NETCONF 2023

SYN Proxy at Scale with BPF

Kuniyuki Iwashima

Kernel Development Engineer
Amazon Web Services

SYN Cookie

SYN
SEQ: X, ACK: 0

SYN+ACK
SEQ: C, ACK: X+1

ACK
SEQ: X+1, ACK: C+1

Acknowledge **ISN**

Encode client info into
Initial Sequence Number

Stateless

Decode client info from
ISN (ACK# - 1)

What's encoded in ISN ?

net/ipv4/syncookies.c :

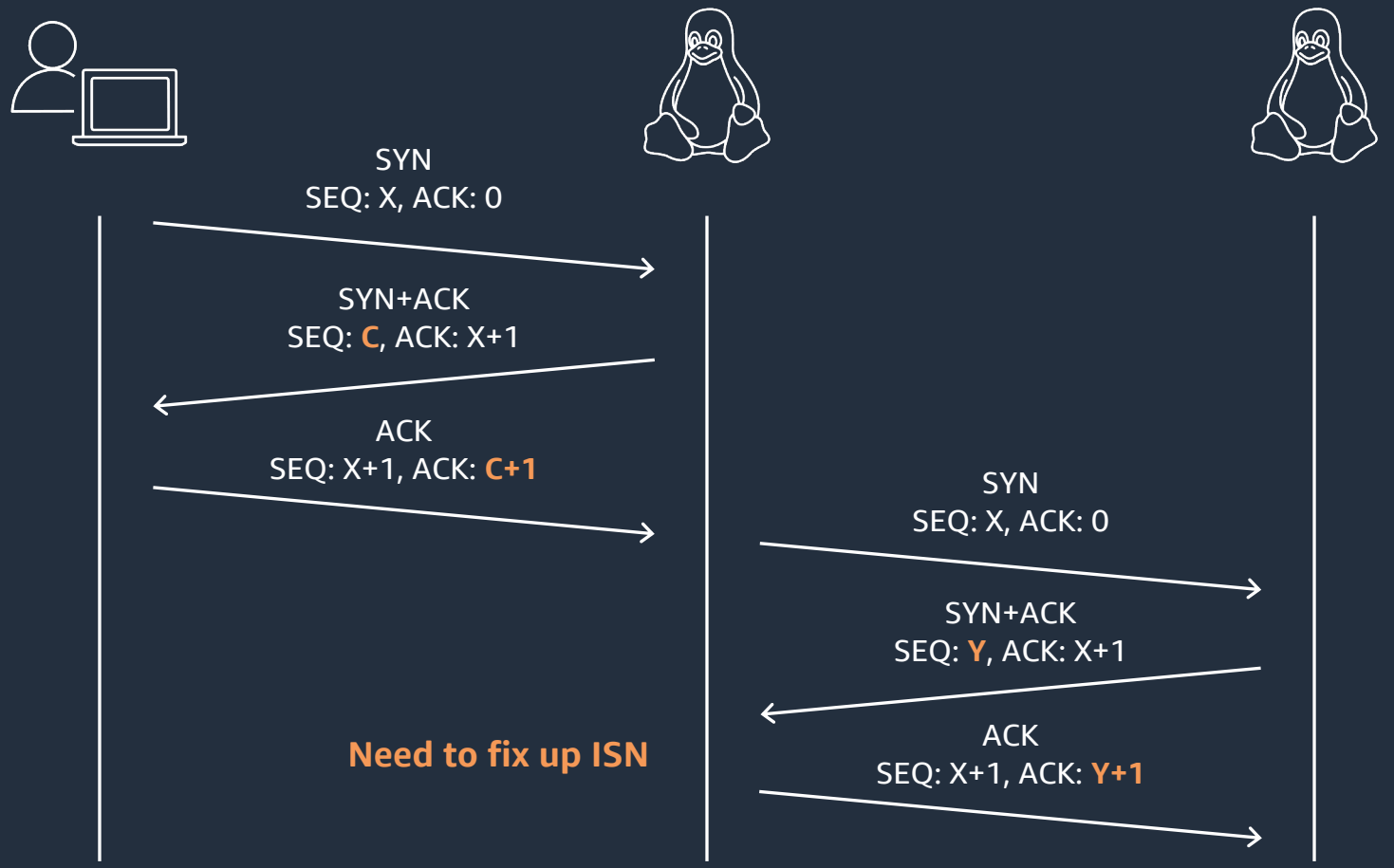
```
static __u32 secure_tcp_syn_cookie(__be32 saddr, __be32 daddr, __be16 sport,
                                   __be16 dport,__u32 sseq, __u32 data)
{
        /* Compute the secure sequence number.
         * The output should be:
         *   HASH(sec1,saddr,sport,daddr,dport,sec1) + sseq + (count * 2^24)
         *      + (HASH(sec2,saddr,sport,daddr,dport,count,sec2) % 2^24).
         * Where sseq is their sequence number and count increases every
         * minute by 1.
         * As an extra hack, we add a small "data" value that encodes the
         * MSS into the second hash value.
         */
        u32 count = tcp_cookie_time();

        return (cookie_hash(saddr, daddr, sport, dport, 0, 0) + sseq + (count << COOKIEBITS) +
                ((cookie_hash(saddr, daddr, sport, dport, count, 1) + data) & COOKIEMASK));
}
```

What's encoded in ISN ?

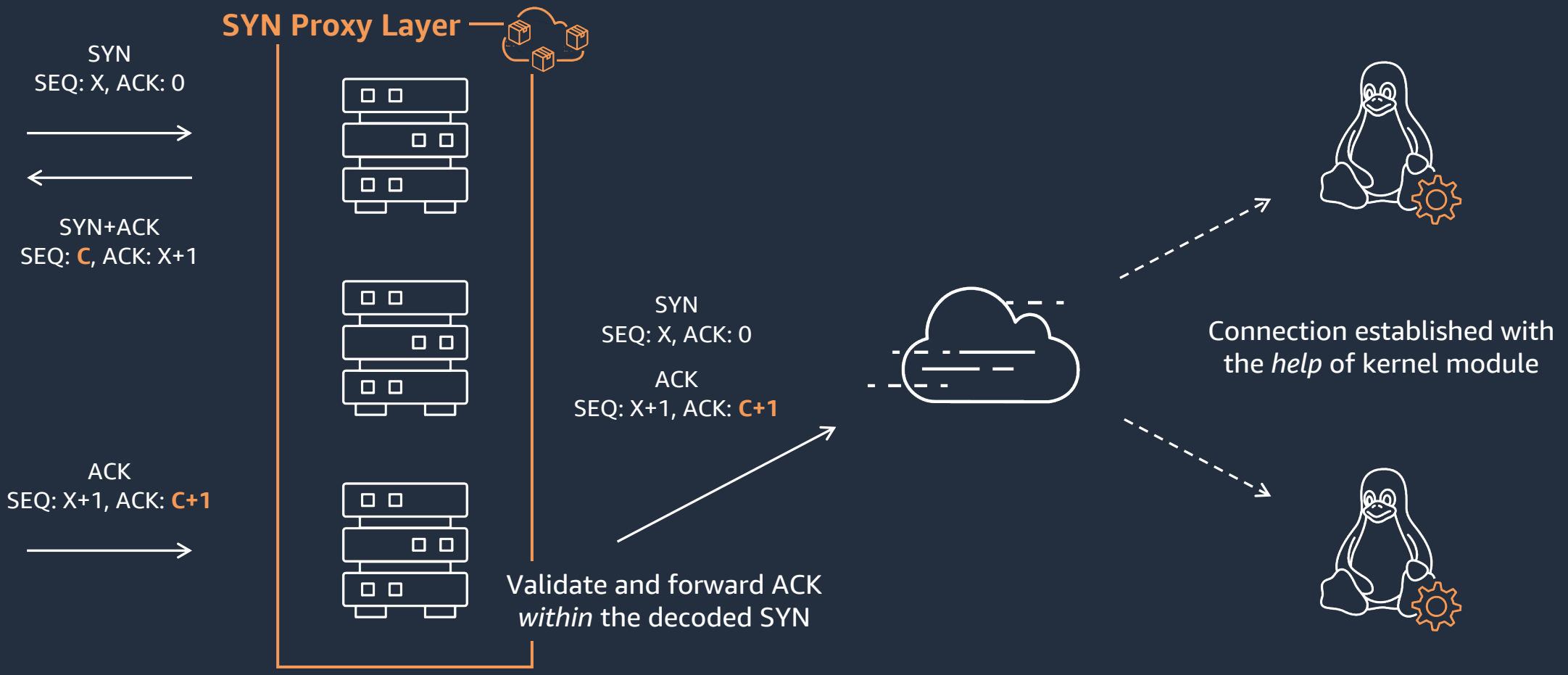
- 4-tuple (src/dst x IP/port)
- Client ISN
- MSS
- SYN Cookie Timestamp
  - Valid for 2 minutes
  - Calculated from **`jiffies`**
- Secret
  - Generated by **`net_get_random_once()`**

Host-specific

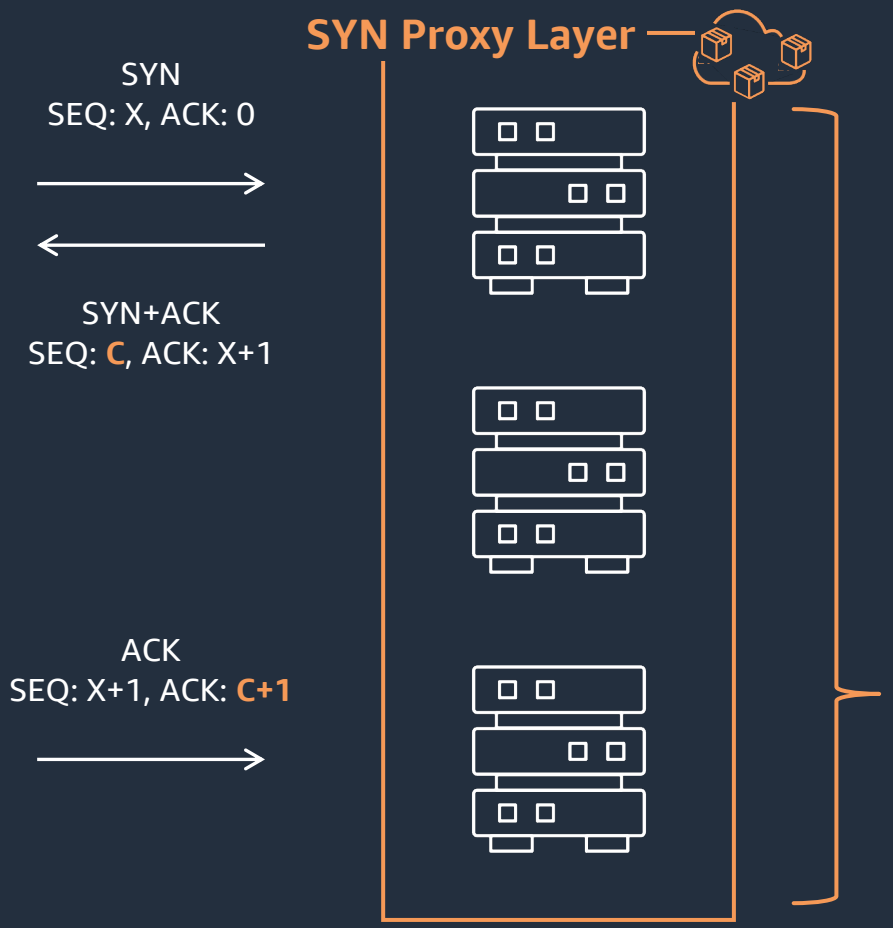
Netfilter SYNPROXY module



SYN Proxy at AWS



SYN Proxy




```
                            1                   2   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 .. 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+  +-+-+
|S| WWW  | MMM | HHHHHHHHHHHHHHHHHHHHHH ..    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+  +-+-+
S: SACK Permitted  W: Window Scale  M: MSS  H: Hash
```

