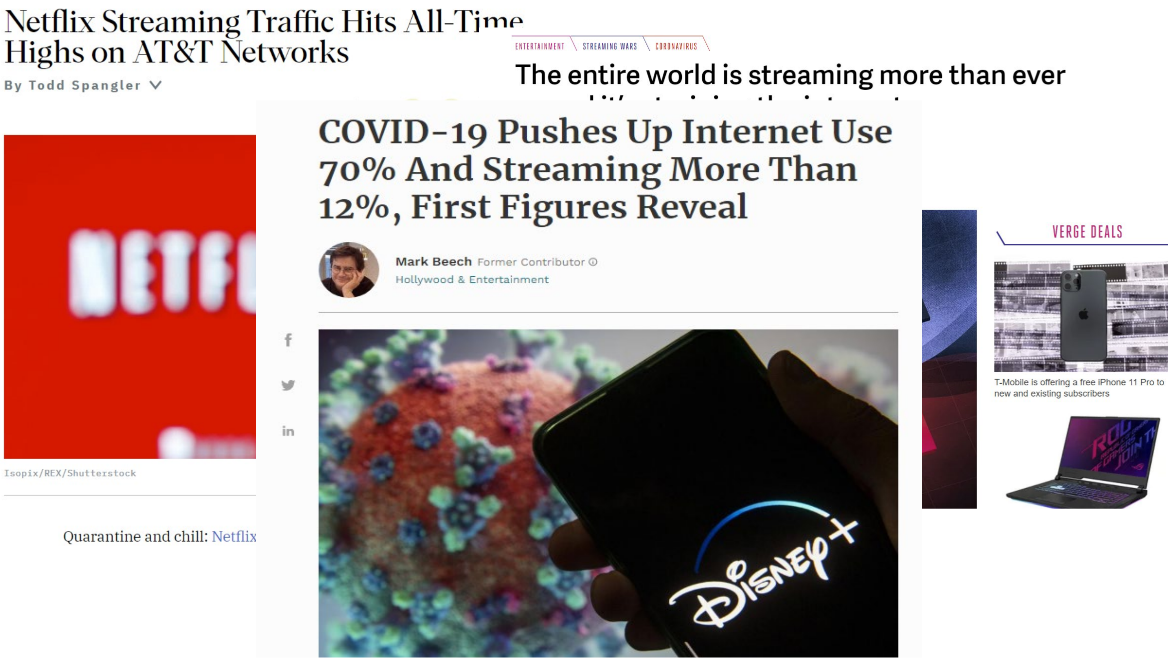
Rearchitecting the TCP Stack for I/O-Offloaded Content Delivery

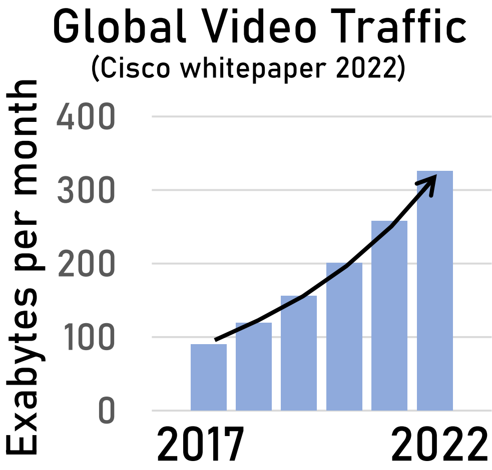
Taehyun Kim, Deondre Martin Ng, Junzhi Gong*, Youngjin Kwon, Minlan Yu*, KyoungSoo Park

KAIST & *Harvard University

Increasing Demand for High-quality Video Streaming

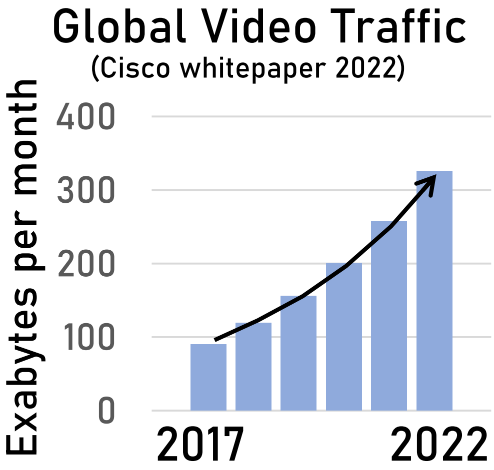
- COVID-19 pandemic (2020~) – “more” rapid increase in video traffic





