TCP-ENO: Encryption Negotiation Option

draft-ietf-tcpinc-tcpeno-06

Andrea Bittau, Dan Boneh, Daniel Giffin, Mark Handley,
David Mazières, and Eric Smith

IETF97

Friday, November 18, 2016

TCP-ENO goals

Facilitate adoption of future TCP encryption protocols (TEPs)

- New TEPs do not require additional TCP option kinds
- New TEPs incrementally deployable, fall back to older ones
- New TEPs compatible with existing TCPINC-aware applications (recall charter requires authentication hooks)

Abstract away details of TEPs

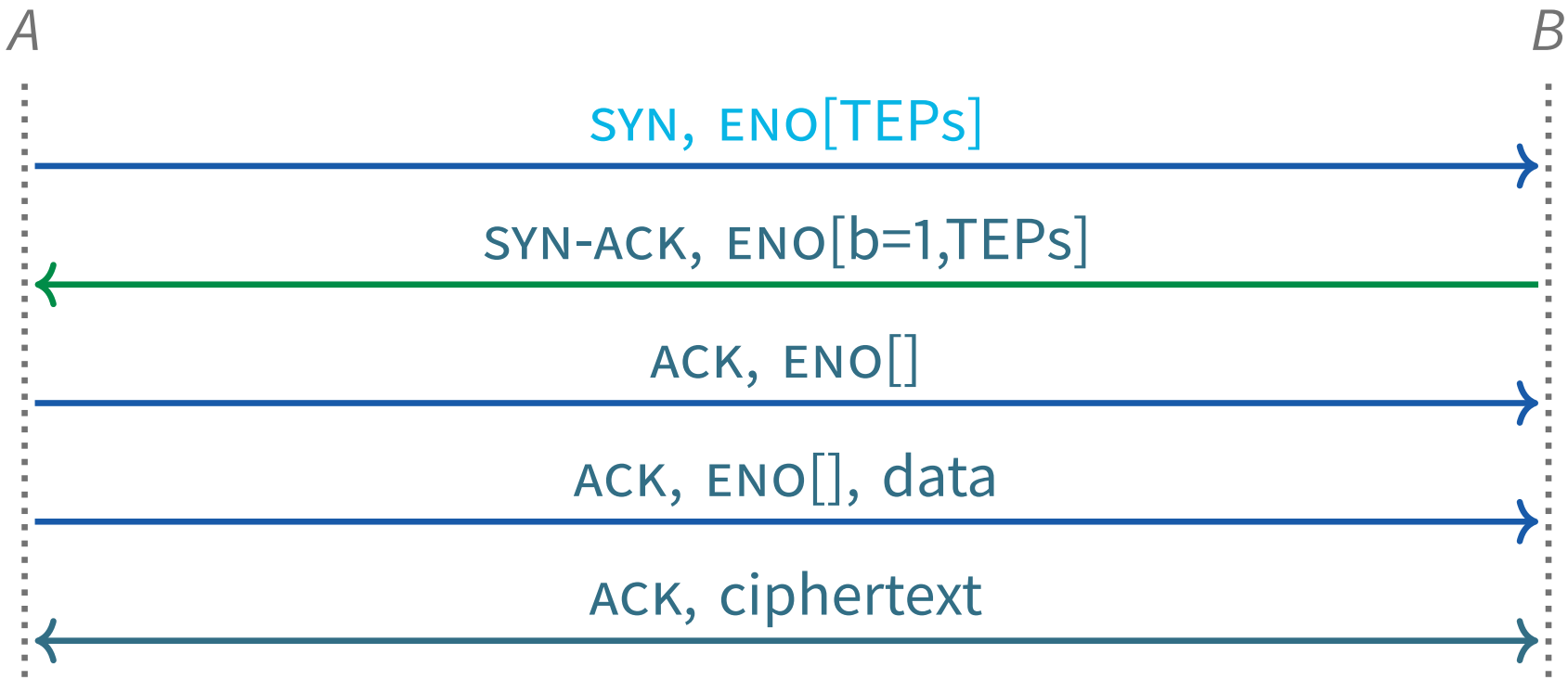
- Opaque session ID allows TEP-agnostic endpoint authentication