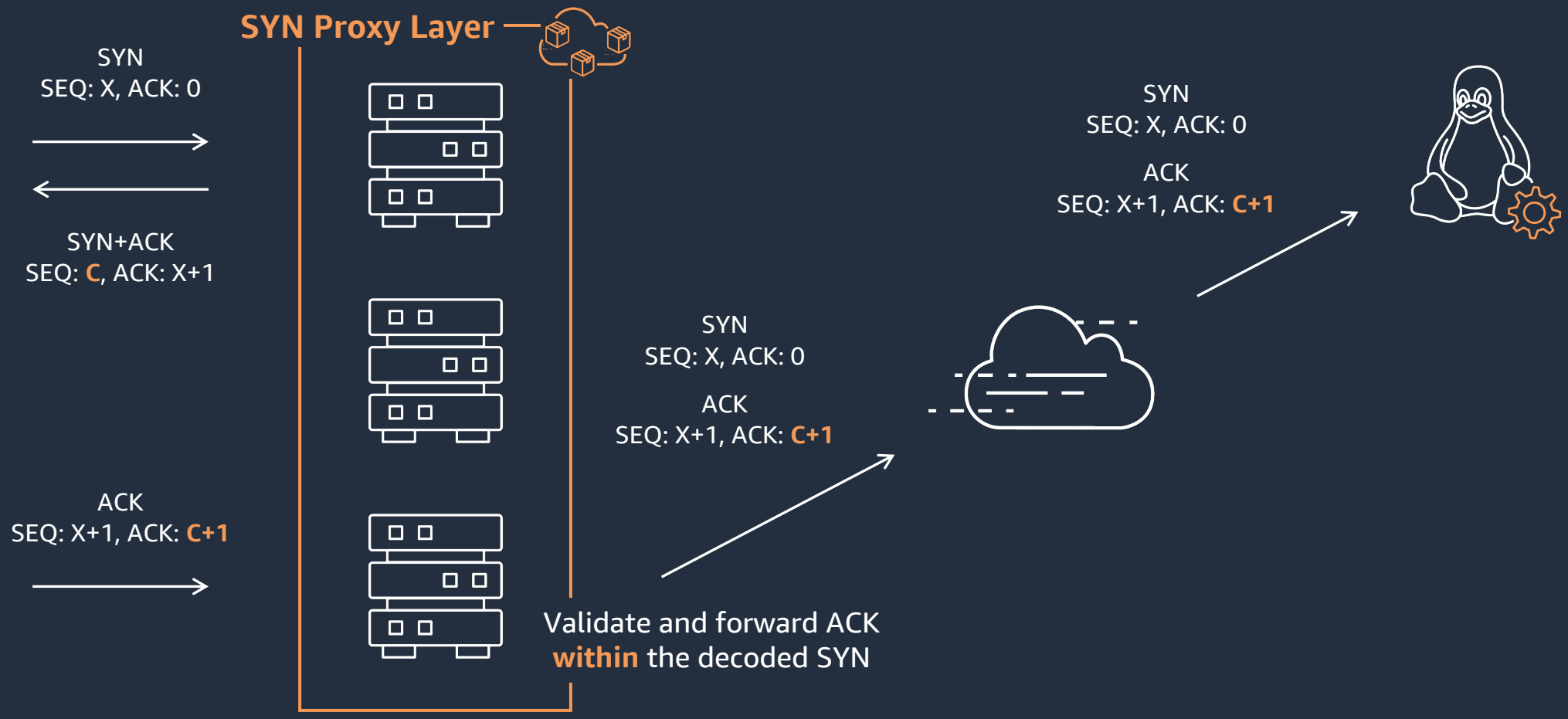
Hash includes:

- 4-tuple
- Client ISN
- Client TCP Options
- **Rolling Salt :** periodically-updated random number