Increasing Demand for High-quality Video Streaming

- COVID-19 pandemic (2020~) – "more" rapid increase in video traffic



CDN server performance is critical for cost-effective service

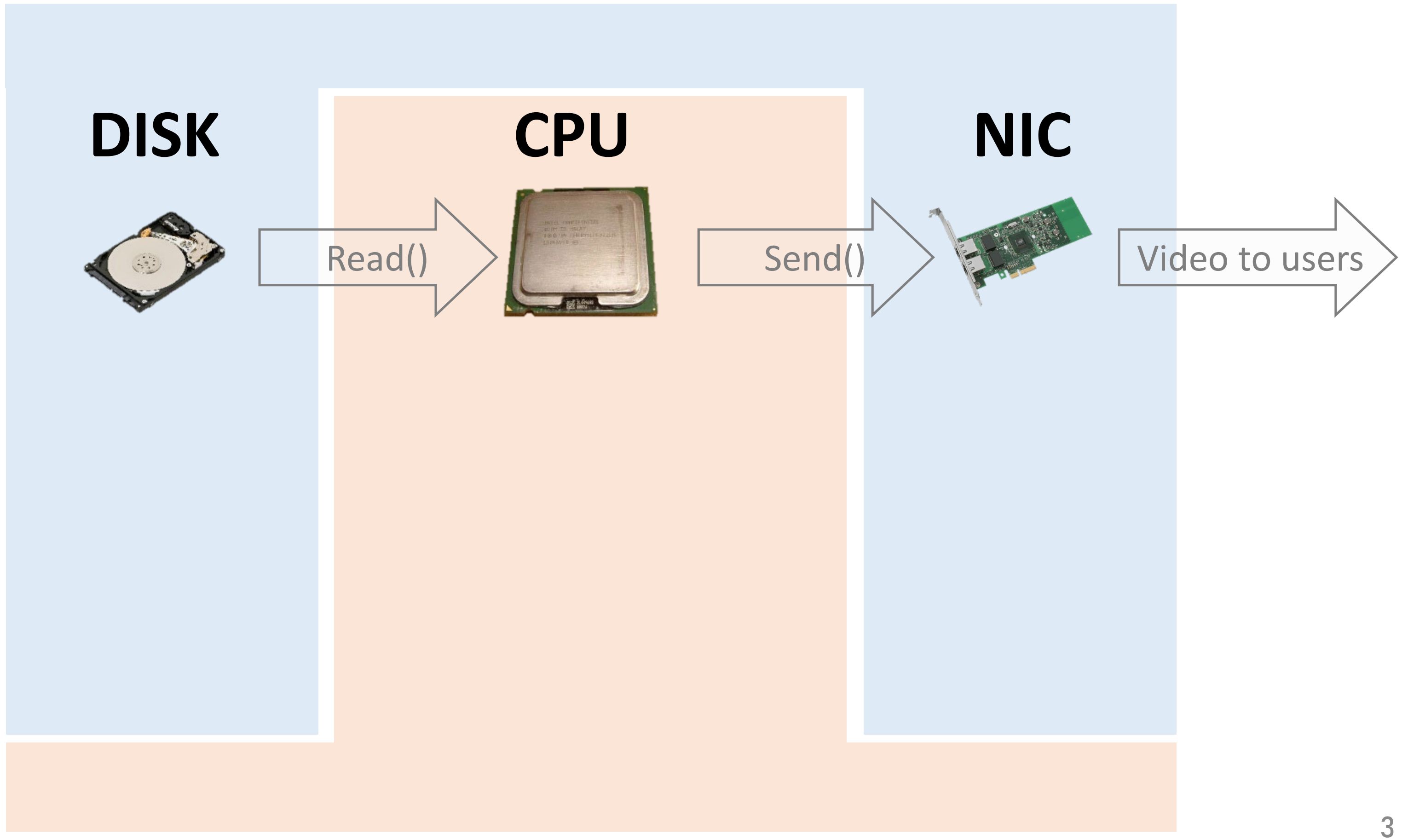
Computing Hardware Development Trend

DISK

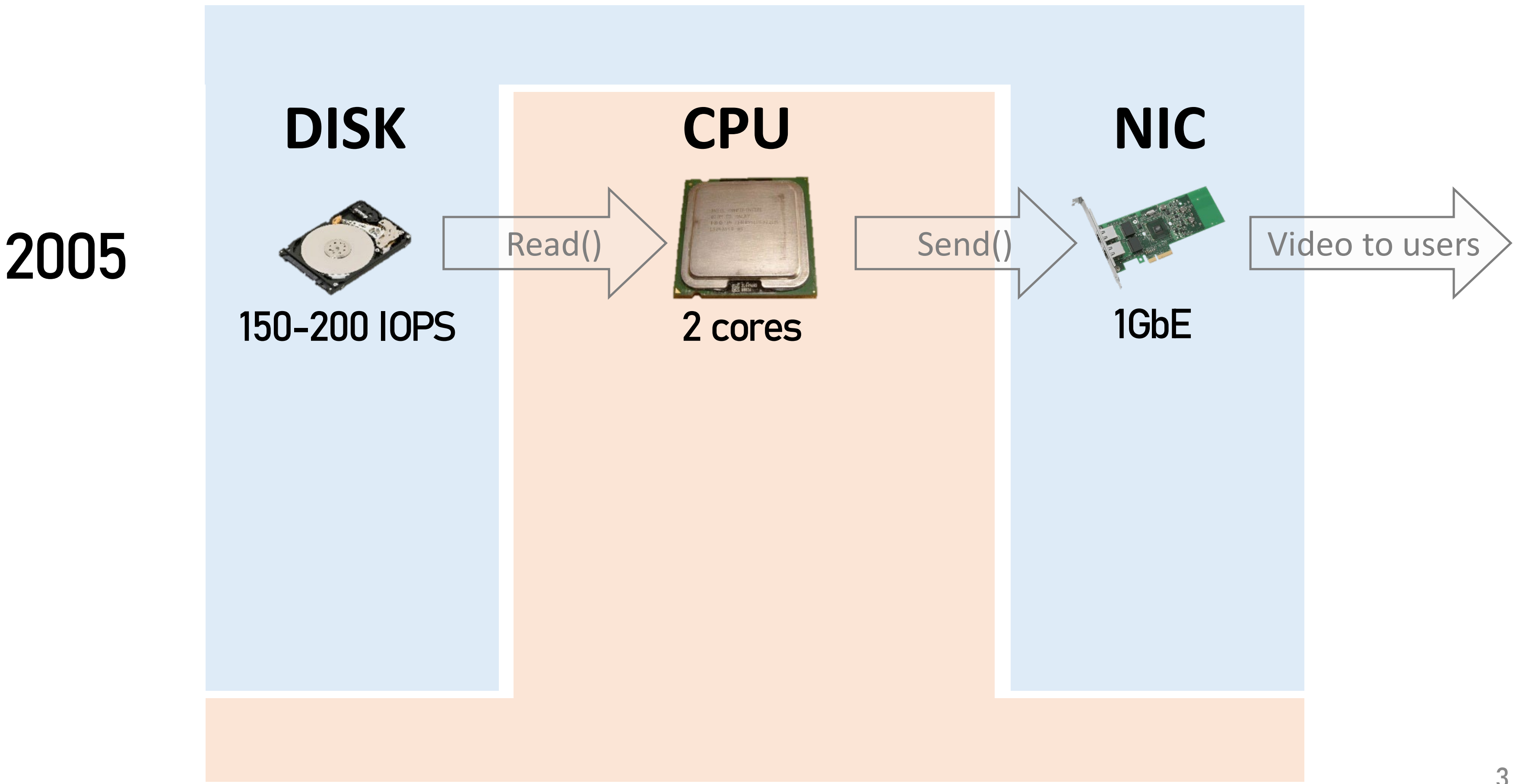
CPU

NIC

Computing Hardware Development Trend



Computing Hardware Development Trend



Computing Hardware Development Trend

+ Rapid performance improvement with **IO devices**

2005

DISK

Read()