Minimize consumption of TCP option space

Avoid unnecessary round trips for connection setup

Revert to unencrypted TCP when encryption not possible

Overview of common case



Active opener *A* advertises supported TEPs

Passive opener *B* chooses a TEP (or ranks TEPs by preference)

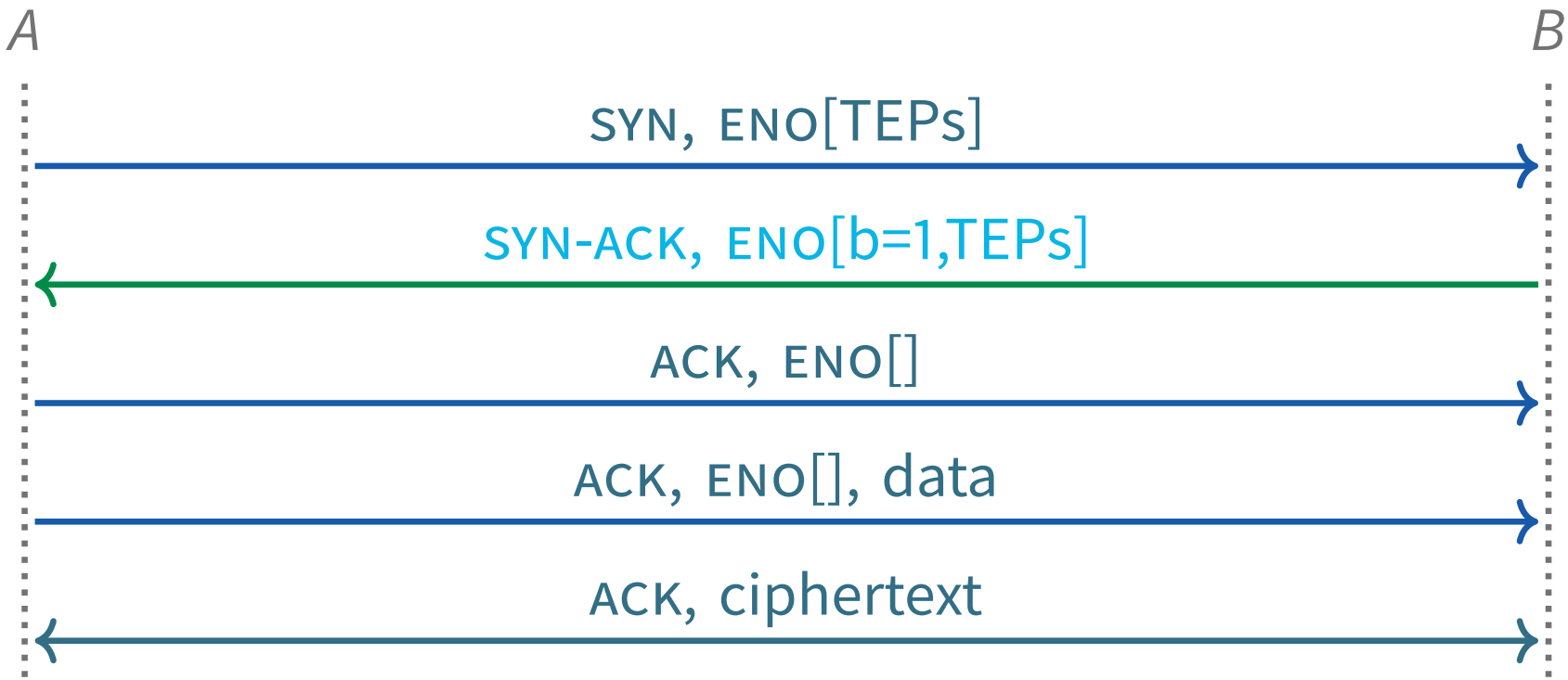
- MUST set global option b=1

***A* sends empty ENO option indicating encryption enabled**

- Keeps sending ENO option until it receives non-SYN segment

If any handshake ENOs missing, revert to unencrypted TCP

Overview of common case



Active opener *A* advertises supported TEPs

Passive opener *B* chooses a TEP (or ranks TEPs by preference)

