Autonomous NIC Offloads

Boris Pismenny Haggai Eran Aviad Yehezkel
Liran Liss Adam Morrison Dan Tsafrir

How to accelerate application layer (L5) computations transparently to software TCP/IP?

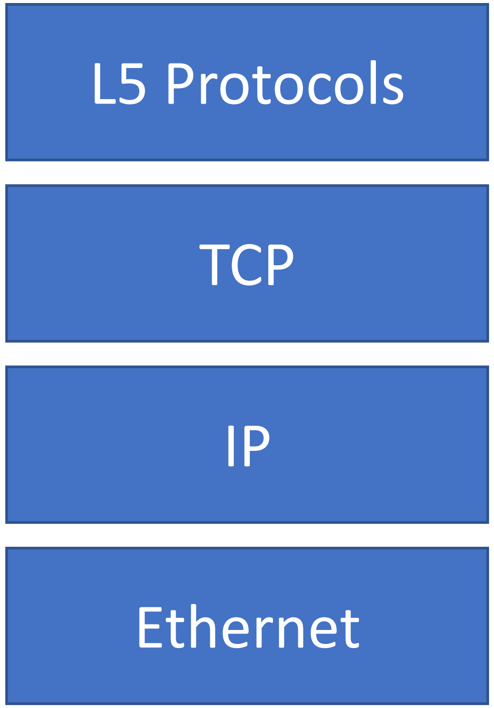
Offloading data-intensive layer-5 protocols

L5P examples

- tls
- nvme-tcp
- http
- grpc
- thrift
- iscsi
- nbd

Computation examples

- encryption
- decryption
- digest
- copy
- pattern matching
- (de)serialization
- (de)compression



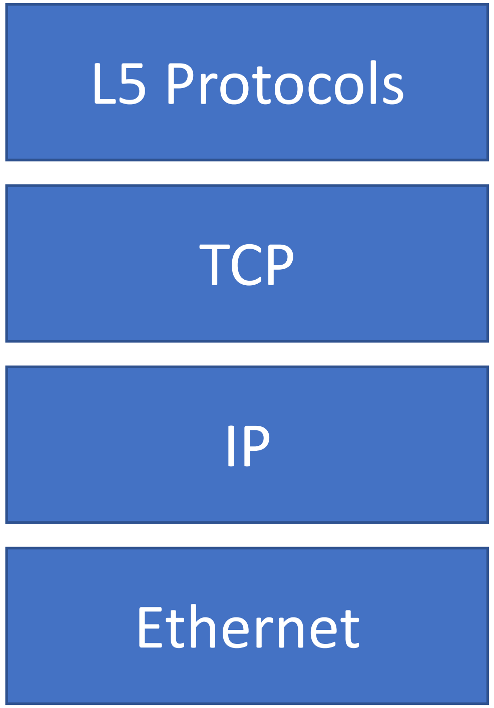
Offloading data-intensive layer-5 protocols

L5P examples

- tls
- nvme-tcp
- http
- grpc
- thrift
- iscsi
- nbd

Computation examples

- encryption
- decryption
- digest
- copy
- pattern matching
- (de)serialization
- (de)compression



L5 Protocols

TCP

IP

Ethernet

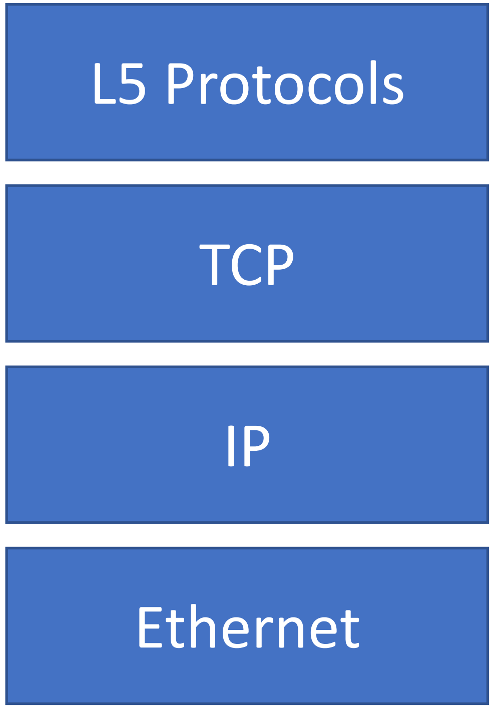
Offloading data-intensive layer-5 protocols

L5P examples

- tls
- nvme-tcp
- http
- grpc
- thrift
- iscsi
- nbd

Computation examples

- encryption
- decryption
- digest
- copy
- pattern matching
- (de)serialization
- (de)compression



L5 Protocols

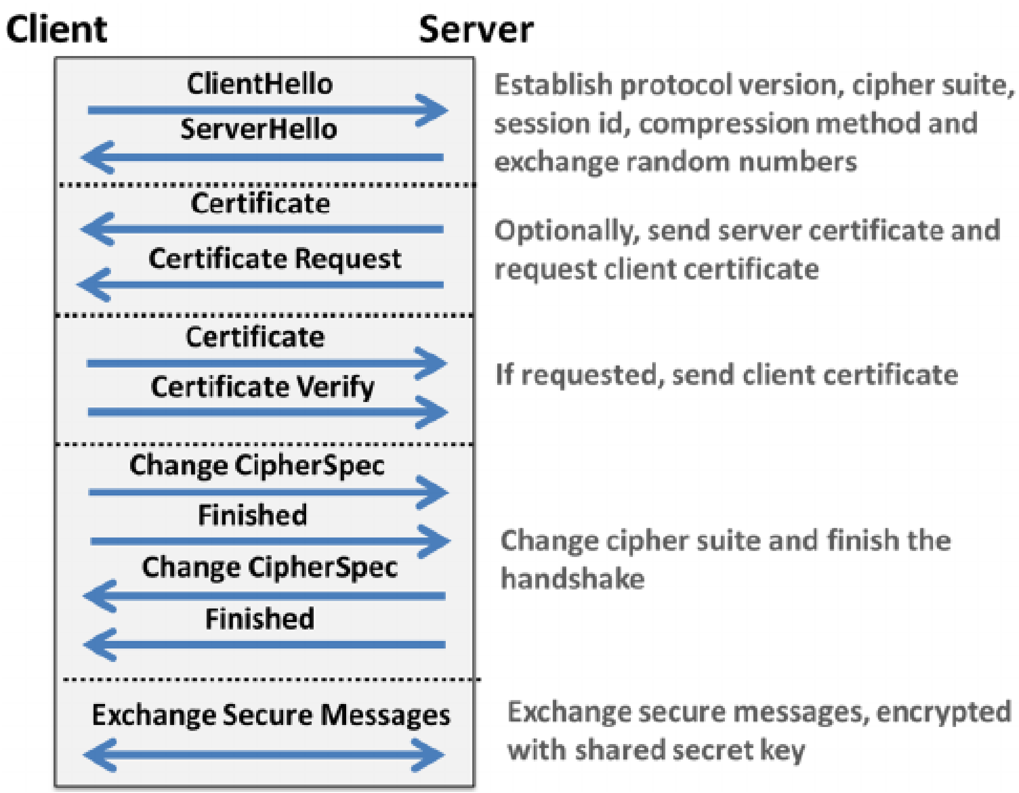
TCP

IP

Ethernet

What is TLS?

- Most popular way to encrypt TCP traffic
- 2 stages
  - Handshake
  - Data transfer



Establish protocol version, cipher suite, session id, compression method and exchange random numbers

Optionally, send server certificate and request client certificate

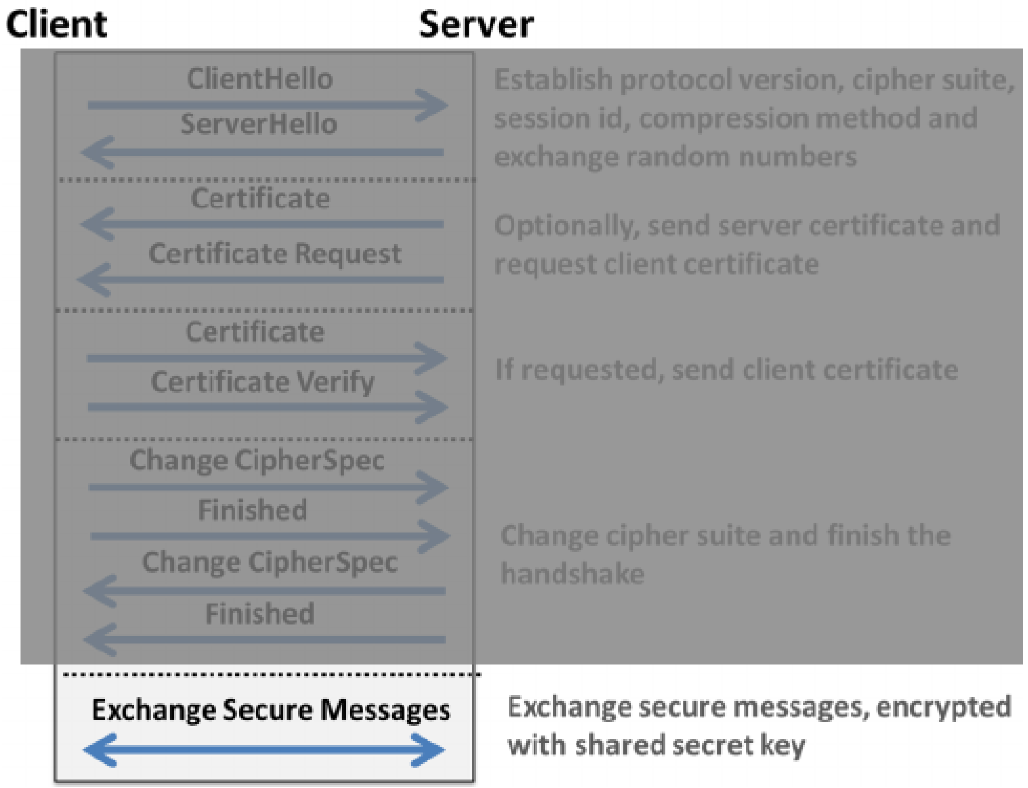
If requested, send client certificate

Change cipher suite and finish the handshake

Exchange secure messages, encrypted with shared secret key

What is TLS?

