IETF118

TCP Extended Data Offset in Linux

Kuniyuki Iwashima

Kernel Development Engineer
Amazon Web Services

Agenda

TCP Extended Data Offset Option

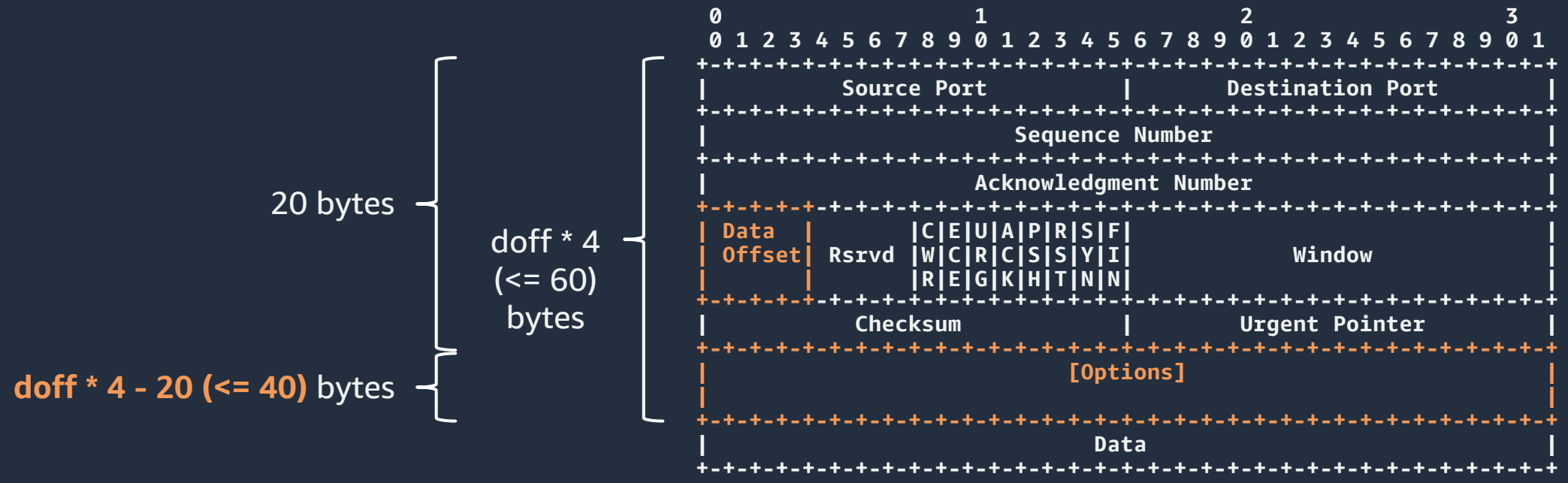
TCP EDO in Linux

Feedback to Draft

TCP Extended Data Offset Option

Data Offset

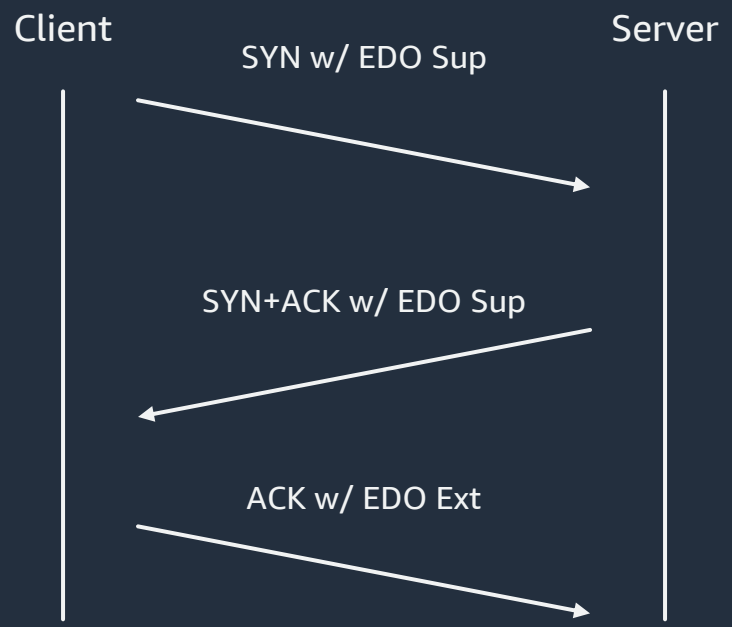
- Number of 32-bit words in the TCP header
- 4-bit field
  - 5 <= doff <= 15