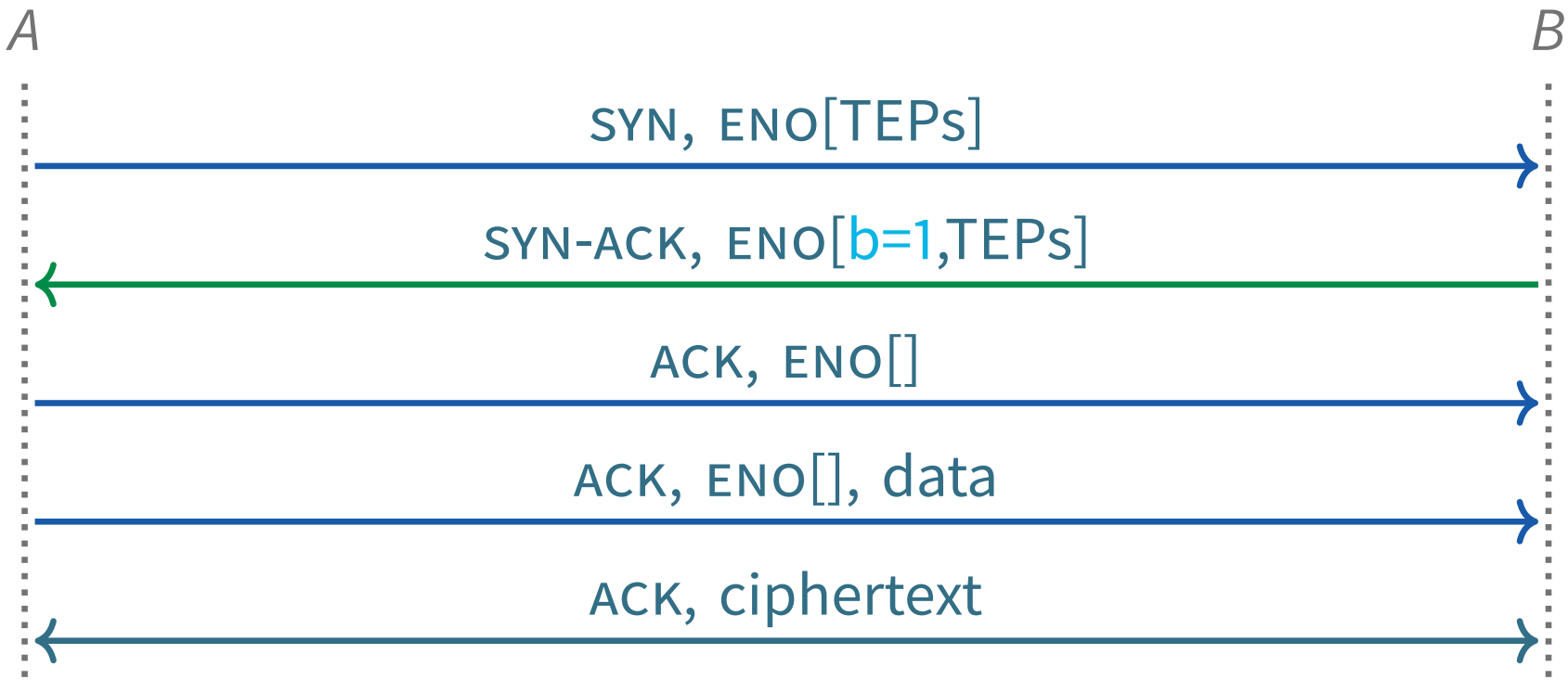
- MUST set global option b=1

***A* sends empty ENO option indicating encryption enabled**

- Keeps sending ENO option until it receives non-SYN segment

If any handshake ENOs missing, revert to unencrypted TCP

Overview of common case



Active opener *A* advertises supported TEPs

Passive opener *B* chooses a TEP (or ranks TEPs by preference)

