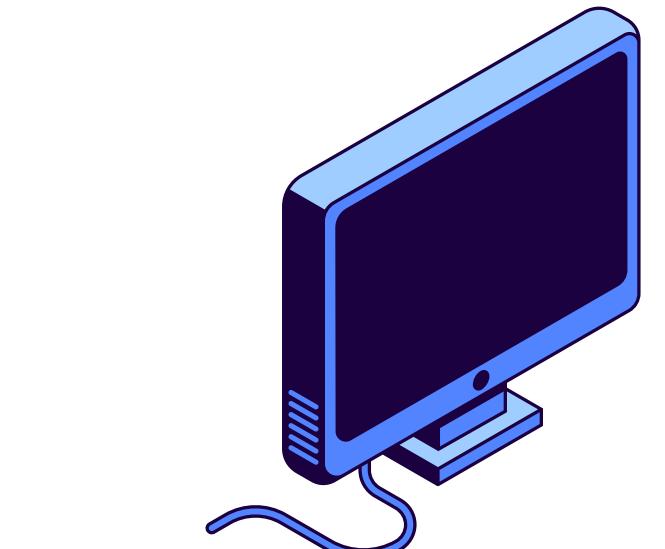
0	1	2	3	4	5	6	7	8	9	0	1	1	1	1	1	1	1	1	1	1	1		
/										/													
/	NAME										/												
/											/												
	TYPE																						
	CLASS																						
	TTL																						
	RDLENGTH																						
/	RDATA										/												
/											/												



XDP
Redirect



**Query RR
UDP Payload**



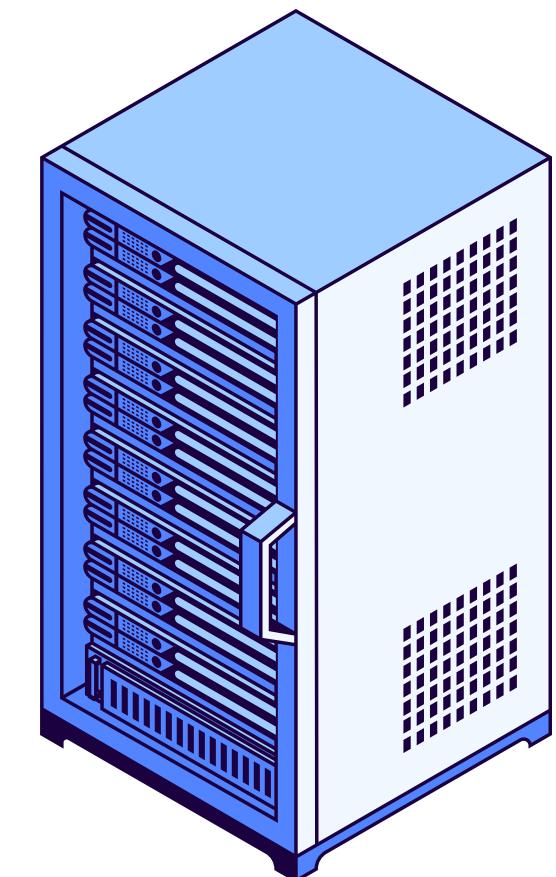
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+	Version IHL Type of Service Total Length	+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+----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

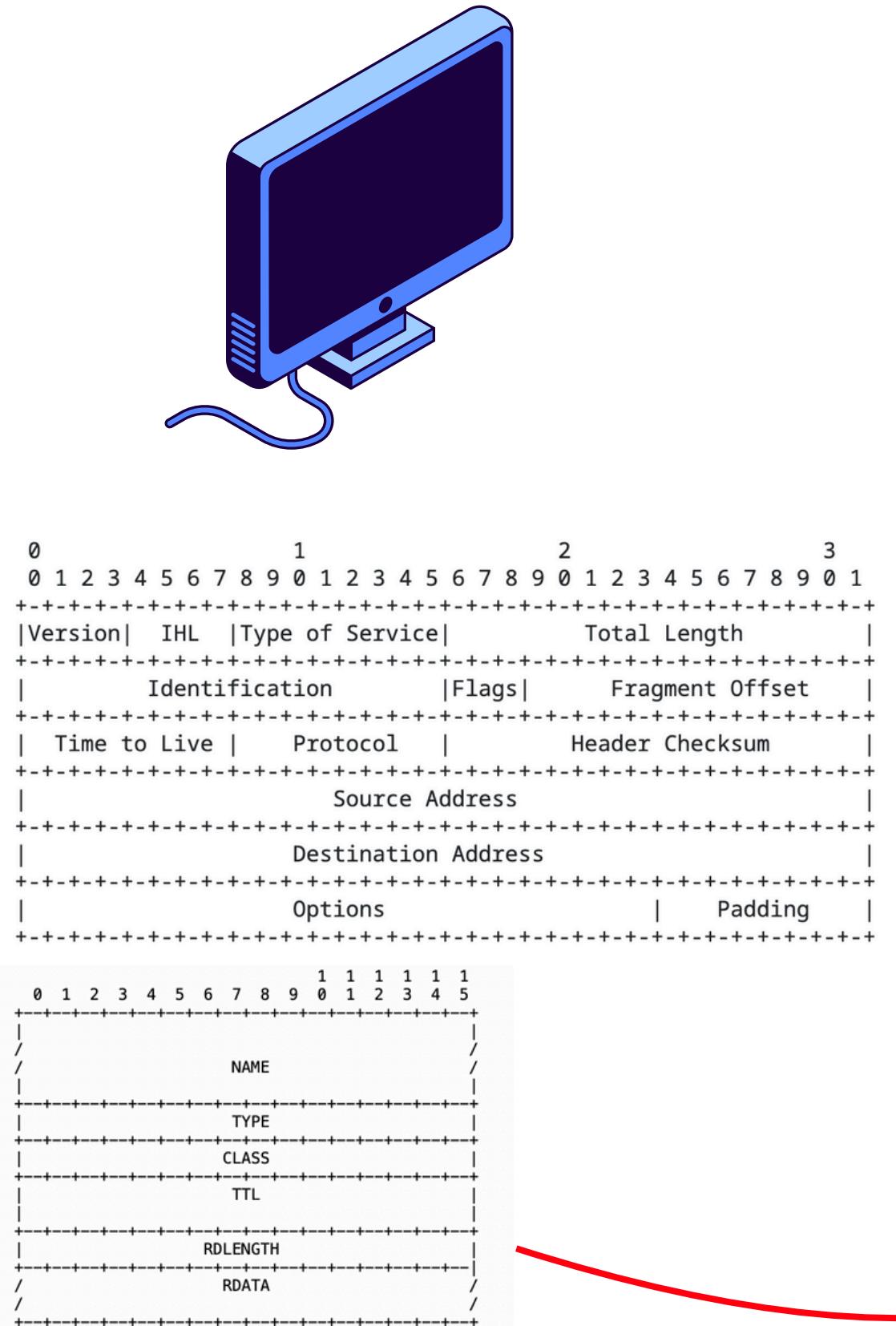
0	1	2	3	4	5	6	7	8	9	0	1	1	1	1	1	1	1	1	1	1	1
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+																					
/																					
/																					
NAME																					
TYPE													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