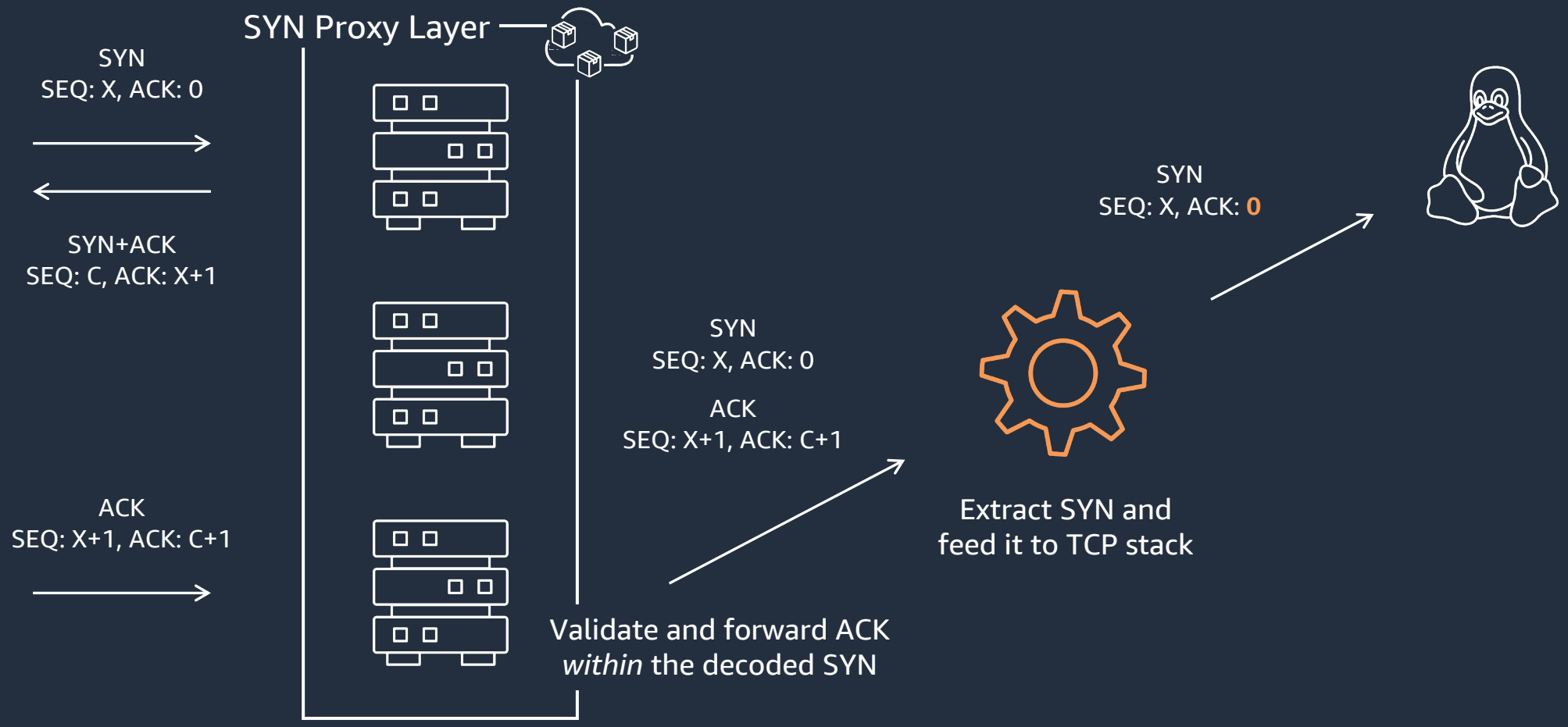
  → All node **share the same secret**

  → Any node can validate SYN cookie **statelessly**

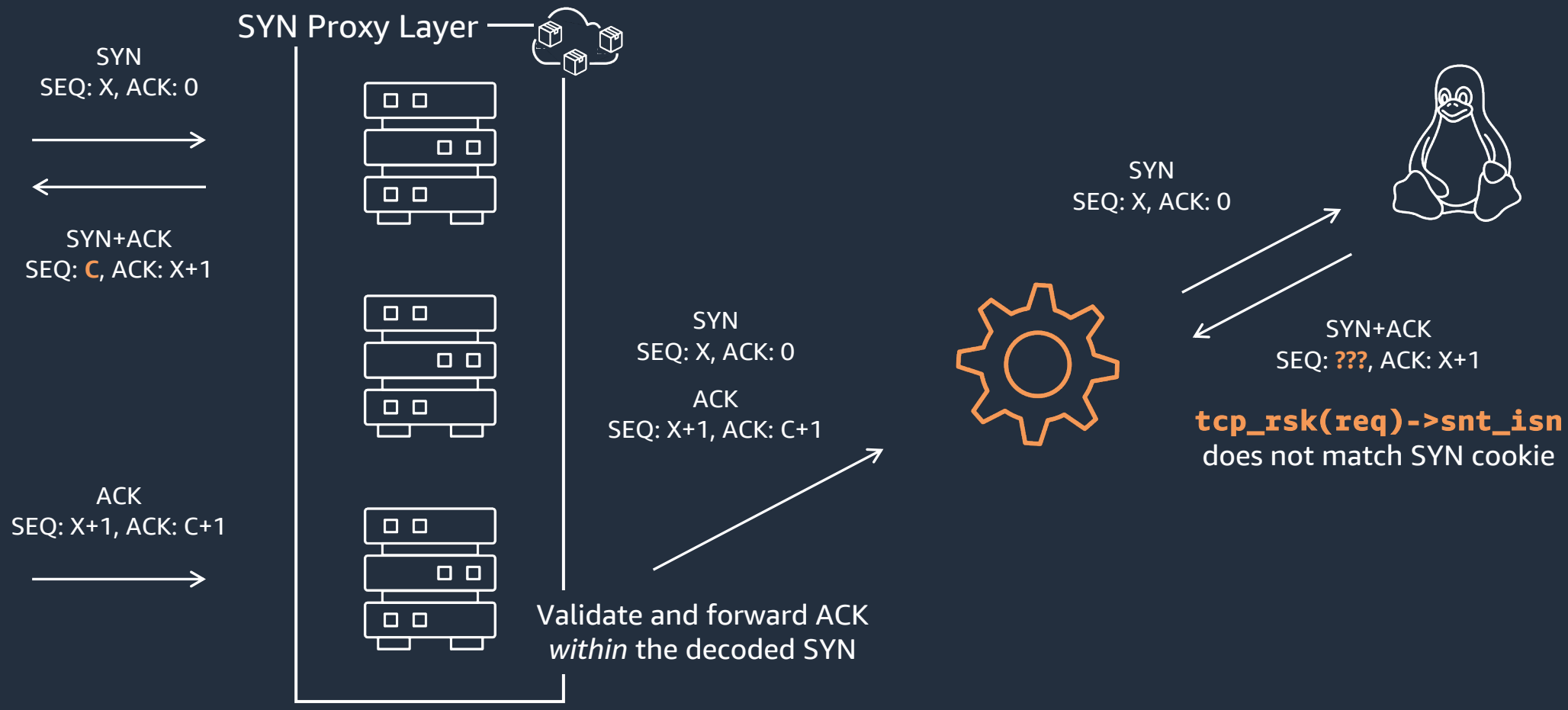
SYN Proxy



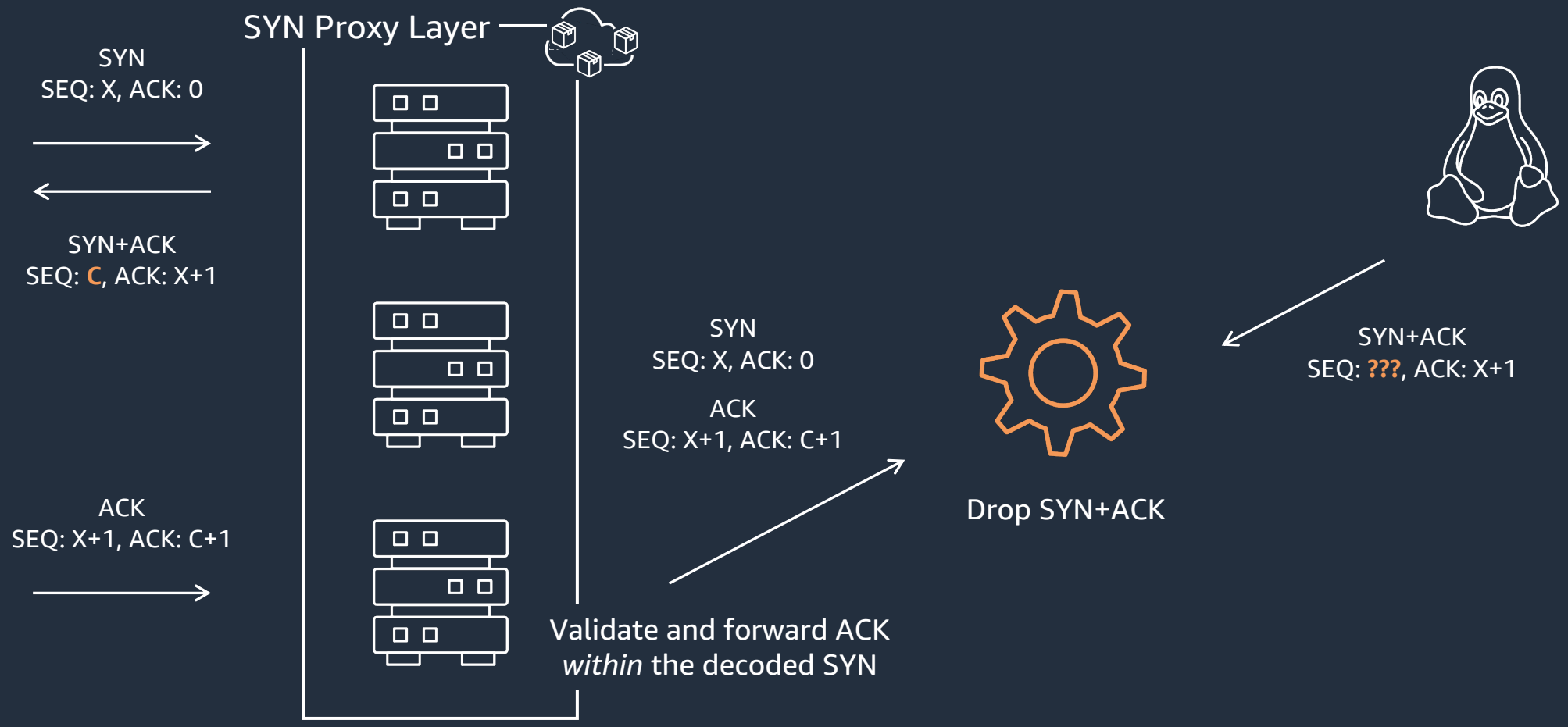
Kernel module



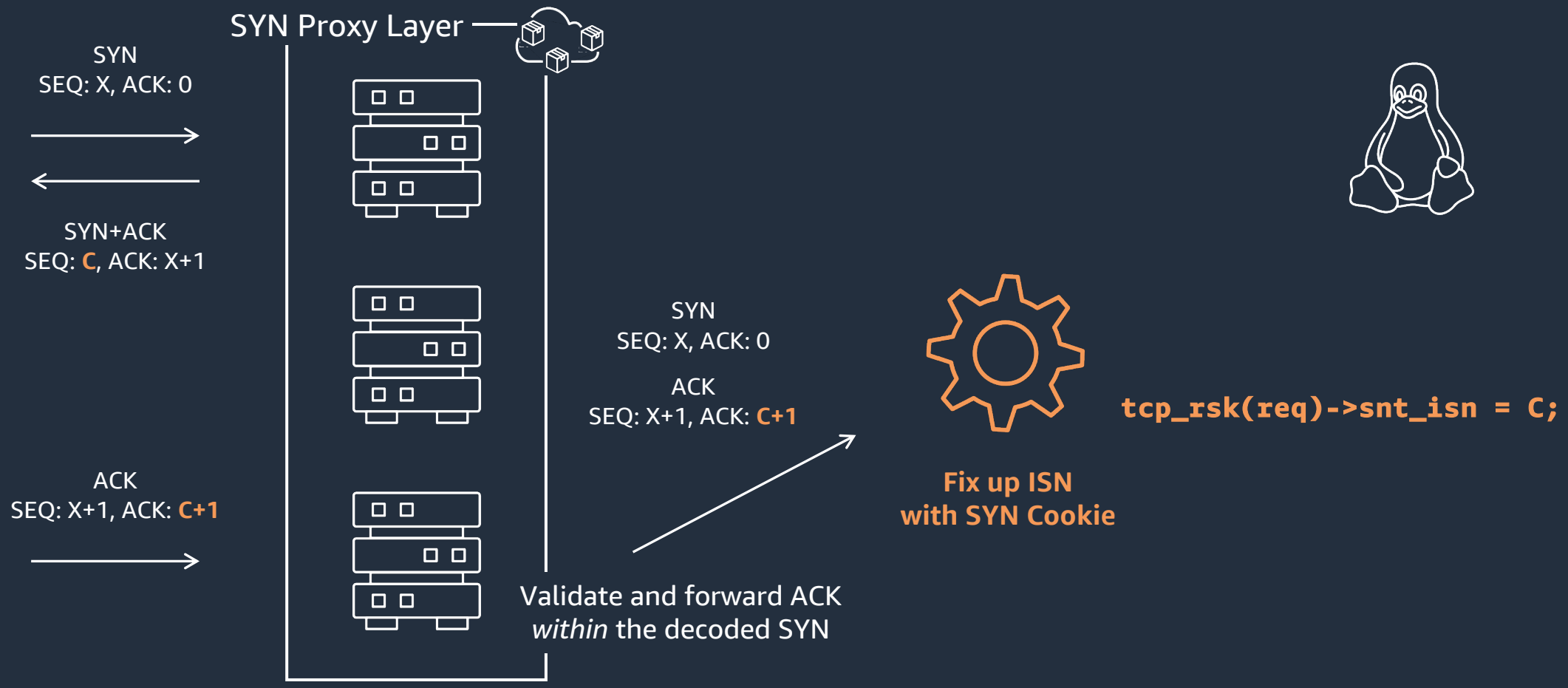
Kernel module



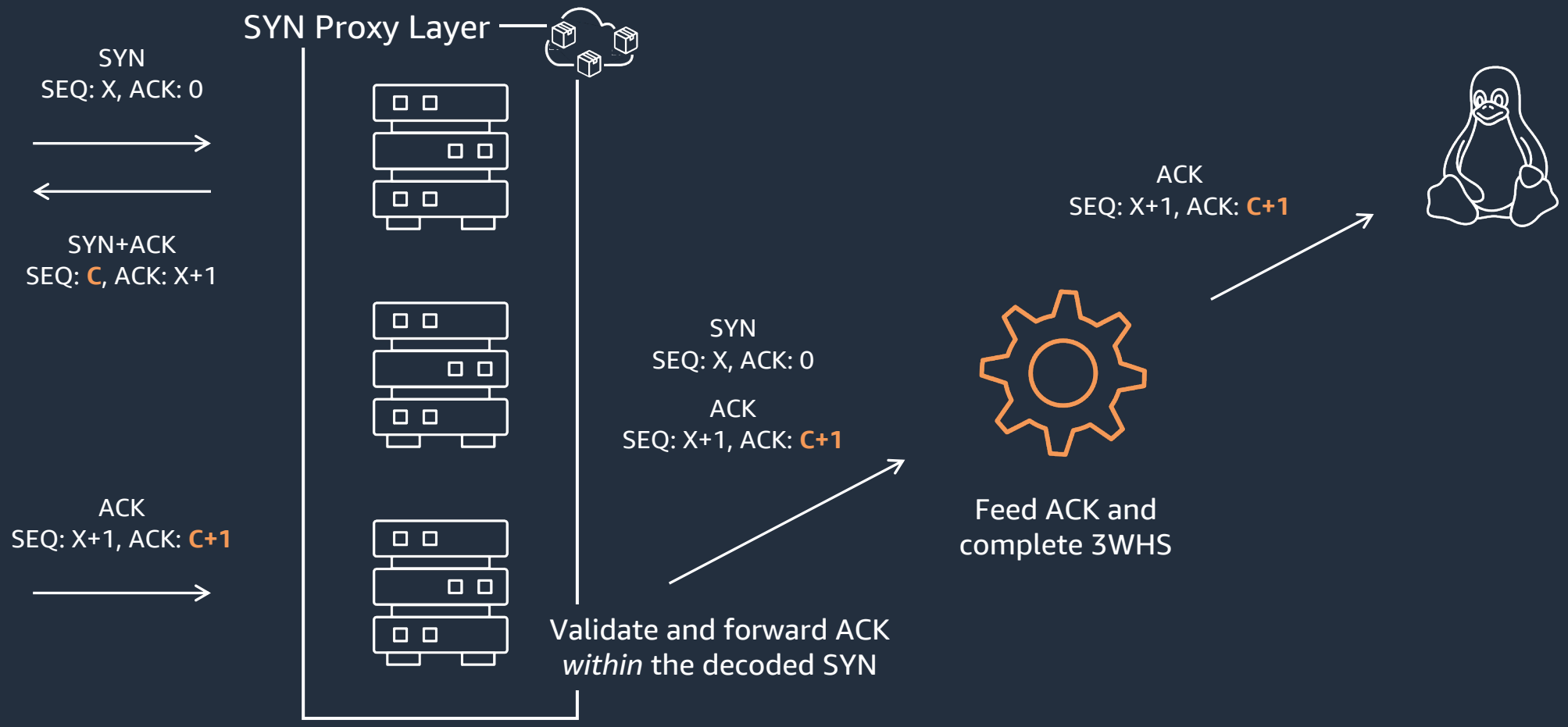
Kernel module



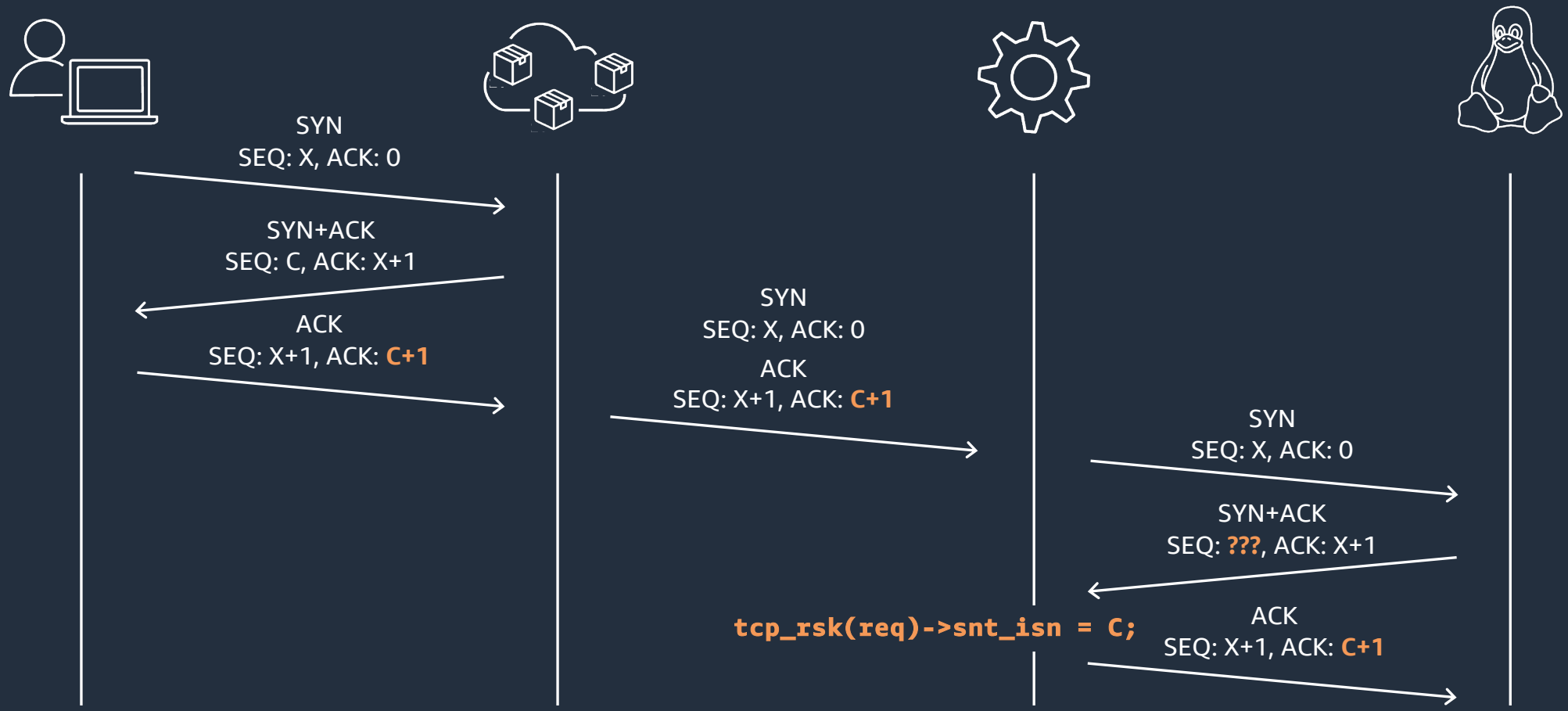
Kernel module



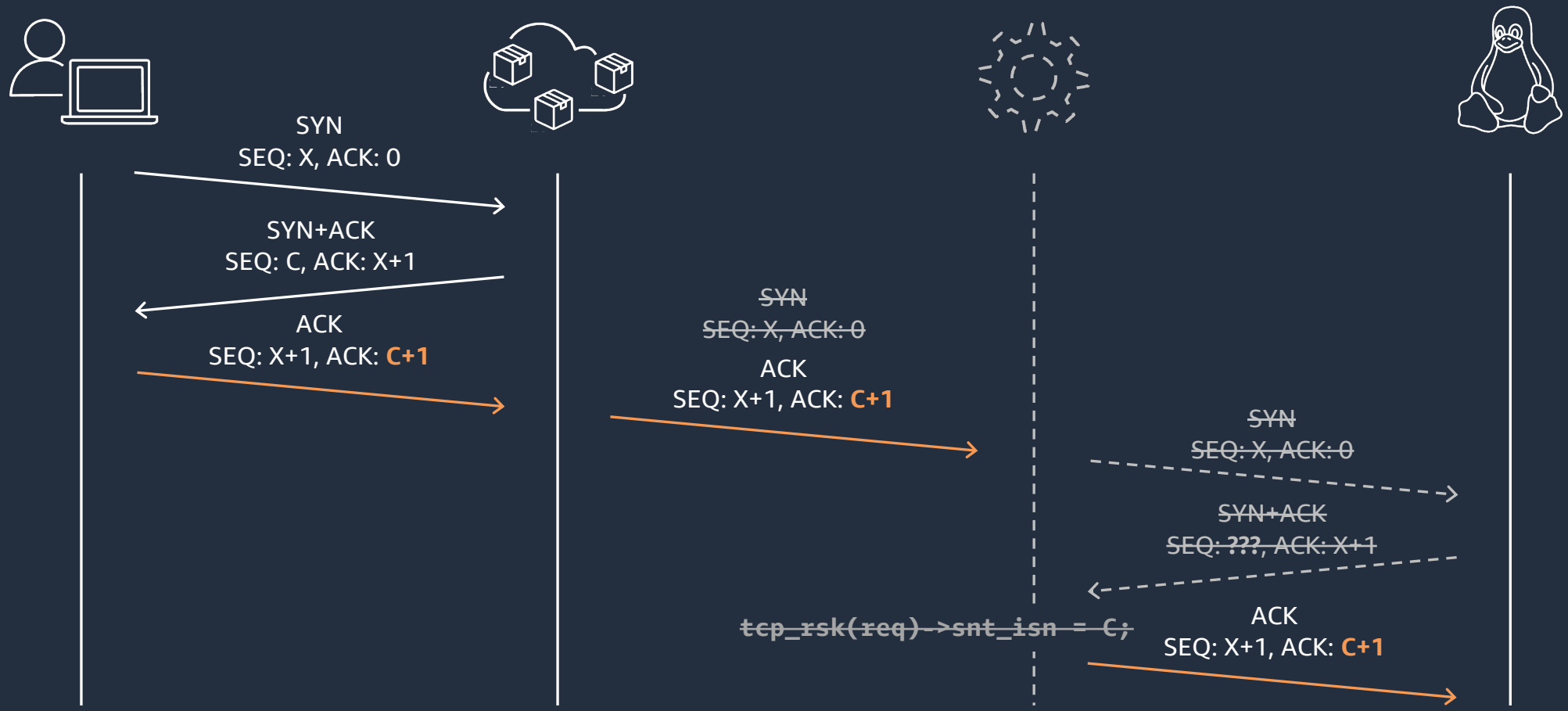
Kernel module



Summary



If we could validate custom SYN Cookie



New SOCK_OPS hooks

SYN
SEQ: X, ACK: 0

SYN+ACK
SEQ: C, ACK: X+1

BPF_SOCK_OPS_GEN_SYNCOOKIE_CB
Encode MSS into ISN

Acknowledge ISN

ACK
SEQ: X+1, ACK: C+1

BPF_SOCK_OPS_CHECK_SYNCOOKIE_CB
Decode MSS from ISN

Prototype

https://github.com/q2ven/linux/tree/bpf_syn_cookie_validation

How to invoke ?

```
#include <linux/bpf.h>
#include <bpf/bpf_helpers.h>

SEC("sockops")
int syncookie(struct bpf_sock_ops *skops)
{
        switch (skops->op) {
        case BPF_SOCK_OPS_TCP_LISTEN_CB:           /* Enable SYN Cookie hooks */
                bpf_sock_ops_cb_flags_set(skops, BPF_SOCK_OPS_SYNCOOKIE_CB_FLAG);
                break;
        case BPF_SOCK_OPS_GEN_SYNCOOKIE_CB:        /* Generate ISN and TS Value. */
                ...
                break;
        case BPF_SOCK_OPS_CHECK_SYNCOOKIE_CB:      /* Validate ISN and TS, and set MSS
                ...                                 * (and WScale, SACK, ECN, etc). */
                break;
        }

        return 1;
}
```

How to invoke ?

```
#include <linux/bpf.h>
#include <bpf/bpf_helpers.h>

SEC("sockops")
int syncookie(struct bpf_sock_ops *skops)
{
        switch (skops->op) {
        case BPF_SOCK_OPS_TCP_LISTEN_CB:            /* Enable SYN Cookie hooks */
                bpf_sock_ops_cb_flags_set(skops, BPF_SOCK_OPS_GEN_SYNCOOKIE_CB_FLAG);
                break;
        case BPF_SOCK_OPS_GEN_SYNCOOKIE_CB:         /* Generate ISN and TS Value. */
                ...
                break;
        case BPF_SOCK_OPS_CHECK_SYNCOOKIE_CB:       /* Validate ISN and TS, and set MSS
                ...                                  * (and WScale, SACK, ECN, etc). */