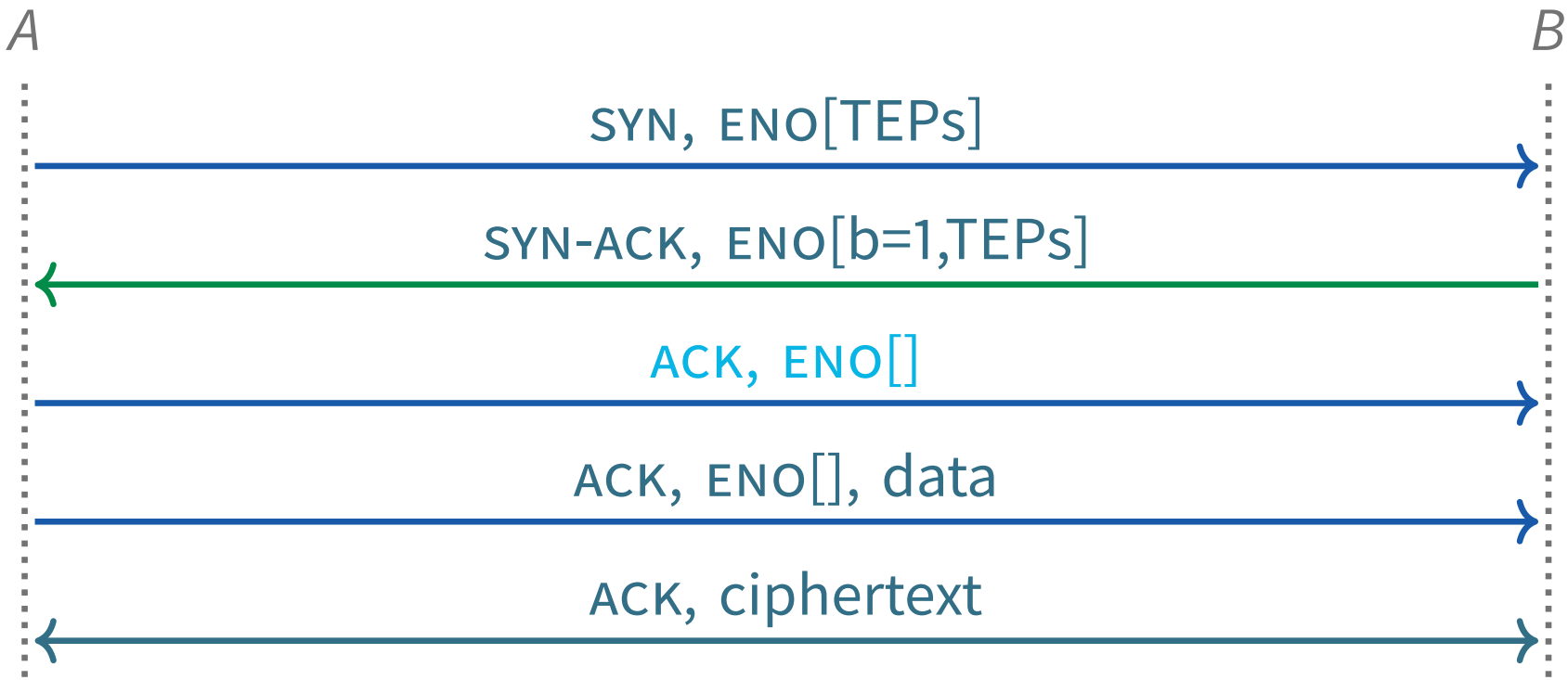
- MUST set global option b=1

***A* sends empty ENO option indicating encryption enabled**

- Keeps sending ENO option until it receives non-SYN segment

If any handshake ENOs missing, revert to unencrypted TCP

Overview of common case



Active opener *A* advertises supported TEPs

Passive opener *B* chooses a TEP (or ranks TEPs by preference)