Extended Data Offset (EDO) Option

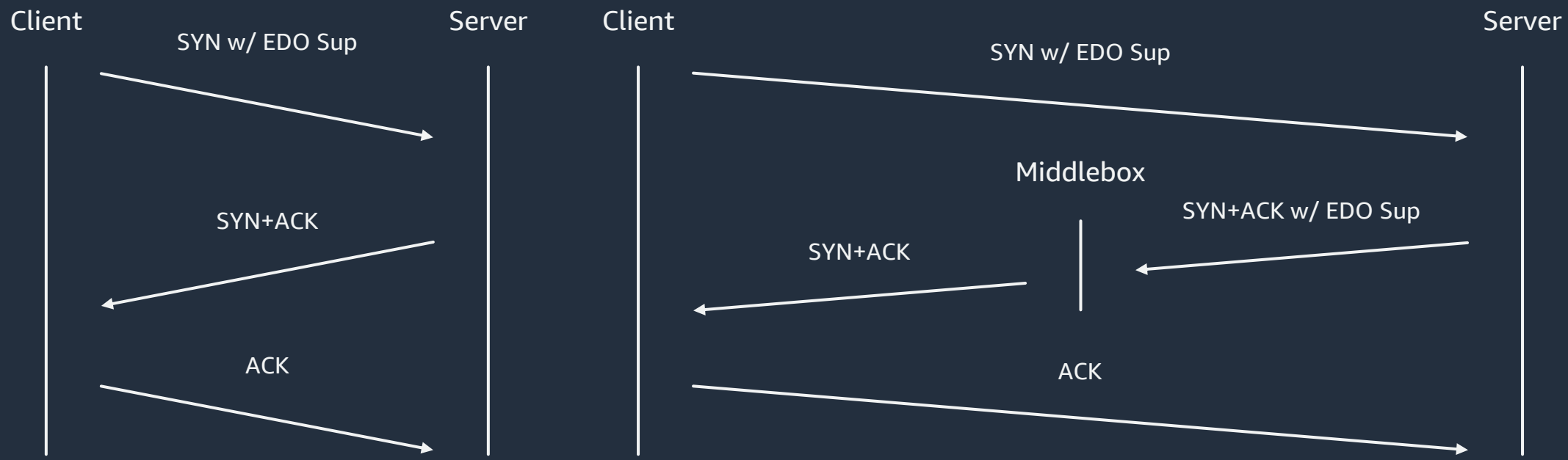
- draft-ietf-tcpm-tcp-edo-13
  - Suggested by J. Touch and Wes Eddy since 2014

- Two types of options
  - EDO Supported
    - Negotiate EDO availability in SYN and SYN+ACK
  - EDO Extension
    - **Override Data Offset** in all **non-SYN** segment (except for RST)
    - Two variants