                break;
        }

        return 1;
}
```

How to invoke ?

```
#include <linux/bpf.h>
#include <bpf/bpf_helpers.h>

SEC("sockops")
int syncookie(struct bpf_sock_ops *skops)
{
        switch (skops->op) {
        case BPF_SOCK_OPS_TCP_LISTEN_CB:            /* Enable SYN Cookie hooks */
                bpf_sock_ops_cb_flags_set(skops, BPF_SOCK_OPS_GEN_SYNCOOKIE_CB_FLAG);
                break;
        case BPF_SOCK_OPS_GEN_SYNCOOKIE_CB:         /* Generate ISN and TS Value. */
                ...
                break;
        case BPF_SOCK_OPS_CHECK_SYNCOOKIE_CB:       /* Validate ISN and TS, and set MSS
                ...                                  * (and WScale, SACK, ECN, etc). */
                break;
        }

        return 1;
}
```

BPF_SOCK_OPS_GEN_SYNCOOKIE_CB

net/ipv4/tcp_input.c :

```
int tcp_conn_request(struct request_sock_ops *rsk_ops,
...
        if (want_cookie) {
                if (BPF_SOCK_OPS_TEST_FLAG(tp, BPF_SOCK_OPS_GEN_SYNCOOKIE_CB_FLAG)) {
                        if (bpf_skops_cookie_init_sequence(sk, req, skb, &isn))
                                goto drop_and_release;
                } else {
                        /* Non-BPF case */
                        isn = cookie_init_sequence(af_ops, sk, skb, &req->mss);
                }

                if (!tmp_opt.tstamp_ok)
                        inet_rsk(req)->ecn_ok = 0;
        }

        tcp_rsk(req)->snt_isn = isn;
...
```

BPF_SOCK_OPS_GEN_SYNCOOKIE_CB

```
static int bpf_skops_cookie_init_sequence(struct sock *sk, struct request_sock *req,
                                          struct sk_buff *skb, __u32 *isn)
{
        struct bpf_sock_ops_kern sock_ops;

        memset(&sock_ops, 0, offsetof(struct bpf_sock_ops_kern, temp));
        sock_ops.op = BPF_SOCK_OPS_GEN_SYNCOOKIE_CB;
        sock_ops.sk = req_to_sk(req);
        sock_ops.args[0] = req->mss;

        bpf_skops_init_skb(&sock_ops, skb, tcp_hdrlen(skb));

        if (BPF_CGROUP_RUN_PROG_SOCK_OPS_SK(&sock_ops, sk)
                return -EPERM;

        *isn = sock_ops.replylong[0];
        tcp_synq_overflow(sk);

        return 0;
}
```