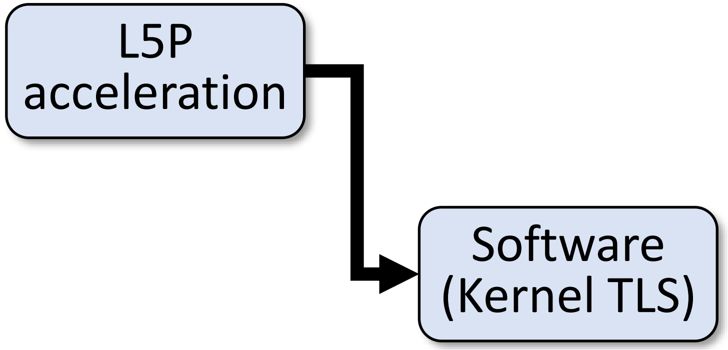
- Most popular way to encrypt TCP traffic
- 2 stages
  - Handshake
  - Data transfer
- We focus on data transfer


Client
Server
ClientHello
ServerHello
Establish protocol version, cipher suite, session id, compression method and exchange random numbers
Certificate
Certificate Request
Optionally, send server certificate and request client certificate
Certificate
Certificate Verify
If requested, send client certificate
Change CipherSpec
Finished
Change CipherSpec
Finished
Change cipher suite and finish the handshake
Exchange Secure Messages
Exchange secure messages, encrypted with shared secret key

Design Space

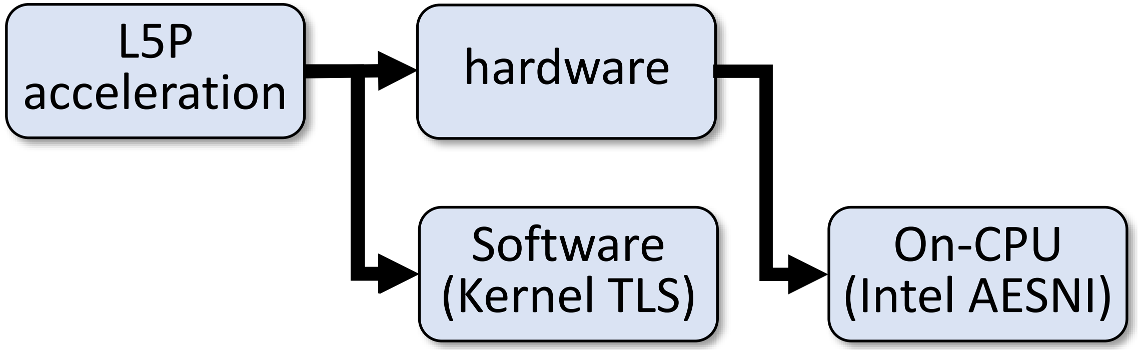
L5P acceleration

Design Space



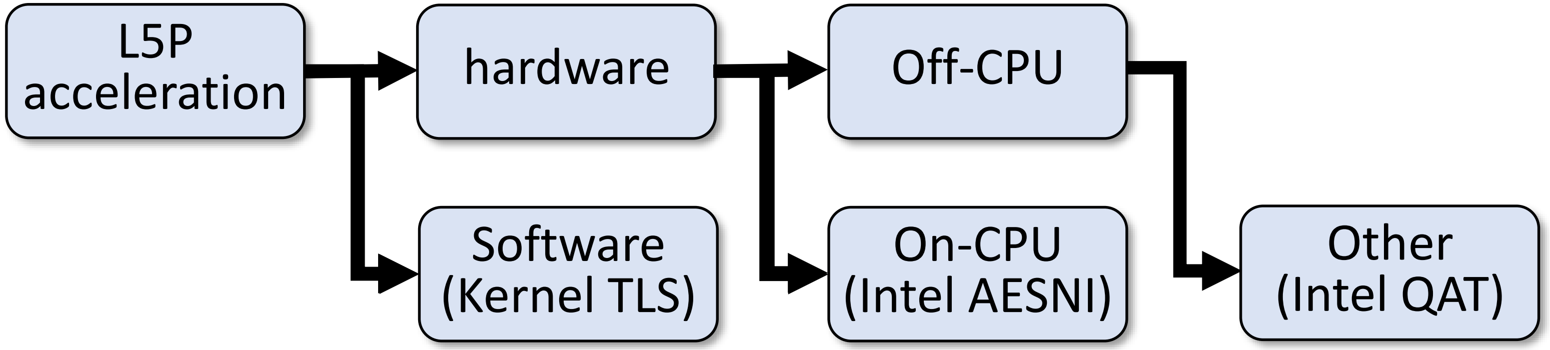
Pros	Cons
No additional hardware	Can’t avoid data intensive computations

Design Space



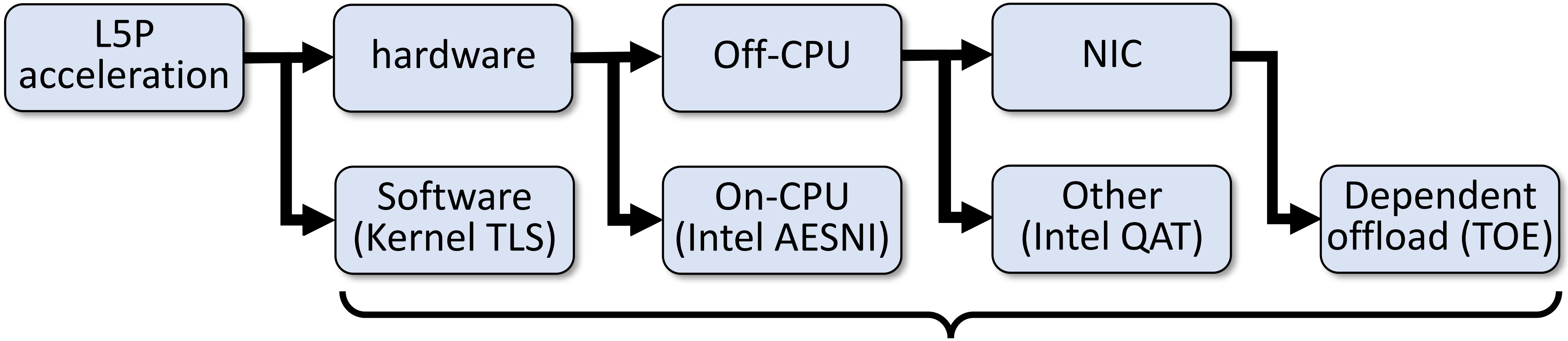
Pros	Cons
Uses fast CPU registers and cache memory	can consume >50% core to compute
Low overhead	

Design Space



Pros	Cons
CPU overhead is independent of data size	Significant parallelism required to outperform on-CPU acceleration