XDP
Redirect

**Query RR
UDP Payload**

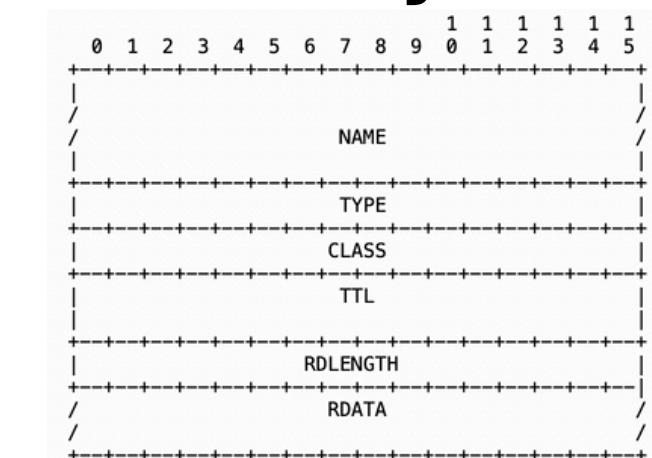




Query RR UDP Payload

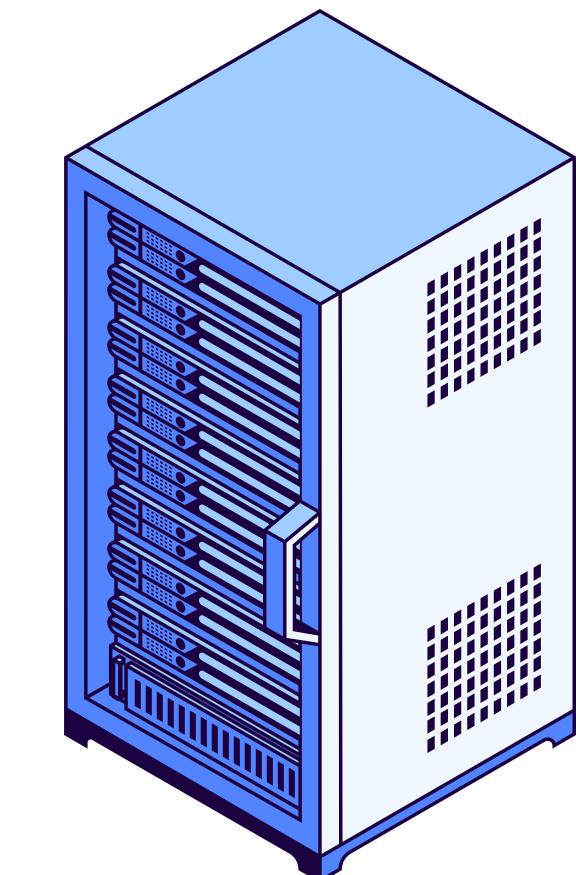


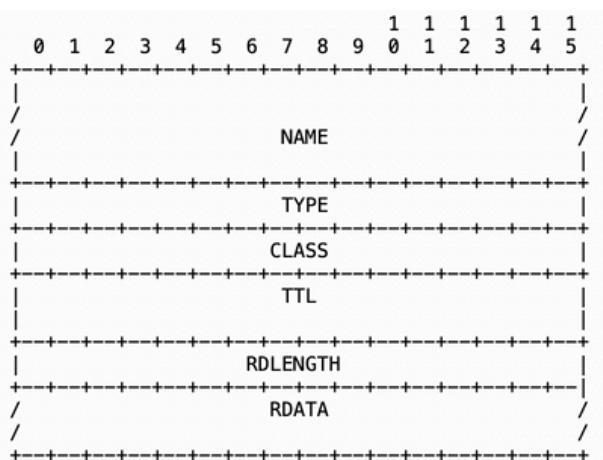
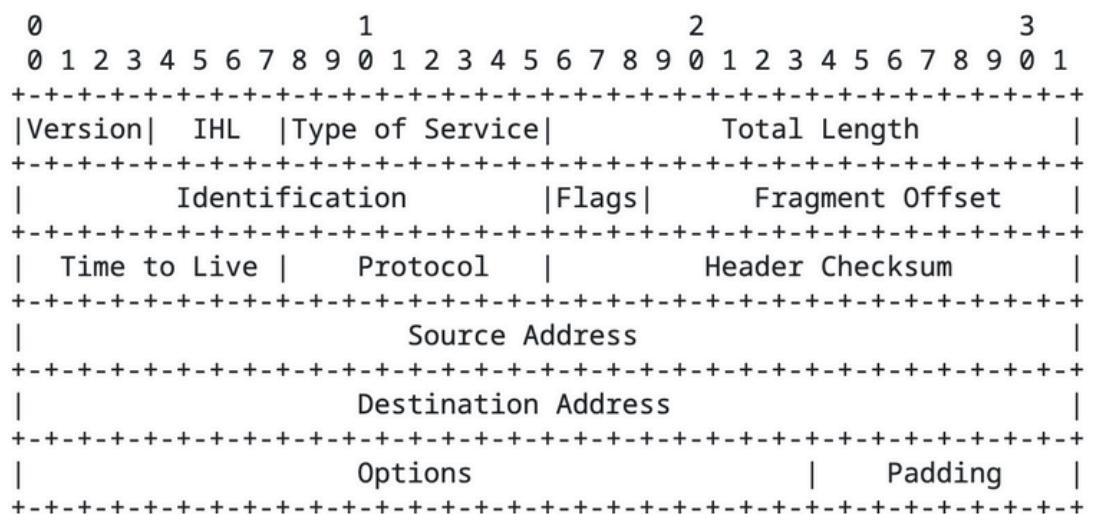
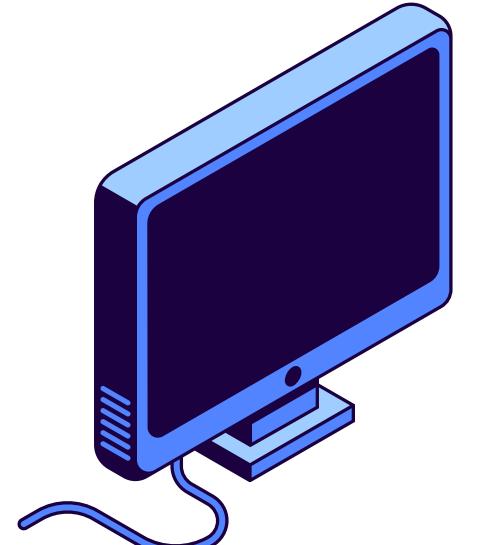
Query RR UDP Payload



`sendto(UDP_SOCK_FD)`

XDP
Redirect

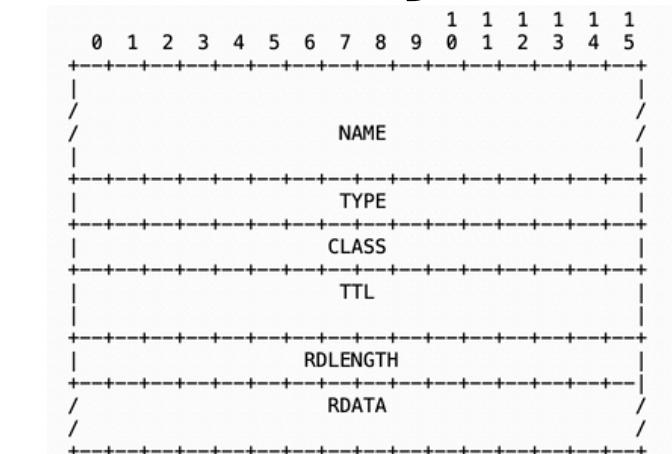




Query RR
UDP Payload



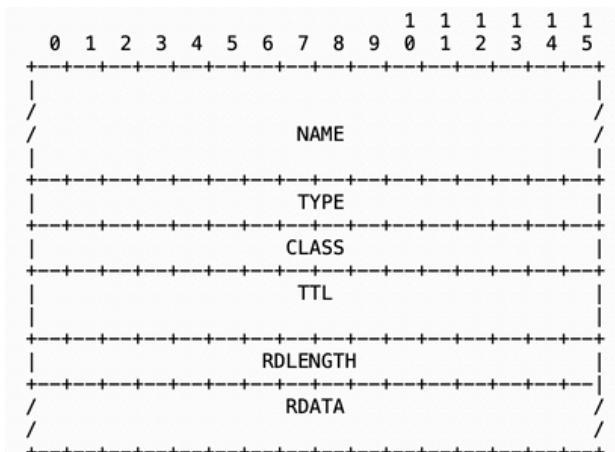
Query RR
UDP Payload



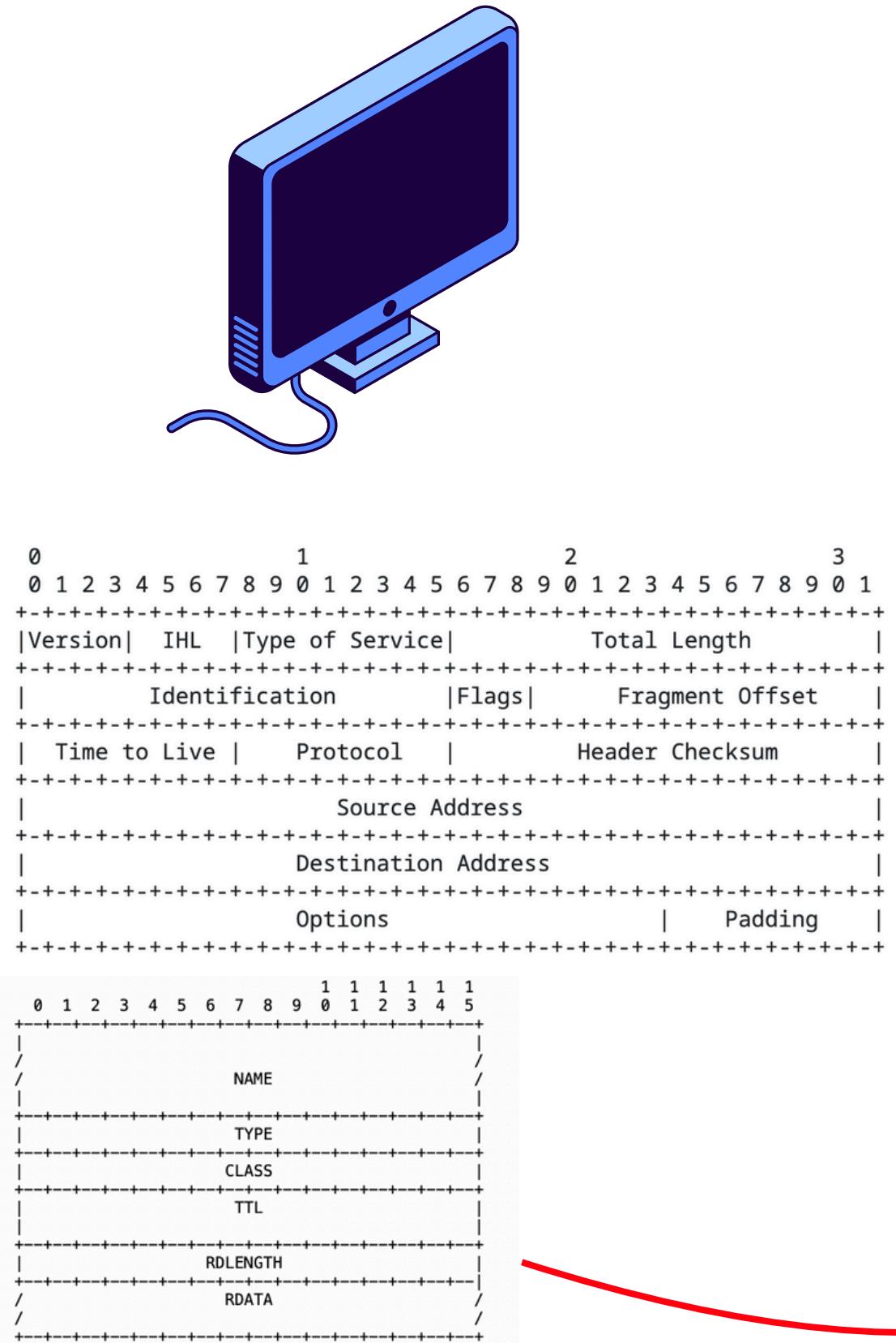
sendto(UDP_SOCK_FD)

XDP
Redirect

Answer RR
UDP Payload



CACHE IT!

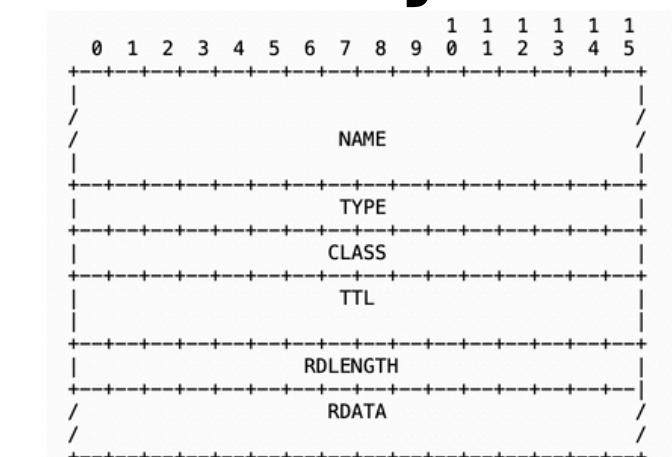


Query RR UDP Payload

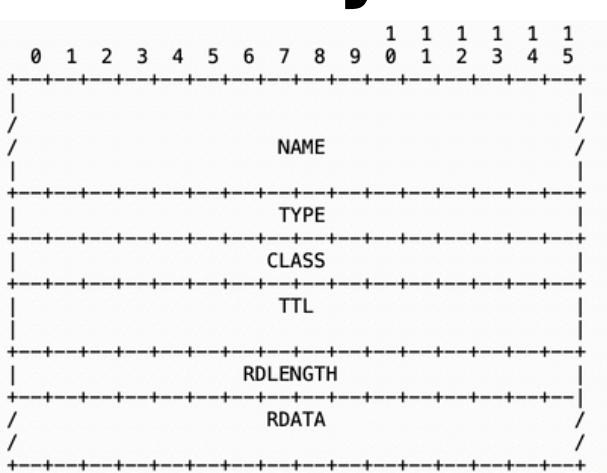


XDP
Redirect

sendto(UDP_SOCK_FD)