BPF_SOCK_OPS_CHECK_SYNCOOKIE_CB

net/ipv4/syncookies.c :

```
struct sock *cookie_v4_check(struct sock *sk, struct sk_buff *skb)
...     /* If true, skip validation for ISN and TS */
        bpf_check = BPF_SOCK_OPS_TEST_FLAG(tp, BPF_SOCK_OPS_CHECK_SYNCOOKIE_CB_FLAG);
        if (!bpf_check) {
                if (tcp_synq_no_recent_overflow(sk))
                        goto out;

                mss = __cookie_v4_check(ip_hdr(skb), th, cookie);
                if (mss == 0) {
                        __NET_INC_STATS(sock_net(sk), LINUX_MIB_SYNCOOKIESFAILED);
                        goto out;
                }
                __NET_INC_STATS(sock_net(sk), LINUX_MIB_SYNCOOKIESRECV);
        }

        if (!bpf_check && !cookie_timestamp_decode(sock_net(sk), &tcp_opt))
                goto out;
```

BPF_SOCK_OPS_CHECK_SYNCOOKIE_CB

```
struct sock *cookie_v4_check(struct sock *sk, struct sk_buff *skb)
...
        req = cookie_tcp_reqsk_alloc(&tcp_request_sock_ops, &tcp_request_sock_ipv4_ops, sk, skb);
...
        sk_rcv_saddr_set(req_to_sk(req), ip_hdr(skb)->daddr);
        sk_daddr_set(req_to_sk(req), ip_hdr(skb)->saddr);

        ireq = inet_rsk(req);
        ireq->ir_num = ntohs(th->dest);
        ireq->ir_rmt_port = th->source;

        treq = tcp_rsk(req);
        treq->snt_isn = cookie;

        if (bpf_check) {
                mss = bpf_skops_cookie_check(sk, req, skb); /* Called after reqsk is allocated
                if (!mss)                                    * and IP/port are initialised
                        goto out_free;                       * for consistent interface
        }                                                    */
```

BPF_SOCK_OPS_CHECK_SYNCOOKIE_CB

```
int bpf_skops_cookie_check(struct sock *sk, struct request_sock *req, struct sk_buff *skb)
{
        struct bpf_sock_ops_kern sock_ops;

        memset(&sock_ops, 0, offsetof(struct bpf_sock_ops_kern, temp));
        sock_ops.op = BPF_SOCK_OPS_CHECK_SYNCOOKIE_CB;
        sock_ops.sk = req_to_sk(req);
        sock_ops.args[0] = tcp_rsk(req)->snt_isn;

        bpf_skops_init_skb(&sock_ops, skb, tcp_hdrlen(skb));

        if (BPF_CGROUP_RUN_PROG_SOCK_OPS_SK(&sock_ops, sk))
                goto err;

        if (sock_ops.replylong[0] < 536) /* Min MSS */
                goto err;

        __NET_INC_STATS(sock_net(sk), LINUX_MIB_SYNCOOKIESRECV);
        return sock_ops.replylong[0];
...
```

User Interface

BPF_SOCK_OPS_GEN_SYNCOOKIE_CB

Input :

- bpf_sock_ops.sk : 4-tuple
- bpf_sock_ops.skb : TCP header
- bpf_sock_ops.args[0] : **MSS**

↓ Encode

Output :

- bpf_sock_ops.replylong[0] : **ISN (SYN Cookie)**

↗ Loop back

BPF_SOCK_OPS_CHECK_SYNCOOKIE_CB

Input :

- bpf_sock_ops.sk : 4-tuple
- bpf_sock_ops.skb : TCP header
- bpf_sock_ops.args[0] : **ISN (SYN Cookie)**

↓ Decode

Output :

- bpf_sock_ops.replylong[0] : **MSS**

Timestamps Option

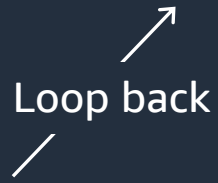
BPF_SOCK_OPS_GEN_SYNCOOKIE_CB

Input :

- bpf_sock_ops.sk : 4-tuple
- bpf_sock_ops.skb : TCP header
- bpf_sock_ops.args[0] : MSS

Output :

- bpf_sock_ops.replylong[0] : ISN (SYN Cookie)
- bpf_sock_ops.**replylong[1]** : **TS** ?
    - Read-only field for now



BPF_SOCK_OPS_CHECK_SYNCOOKIE_CB

Input :

- bpf_sock_ops.sk : 4-tuple
- bpf_sock_ops.skb : TCP header
- bpf_sock_ops.args[0] : ISN (SYN Cookie)
- bpf_sock_ops.**args[1]** : **TS** ?

Output :

- bpf_sock_ops.replylong[0] : MSS

Timestamp is not sent as is

net/ipv4/tcp_output.c :

```
static unsigned int tcp_synack_options(const struct sock *sk,
...
        if (likely(ireq->tstamp_ok)) {
                opts->options |= OPTION_TS;
                opts->tsval = tcp_skb_timestamp(skb) + tcp_rsk(req)->ts_off;
                opts->tsecr = READ_ONCE(req->ts_recent);
                remaining -= TCPOLEN_TSTAMP_ALIGNED;
        }
...
}

struct sk_buff *tcp_make_synack(const struct sock *sk, struct dst_entry *dst,
...
        if (unlikely(synack_type == TCP_SYNACK_COOKIE && ireq->tstamp_ok))
                skb_set_delivery_time(skb, cookie_init_timestamp(req, now), true);

         tcp_options_write(th, NULL, &opts);
```

Timestamp is not sent as is

net/ipv4/tcp_output.c :

```
int tcp_conn_request(struct request_sock_ops *rsk_ops,
...
        if (tmp_opt.tstamp_ok)
                tcp_rsk(req)->ts_off = af_ops->init_ts_off(net, skb);
...
}
```

net/core/secure_seq.c :

```
u32 secure_tcp_ts_off(const struct net *net, __be32 saddr, __be32 daddr)
{
...
        ts_secret_init(); /* Host-specific..., kill scalability !! */
        return siphash_2u32((__force u32)saddr, (__force u32)daddr, &ts_secret);
}
```

Timestamp is not sent as is

net/ipv4/syncookies :

```
struct sock *cookie_v4_check(struct sock *sk, struct sk_buff *skb)
...
        if (tcp_opt.saw_tstamp && tcp_opt.rcv_tsecr) {              /* Unnecessary for
                tsoff = secure_tcp_ts_off(sock_net(sk),              * BPF Cookie Validation
                                          ip_hdr(skb)->daddr,        */
                                          ip_hdr(skb)->saddr);
                tcp_opt.rcv_tsecr -= tsoff;
        }

        if (!cookie_timestamp_decode(sock_net(sk), &tcp_opt))
                goto out;

...
```

No Timestamp Adjustment during 3WHS

- **`BPF_SOCK_OPS_GEN_SYNCOOKIE_CB`**
  - Need to clear `tcp_rsk(req)->ts_off` after invoking
- **`BPF_SOCK_OPS_CHECK_SYNCOOKIE_CB`**
  - Must not adjust `tcp_opt.rcv_tsecr` before invoking

Timestamp Adjustment after 3WHS

- SYN+ACK : **Any value**
- After 3WHS : `tp->tsoffset`

Gap

`tcp_ns_to_ts(tcp_clock_ns())`

Timestamp Adjustment after 3WHS

Need to reset `tp->tsoffset` with

```
tcp_opt.rcv_tsecr - tcp_ns_to_ts(tcp_clock_ns())
```

to fill the gap

- SYN+ACK : Any value
- After 3WHS : `tcp_ns_to_ts(tcp_clock_ns())`
  `+ tp->tsoffset`
  = Initial TS + delta

Other SYN info

BPF_SOCK_OPS_GEN_SYNCOOKIE_CB

Input :

- bpf_sock_ops.sk : 4-tuple
- bpf_sock_ops.skb : TCP header
- bpf_sock_ops.args[0] : MSS
- bpf_sock_ops.**args[1]** : **sack_ok, etc** ?

↓ Encode

Output :

- bpf_sock_ops.replylong[0] : ISN (SYN Cookie)
- bpf_sock_ops.replylong[1] : TS ?

Loop back ↗

BPF_SOCK_OPS_CHECK_SYNCOOKIE_CB

Input :

- bpf_sock_ops.sk : 4-tuple
- bpf_sock_ops.skb : TCP header
- bpf_sock_ops.args[0] : ISN (SYN Cookie)
- bpf_sock_ops.args[1] : TS ?

↓ Decode

Output :

- bpf_sock_ops.replylong[0] : MSS
- bpf_sock_ops.**replylong[1]** : **sack_ok , etc** ?

How to format other SYN info ?

net/ipv4/syncookies :

```
/* TCP Timestamp: 6 lowest bits of timestamp sent in the cookie SYN-ACK
 * stores TCP options:
 *
 * MSB                                LSB
 * | 31 ...   6 |  5  |  4   | 3 2 1 0 |
 * |  Timestamp | ECN | SACK | WScale  |
 *
 * When we receive a valid cookie-ACK, we look at the echoed tsval (if
 * any) to figure out which TCP options we should use for the rebuilt
 * connection.
 *
 * A WScale setting of '0xf' (which is an invalid scaling value)
 * means that original syn did not include the TCP window scaling option.
 */
#define TS_OPT_WSCALE_MASK 0xf
#define TS_OPT_SACK BIT(4)
#define TS_OPT_ECN BIT(5)
```

How to format other SYN info ?

include/uapi/linux/bpf.h :

```
/* arg[1] value for BPF_SOCK_OPS_GEN_SYNCOOKIE_CB and
 * replylong[1] for BPF_SOCK_OPS_CHECK_SYNCOOKIE_CB.
 *
 * MSB                                   LSB
 * | 31 ... | 6  | 5   | 4    | 3 2 1 0 |
 * |    ... | TS | ECN | SACK | WScale  |
 */
enum {
        /* 0xf is invalid thus means that SYN did not have WScale. */
        BPF_SYNCOOKIE_WSCALE_MASK         = (1 << 4) - 1,

        BPF_SYNCOOKIE_SACK                = (1 << 4),
        BPF_SYNCOOKIE_ECN                 = (1 << 5),

        /* Only available for BPF_SOCK_OPS_GEN_SYNCOOKIE_CB to check if SYN has TS */
        BPF_SYNCOOKIE_TS                  = (1 << 6),
};
```

User Interface

BPF_SOCK_OPS_GEN_SYNCOOKIE_CB

Input :

- bpf_sock_ops.sk : 4-tuple
- bpf_sock_ops.skb : TCP header
- bpf_sock_ops.args[0] : MSS
- bpf_sock_ops.args[1] : sack_ok, etc

↓ Encode

Loop back ↗

Output :

- bpf_sock_ops.replylong[0] : ISN (SYN Cookie)
- bpf_sock_ops.replylong[1] : TS

BPF_SOCK_OPS_CHECK_SYNCOOKIE_CB

Input :

- bpf_sock_ops.sk : 4-tuple
- bpf_sock_ops.skb : TCP header
- bpf_sock_ops.args[0] : ISN (SYN Cookie)
- bpf_sock_ops.args[1] : TS

↓ Decode

Output :