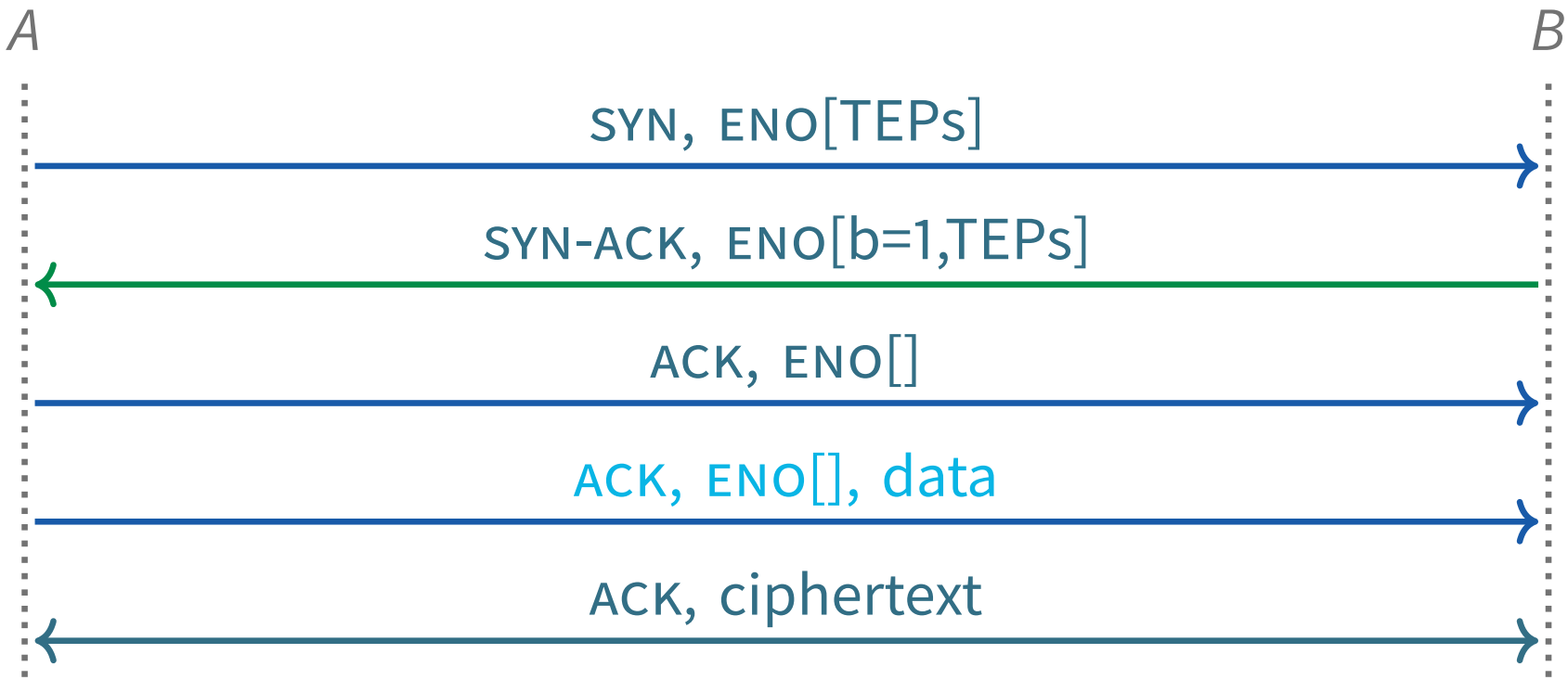
- MUST set global option b=1

***A* sends empty ENO option indicating encryption enabled**

- Keeps sending ENO option until it receives non-SYN segment

If any handshake ENOs missing, revert to unencrypted TCP

Overview of common case



Active opener *A* advertises supported TEPs

Passive opener *B* chooses a TEP (or ranks TEPs by preference)