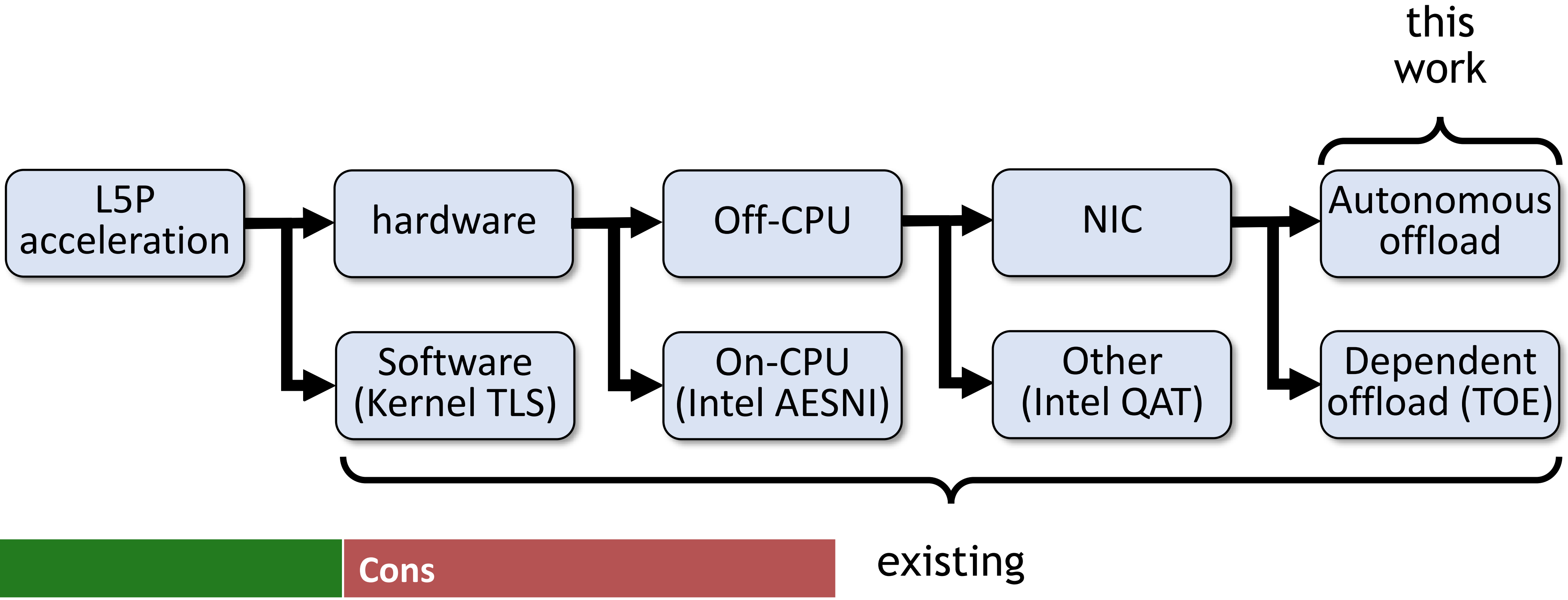
Design Space



existing

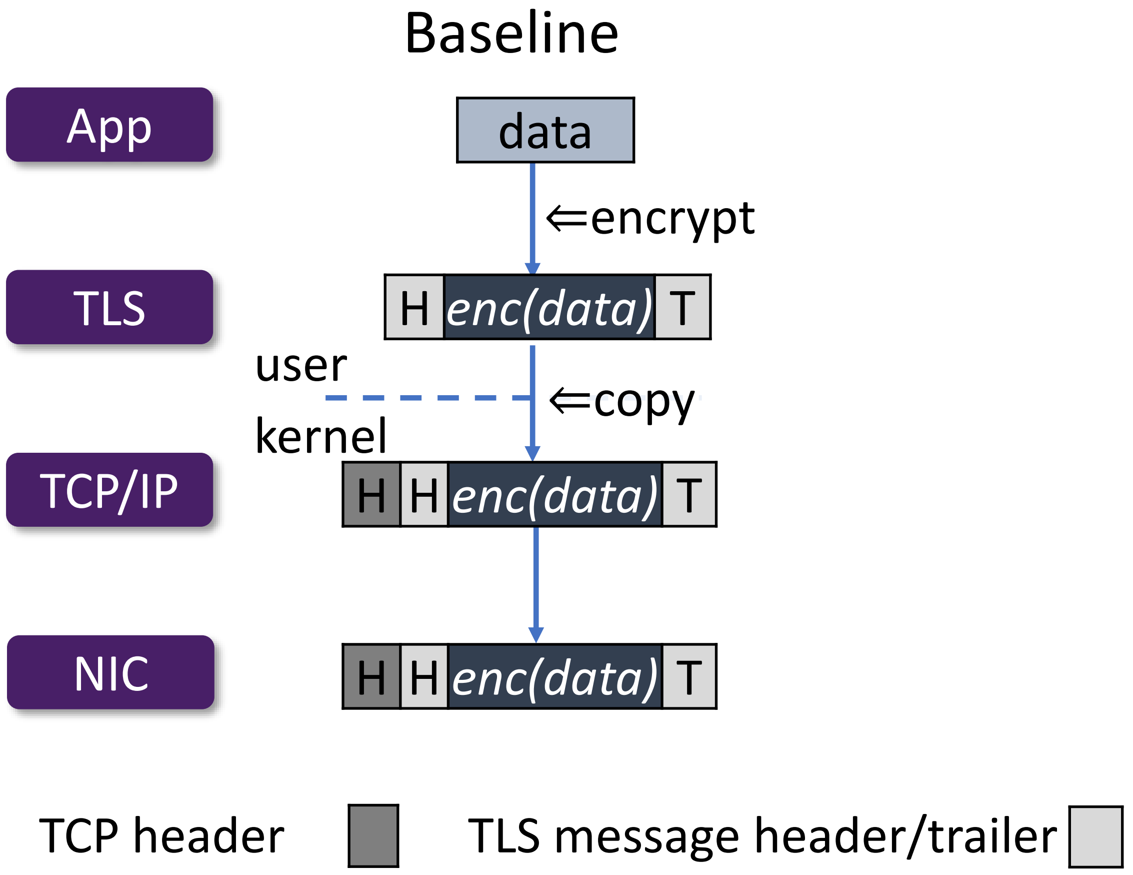
Pros	Cons
Eliminates CPU overhead	Depends on offloading: TCP, IP, routing, QoS, firewall, etc.

Design Space

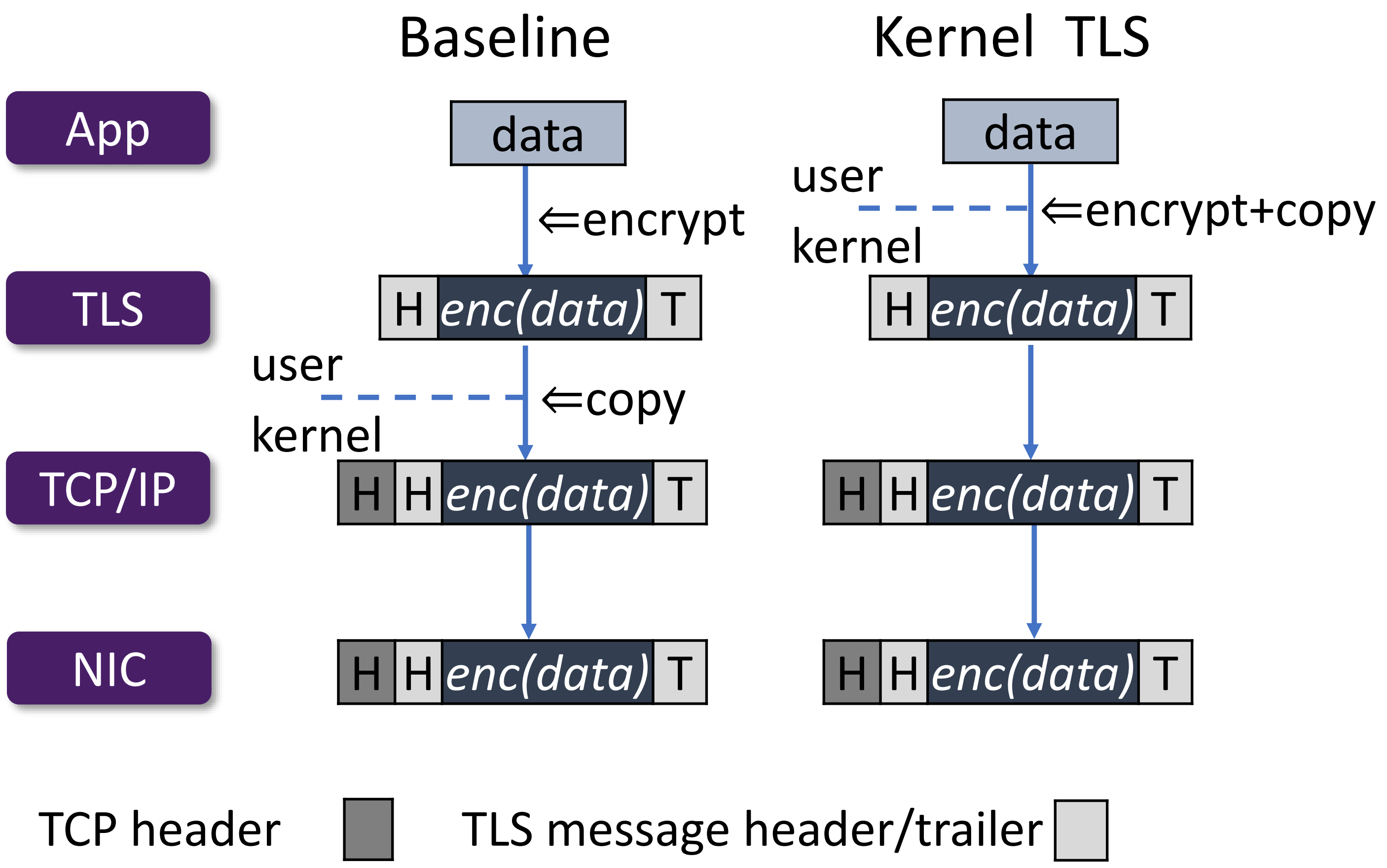


Pros	Cons
Eliminates CPU overhead	Overhead on recovery from reordering/loss
Works with software TCP, IP, routing, QoS, firewall, etc.	