150-200 IOPS

CPU

Send()

2 cores

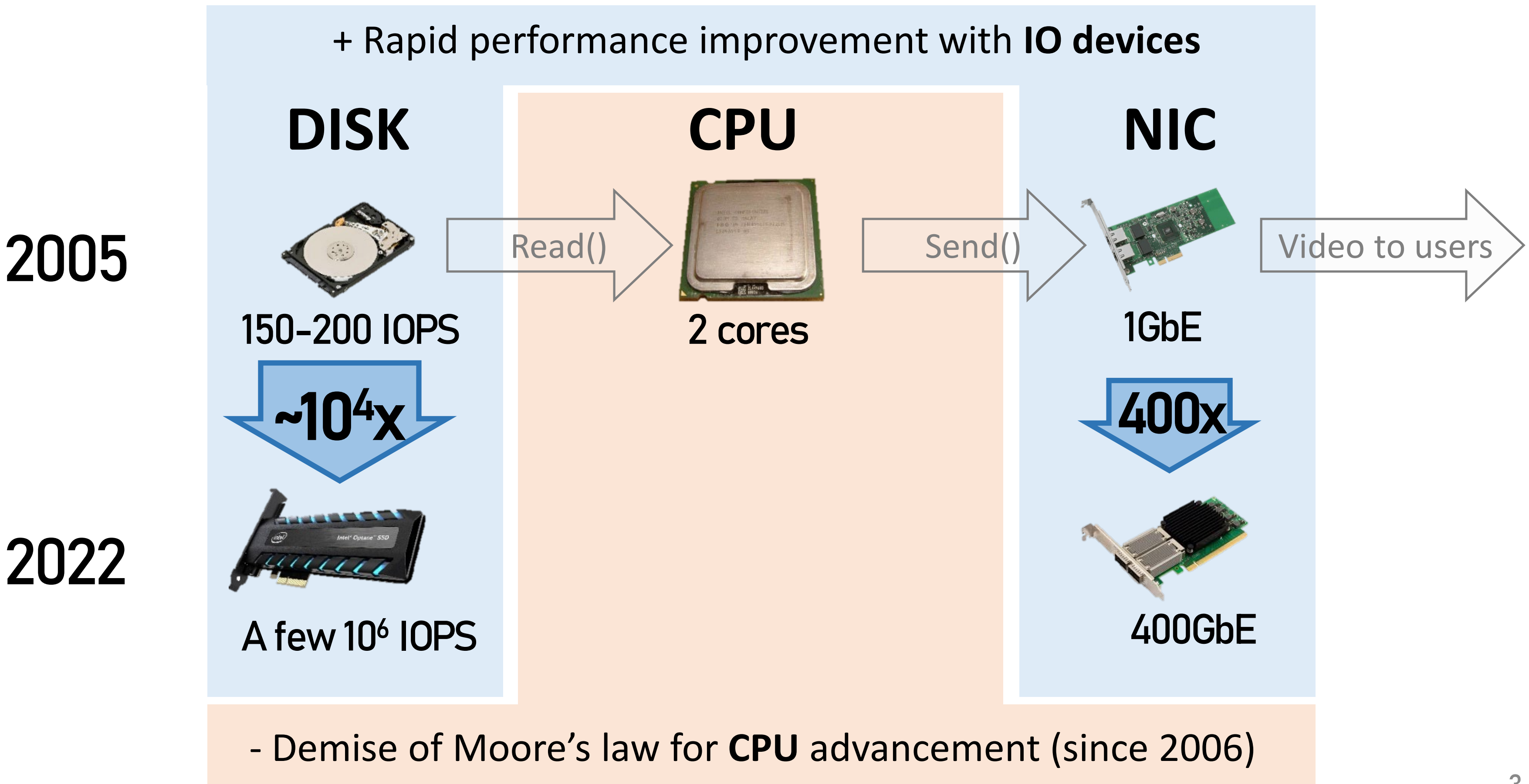
NIC

Video to users

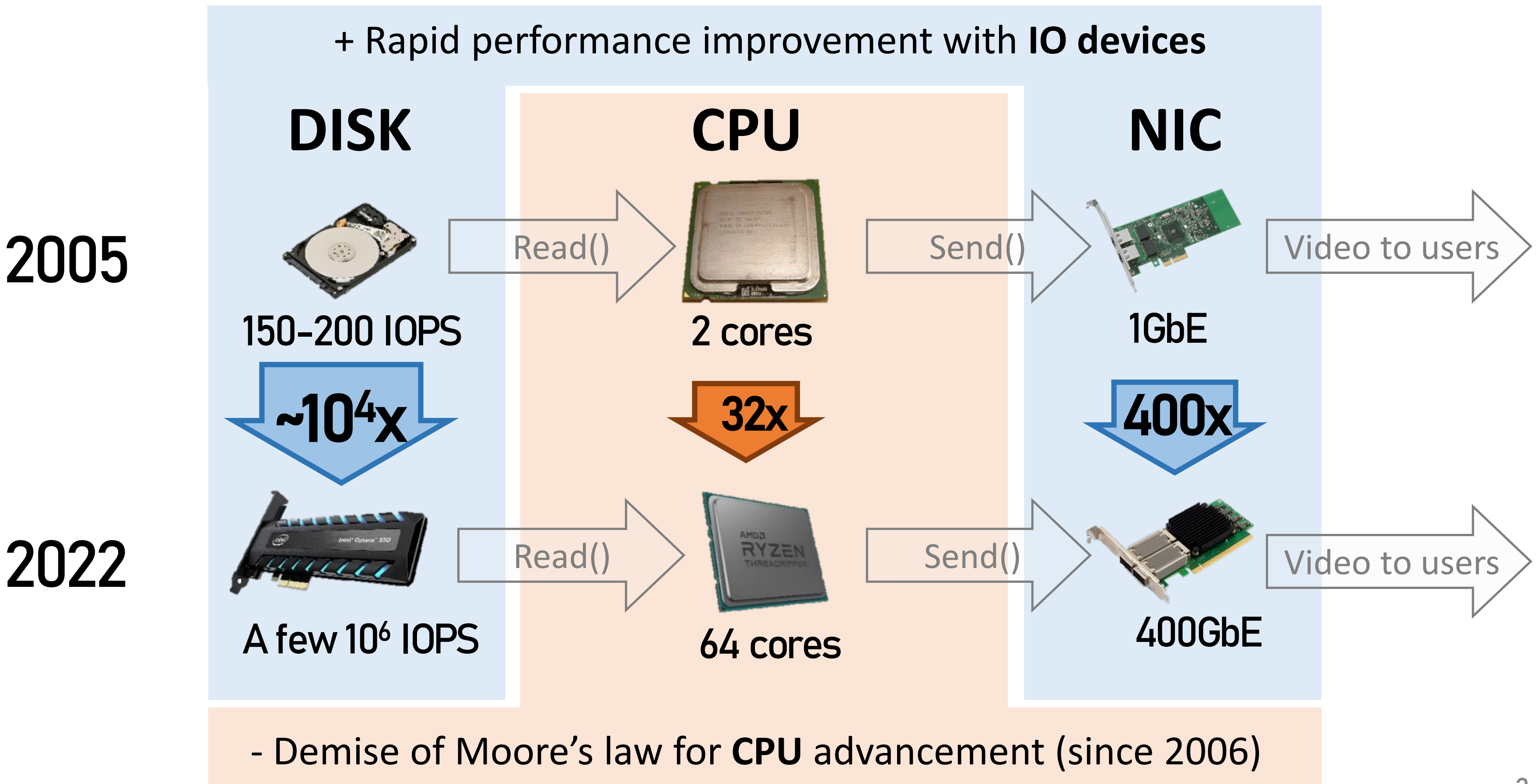
1GbE

- Demise of Moore's law for **CPU** advancement (since 2006)

Computing Hardware Development Trend



Computing Hardware Development Trend



CPU Consumption for HTTP Video Streaming Server

- Typical server operations
  - Read an HTTP request
  - Read a file chunk for the request
  - Send the response
- Result

Benchmark setting
- nginx (v.1.80.0) on Linux
- 300KB files on 4x Optane NVMe
- 100Gbps NIC
- Single core of Xeon Silver 4210

CPU Consumption for HTTP Video Streaming Server

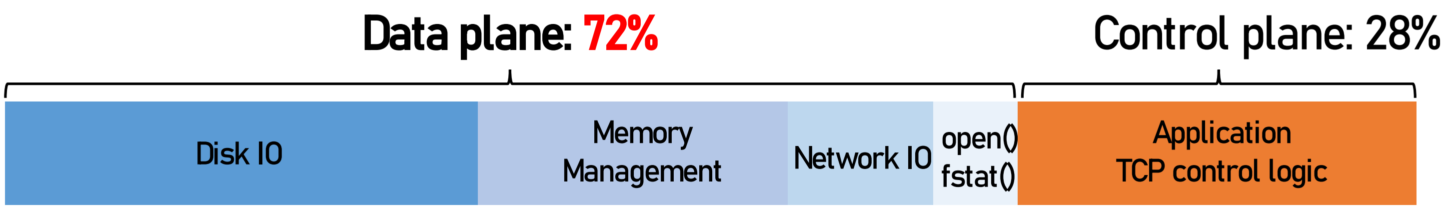
- Typical server operations