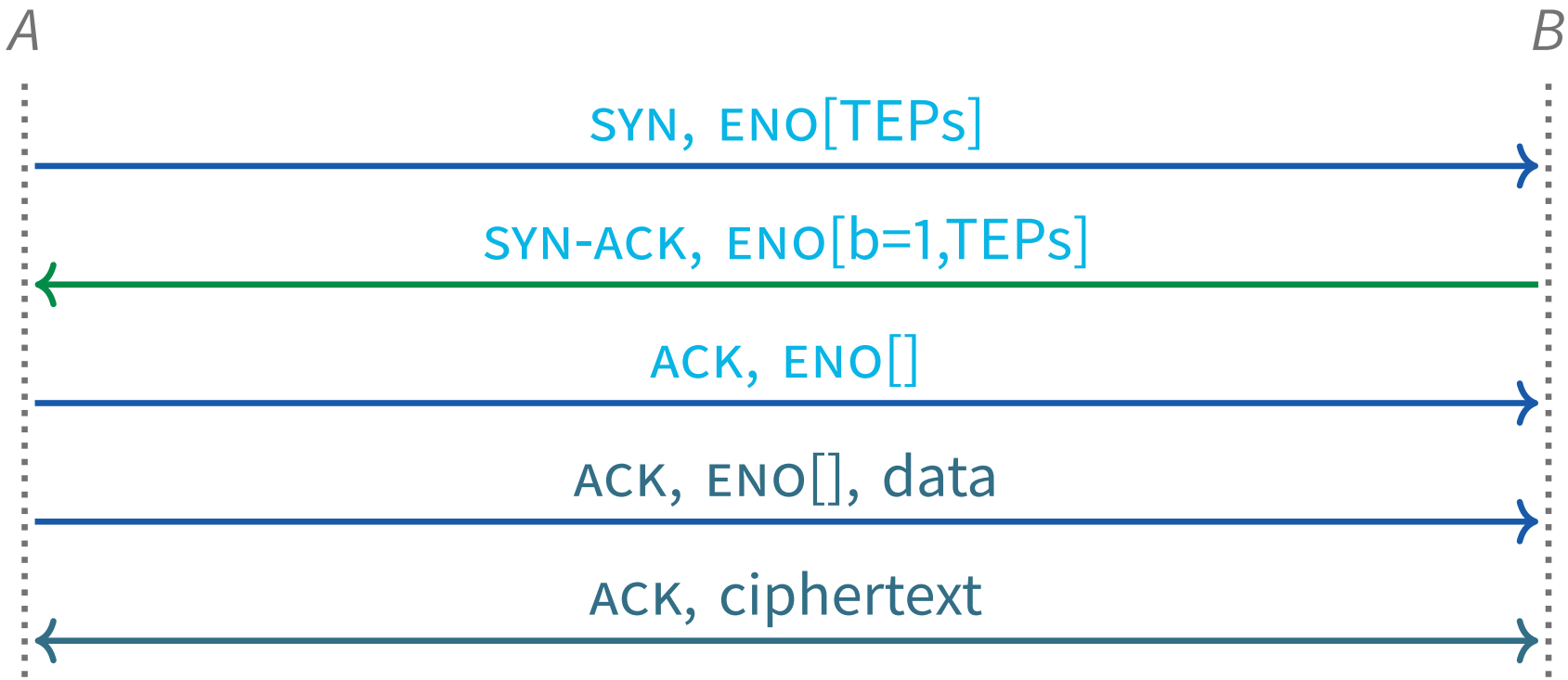
- MUST set global option b=1

***A* sends empty ENO option indicating encryption enabled**

- Keeps sending ENO option until it receives non-SYN segment

If any handshake ENOs missing, revert to unencrypted TCP

Overview of common case



Active opener *A* advertises supported TEPs

Passive opener *B* chooses a TEP (or ranks TEPs by preference)

- MUST set global option b=1

***A* sends empty ENO option indicating encryption enabled**

- Keeps sending ENO option until it receives non-SYN segment

If any handshake ENOs missing, revert to unencrypted TCP

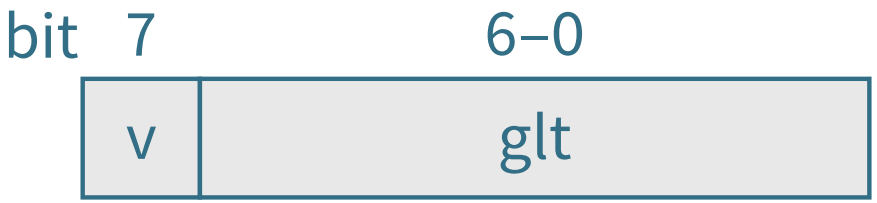
ENO option contents

SYN-form ENO is a container for a set of *suboptions*:

Non-SYN-form ENO is just a flag:

- Non-SYN-form contents MUST be 0 bytes unless defined by TEP

Initial suboption byte



glt	v	meaning
`0x00-0x1f`	`0`	Global suboption (was general suboption)
`0x00-0x1f`	`1`	Length byte (no more length word)
`0x20-0x7f`	`0`	TEP Id without data
`0x20-0x7f`	`1`	TEP Id followed by data

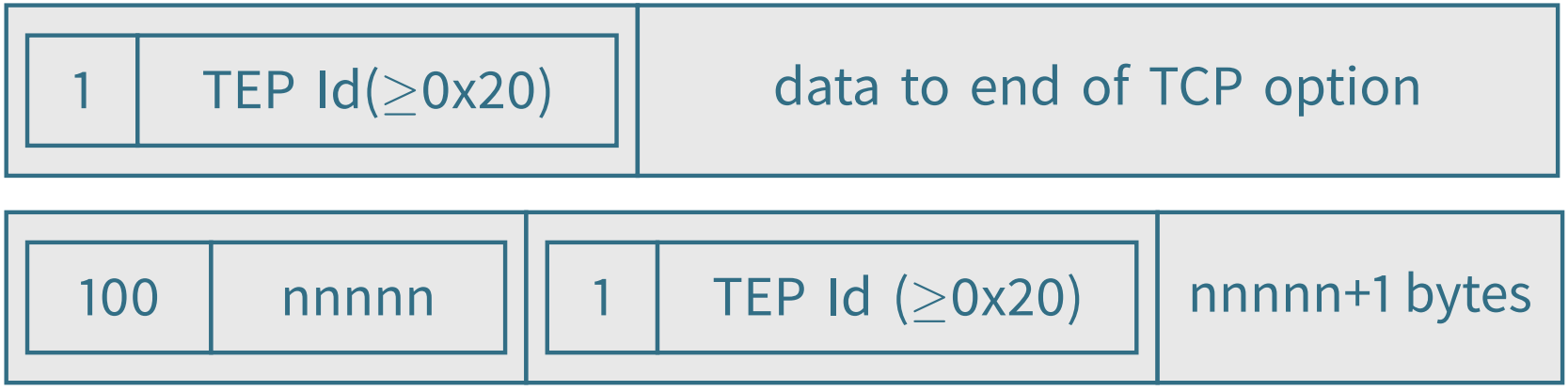
v = Variable-length data indicator

glt = Global suboption, Length byte, or TEP Id

TEP identifier suboption format

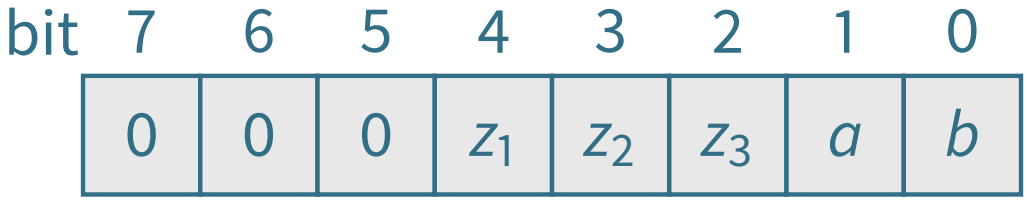
Single-byte TEP identifier suboption

TEP identifier suboption with suboption data



[not drawn to scale]

Global suboption format



b – Passive role bit

- Required to be 1 for all passive openers
- Disable ENO if both sides have same value (eliminated p bit)

a – Application-aware bit

- Intention: modify application protocol to incorporate session ID
- Mandatory application aware mode disables ENO if peer has $a = 0$

z_1, z_2, z_3 – Reserved (send as 0, ignore on receipt)

- No more m, but name z bits for easier future use
- Ideally z_3 can play the role of m in some future RFC

Ignore all but first global suboption byte in ENO

New: Data in SYN segments (§4.7)

The last TEP is a SYN segment is termed the SYN TEP

- The SYN TEP governs the meaning of data in that SYN segment
- Hosts MUST NOT send SYN data unless use defined by SYN TEP

Safeguard: REQUIRE discarding SYN data if:

- SYN TEP is not ultimately the negotiated TEP (including ENO fails), or
- Non-empty TFO or other TCP option indicates conflicting meaning for SYN data.