Software specialization



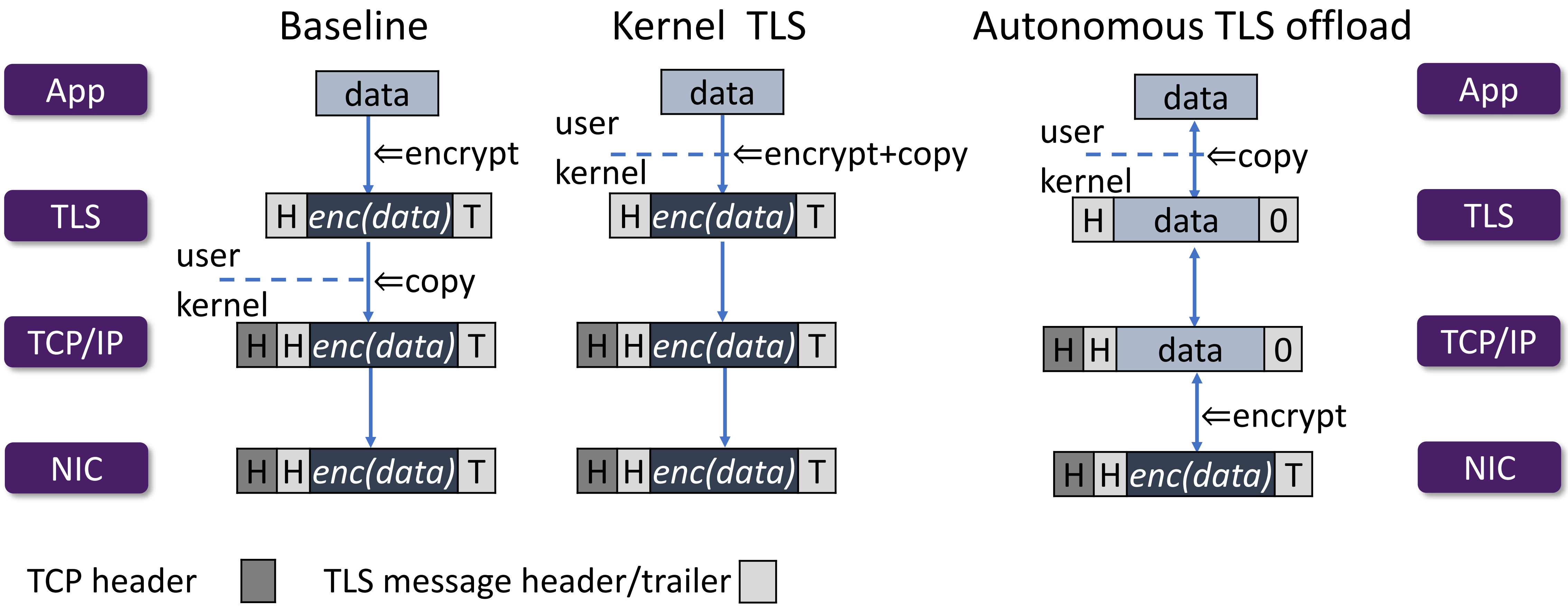
Software specialization



Kernel TLS enables

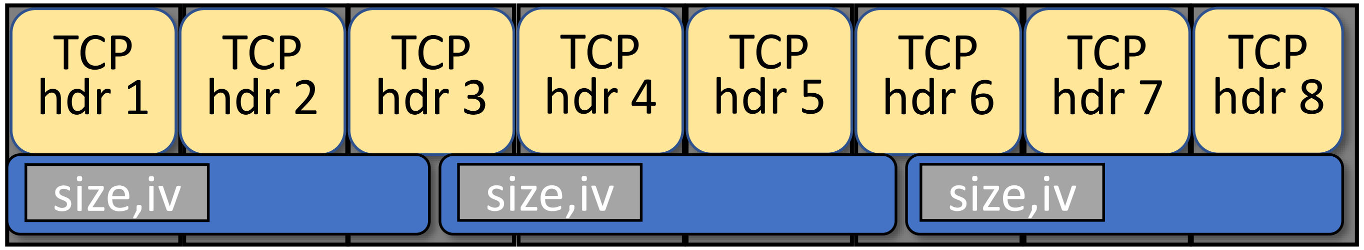
- Cross layer optimization
- Direct communication between NIC and TLS layers

Autonomous NIC offload: TLS



Transmit offload in-sequence

- In-order offload is the simplest
  - Incrementally offload using NIC contexts



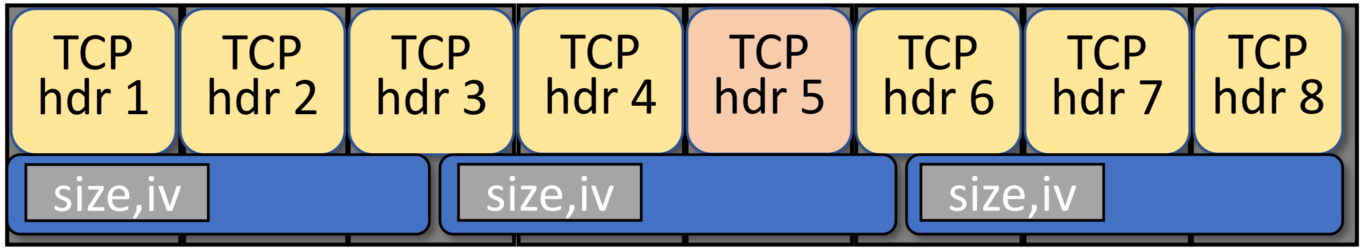
NIC contexts

Static state
- crypto keys

Dynamic state
- expected TCP seq
- current msg offset
- current msg size
- current msg IV
- msg ICV state

Transmit offload out-of-sequence

- Wrong dynamic NIC context state
- Context recovery needs only the message prefix
  - Driver can get the prefix from software TLS



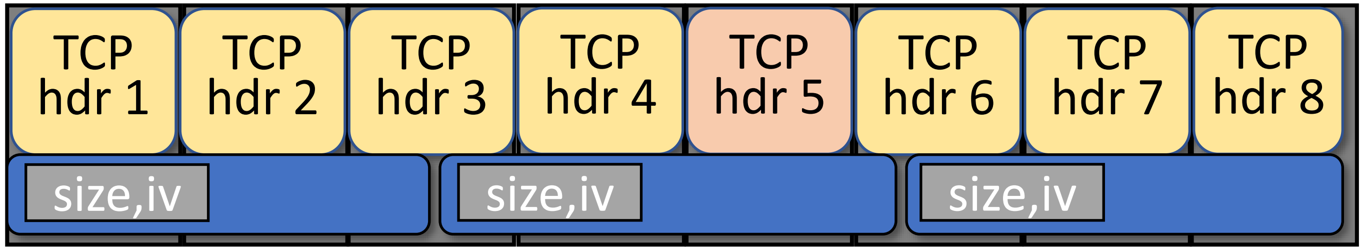
NIC contexts

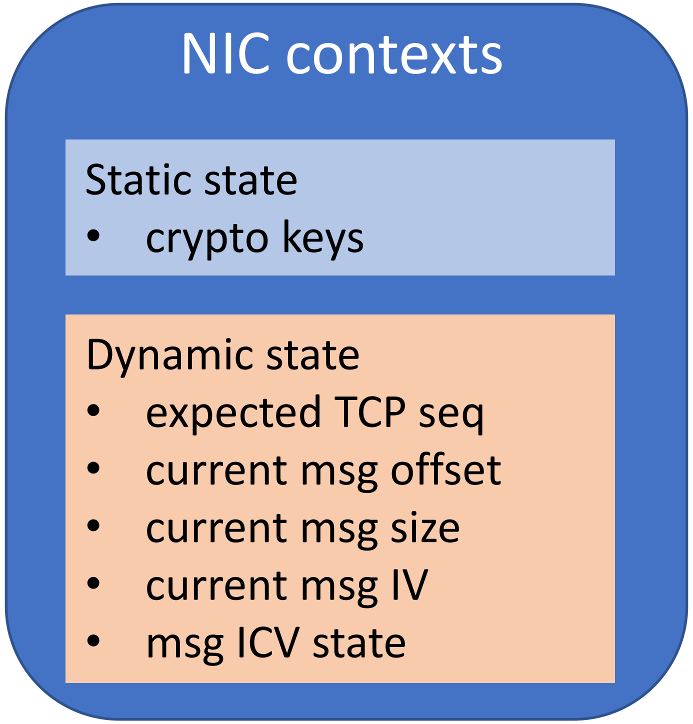
Static state
- crypto keys

Dynamic state
- expected TCP seq
- current msg offset
- current msg size
- current msg IV
- msg ICV state

Transmit offload out-of-sequence

- Wrong dynamic NIC context state
- Context recovery needs only the message prefix
  - Driver can get the prefix from software TLS