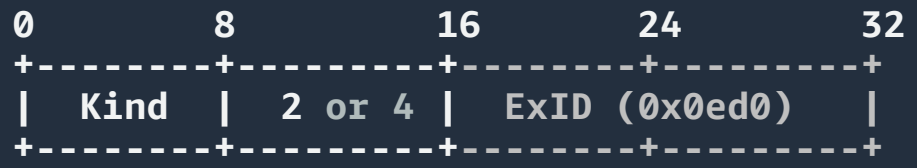
Zero-cost fallback

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

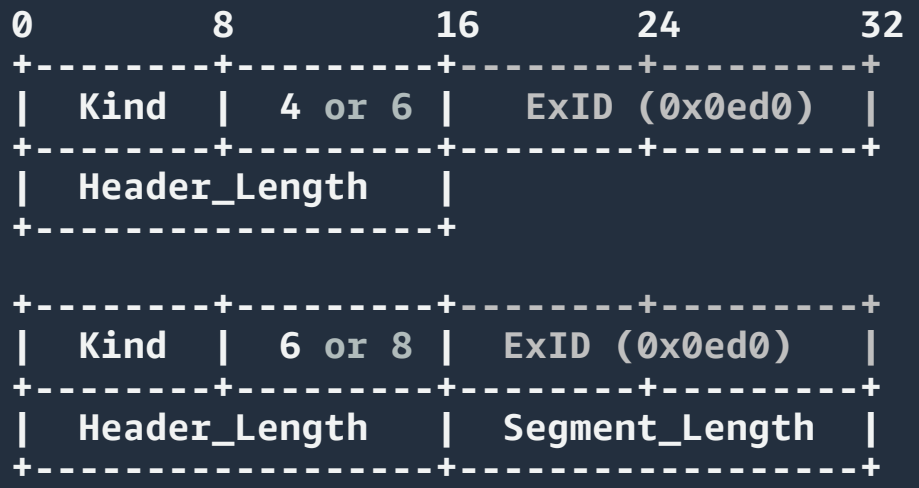


Option format

EDO Supported



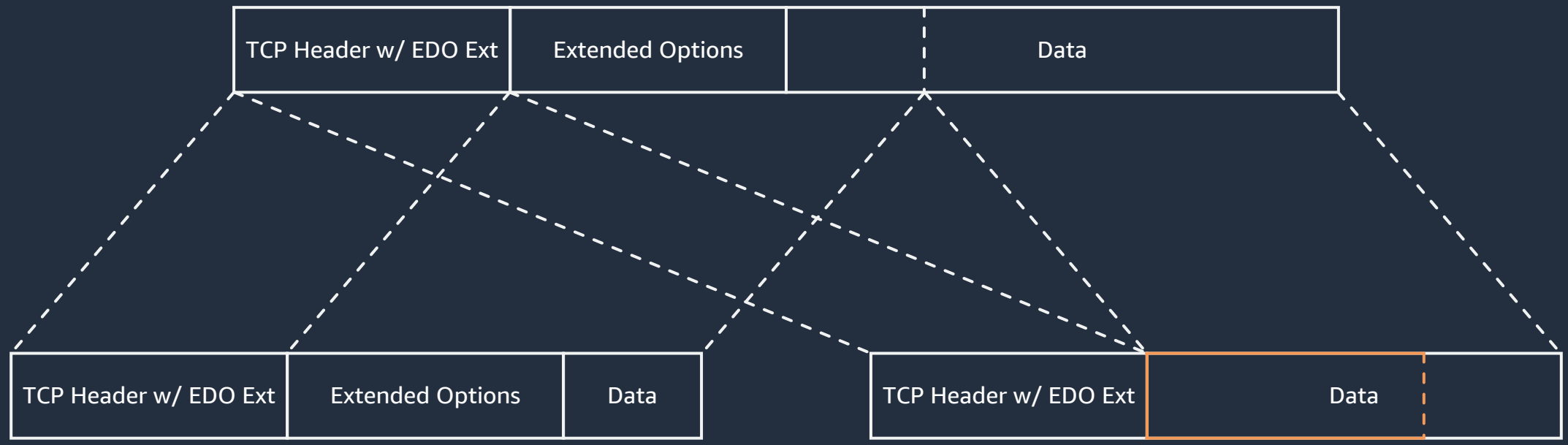
EDO Extension



Header Length : Override Data Offset

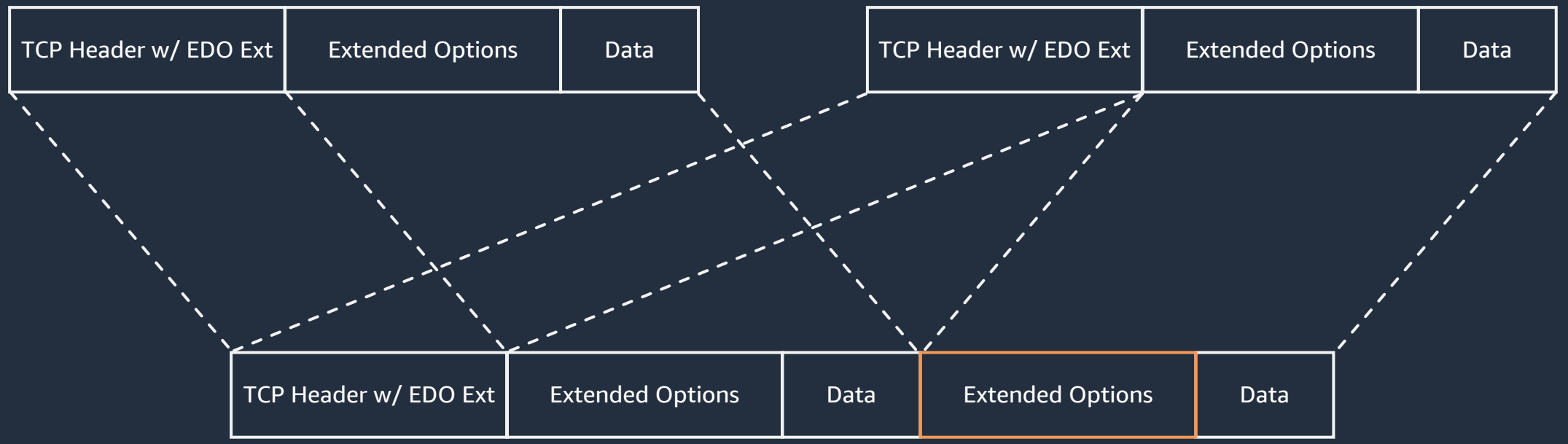
Segment Length : Detect merge/split (IP Payload Length)