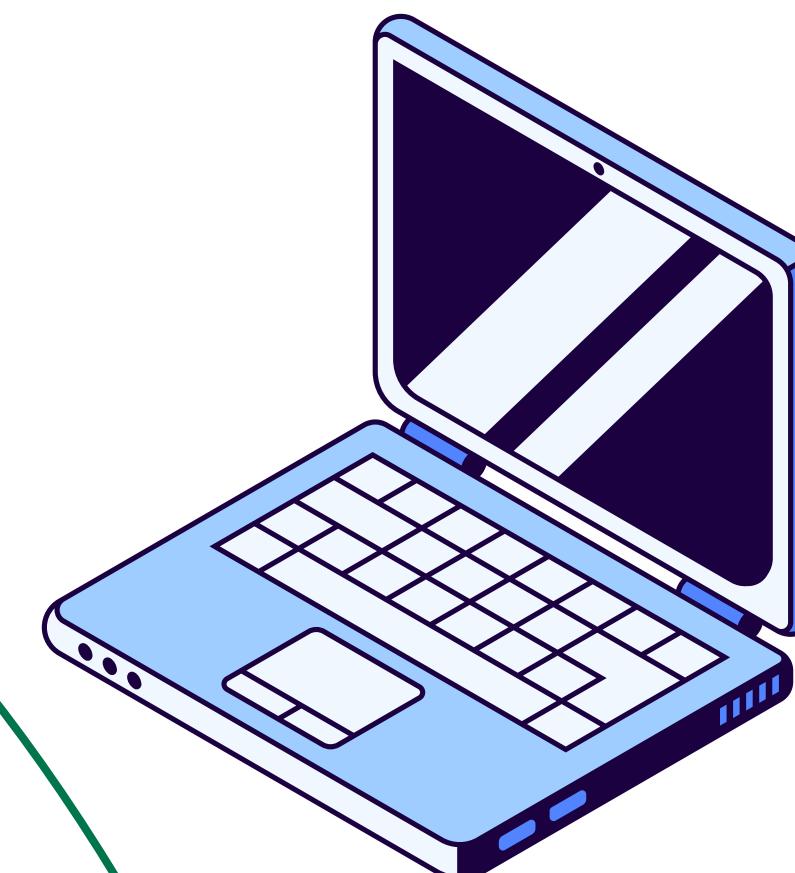
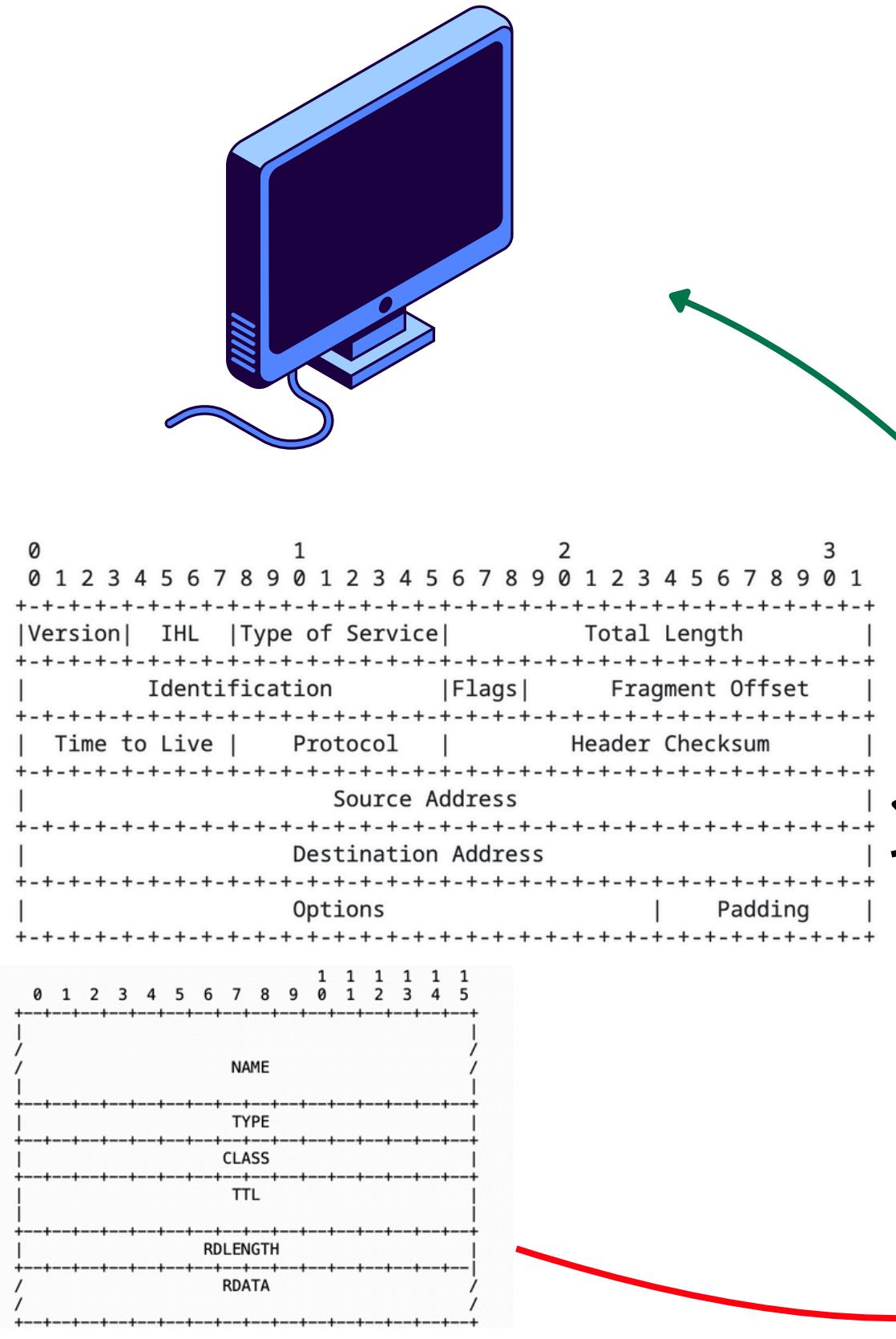


`recvfrom(UDP_SOCK_FD)`



Answer RR UDP Payload

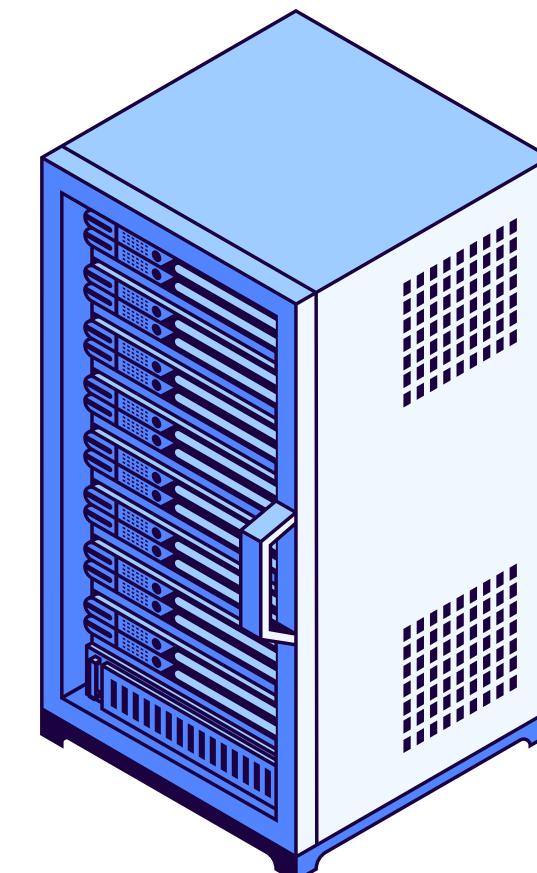
CACHE IT!



Query RR UDP Payload

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

`sendto(UDP_SOCK_FD)`



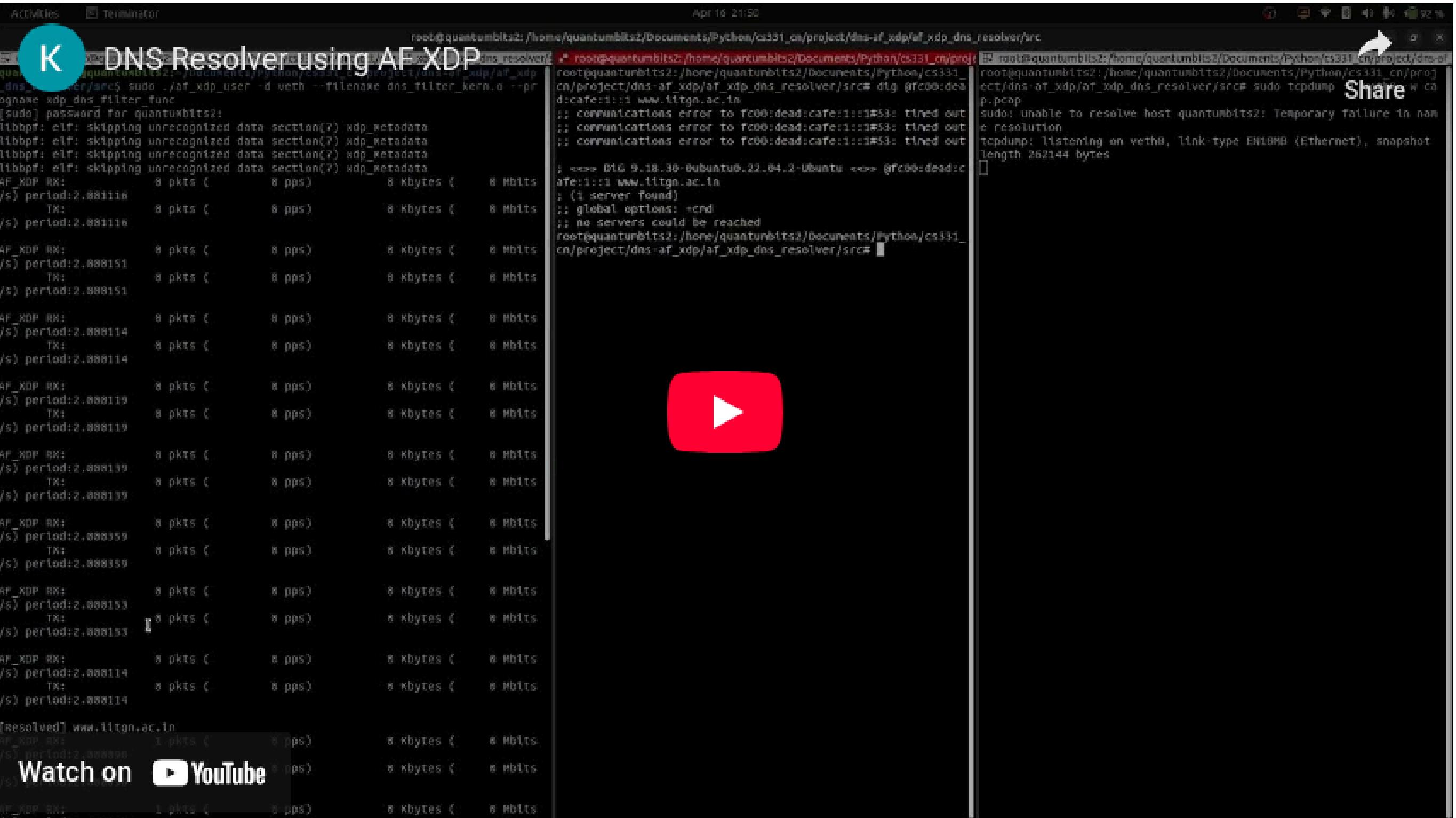
Answer RR UDP Payload

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

`recvfrom(UDP_SOCK_FD)`

CACHE IT!