Static state

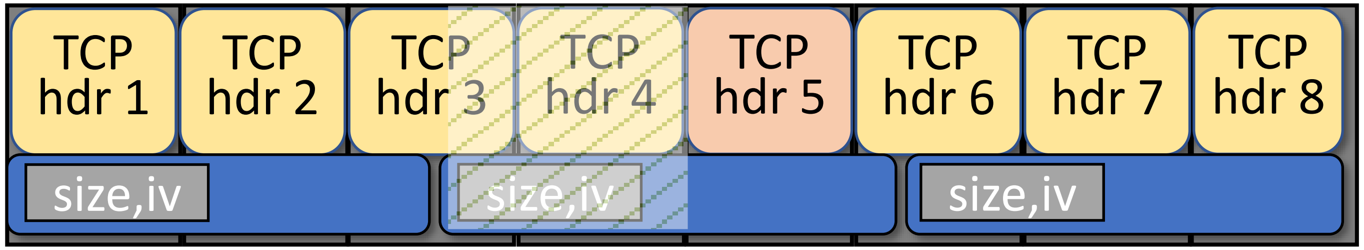
- crypto keys

Dynamic state

- expected TCP seq
- current msg offset
- current msg size
- current msg IV
- msg ICV state

Transmit offload out-of-sequence

- Wrong dynamic NIC context state
- Context recovery needs only the message prefix
  - Driver can get the prefix from software TLS



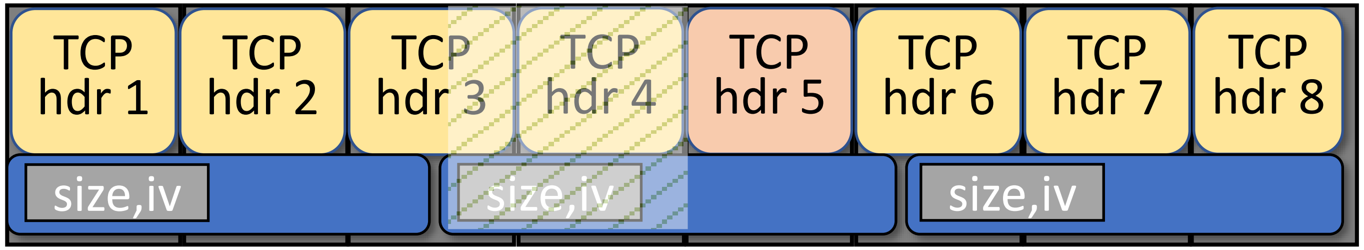
NIC contexts

Static state
- crypto keys

Dynamic state
- expected TCP seq
- current msg offset
- current msg size
- current msg IV
- msg ICV state

Transmit offload out-of-sequence

- Wrong dynamic NIC context state
- Context recovery needs only the message prefix
  - Driver can get the prefix from software TLS



NIC contexts

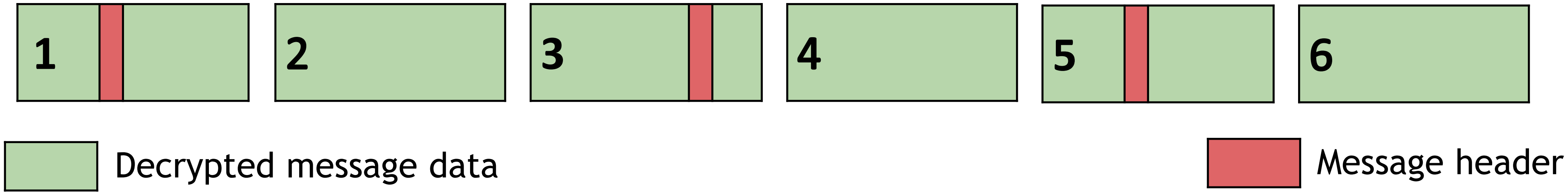
Static state
- crypto keys

Dynamic state
- expected TCP seq
- current msg offset
- current msg size
- current msg IV
- msg ICV state

- Reuse TCP transmit buffer for storing data
  - TCP ACKs release data in TLS record granularity

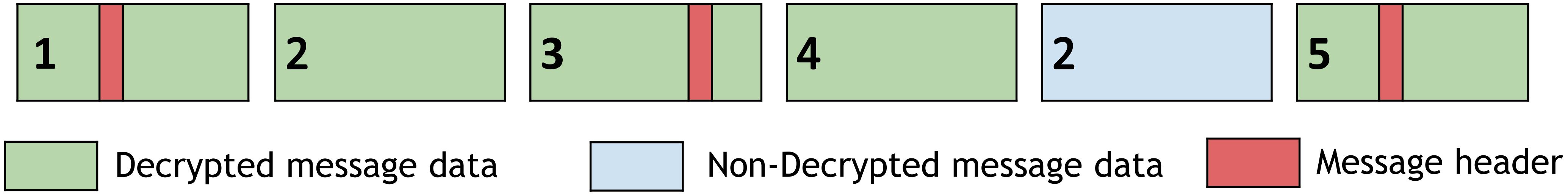
Receive offload in-sequence

- NIC offload Implementation is simple
  - Incrementally offload using NIC contexts
- Hardware reports one bit per packet
  - is packet decrypted and authenticated?



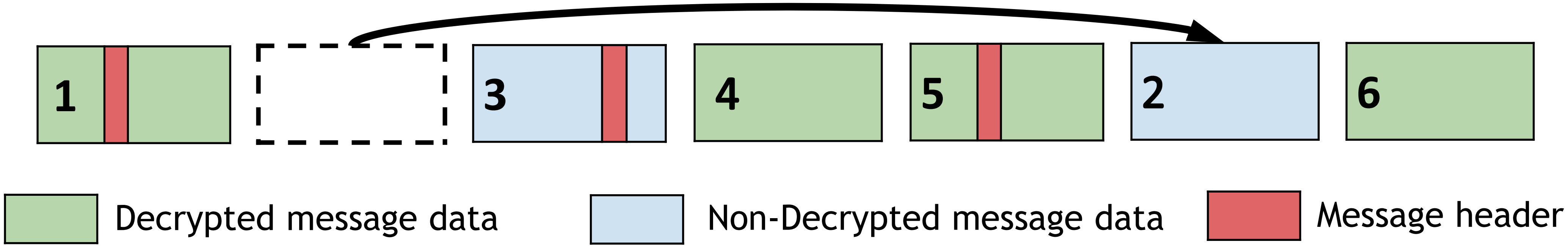
Receive offload retransmission

- Retransmissions bypass offload
  - Software fallback



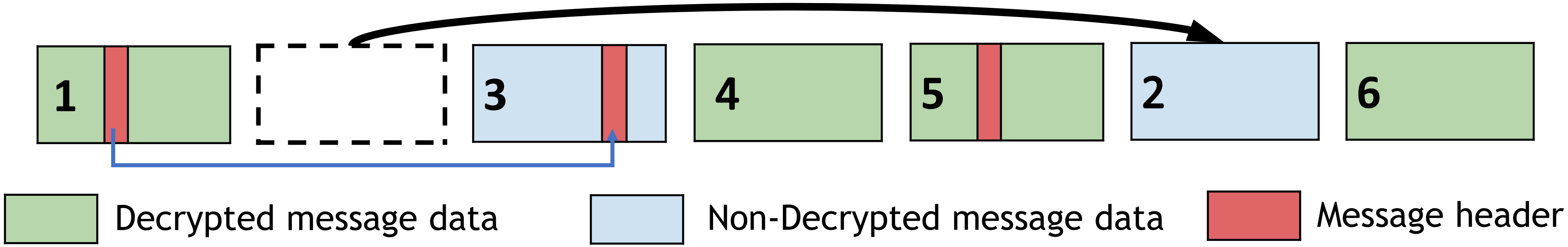
Receive offload data reordering

- Record data reordering
  - Hardware skips to the next record
  - Continues offloading



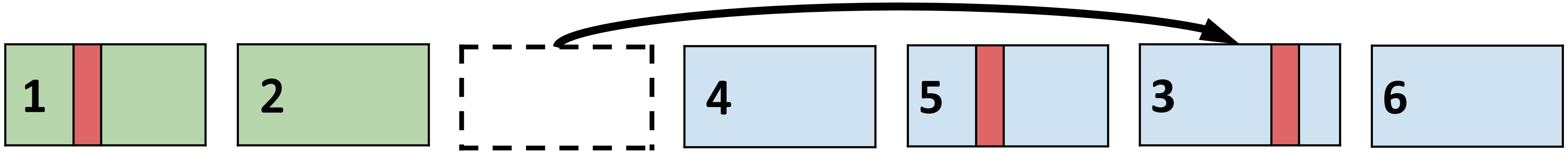
Receive offload data reordering

- Record data reordering
  - Hardware skips to the next record
  - Continues offloading



Receive offload header reordering

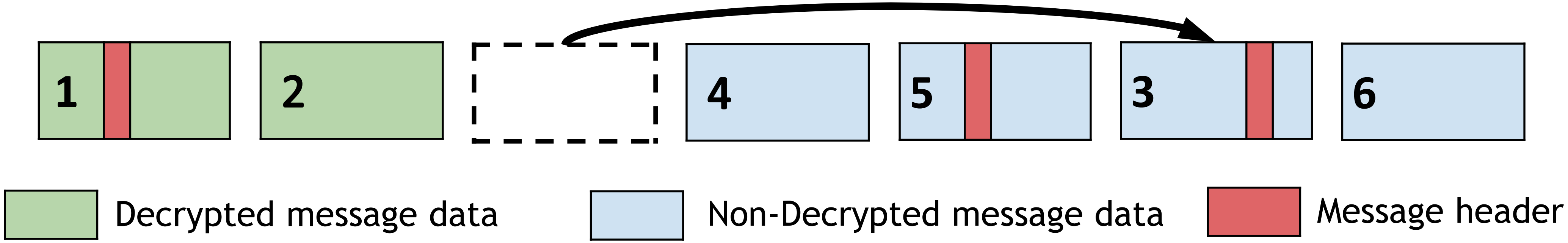
- Record header reordering
  - Stops hardware NIC offloading
  - Software must recover NIC context to continue