If EDO packet is split without considering EDO



Partial data is parsed as Extended Options

If EDO packet is merged without considering EDO



TCP EDO in Linux

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
  by Yoshifumi Nishida

uAPI

Switch EDO via sysctl (per-netns) or setsockopt (per-socket)

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

- 0 <= val <= 2
  - 0 : Disabled
  - 1 : Use Header_Length
  - 2 : Use Header_Length and Segment_Length

Debug sysctl

Add NOPs after EDO Extension

- `sysctl -w net.ipv4.tcp_nops=val`

- 0 <= val <= 255

Demo

```
$ sudo sysctl -wq net.ipv4.tcp_ext_data_offset=2

$ sudo sysctl -wq net.ipv4.tcp_nops=32

$ python3

>>> from socket import *
>>>
>>> server = socket(AF_INET, SOCK_STREAM)
>>> server.listen()
>>>
>>> client = socket(AF_INET, SOCK_STREAM)
>>> client.connect(server.getsockname())
>>>
>>> child, _ = server.accept()
>>>
>>> client.send(b'hello’)
5
>>> child.recv(1024)
b'hello'
```

Demo

```
$ sudo tcpdump -i lo -nnt

IP 127.0.0.1.36406 > 127.0.0.1.50429: Flags [S], seq 3457883950, win 33280, options [exp-0ed0,
mss 65495,sackOK,TS val 2440410000 ecr 0,nop,wscale 7], length 0

IP 127.0.0.1.50429 > 127.0.0.1.36406: Flags [S.], seq 1988941892, ack 3457883951, win 33280,
options [exp-0ed0,mss 65495,sackOK,TS val 2440410000 ecr 2440410000,nop,wscale 7], length 0

IP 127.0.0.1.36406 > 127.0.0.1.50429: Flags [.], seq 1:13, ack 1, win 260, options [exp-0ed0,
nop,nop,nop,nop,nop,nop,nop,nop,nop,nop,nop,nop,nop,nop,nop,nop,nop,nop,nop,nop,nop,nop,nop,n
op,nop,nop,nop,nop,nop,nop,nop], length 12

IP 127.0.0.1.36406 > 127.0.0.1.50429: Flags [P.], seq 1:18, ack 1, win 260, options [exp-0ed0,
nop,nop,nop,nop,nop,nop,nop,nop,nop,nop,nop,nop,nop,nop,nop,nop,nop,nop,nop,nop,nop,nop,nop,n
op,nop,nop,nop,nop,nop,nop,nop], length 17

IP 127.0.0.1.50429 > 127.0.0.1.36406: Flags [.], seq 1:13, ack 6, win 260, options [exp-0ed0,
nop,nop,nop,nop,nop,nop,nop,nop,nop,nop,nop,nop,nop,nop,nop,nop,nop,nop,nop,nop,nop,nop,nop,n
op,nop,nop,nop,nop,nop,nop,nop], length 12
```