Demonstration



User Program (XSK - XDP Socket)

```
#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

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);

    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)

```
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)

```
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)

```
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)

The screenshot shows a Wireshark capture window titled "cap_first.pcap". The main pane displays a list of network packets, and the bottom pane provides a detailed view of the selected packet (packet 30). A pink arrow highlights the transition from the first packet to the last packet.

Packets List:

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	fc00:dead:cafe:1::2	fc00:dead:cafe:1::1	DNS	90	Standard query 0x0000 A google.com
2	0.001134	fc00:dead:cafe:1::2	fc00:dead:cafe:1::1	DNS	92	Standard query 0x0001 A facebook.com
3	0.001145	fc00:dead:cafe:1::2	fc00:dead:cafe:1::1	DNS	95	Standard query 0x0002 A doubleclick.net
4	0.002219	fc00:dead:cafe:1::2	fc00:dead:cafe:1::1	DNS	100	Standard query 0x0003 A google-analytics.com
5	0.002227	fc00:dead:cafe:1::2	fc00:dead:cafe:1::1	DNS	92	Standard query 0x0004 A akamaihd.net
6	0.003332	fc00:dead:cafe:1::2	fc00:dead:cafe:1::1	DNS	101	Standard query 0x0005 A googlesyndication.com
7	0.003350	fc00:dead:cafe:1::2	fc00:dead:cafe:1::1	DNS	94	Standard query 0x0006 A googleapis.com
8	0.004457	fc00:dead:cafe:1::2	fc00:dead:cafe:1::1	DNS	100	Standard query 0x0007 A googleadservices.com
9	0.004475	fc00:dead:cafe:1::2	fc00:dead:cafe:1::1	DNS	92	Standard query 0x0008 A facebook.net
10	0.005560	fc00:dead:cafe:1::2	fc00:dead:cafe:1::1	DNS	91	Standard query 0x0009 A youtube.com
11	0.005568	fc00:dead:cafe:1::2	fc00:dead:cafe:1::1	DNS	91	Standard query 0x00a A twitter.com
12	0.006691	fc00:dead:cafe:1::2	fc00:dead:cafe:1::1	DNS	101	Standard query 0x00b A scorecardresearch.com
13	0.006715	fc00:dead:cafe:1::2	fc00:dead:cafe:1::1	DNS	93	Standard query 0x00c A microsoft.com
14	0.008581	fc00:dead:cafe:1::2	fc00:dead:cafe:1::1	DNS	89	Standard query 0x00d A ytimg.com
15	0.008591	fc00:dead:cafe:1::2	fc00:dead:cafe:1::1	DNS	101	Standard query 0x00e A googleusercontent.com
16	0.009660	fc00:dead:cafe:1::2	fc00:dead:cafe:1::1	DNS	89	Standard query 0x00f A apple.com
17	0.009667	fc00:dead:cafe:1::2	fc00:dead:cafe:1::1	DNS	92	Standard query 0x010 A msftncsi.com
18	0.010733	fc00:dead:cafe:1::2	fc00:dead:cafe:1::1	DNS	88	Standard query 0x011 A 2mdn.net
19	0.010740	fc00:dead:cafe:1::2	fc00:dead:cafe:1::1	DNS	101	Standard query 0x012 A googletagservices.com
20	0.011806	fc00:dead:cafe:1::2	fc00:dead:cafe:1::1	DNS	89	Standard query 0x013 A adnxs.com
21	0.011813	fc00:dead:cafe:1::2	fc00:dead:cafe:1::1	DNS	89	Standard query 0x014 A yahoo.com
22	0.012879	fc00:dead:cafe:1::2	fc00:dead:cafe:1::1	DNS	95	Standard query 0x015 A serving-sys.com
23	0.012887	fc00:dead:cafe:1::2	fc00:dead:cafe:1::1	DNS	90	Standard query 0x016 A akadns.net
24	0.013952	fc00:dead:cafe:1::2	fc00:dead:cafe:1::1	DNS	91	Standard query 0x017 A bluekai.com
25	0.013959	fc00:dead:cafe:1::2	fc00:dead:cafe:1::1	DNS	89	Standard query 0x018 A ggptt.com
26	0.015048	fc00:dead:cafe:1::2	fc00:dead:cafe:1::1	DNS	98	Standard query 0x019 A rubiconproject.com
27	0.015062	fc00:dead:cafe:1::2	fc00:dead:cafe:1::1	DNS	92	Standard query 0x01a A verisign.com
28	0.016137	fc00:dead:cafe:1::2	fc00:dead:cafe:1::1	DNS	91	Standard query 0x01b A addthis.com
29	0.016149	fc00:dead:cafe:1::2	fc00:dead:cafe:1::1	DNS	95	Standard query 0x01c A crashlytics.com
30	0.016851	fc00:dead:cafe:1::1	fc00:dead:cafe:1::2	DNS	106	Standard query response 0x000 A google.com A 142.251.42.14
31	0.017236	fc00:dead:cafe:1::2	fc00:dead:cafe:1::1	DNS	93	Standard query 0x01d A amazonaws.com
32	0.017248	fc00:dead:cafe:1::2	fc00:dead:cafe:1::1	DNS	94	Standard query 0x01e A quantserve.com

Selected Packet Details: (Packet 30)

- User Datagram Protocol, Src Port: 37633, Dst Port: 53
- Domain Name System (response)
 - Transaction ID: 0x0000
 - Flags: 0x8180 Standard query response, No error
 - Questions: 1
 - Answer RRs: 1
 - Authority RRs: 0
 - Additional RRs: 0
- Queries
 - google.com: type A, class IN, addr 142.251.42.14
- Answers
 - [Unsolicited: True]

Hex and ASCII panes: The bottom right corner shows the raw hex and ASCII representations of the selected DNS response packet.

Metrics: Cache Hits (dnsperf)

x10 Speedup!

The screenshot shows a Wireshark capture window titled "cap_cache.pcap". The list pane displays 17 DNS frames. Frame 5 is highlighted with a pink arrow and is expanded in the details and bytes panes.

Frame 5 Details:

- Frame 5: 108 bytes on wire (864 bits), 108 bytes captured (864 bits)
- Ethernet II, Src: 06:44:b6:11:6d:ef (06:44:b6:11:6d:ef), Dst: 9a:41:79:7e:6d:3a (9a:41:79:7e:6d:3a)
- Internet Protocol Version 6, Src: fc00:dead:cafe:1::1, Dst: fc00:dead:cafe:1::2
- User Datagram Protocol, Src Port: 56815, Dst Port: 53
- Domain Name System (response)

Frame 5 Bytes:

0000	9a 41 79 7e 6d 3a 06 44 b6 11 6d ef 86 dd 60 0d
0010	72 08 00 36 11 40 fc 00 de ad ca fe 00 01 00 00
0020	00 00 00 00 00 01 fc 00 de ad ca fe 00 01 00 00
0030	00 00 00 00 00 02 dd ef 00 35 00 36 89 30 00 01
0040	81 80 00 01 00 01 00 00 00 00 08 66 61 63 65 62
0050	6f 6f 6b 03 63 6f 6d 00 00 01 00 01 c0 0c 00 01
0060	00 01 00 00 00 3c 00 04 39 90 b0 01

Frame 5 Summary:

- Transaction ID: 0x0001
- Flags: 0x8180 Standard query response, No error
- Questions: 1
- Answer RRs: 1
- Authority RRs: 0
- Additional RRs: 0
- Queries
- Answers
 - facebook.com: type A, class IN, addr 57.144.176.1
[Unsolicited: True]

Comparison: First Hits vs. Cache Hits

cap_first.pcap:

Average Latency = 26.322772 seconds

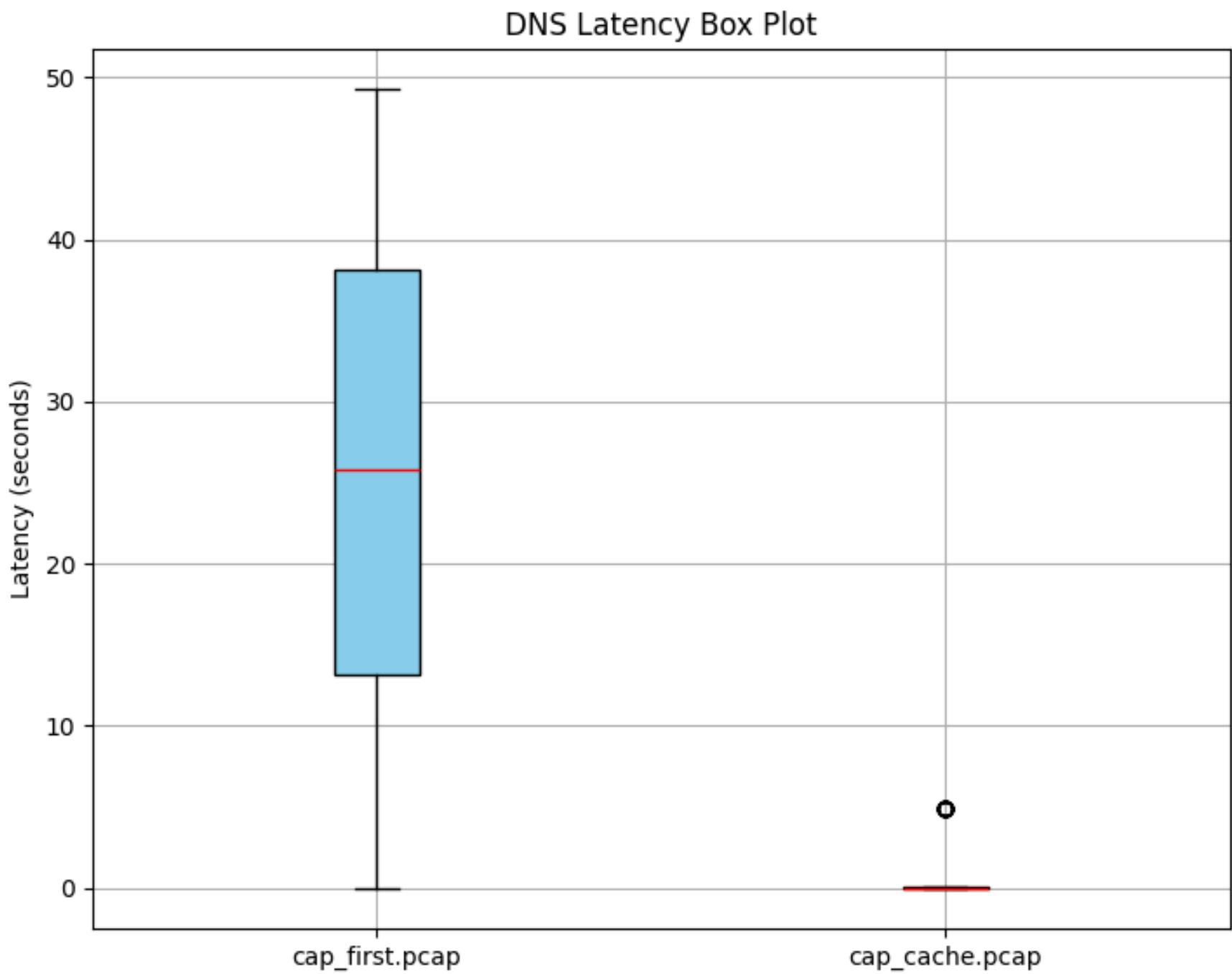
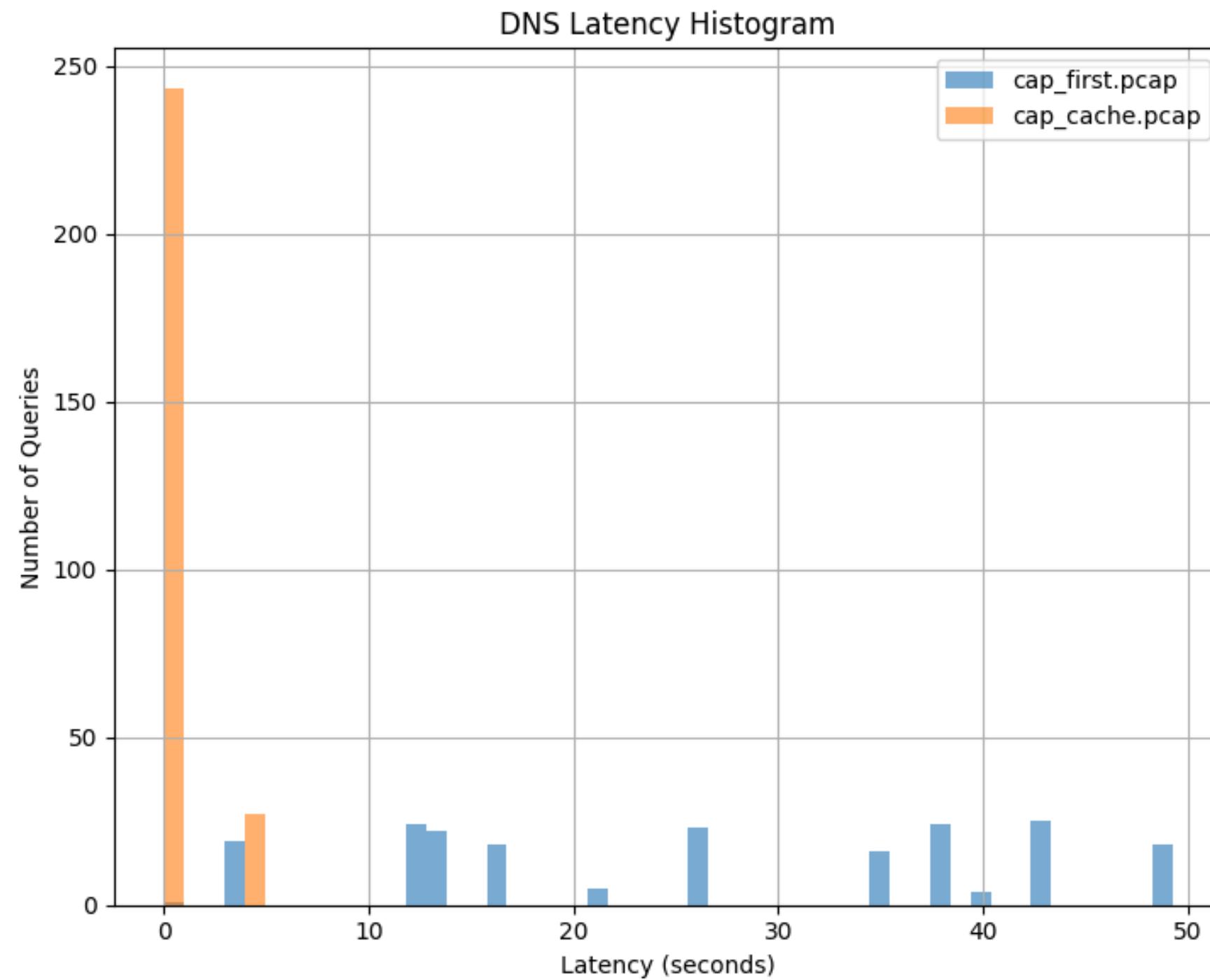
Median Latency = 25.766722 seconds

cap_cache.pcap:

Average Latency = 0.512208 seconds

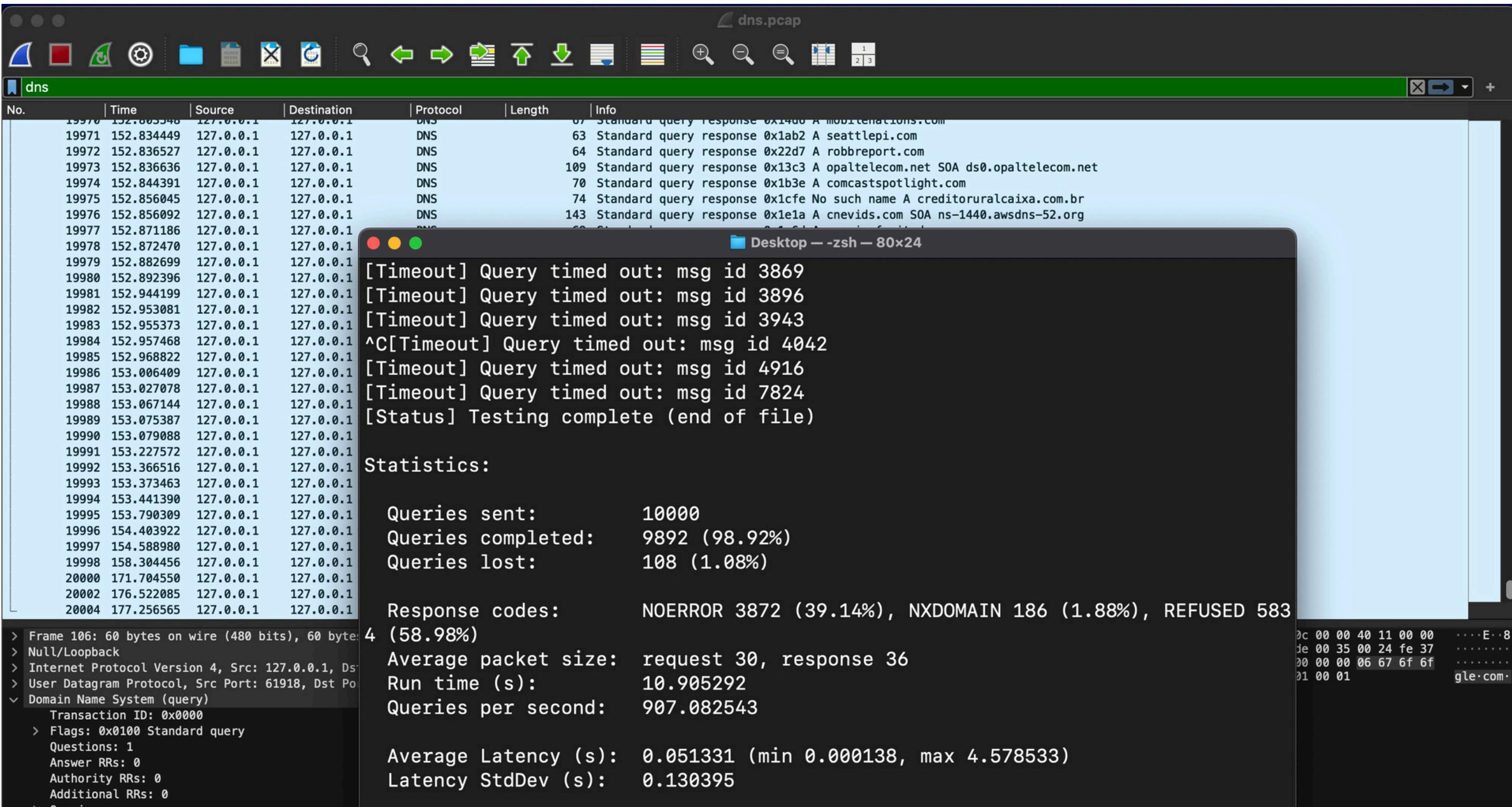
Median Latency = 0.022440 seconds

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

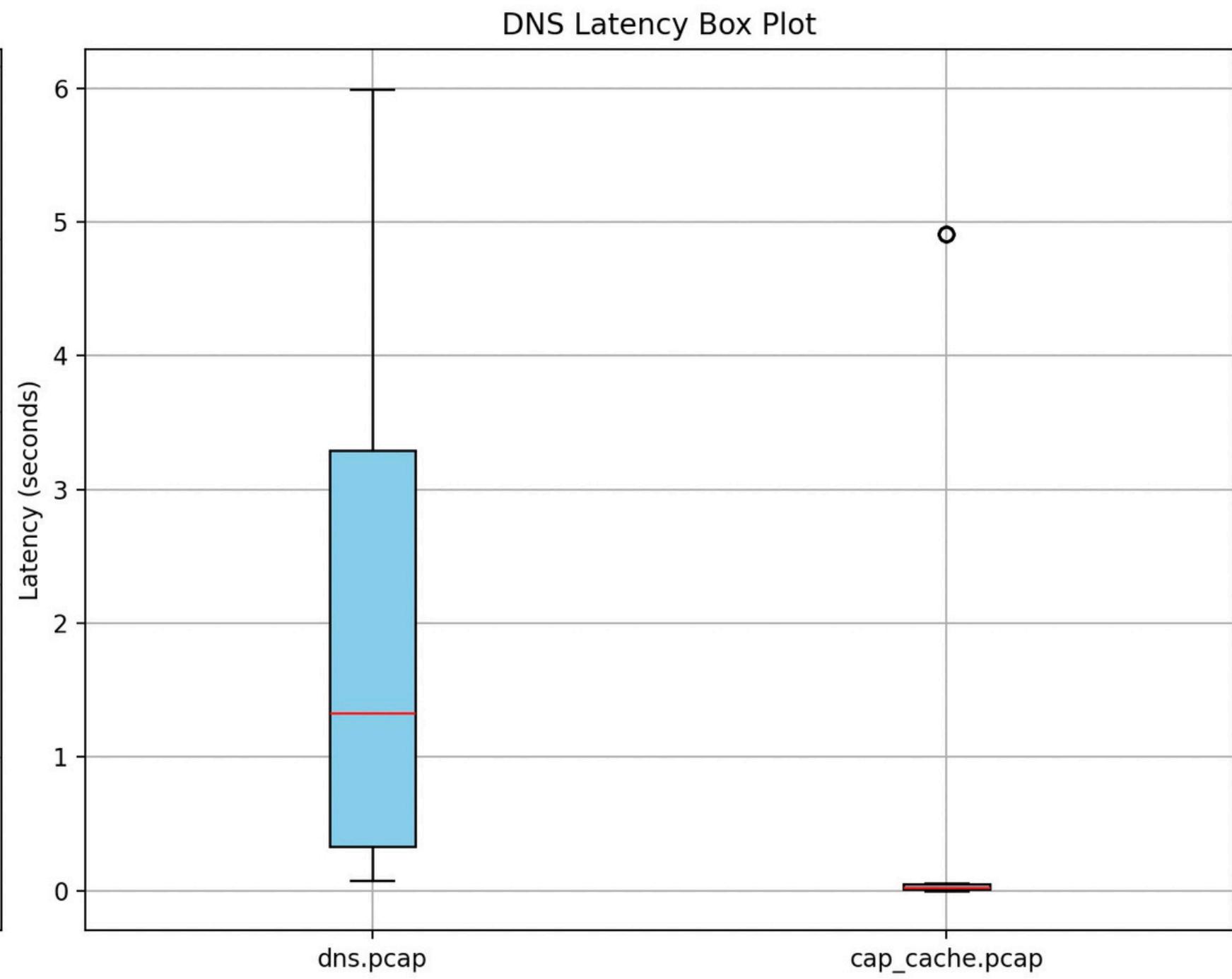
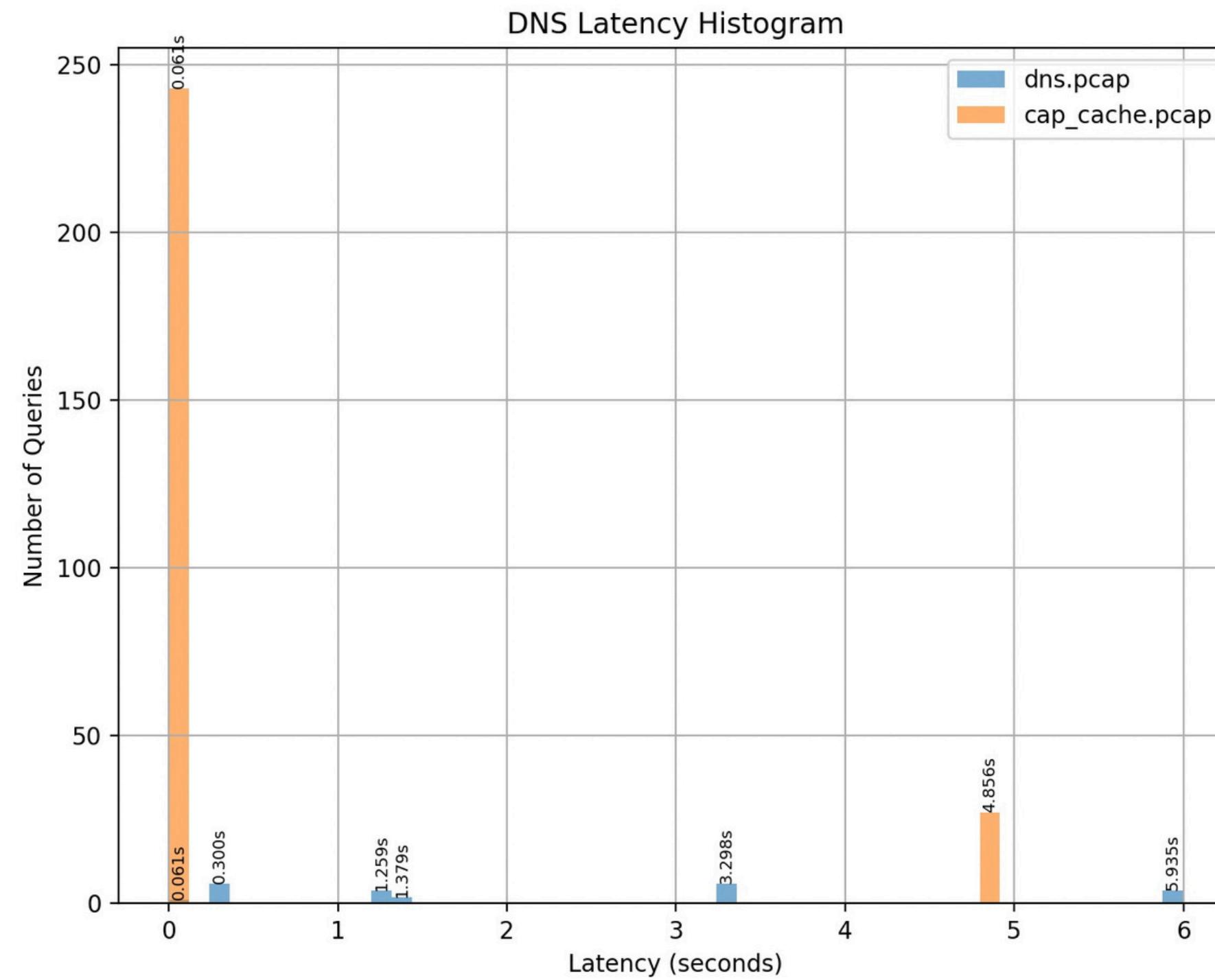
Median Latency = 1.328159 seconds