🟩 Decrypted message data ▭ Non-Decrypted message data 🟥 Message header

Receive offload recovery problem

- NIC context recovery on receive is non-trivial:
  - Stopping packets to recover NIC context is impossible
    - Packets keep coming
  - Software alone cannot recover during traffic
    - Need to combine software and hardware

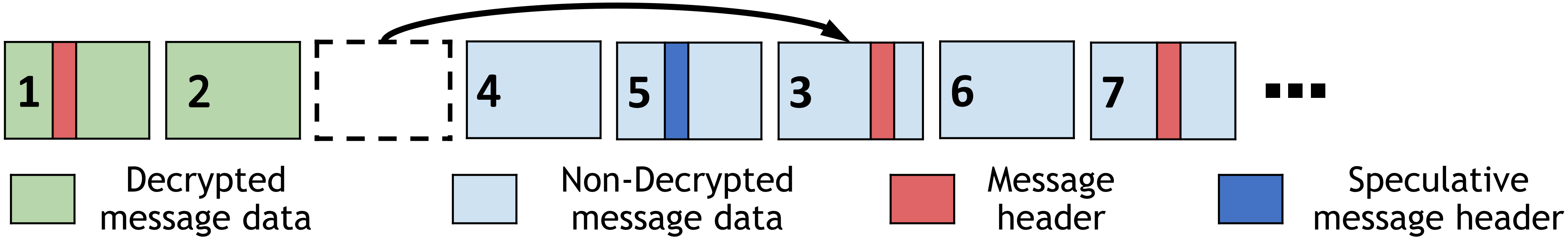


Receive offload recovery solution

NIC context recovery relies on:

(1) Speculatively finding TLS message header magic pattern

- TLS message type and version (0x170303)



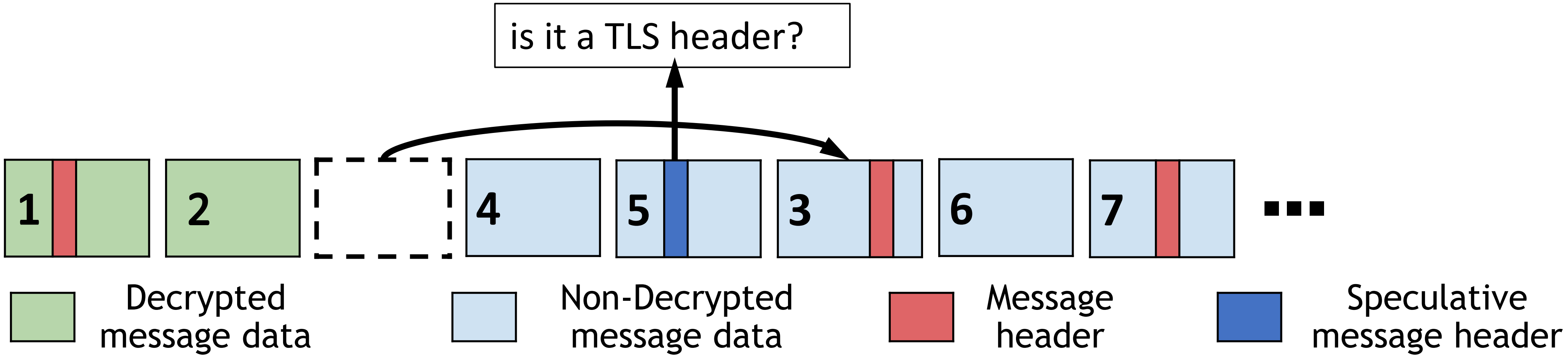
Receive offload recovery solution

NIC context recovery relies on:

(1) Speculatively finding TLS message header magic pattern

- TLS message type and version (0x170303)

(2) Requesting software to confirm that this is indeed a TLS header, while



Receive offload recovery solution

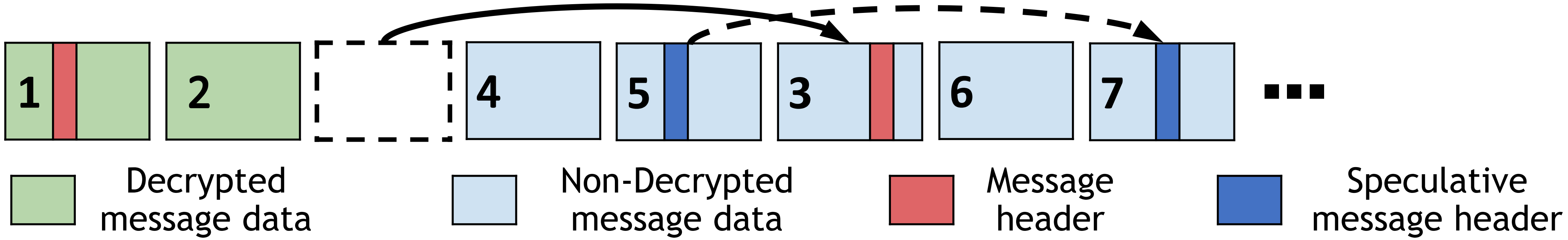
NIC context recovery relies on:

(1) Speculatively finding TLS message header magic pattern

- TLS message type and version (0x170303)

(2) Requesting software to confirm that this is indeed a TLS header, while

(3) Tracking subsequent messages using the message header’s length field



Receive offload recovery solution

NIC context recovery relies on:

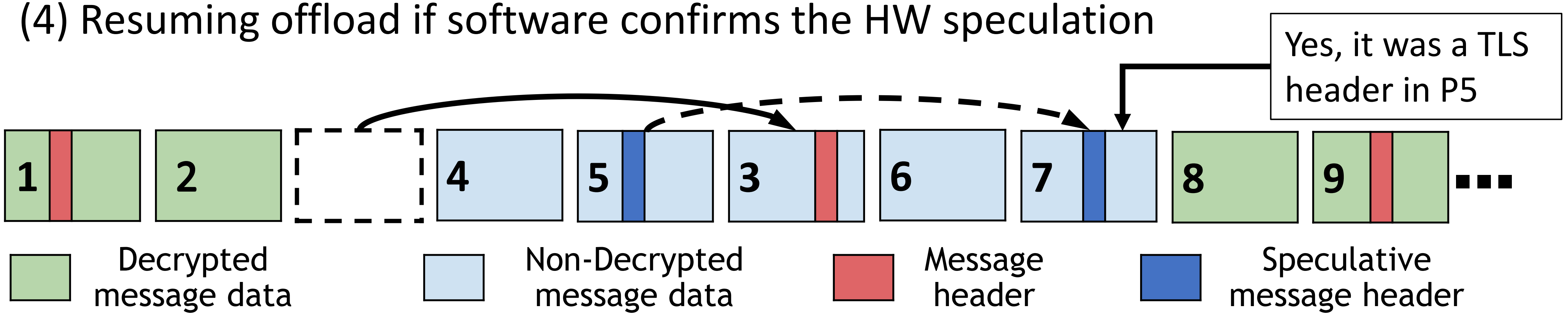
(1) Speculatively finding TLS message header magic pattern

- TLS message type and version (0x170303)

(2) Requesting software to confirm that this is indeed a TLS header, while

(3) Tracking subsequent messages using the message header's length field

(4) Resuming offload if software confirms the HW speculation



Yes, it was a TLS header in P5
1
2
4
5
3
6
7
8
9
...
Decrypted message data
Non-Decrypted message data
Message header
Speculative message header

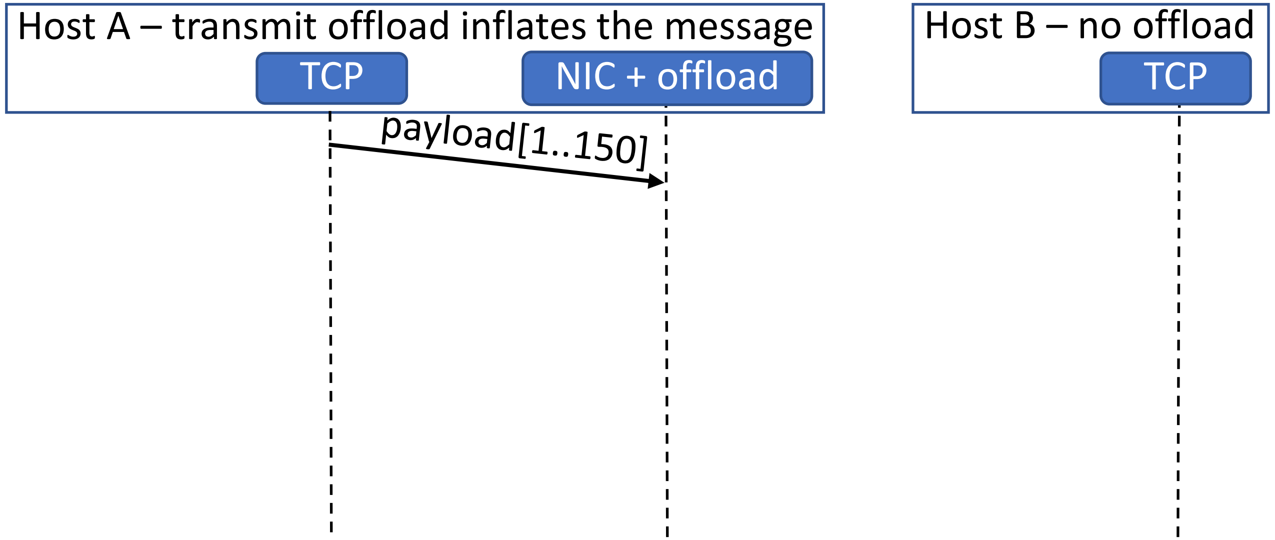
Autonomous offload properties

- What computations are autonomously offloadable?
  - Most computations, but not all