- bpf_sock_ops.replylong[0] : MSS
- bpf_sock_ops.replylong[1] : sack_ok , etc

Sample

```
        switch (skops->op) {
...
        case BPF_SOCK_OPS_GEN_SYNCOOKIE_CB:
                skops->replylong[0] = skops->args[0];
                skops->replylong[1] = skops->args[1];
                break;
        case BPF_SOCK_OPS_CHECK_SYNCOOKIE_CB:
                skops->replylong[0] = skops->args[0];
                skops->replylong[1] = skops->args[1];
                break;
        }
```

```
Flags [S], seq 4191364364, win 33280, options [mss 65476,sackOK,TS val 396321828 ecr 0,nop,wscale 7], length 0
Flags [S.], seq 65476, ack 4191364365, win 33280, options [mss 65476,sackOK,TS val 87 ecr 396321828,nop,wscale 7]...
Flags [.], ack 1, win 260, options [nop,nop,TS val 396321834 ecr 87], length 0
Flags [F.], seq 1, ack 1, win 260, options [nop,nop,TS val 122 ecr 396321834], length 0
Flags [F.], seq 1, ack 2, win 260, options [nop,nop,TS val 396321870 ecr 122], length 0
Flags [.], ack 2, win 260, options [nop,nop,TS val 123 ecr 396321870], length 0
```

NETCONF 2023

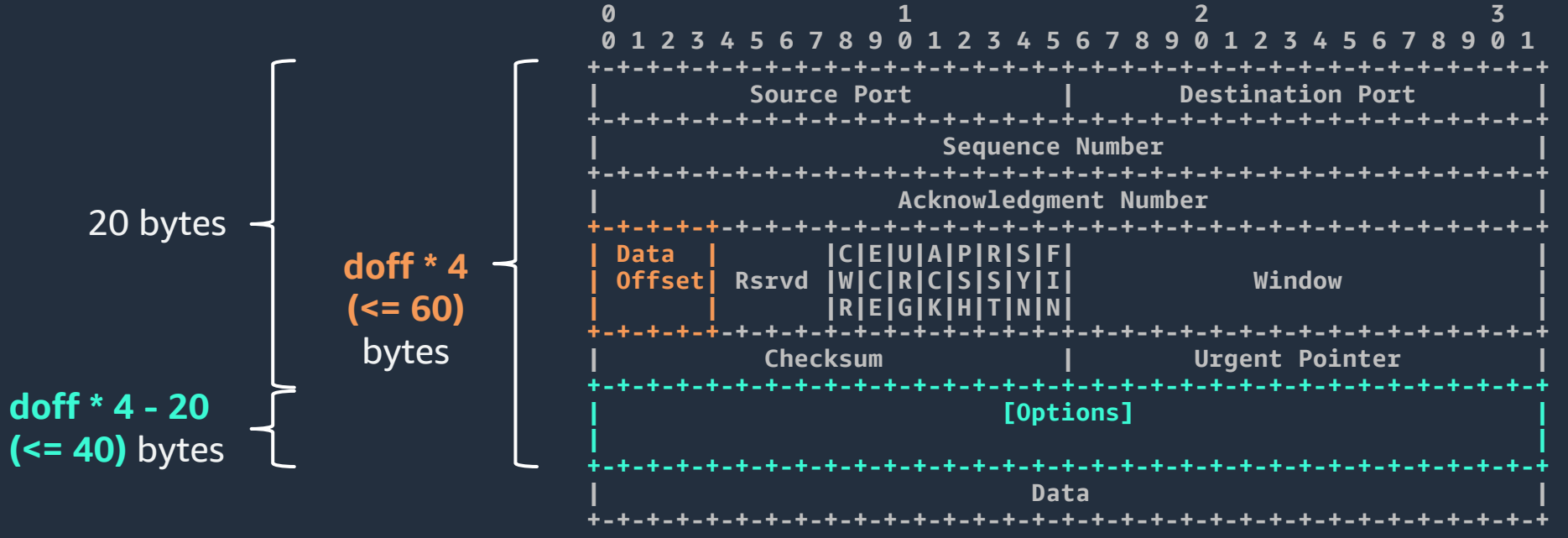
TCP Extended Data Offset

Kuniyuki Iwashima

Kernel Development Engineer
Amazon Web Services

Data Offset

- Number of 32-bit words in the TCP header
- 4-bit field
  - 5 <= doff <= 15



Option Length

In SYN segments

- MSS : 4 bytes
- SACK Permitted : 2 bytes
- TS : 10 bytes
- WScale : 3 bytes
- MPTCP
  - MP_CAPABLE : 4 or 12 bytes
  - MP_JOIN : 12 or 16 bytes
- AO : 16 bytes
- MD5 : 18 bytes

etc

In non-SYN segments

- SACK : 10 + 8n bytes (n < 3)
- TS : 10 bytes
- MPTCP
  - MP_CAPABLE : 20, 22 or 24 bytes
  - MP_JOIN : 24 bytes
  - DSS : 4 ~ 28 bytes

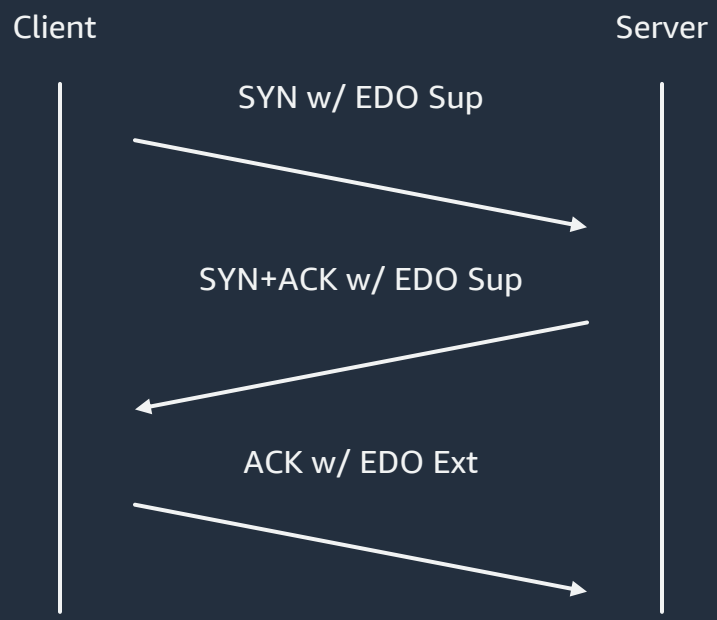
  etc
- AO : 16 bytes
- MD5 : 18 bytes

etc

Extended Data Offset (EDO) Option

Increase the option space in **non-SYN** segment

- EDO Supported : Negotiated in SYN and SYN+ACK
- EDO Extension : Override Data Offset in non-SYN segment

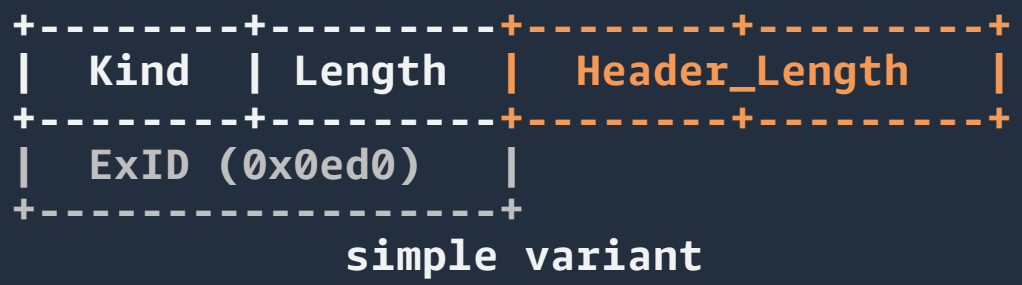


EDO Supported

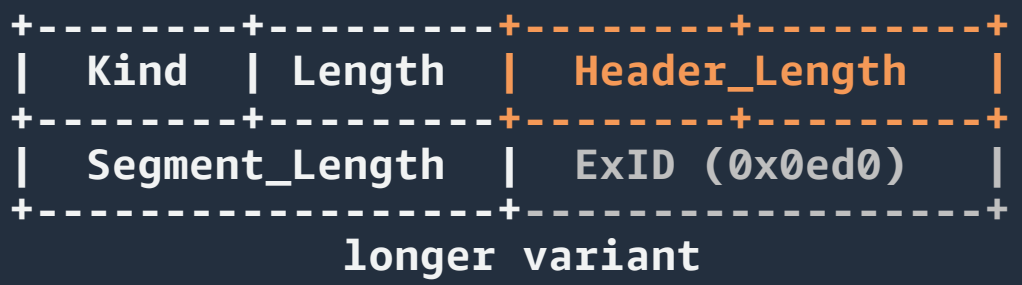
- Exchanged in SYN and SYN+ACK
- Indicate EDO capability

EDO Extension

- **Must be included in all segments** except for RST
- **Override Data Offset** by Header_Length
  - Data Offset <= Header_Length <= IP Payload Length
- Detect merge/split by Segment_Length
  - Segment Length == IP Payload Length