`cap_cache.pcap:`

Average Latency = 0.512208 seconds

Median Latency = 0.022440 seconds

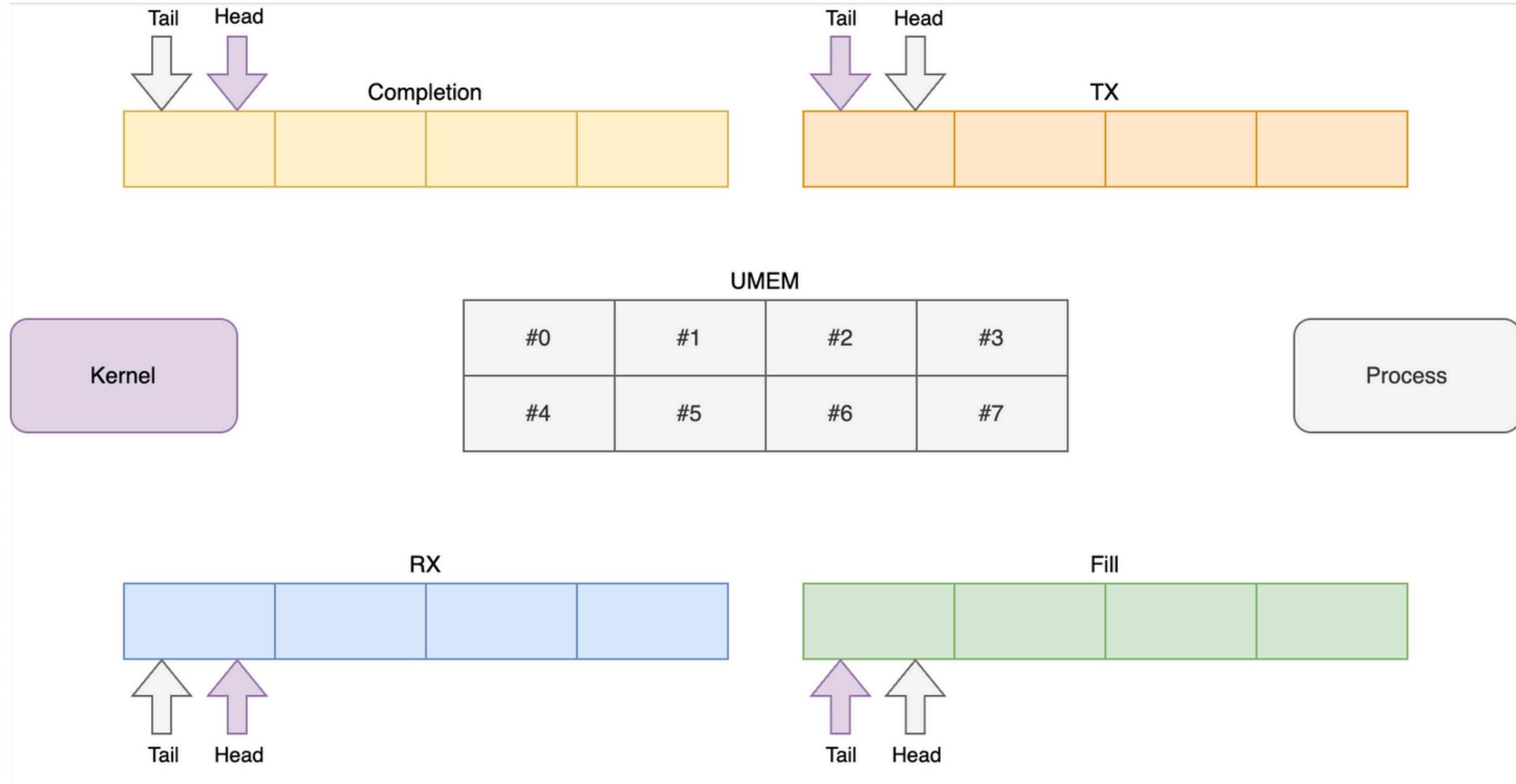
Comparison: Direct DNS vs. Cache Hits



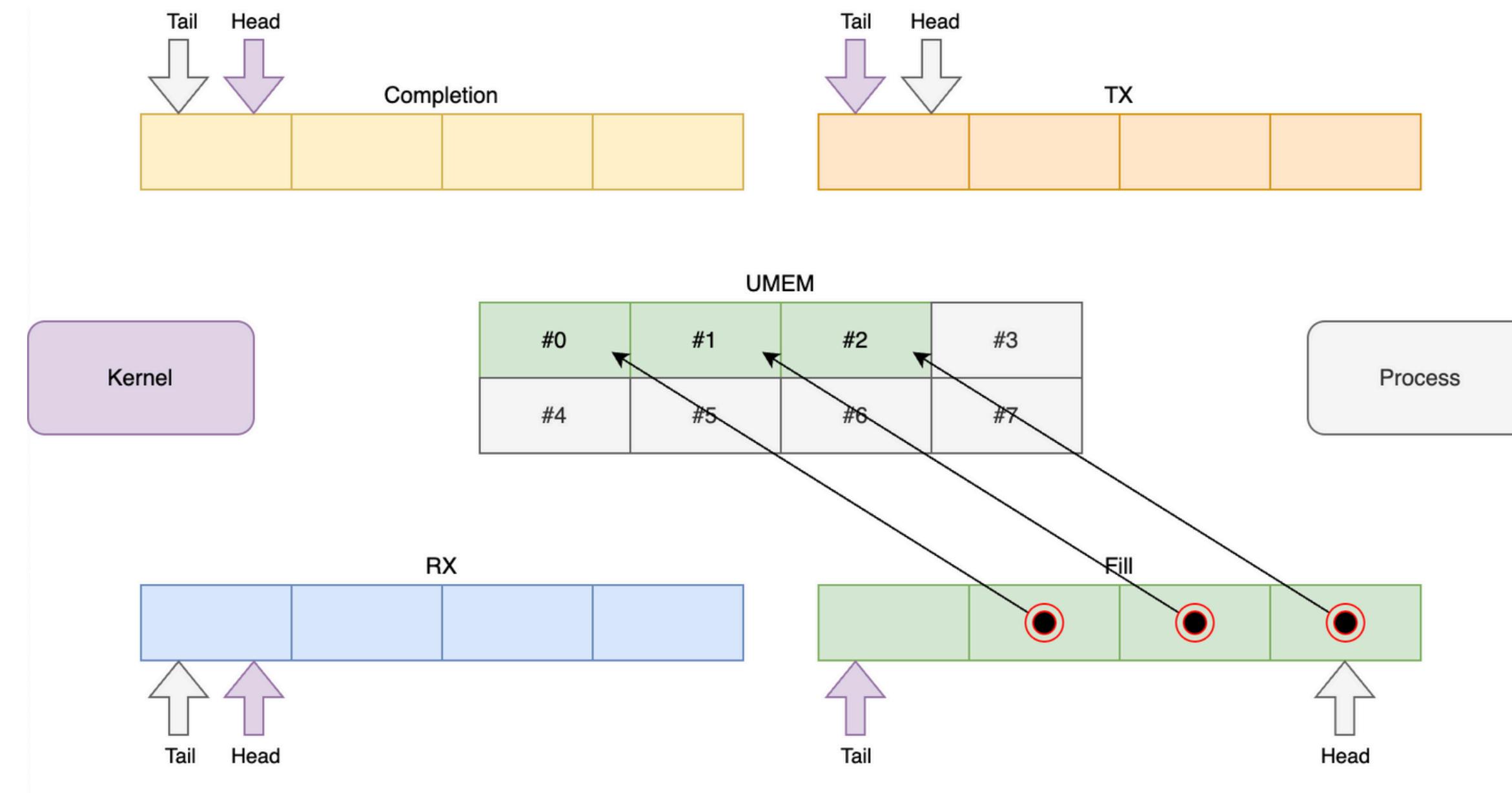
Appendix

XDP Socket Working

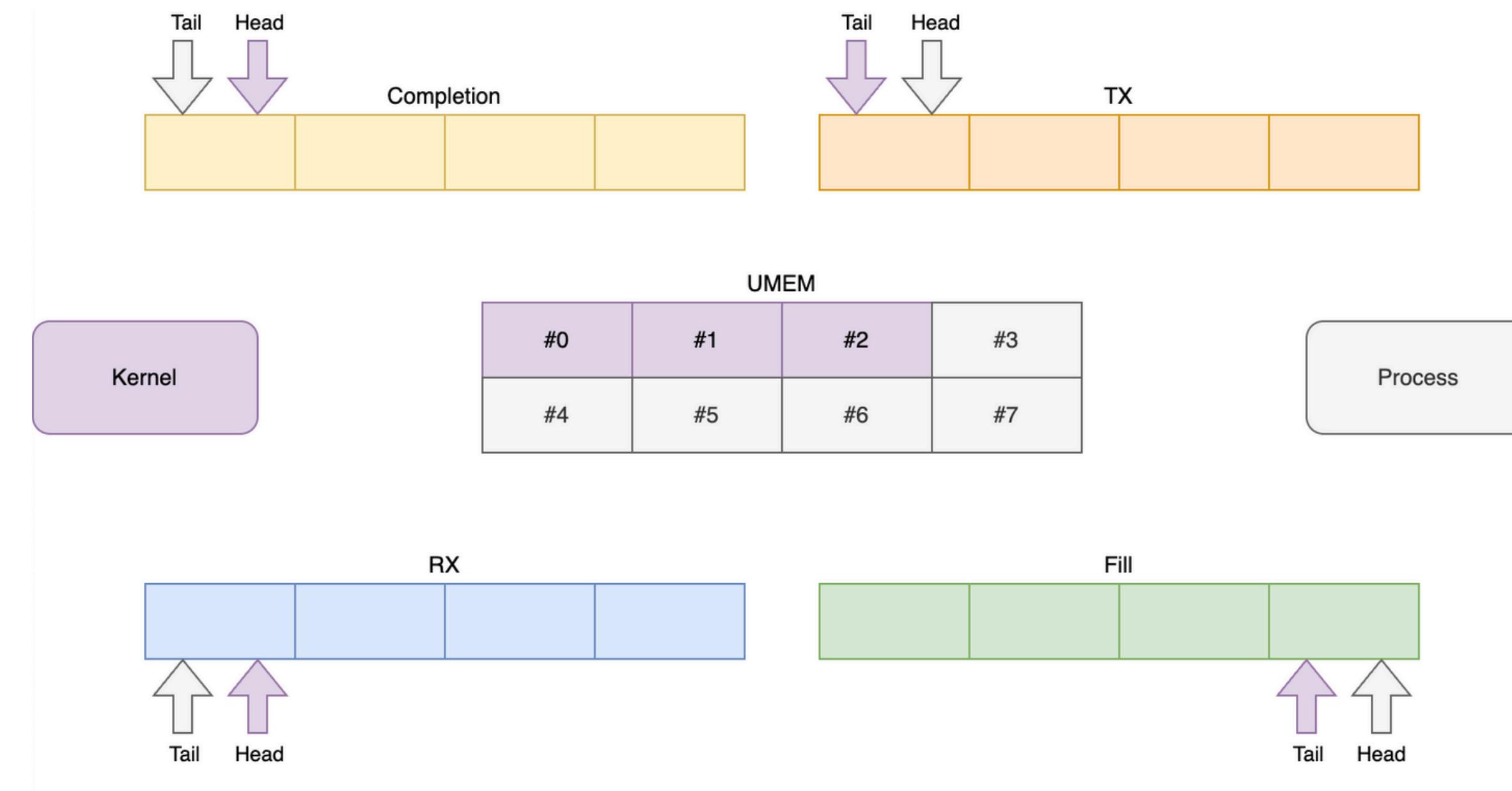
XSK - XDP Socket



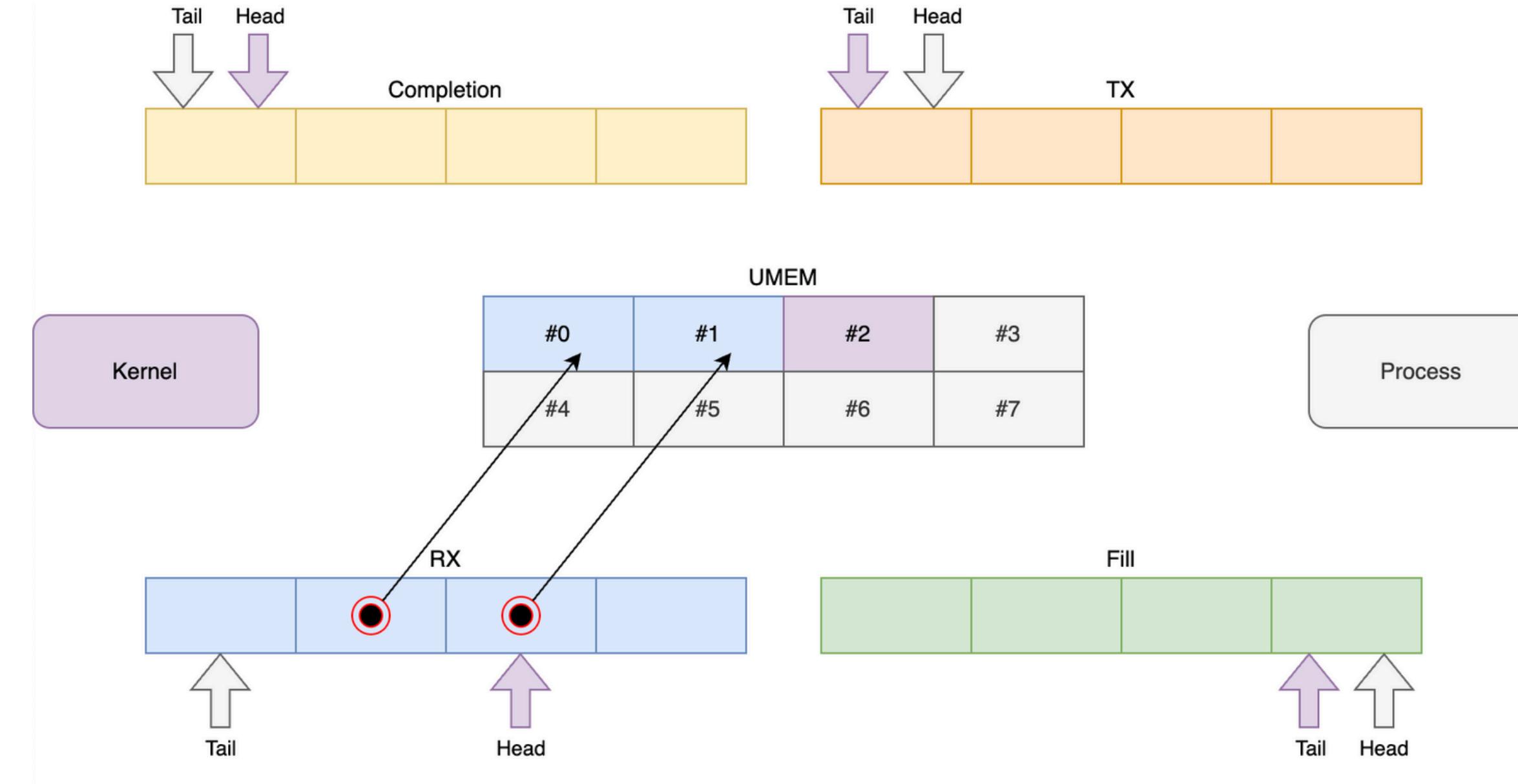