Demo

```
$ sudo tcpdump -i lo -nnt

IP 127.0.0.1.36406 > 127.0.0.1.50429: Flags [S], seq 3457883950, win 33280, options [exp-edo
supported,mss 65495,sackOK,TS val 2440410000 ecr 0,nop,wscale 7], length 0

IP 127.0.0.1.50429 > 127.0.0.1.36406: Flags [S.], seq 1988941892, ack 3457883951, win 33280,
options [exp-edo supported,mss 65495,sackOK,TS val 2440410000 ecr 2440410000,nop,wscale 7],
length 0

IP 127.0.0.1.36406 > 127.0.0.1.50429: Flags [.], ack 1, win 260, options [exp-edo hlen 72
seglen 72,nop,nop,nop,nop,nop,nop,nop,nop,nop,nop,nop,nop,nop,nop,nop,nop,nop,nop,nop,nop,
nop,nop,nop,nop,nop,nop,nop,nop,nop,nop,nop,nop,TS val 2440410000 ecr 2440410000], length 0

IP 127.0.0.1.36406 > 127.0.0.1.50429: Flags [P.], seq 1:6, ack 1, win 260, options [exp-edo
hlen 72 seglen 77,nop,nop,nop,nop,nop,nop,nop,nop,nop,nop,nop,nop,nop,nop,nop,nop,nop,nop,
nop,nop,nop,nop,nop,nop,nop,nop,nop,nop,nop,nop,nop,nop,TS val 2440416759 ecr2440410000],
length 5

IP 127.0.0.1.50429 > 127.0.0.1.36406: Flags [.], ack 6, win 260, options [exp-edo hlen 72
seglen 72,nop,nop,nop,nop,nop,nop,nop,nop,nop,nop,nop,nop,nop,nop,nop,nop,nop,nop,nop,nop,
nop,nop,nop,nop,nop,nop,nop,nop,nop,nop,nop,nop,TS val 2440416759 ecr 2440416759], length 0
```

Notable Changes in Linux (1/2)

- Reallocate skb linear buffer if options >= 40 bytes
  - **`MAX_TCP_HEADER`** assumes TCP header length <= 60 bytes
- Reload TCP/IP header pointers after parsing options
  - TCP/IP headers are pulled in the linear buffer
  - Parser pulls extended options area and may change the buffer layout
- Correct Data Offset uses
  - Options in extended area not to be ACKed or passed to userspace
  - MD5, TCP-AO, and MPTCP need to be aware of EDO

Notable Changes in Linux (2/2)

- Disable fast processing
  - TCP coalescing, Header prediction

- Disable GSO and GRO automatically
  - GSO is disabled when EDO is negotiated successfully
  - GRO is disabled by parsing options (not acceptable approach)
    - Another approach is trick GRO with NOPs
      - 2n-th packet : EDO Ext, NOP, NOP, ...
      - (2n + 1)-th packet : NOP, NOP, EDO Ext, ...

Performance

Single flow test with iperf3 between two EC2 instances (c6i.32xlarge)

- Baseline (dbc0fd481cd0) : 23.97 Gbps
- Non-EDO w/ EDO patches : 23.97 Gbps (-0%)
- EDO (Segment_Length) : 23.71 Gbps (-1.1%)
- EDO (Segment_Length) + 64 NOPs : 23.25 Gbps (-3.0%)

Feedback to Draft

Multiple EDO Extension

>> EDO Extension MUST not be used more than once in each segment.

Data Offset : 15
EDO Ext 1 : Header_Length == 1000 bytes offset == 20 bytes
EDO Ext 2 : Header_Length == 500 bytes offset == 28 bytes
EDO Ext 3 : Header_Length == 1500 bytes offset == 36 bytes
EDO Ext 4 : Header_Length == 400 bytes offset == 44 bytes
...

Cross SYN and Fallback

>> The EDO Extension option **MAY** be used only if confirmed when the connection transitions to the ESTABLISHED state, e.g., **a client is enabled after receiving the EDO Supported option in the SYN/ACK** and the server is enabled after seeing the EDO Extension option in the final ACK of the three-way handshake.

Cross SYN and Fallback

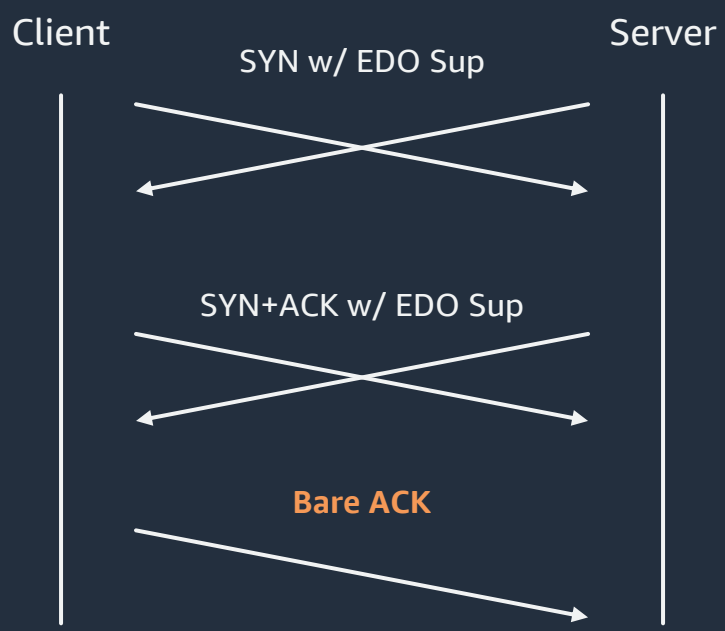
>> The EDO Extension option **MAY** be used only if confirmed when the connection transitions to the ESTABLISHED state, e.g., ...