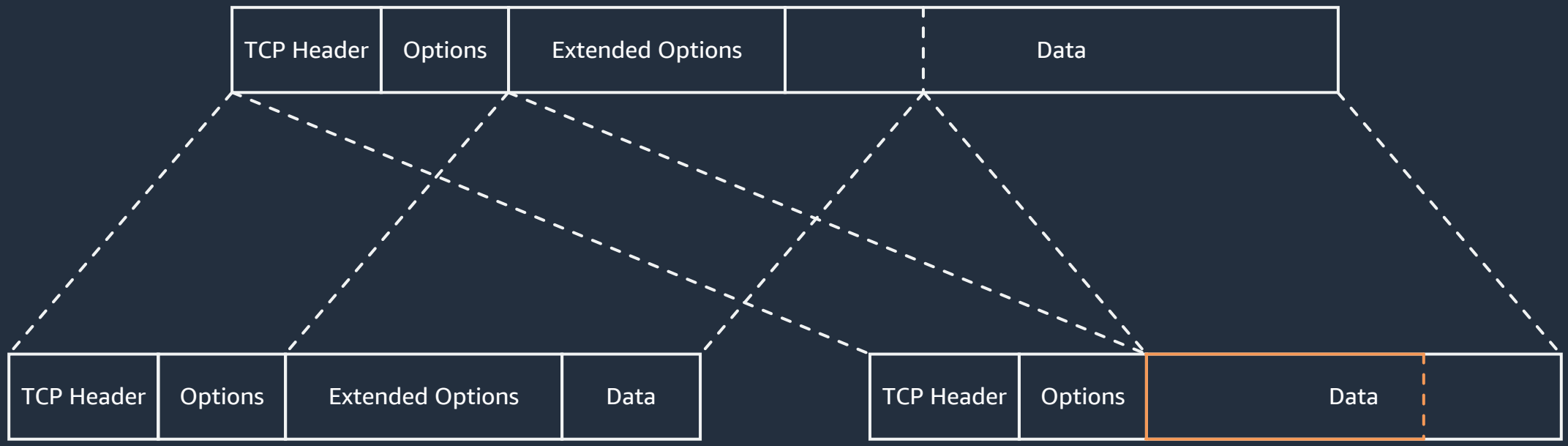


simple variant



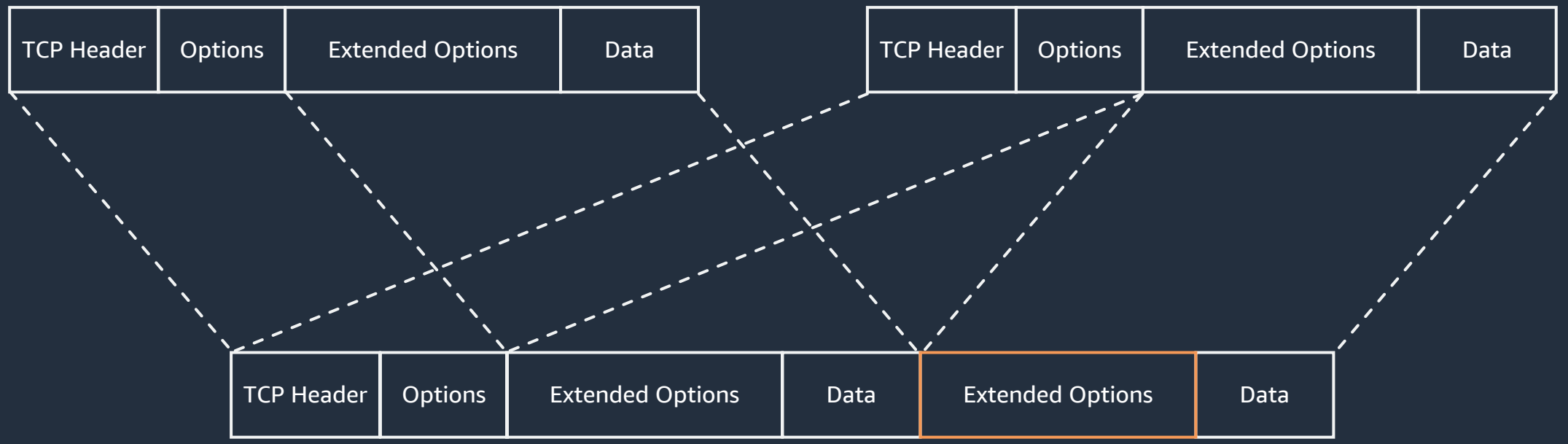
longer variant

If EDO packet is split



Partial data is parsed as Extended Options

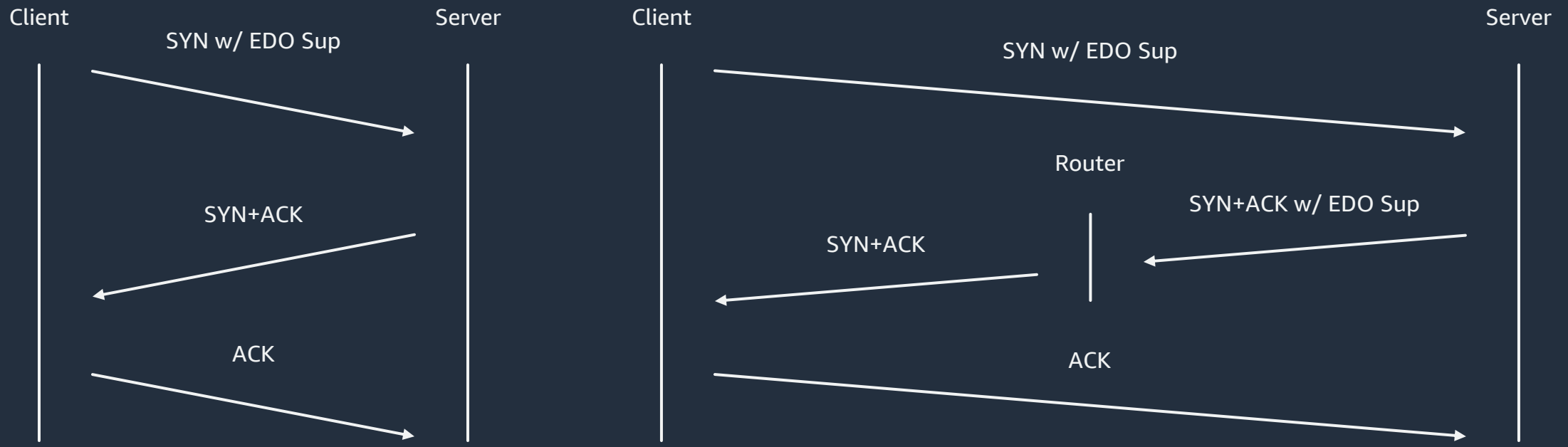
If EDO packet is merged



Extended Options are ACKed

Zero-cost fallback

- Client falls back if EDO Supported is not in SYN+ACK
- Server falls back if EDO Extension is not in ACK



Prototypes

- Linux : https://github.com/q2ven/linux/tree/edo
- packetdrill : https://github.com/q2ven/packetdrill/tree/edo
- tcpdump : https://github.com/nsdyoshi/tcpdump/tree/tcp-edo-option

uAPI

- `sysctl -w net.ipv4.tcp_ext_data_offset=$val`
- `setsockopt(SOL_TCP, TCP_EXT_DATA_OFFSET, val,...)`

- 0 : Disabled
- 1 : Use simple variant (Header_Length)
- 2 : Use longer variant (Header_Length & Segment_Length)

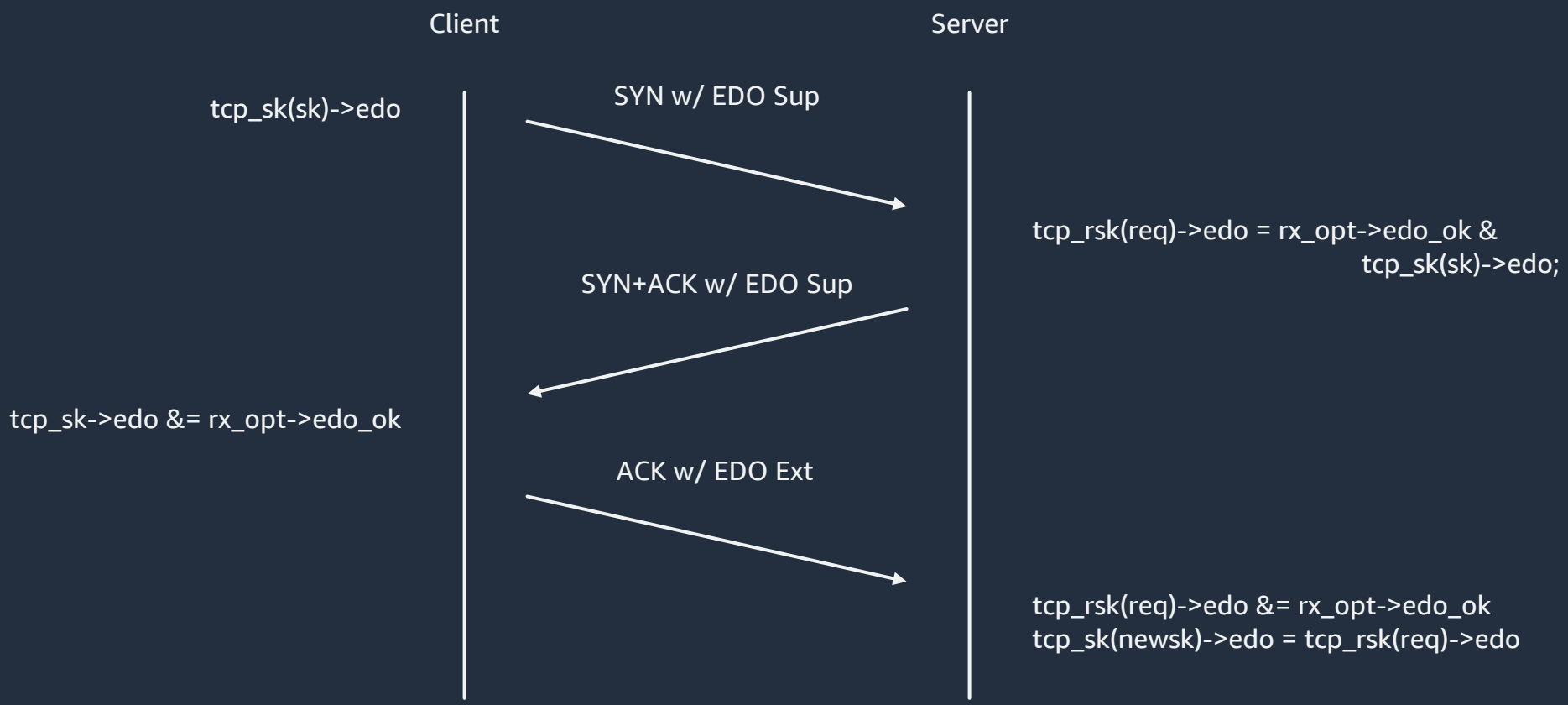
EDO-related field

```
enum {
        TCP_EDO_OFF,
        TCP_EDO_HDR,
        TCP_EDO_HDR_SEG,
};


void tcp_init_sock(struct sock *sk)
{
...
        switch (READ_ONCE(sock_net(sk)->ipv4.sysctl_tcp_ext_data_offset)) {
        case TCP_EDO_HDR_SEG:
                tp->edo_seg = 1;
                fallthrough;
        case TCP_EDO_HDR:
                tp->edo = 1;
                break;
        }
...
}
```

EDO-related field

- `tcp_sk(sk)->{edo,edo_seg}`
- `tcp_rsk(req)->{edo,edo_seg}`
- `inet_twsk(sk)->{tw_edo,tw_edo_seg}`

EDO Negotiation



Send EDO Extension

net/ipv4/tcp_output.c :

```
static unsigned int tcp_established_options(struct sock *sk, struct sk_buff *skb, ...
{
        unsigned int size = 0, limit = MAX_TCP_OPTION_SPACE;
...
        if (tp->edo) {
                opts->options |= OPTION_EDO_EXT_HDR;
                if (tp->edo_seg)
                        opts->options |= OPTION_EDO_EXT_SEG;

                size += TCPOLEN_EXP_EDO_EXT_ALIGNED;
                limit = tp->mss_cache;
        }

        if (sk_is_mptcp(sk)) {
                const unsigned int remaining = limit - size;
                ...
        }
```

Send EDO Extension

net/ipv4/tcp_output.c :

```
static int __tcp_transmit_skb(struct sock *sk, struct sk_buff *skb, ...
{
...
        } else {
                tcp_options_size = tcp_established_options(sk, skb, &opts, &md5);
                if (unlikely(tcp_options_size > MAX_TCP_OPTION_SPACE) &&
                    pskb_expand_head(skb, tcp_options_size - MAX_TCP_OPTION_SPACE,
                                     0, GFP_ATOMIC))
                        return -ENOBUFS;

...
        }
...
        if (tcp_header_size < MAX_TCP_OPTION_SPACE + sizeof(struct tcphdr))
                *(((__be16 *)th) + 6) = htons(((tcp_header_size >> 2) << 12) | tcb->tcp_flags);
        else
                *(((__be16 *)th) + 6) = htons((15 << 12) | tcb->tcp_flags);
```

Parse EDO Extension

net/ipv4/tcp_input.c :

```
int tcp_parse_options(..., bool parse_edo_ext)
{
...
        case TCPOPT_EXP:
                if (opsize >= TCPOLEN_EXP_EDO_SUPPORTED &&
                    get_unaligned_be16(ptr) == TCPOPT_EDO_MAGIC) {
...
                        if (opt_rx->edo_ok) /* Parse only once */
                                return -EINVAL;
                        if (!parse_edo_ext) /* Non-EDO socket */
                                break;

                        parse_edo_ext = false;
                        ret = tcp_parse_edo_extension(skb, &th, &ptr, &length, opsize);
                        if (ret < 0)
                                return ret;
                        opt_rx->edo_ok = 1;
```