XSK - XDP Socket



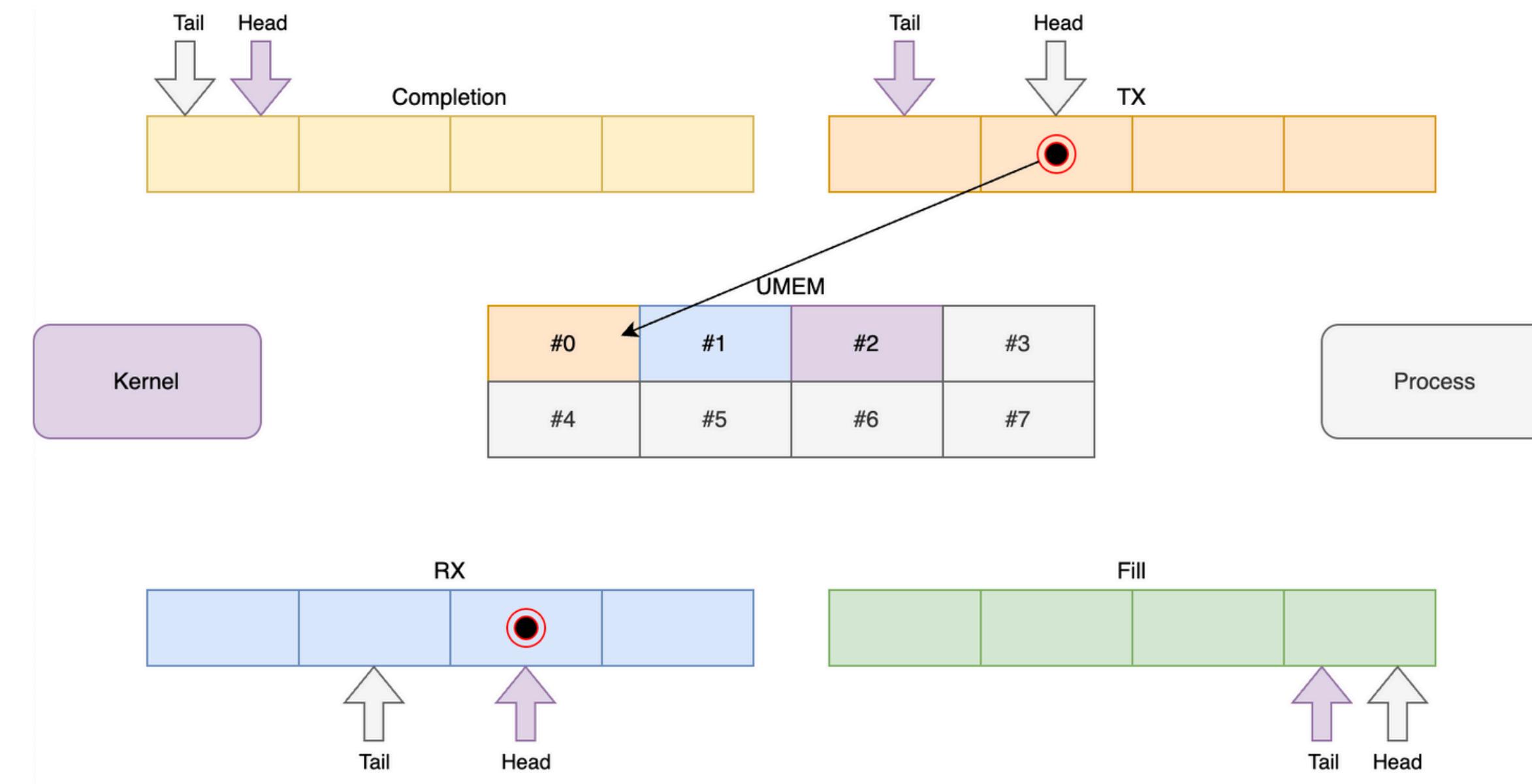
XSK - XDP Socket



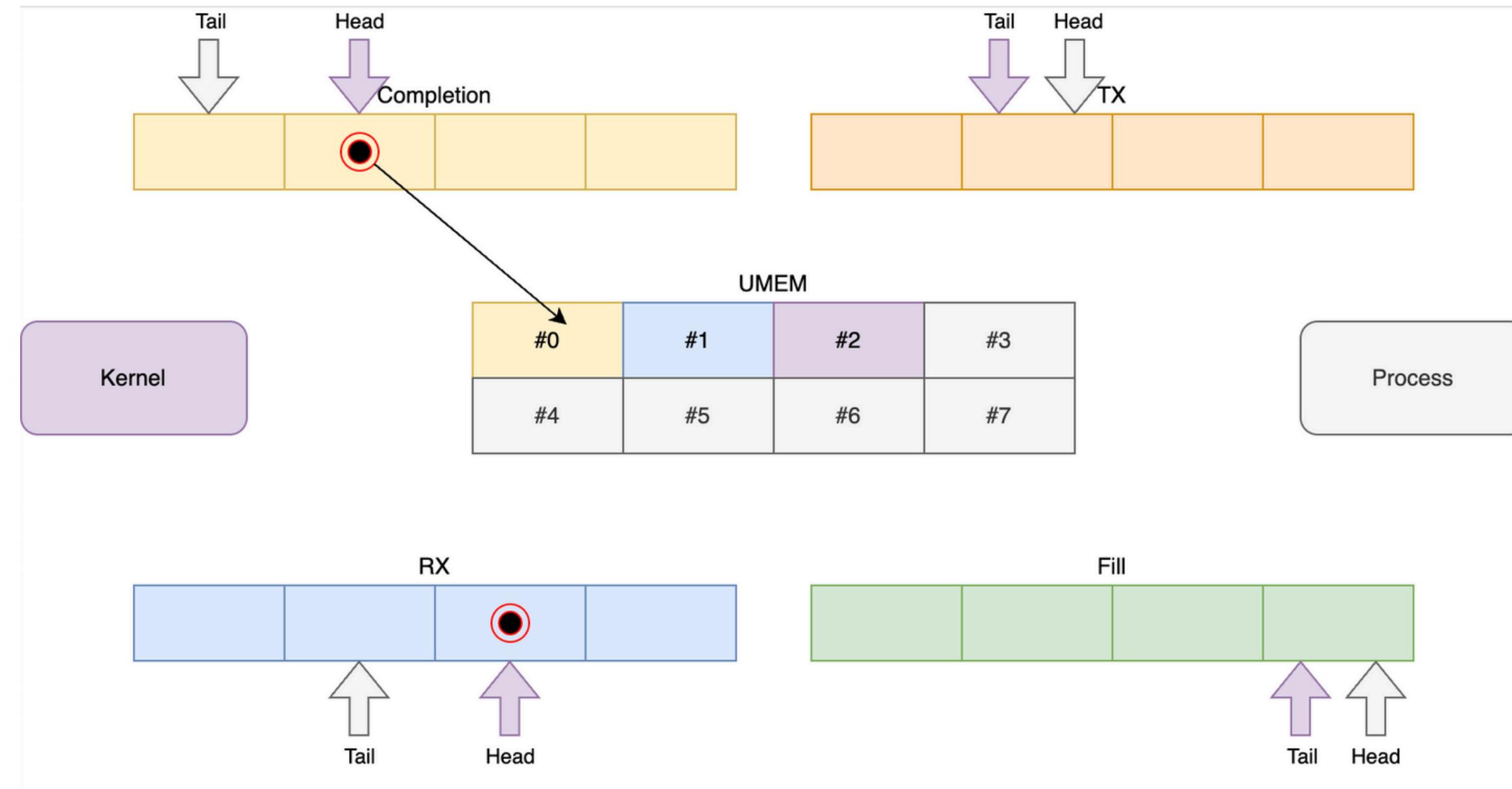
XSK - XDP Socket



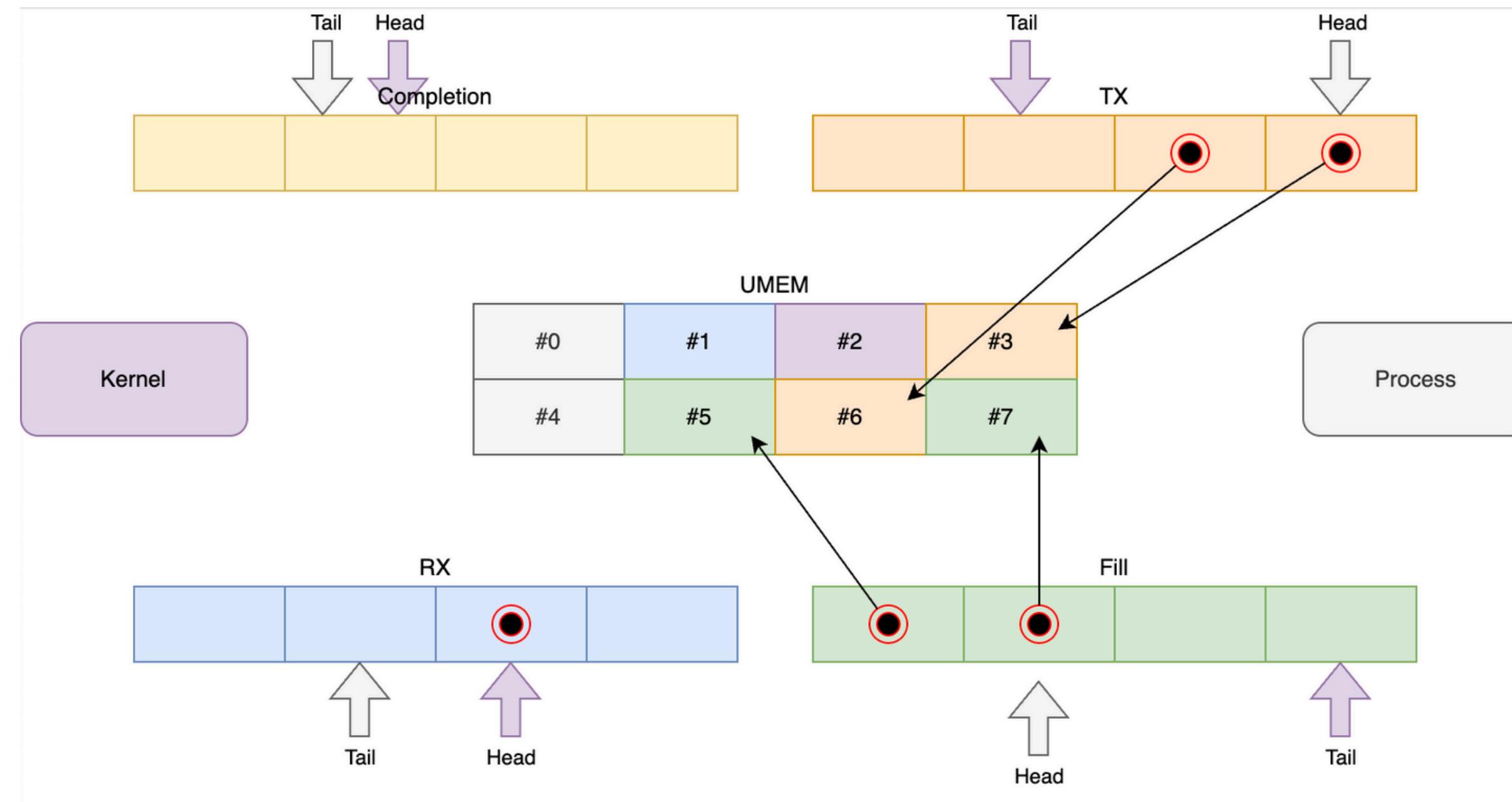
XSK - XDP Socket



XSK - XDP Socket



XSK - XDP Socket



THANK YOU

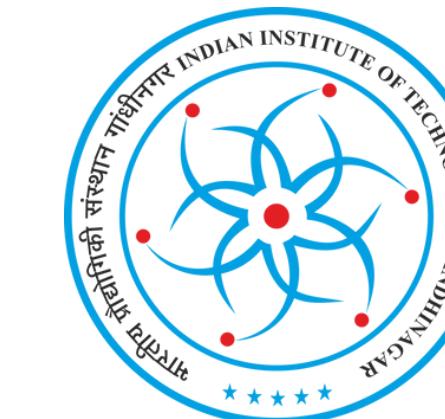
TEAM - 13

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