  - Read an HTTP request
  - Read a file chunk for the request
  - Send the response
- Result
  - Observation 1: CPU cycles are 100% utilized

Benchmark setting
- nginx (v.1.80.0) on Linux
- 300KB files on 4x Optane NVMe
- 100Gbps NIC
- Single core of Xeon Silver 4210

CPU Consumption for HTTP Video Streaming Server

- Typical server operations
  - Read an HTTP request
  - Read a file chunk for the request
  - Send the response
- Result
  - Observation 1: CPU cycles are 100% utilized
  - Observation 2: data plane dominates CPU consumption

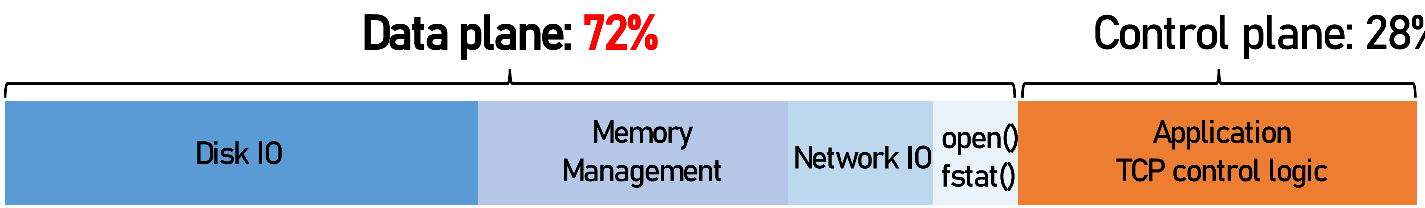
Benchmark setting
- nginx (v.1.80.0) on Linux
- 300KB files on 4x Optane NVMe
- 100Gbps NIC
- Single core of Xeon Silver 4210