Parse EDO Extension

net/ipv4/tcp_input.c :

```
static int tcp_parse_edo_extension(struct sk_buff *skb, const struct tcphdr **thp,
                                   const unsigned char **ptr, int *left, int opsize)
{
...
        hdr_len = get_unaligned_be16(*ptr + 2);
        if (hdr_len == th->doff * 4)
                return 0;
...
        if (!pskb_may_pull(skb, hdr_len)) /* iph and th must be reloaded at callers. */
                return -ENOMEM;

        *thp = th = tcp_hdr(skb);          /* Not to ACK options in extended area */
        TCP_SKB_CB(skb)->end_seq = TCP_SKB_CB(skb)->seq + th->fin + skb->len - hdr_len;
        parsed = (th->doff << 2) - (*left - 2);
        *ptr = (const unsigned char *)th + parsed; /* Continue parsing in
        *left = hdr_len - parsed;                                * tcp_parse_options */
        return 0;
```

Parse EDO Extension

net/ipv4/tcp_input.c :

```
static void tcp_data_queue(struct sock *sk, struct sk_buff *skb)
{
...
        if (tp->edo) {    /* Don't expose TCP options in extended area to user space */
                u32 data_len;

                data_len = TCP_SKB_CB(skb)->end_seq - TCP_SKB_CB(skb)->seq - tcp_hdr(skb)->fin;

                __skb_pull(skb, skb->len - data_len);
        } else {
                __skb_pull(skb, tcp_hdr(skb)->doff * 4);
        }

...
```

EDO and other options

Options that depend on other options MUST also be augmented to interpret the EDO Extension option to operate correctly

- MD5
- TCP-AO (upcoming)
- MPTCP

EDO and MD5

MD5 hashes

1. The TCP pseudo-header (in the order: source IP address, destination IP address, zero-padded protocol number, and segment length)
2. The TCP header, **excluding options**, and assuming a checksum of zero
3. The **TCP segment data** (if any)
4. An independently-specified key or password, known to both TCPs and presumably connection-specific

EDO and MD5

```
--- a/net/ipv4/tcp.c
+++ b/net/ipv4/tcp.c
@@ -4511,18 +4511,19 @@ tcp_inbound_md5_hash(const struct sock *sk, const struct sk_buff *skb,
...
+       header_len = skb->len - TCP_SKB_CB(skb)->end_seq + TCP_SKB_CB(skb)->seq
+                + th->syn + th->fin;
...
        if (family == AF_INET)
-               genhash = tcp_v4_md5_hash_skb(newhash, hash_expected, NULL, skb);
+               genhash = tcp_v4_md5_hash_skb(newhash, hash_expected, NULL, skb, header_len);


--- a/net/ipv4/tcp_ipv4.c
+++ b/net/ipv4/tcp_ipv4.c
@@ -1525,7 +1525,7 @@ int tcp_v4_md5_hash_skb(char *md5_hash, const struct tcp_md5sig_key *key,
...
-       if (tcp_md5_hash_skb_data(hp, skb, th->doff << 2))
+       if (tcp_md5_hash_skb_data(hp, skb, header_len))
                goto clear_hash;
...
```

EDO and MD5

```
const u8 *tcp_parse_md5sig_option(struct sk_buff *skb, const struct tcphdr *th, bool parse_edo_ext)
{
...
        /* If not enough data remaining, we can short cut */
        while (length >= TCPOLEN_EXP_EDO_EXT_HDR) {
...
                } else if (parse_edo_ext && !th->syn && opcode == TCPOPT_EXP_EDO &&
                           opsize >= TCPOLEN_EXP_EDO_EXT_HDR &&
                           get_unaligned_be16(ptr) == TCPOPT_EDO_MAGIC) {
                        int ret;

                        if (parsed_edo_ext)
                                return ERR_PTR(-EINVAL);

                        ret = tcp_parse_edo_extension(skb, &th, &ptr, &length, opsize);
                        if (ret < 0)
                                return ERR_PTR(ret);

                        parsed_edo_ext = true;
                }
```

EDO and MPTCP

```
void mptcp_get_options(const struct sk_buff *skb, struct mptcp_options_received *mp_opt,
                       bool parse_edo_ext)
{
...
        if (opcode == TCPOPT_MPTCP)
                mptcp_parse_option(skb, ptr, opsize, hdr_len, mp_opt);

        if (parse_edo_ext && !th->syn && opcode == TCPOPT_EXP_EDO &&
            (opsize == TCPOLEN_EXP_EDO_EXT_HDR || opsize == TCPOLEN_EXP_EDO_EXT_SEG) &&
             get_unaligned_be16(ptr) == TCPOPT_EDO_MAGIC) {
                u8 parsed;

                /* Here, we have already pulled EDO Header Length */
                parse_edo_ext = false;
                parsed = hdr_len - (length - 2);
                hdr_len = get_unaligned_be16(ptr + 2);
                length = hdr_len - parsed;
        }
...
```

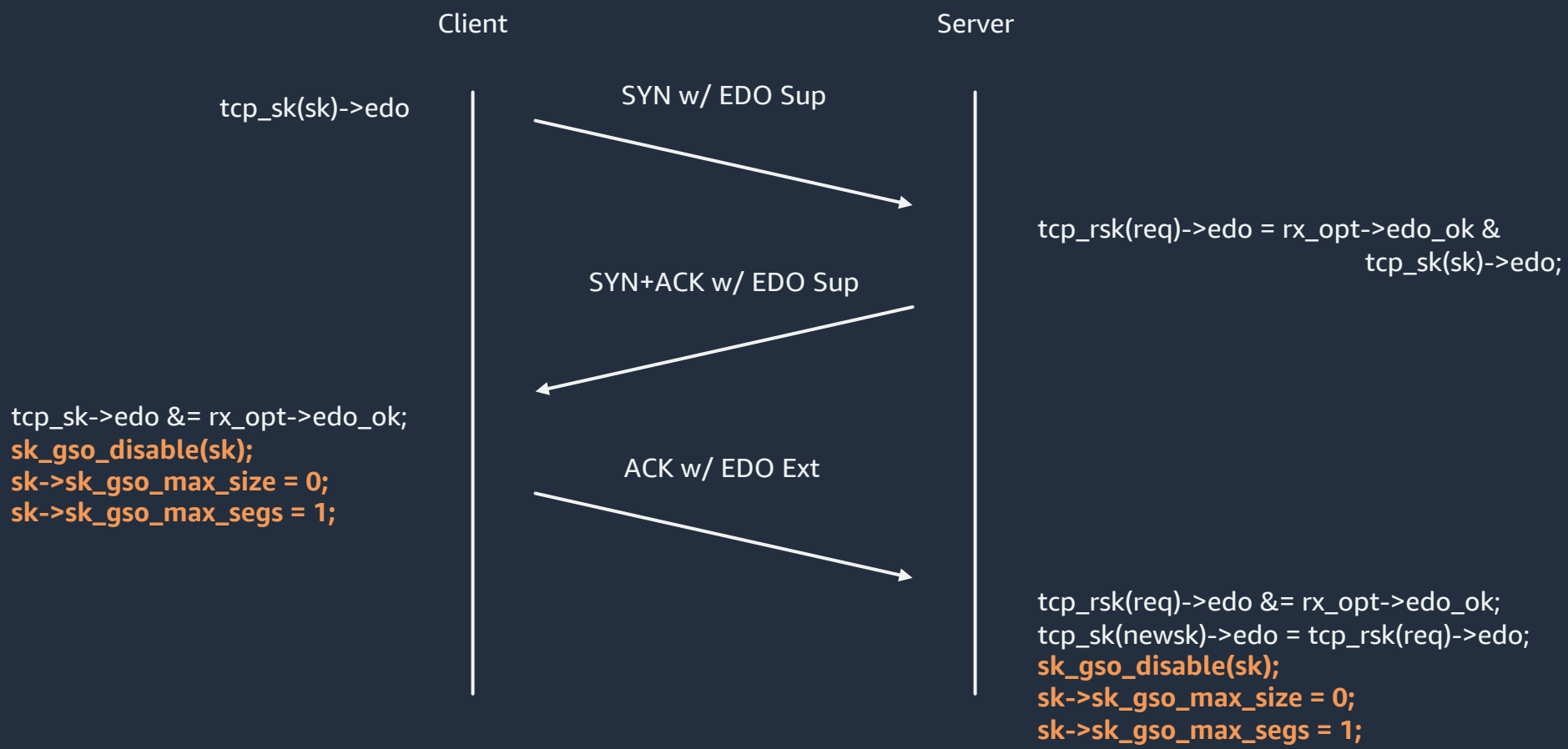
EDO and MPTCP

```
--- a/net/mptcp/options.c
+++ b/net/mptcp/options.c
@@ -21,7 +21,7 @@ static bool mptcp_cap_flag_sha256(u8 flags)
 }

 static void mptcp_parse_option(const struct sk_buff *skb,
-       const unsigned char *ptr, int opsize,
+       const unsigned char *ptr, int opsize, int hdr_len,
        struct mptcp_options_received *mp_opt)
 {
  u8 subtype = *ptr >> 4;
@@ -35,7 +35,7 @@ static void mptcp_parse_option(const struct sk_buff *skb,
  case MPTCPOPT_MP_CAPABLE:
        /* strict size checking */
        if (!(TCP_SKB_CB(skb)->tcp_flags & TCPHDR_SYN)) {
-               if (skb->len > tcp_hdr(skb)->doff << 2)
+               if (skb->len > hdr_len)
                        expected_opsize = TCPOLEN_MPTCP_MPC_ACK_DATA;
                else
                        expected_opsize = TCPOLEN_MPTCP_MPC_ACK;
```

EDO Extension must be parsed

```
--- a/include/net/tcp.h
+++ b/include/net/tcp.h
@@ -703,6 +703,12 @@ static inline void __tcp_fast_path_on(struct tcp_sock *tp, u32 snd_wnd)
        if (sk_is_mptcp((struct sock *)tp))
                return;
...
+       if (tp->edo)
+               return;
...


--- a/net/ipv4/tcp_ipv4.c
+++ b/net/ipv4/tcp_ipv4.c
@@ -1869,7 +1869,7 @@ bool tcp_add_backlog(struct sock *sk, struct sk_buff *skb,
...
-           thtail->doff != th->doff ||
+           thtail->doff != th->doff || tcp_sk(sk)->edo ||
            memcmp(thtail + 1, th + 1, hdrlen - sizeof(*th)))
                goto no_coalesce;
```

EDO and GSO



EDO and GRO

- Option 1 : No support
- Option 2 : Parse TCP options at GRO
- Option 3 : Trick GRO by NOPs
  - 2n -th packet : NOP NOP NOP NOP EDO-Ext ...
  - (2n-1)-th packet : EDO-Ext NOP NOP NOP NOP ...

Performance

Single flow test with iperf3 (avg of 5)

```
taskset -c 0 iperf3 -s
taskset -c 1 iperf3 -c localhost
```

- Baseline (5a1b322cb0b7) : 54.76 Gbps
- With series : 53.30 Gbps (-2.66 %)
- EDO w/o GRO : 52.66 Gbps (-3.83 %)

Thank you!