- What L5Ps are autonomously offloadable?
  - Many L5Ps, but not all

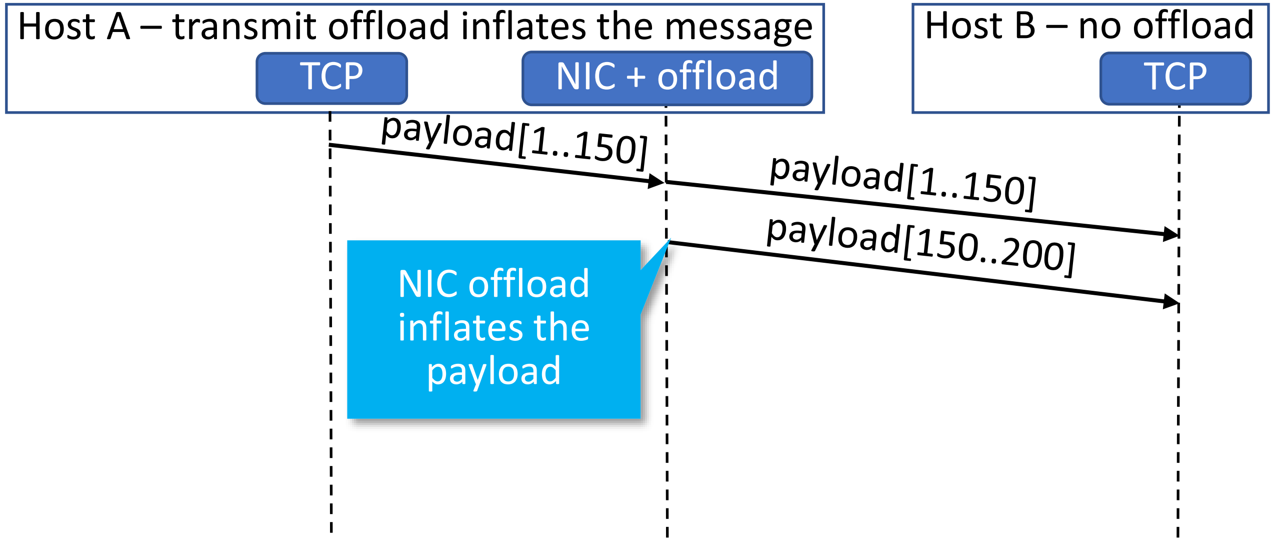
When computation is autonomously offloadable?

- On transmit, it must be size-preserving
  - This precludes transmit compression offloads



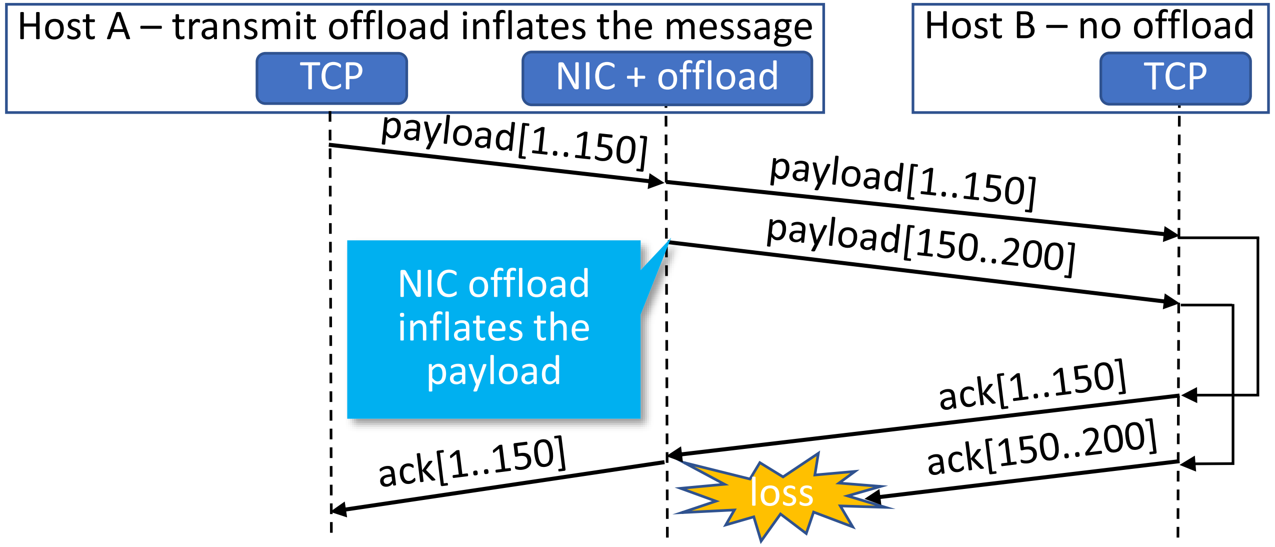
When computation is autonomously offloadable?

- On transmit, it must be size-preserving
  - This precludes transmit compression offloads



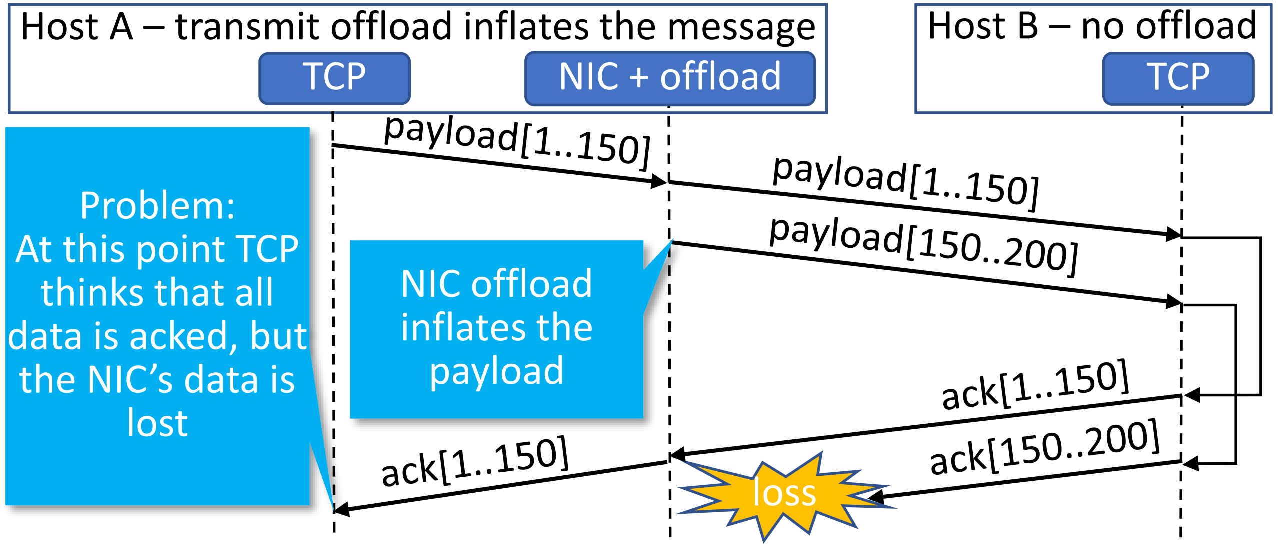
When computation is autonomously offloadable?

- On transmit, it must be size-preserving
  - This precludes transmit compression offloads



When computation is autonomously offloadable?

- On transmit, it must be size-preserving
  - This precludes transmit compression offloads


Host A – transmit offload inflates the message
TCP
NIC + offload
Host B – no offload
TCP
payload[1..150]
payload[1..150]
payload[150..200]
Problem:
At this point TCP thinks that all data is acked, but the NIC's data is lost
NIC offload inflates the payload
ack[1..150]
ack[150..200]
ack[1..150]
loss

When computation is autonomously offloadable?

- On transmit, it must be size-preserving
- It is computable on TCP packets of any size
  - This precludes some block ciphers (AES-CBC) which operate on 16B blocks

When computation is autonomously offloadable?

- On transmit, it must be size-preserving
- It is computable on TCP packets of any size
- It uses constant-size message-independent state
  - It cannot depend on all stream payload
  - It can depend on message metadata (message sequence)

When computation is autonomously offloadable?