CPU Consumption for HTTP Video Streaming Server

- Typical server operations
  - Read an HTTP request
  - Read a file chunk for the request
  - Send the response
- Result
  - Observation 1: CPU cycles are 100% utilized
  - Observation 2: data plane dominates CPU consumption

Benchmark setting
- nginx (v.1.80.0) on Linux
- 300KB files on 4x Optane NVMe
- 100Gbps NIC
- Single core of Xeon Silver 4210



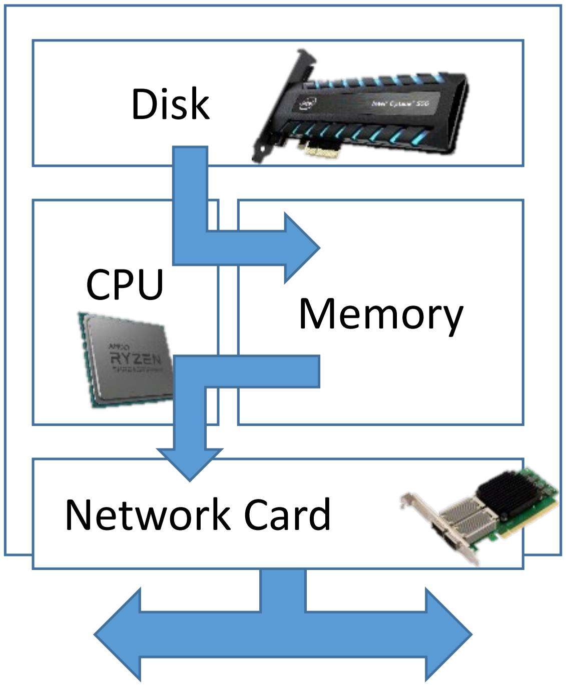
Why CPU bottleneck for IO-bound workload?

Root cause: “CPU-centric” OS Abstractions

- Modern OS designed with “implicit” assumptions
  - CPU is fast but IO devices are slow
  - CPU is never bottleneck for IO-bound workloads

Root cause: “CPU-centric” OS Abstractions

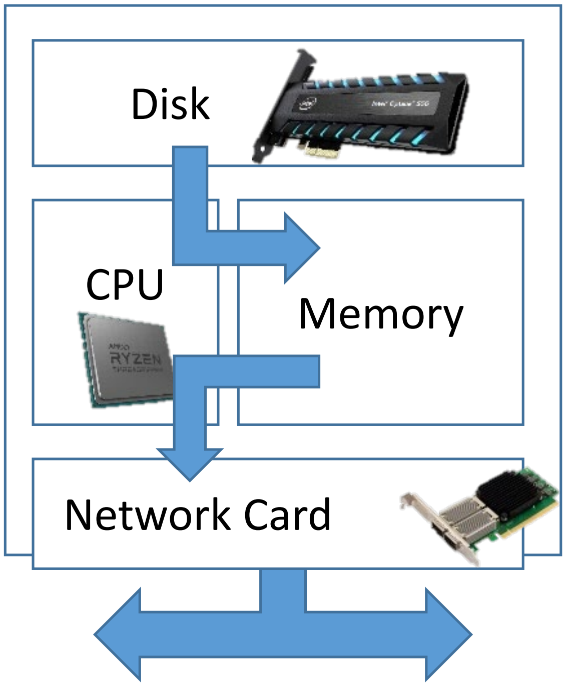
- Modern OS designed with “implicit” assumptions
  - CPU is fast but IO devices are slow
  - CPU is never bottleneck for IO-bound workloads
- Artifact: CPU-centric IO operations
  - size_t read(fd, buf, size); // Disk→Memory
  - size_t write(fd, buf, size); // Memory→IO device
  - All content must be brought to ”main” memory first!
  - Memory (or CPU) becomes the bottleneck



Root cause: “CPU-centric” OS Abstractions

- Modern OS designed with “implicit” assumptions
  - CPU is fast but IO devices are slow
  - CPU is never bottleneck for IO-bound workloads

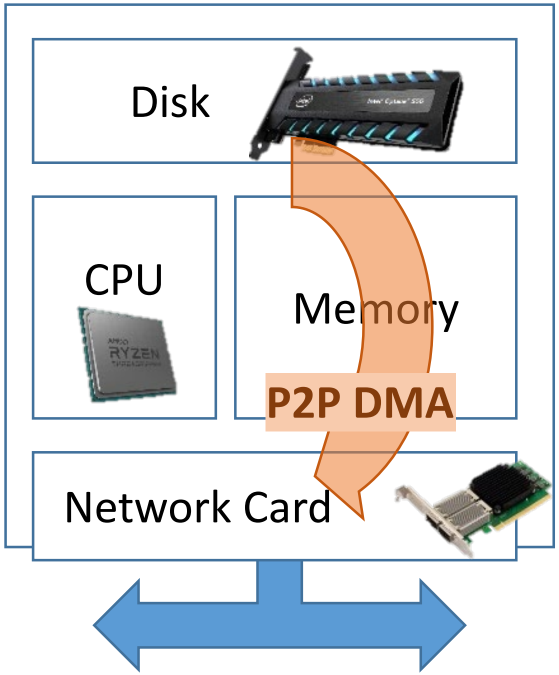
- Artifact: CPU-centric IO operations
  - size_t read(fd, buf, size); // Disk→Memory
  - size_t write(fd, buf, size); // Memory→IO device
  - All content must be brought to ”main” memory first!
  - Memory (or CPU) becomes the bottleneck



How to avoid CPU bottleneck for IO-bound workload?