Safeguard: Don’t trust non-ENO hosts to discard bad SYN data

- If SYN TEP governs data but passive opener does not support ENO, might cache data even without ACKing it
- Hence, MUST abort connection if SYN-only+ENO+data followed by SYN-ACK without ENO, even if SYN-ACK does not ack bad SYN data

To avoid resets, SHOULD avoid SYN-only data by default

- Suggest mandatory encryption mode to enable such SYN data

Improvements to TEP requirements (§5)

> *TEPs MUST protect and authenticate the end-of-file marker conveyed by TCP's FIN flag. . . .*
>
> *TEPs MUST prevent corrupted packets from causing urgent data to be delivered when none has been sent. . . . A TEP MAY disable urgent data functionality by clearing the URG flag on all received segments and returning errors in response to sender-side urgent-data API calls. Implementations SHOULD avoid negotiating TEPs that disable urgent data by default. The exception is when applications and protocols are known never to send urgent data.*

Goal: avoid updating RFC793 without precluding TCP-use-TLS

- Phrase everything in terms of protecting TCP functionality
- Can't break urgent data [RFC6093] by default
- Leave big loophole since most apps known not to use urgent data

Improvements to TEP requirements (§5)

> *TEPs MUST protect and authenticate the end-of-file marker conveyed by TCP's FIN flag. . . .*
>
> *TEPs MUST prevent corrupted packets from causing urgent data to be delivered when none has been sent. . . . A TEP MAY disable urgent data functionality by clearing the URG flag on all received segments and returning errors in response to sender-side urgent-data API calls. Implementations SHOULD avoid negotiating TEPs that disable urgent data by default. The exception is when applications and protocols are known never to send urgent data.*

Goal: avoid updating RFC793 without precluding TCP-use-TLS

- Phrase everything in terms of protecting TCP functionality
- Can't break urgent data [RFC6093] by default
- Leave big loophole since most apps known not to use urgent data

Improvements to TEP requirements (§5)

> *TEPs MUST protect and authenticate the end-of-file marker conveyed by TCP's FIN flag. . . .*
>
> *TEPs MUST prevent corrupted packets from causing urgent data to be delivered when none has been sent. . . . A TEP MAY disable urgent data functionality by clearing the URG flag on all received segments and returning errors in response to sender-side urgent-data API calls. Implementations SHOULD avoid negotiating TEPs that disable urgent data by default. The exception is when applications and protocols are known never to send urgent data.*

Goal: avoid updating RFC793 without precluding TCP-use-TLS

- Phrase everything in terms of protecting TCP functionality
- Can't break urgent data [RFC6093] by default
- Leave big loophole since most apps known not to use urgent data

Changes since Berlin

Terminology changes:

- spec→TEP, general suboption→global suboption, SYN TEP

No more length word (max 32 bytes for all but last suboption)

No more global m bit; name z_1, z_2, z_3 in global suboption

Specify use of data in SYN segments

Several SHOULDs are now MUSTs

- Remaining SHOULDs make clear what exceptions might be

Improved wording for TEP requirements

- Forward secrecy a MUST at TEP level, a SHOULD for implementation
- FIN, URG preserve RFC793 but add authentication requirements

Still to do

Optional way to signal ENO implemented but disabled?

- Maybe permit SYN ENO option with just **b** bit, no TEP Ids?
- Might facilitate deployment of TEPs with SYN data
- Might facilitate data gathering

Add TCP_ENO_MANDATORY socket option to API doc

Get dedicated TCP option (preferably 'E' – 69)

Ideally not too much else before RFC...

Work needed for follow-on/companion documents:

- TCP-ENO middlebox probing
- How to multiplex experimental spec ID 0x20 (ExID-like mechanism)
- Define how to do application-independent endpoint authentication (probably co-opting z_3).