SYN_RECV is split to 2 states in Linux

- TCP_NEW_SYN_RECV : Passive open
- TCP_SYN_RECV : Passive Fast Open, **Cross SYN**, Self-connect

TCP_SYN_RECV and TCP_ESTABLISHED share the same function to parse TCP options

→ Fallback support for Cross SYN is expensive



Cross SYN and Fallback

>> The EDO Extension option **MAY** be used only if confirmed when the connection transitions to the ESTABLISHED state **for passive connection**, e.g., ~~a client is enabled after receiving the EDO Supported option in the SYN/ACK and~~ the server is enabled after seeing the EDO Extension option in the final ACK of the three-way handshake.

>> A client MUST be enabled after receiving EDO Supported in SYN/ACK.

Thank you!

Reallocate skb linear buffer if options >= 40 bytes

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;
                ...
        }
```

Reallocate skb linear buffer if options >= 40 bytes

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);
```

Reload TCP/IP header pointers after parsing options

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;
```

Reload TCP/IP header pointers after parsing options

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);
        parsed = *ptr - (const unsigned char *)*thp;
...
        if (!pskb_may_pull(skb, hdr_len)) /* iph and th must be reloaded at callers. */
                return -ENOMEM;

        *thp = tcp_hdr(skb);

        /* Not to ACK options in extended area */
        TCP_SKB_CB(skb)->end_seq = TCP_SKB_CB(skb)->seq + (*thp)->fin + skb->len - hdr_len;

        *ptr = (const unsigned char *)*thp + parsed;         /* Continue parsing in
        *left = hdr_len - parsed + 2;                         * tcp_parse_options() */
```

Correct Data Offset uses (ACK)

net/ipv4/tcp_input.c :

```
static void tcp_data_queue(struct sock *sk, struct sk_buff *skb)
{
...
        if (tp->edo) {
                u32 data_len;

                /* Don’t expose TCP options in extended area to user space */
                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);
        }
...
```

Correct Data Offset uses (MD5)

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

Correct Data Offset uses (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-aware 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;
                }
```

Disable fast processing

```
--- 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;
```

Cross SYN packetdrill test

```
`../common/defaults.sh`

0  socket(..., SOCK_STREAM, IPPROTO_TCP) = 3
+0 fcntl(3, F_SETFL, O_RDWR|O_NONBLOCK) = 0
+0 setsockopt(3, SOL_TCP, TCP_EXT_DATA_OFFSET, [TCP_EDO_HDR_SEG], 4) = 0

+0 connect(3, ..., ...) = -1 EINPROGRESS (Operation now in progress)

+0 > S  0:0(0) <edoOK,mss 1440,sackOK,TS val 1000 ecr 0,nop,wscale 8>

// Cross SYN has EDO Supported
+0 < S  0:0(0) win 1000 <edoOK, mss 1000>

+0 > S. 0:0(0) ack 1 <edoOK, mss 1440,sackOK,TS val 3308134035 ecr 0,nop,wscale 8>

// No EDO Extension, fall back to non-EDO
+0 < .  1:1(0) ack 1 win 1000

+0 write(3, ..., 100) = 100
+0 > P. 1:101(100) ack 1
```

Cross SYN packetdrill test

```
outbound sniffed packet: 0.001430 S 3464997220:3464997220(0) win 65535 <edoOK,mss 1440,sackOK,TS
val 233119379 ecr 0,nop,wscale 8>

inbound injected packet: 0.003027 S 0:0(0) win 1000 <edoOK,mss 1000>

outbound sniffed packet: 0.003478 S. 3464997220:3464997220(0) ack 1 win 65535 <edoOK,mss
1440,sackOK,TS val 233119381 ecr 0,nop,wscale 8>

inbound injected packet: 0.005526 . 1:1(0) ack 3464997221 win 1000

write syscall: 1693367983.750451

fallback-client-ack.pkt:17: runtime error in write call: Expected result 100 but got -1 with
errno 11 (Resource temporarily unavailable)
```