Opportunity in the Solution Space

- Modern PCIe devices support **P2PDMA**
  - Peer-to-peer DMA without CPU intervention
  - No main memory copy if the DMA devices have memory
- **Programmability** in IO devices
  - SmartNICs & computational SSDs
  - Arm SOC, FPGA-based, or ASIC-based
- Approach: SmartNIC as the hub for NVMe disk IOs

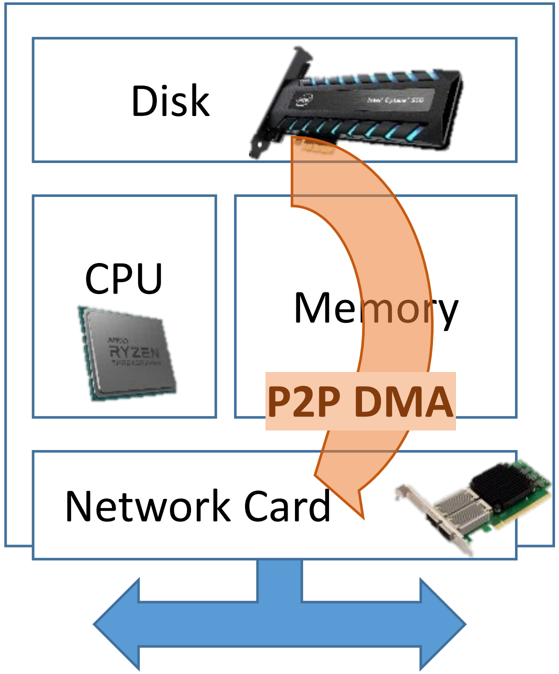


Opportunity in the Solution Space

- Modern PCIe devices support **P2PDMA**
  - Peer-to-peer DMA without CPU intervention
  - No main memory copy if the DMA devices have memory

- **Programmability** in IO devices
  - SmartNICs & computational SSDs
  - Arm SOC, FPGA-based, or ASIC-based

- Approach: SmartNIC as the hub for NVMe disk IOs

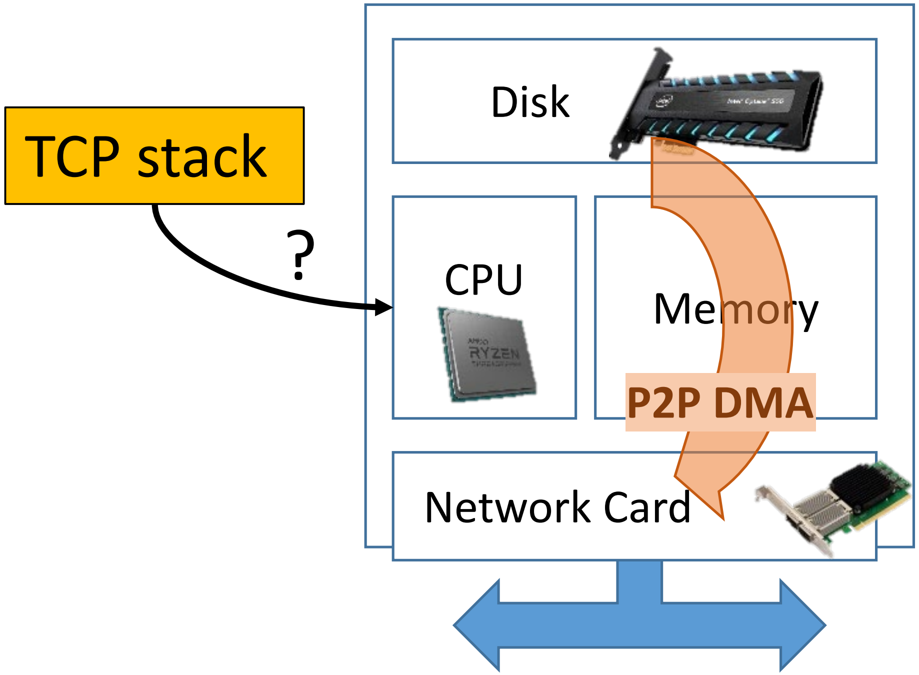


Key design issue: where to place TCP stack?