- On transmit, it must be size-preserving
- It is computable on TCP packets of any size
- It uses constant-size message-independent state

- Many computations fit this requirement
  - encryption
  - decryption
  - digest
  - copy
  - pattern matching

When L5Ps are autonomously offloadable?

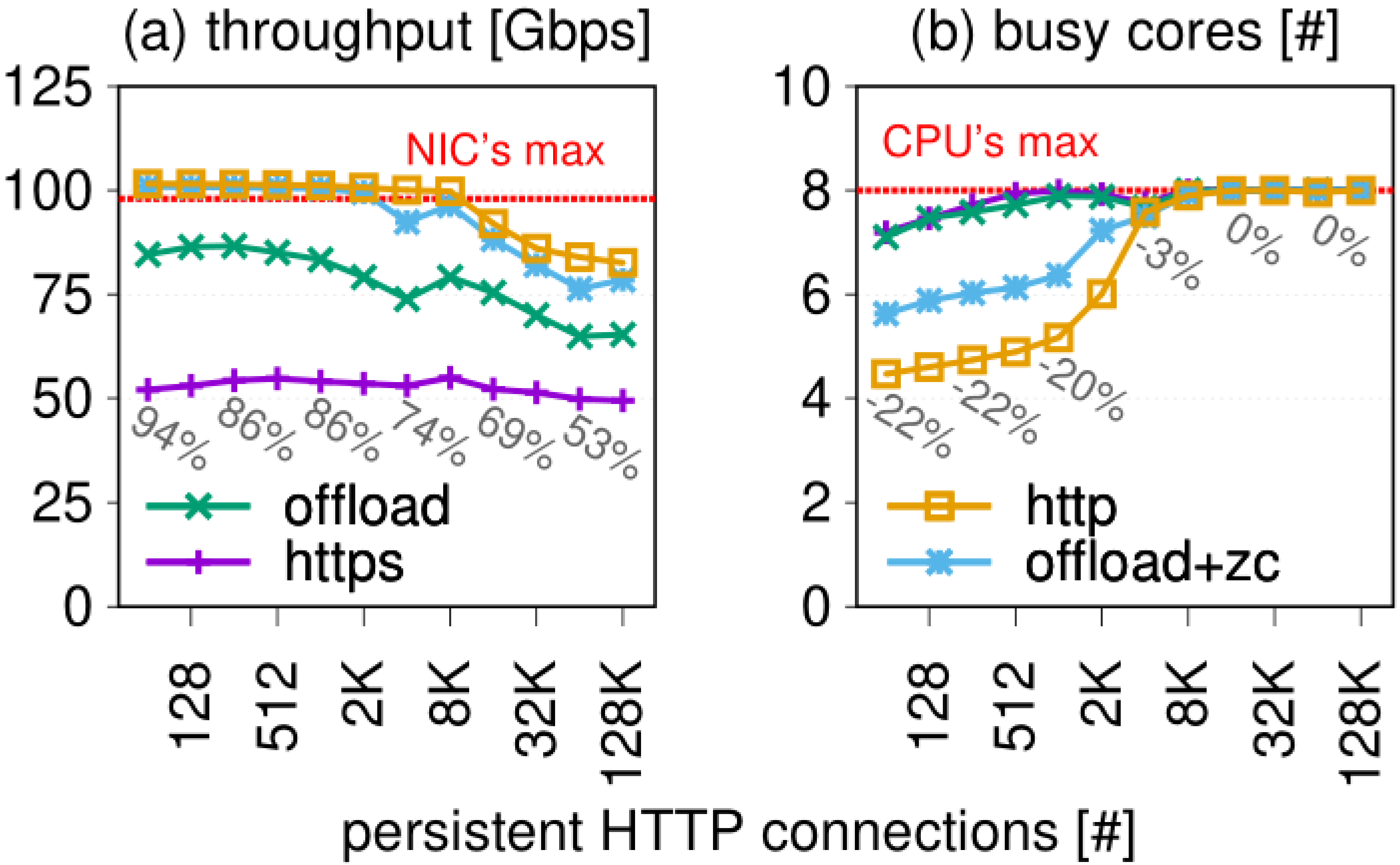
- The protocol message header must contain:
    1. Message length field
    2. Plaintext magic pattern (version/opcode)
- Together these enable hardware-driven NIC context reconstruction

- Many protocols fit this requirement
    - http/2
    - memcached
    - iscsi
    - smb
    - thrift
    - grpc
    - nbd

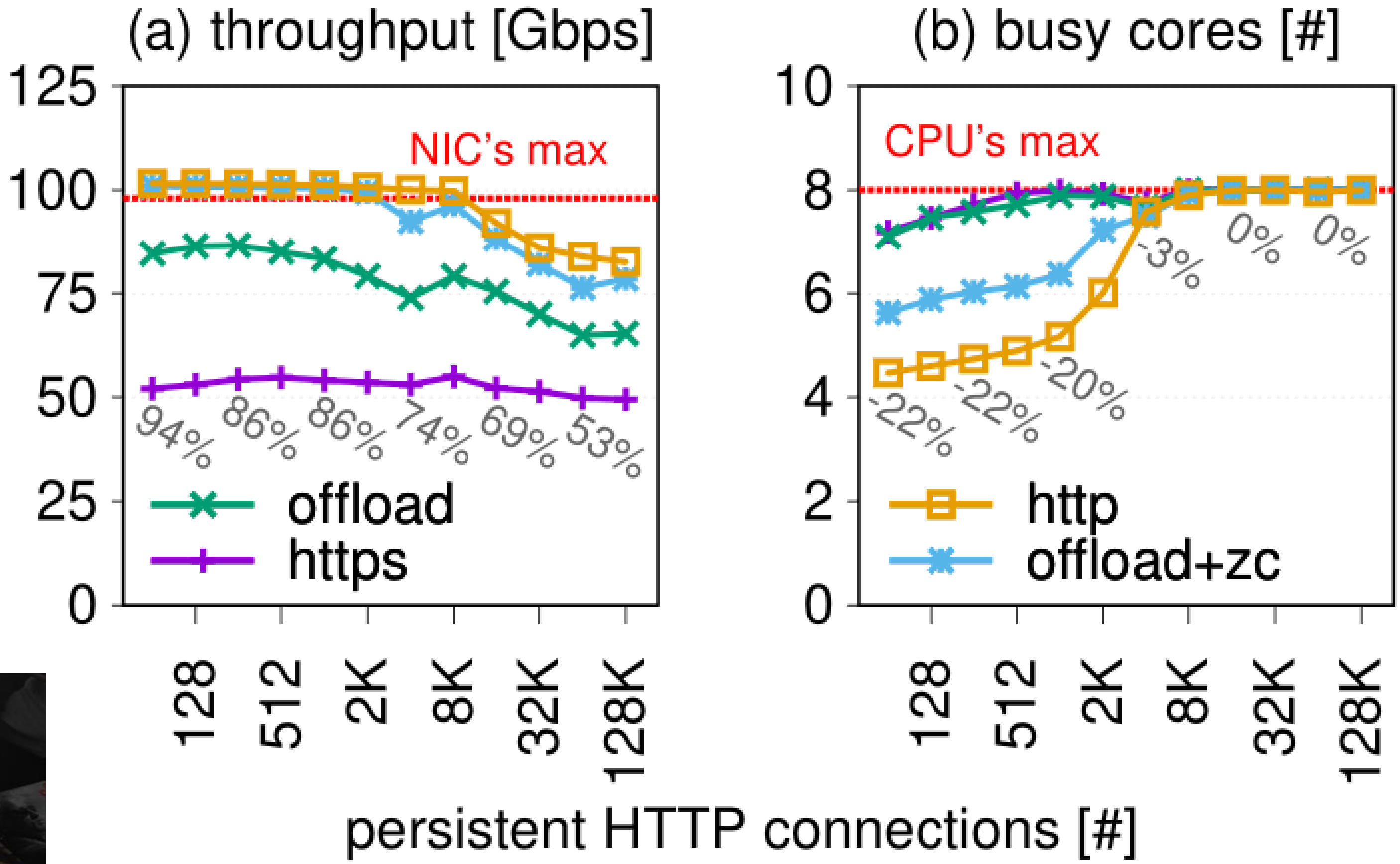
Implementation

- TLS crypto offload is available in Mellanox ConnectX6-Dx NICs:
  - OpenSSL: 1381 LoC (available upstream)
  - Linux kernel: 2223 LoC (available upstream)
  - Mellanox NIC driver: 2095 LoC (available upstream)

TLS sendfile scalability



TLS sendfile scalability



Serving Netflix Video Traffic at ~~400Gb/s~~ 800Gb/s and Beyond

Drew Gallatin
NAB Show, April 2022

Conclusion

- Autonomous NIC offloads is a framework for accelerating L5P computations efficiently while cooperating with software TCP/IP
- Autonomous NIC offloads is applicable to most protocols and computations
- Evaluation shows our approach improves throughput by up to 3.3x, and reduces CPU utilization by up to 60% and latency by up to 30%

Thank you

Contact info:

Boris Pismenny

borispi@cs.technion.ac.il