Challenges in Operating TCP Stack

Option1: TCP stack on CPU side

- No disk content available on CPU



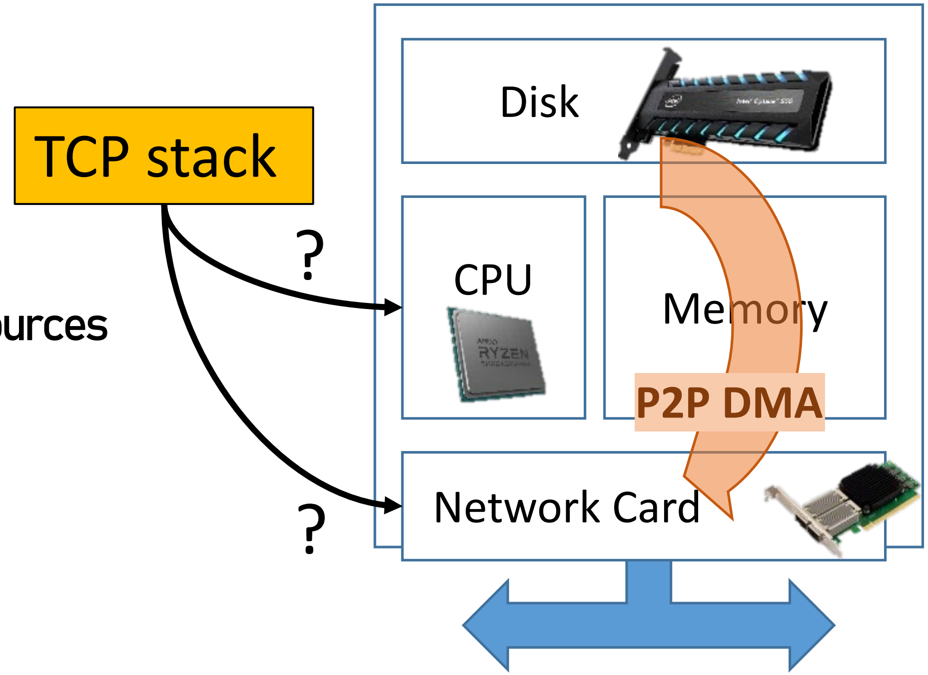
Challenges in Operating TCP Stack

Option1: TCP stack on **CPU side**

- No disk content available on CPU

Option2: TCP stack on **SmartNIC**

- Performance limited by SmartNIC resources



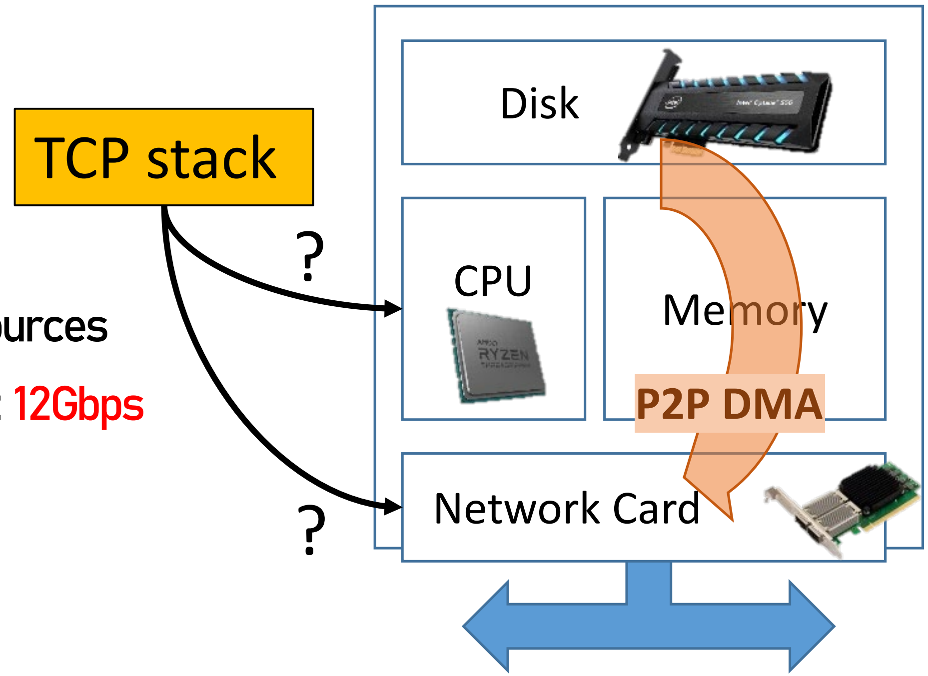
Challenges in Operating TCP Stack

Option1: TCP stack on **CPU side**

- No disk content available on CPU

Option2: TCP stack on **SmartNIC**

- Performance limited by SmartNIC resources
- Lighttpd with Linux TCP on Bluefield-2: 12Gbps
  - On Xeon Silver 4210: 59Gbps



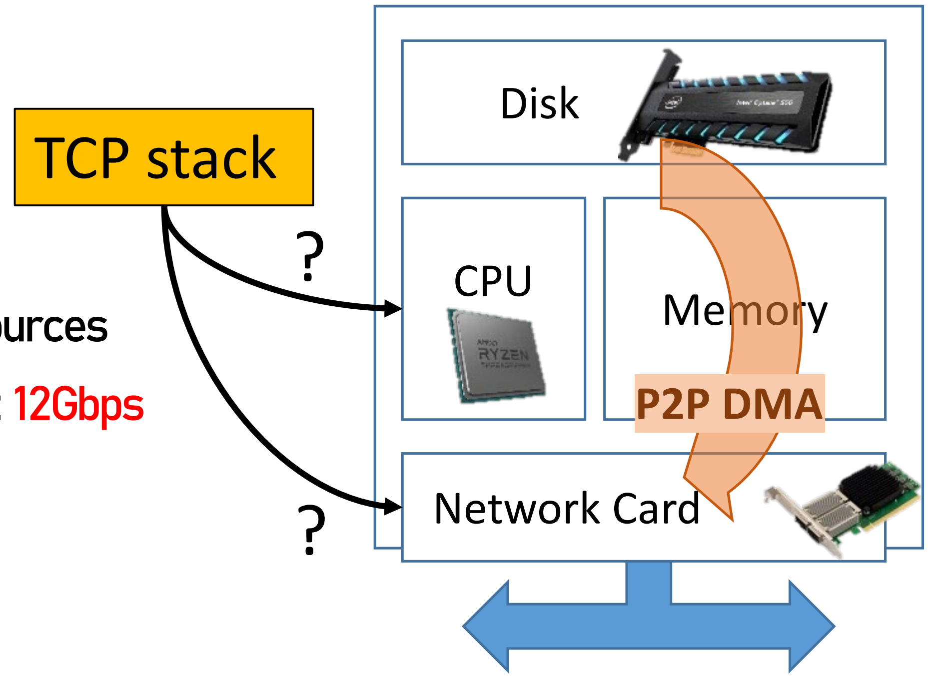
Challenges in Operating TCP Stack

Option1: TCP stack on **CPU side**

- No disk content available on CPU

Option2: TCP stack on **SmartNIC**

- Performance limited by SmartNIC resources
- Lighttpd with Linux TCP on Bluefield-2: 12Gbps
  - On Xeon Silver 4210: 59Gbps

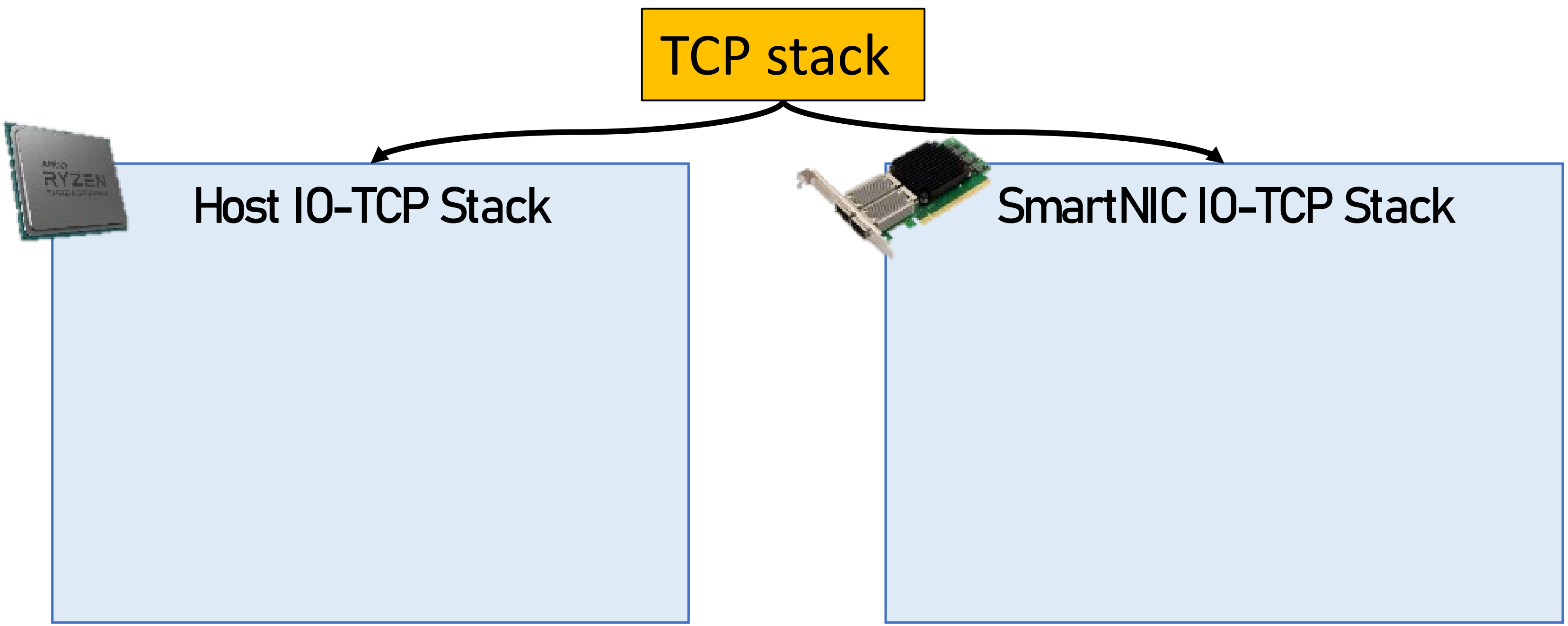


Hybrid solution?

Separate IO-intensive Data Plane from TCP stack?

IO-TCP: Split TCP Stack Architecture for Content Delivery

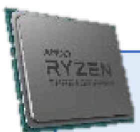
- Separation of TCP control/data planes



IO-TCP: Split TCP Stack Architecture for Content Delivery

- Separation of TCP control/data planes

TCP stack



Host IO-TCP Stack

- Reliable data delivery
- Congestion control
- Flow control
- Header generation

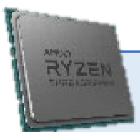
Low latency control logic

SmartNIC IO-TCP Stack

IO-TCP: Split TCP Stack Architecture for Content Delivery

- Separation of TCP control/data planes

TCP stack



Host IO-TCP Stack

- Reliable data delivery
- Congestion control
- Flow control
- Header generation

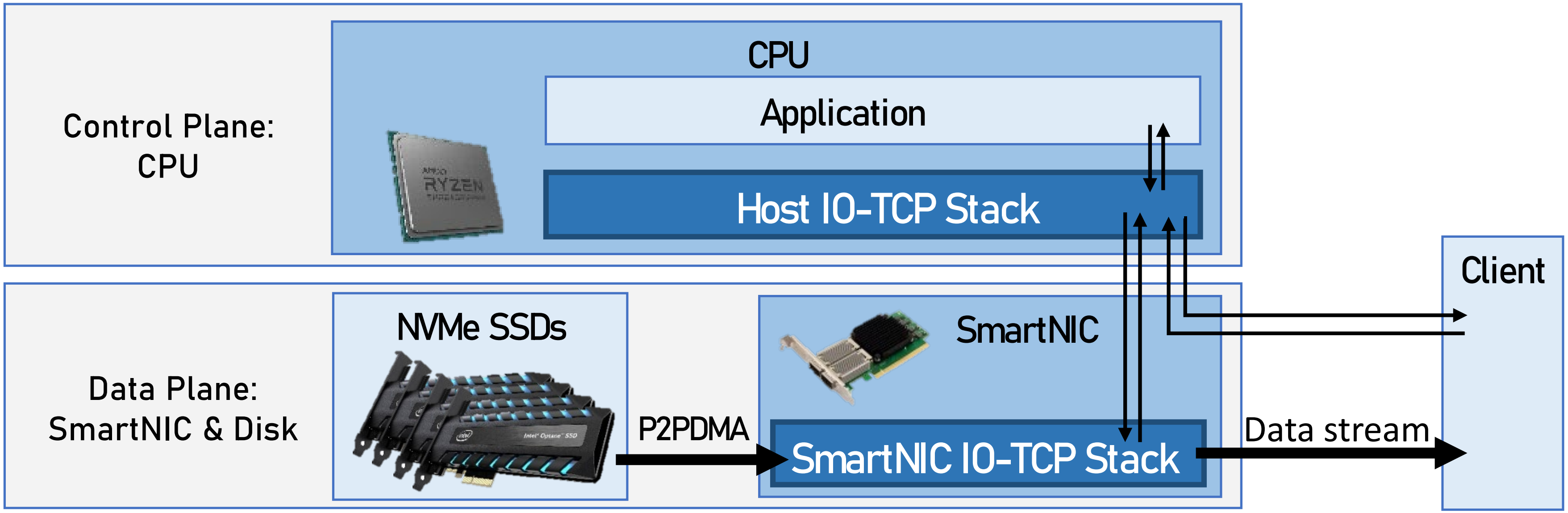
Low latency control logic

SmartNIC IO-TCP Stack

- Read file from disk
- Send packet

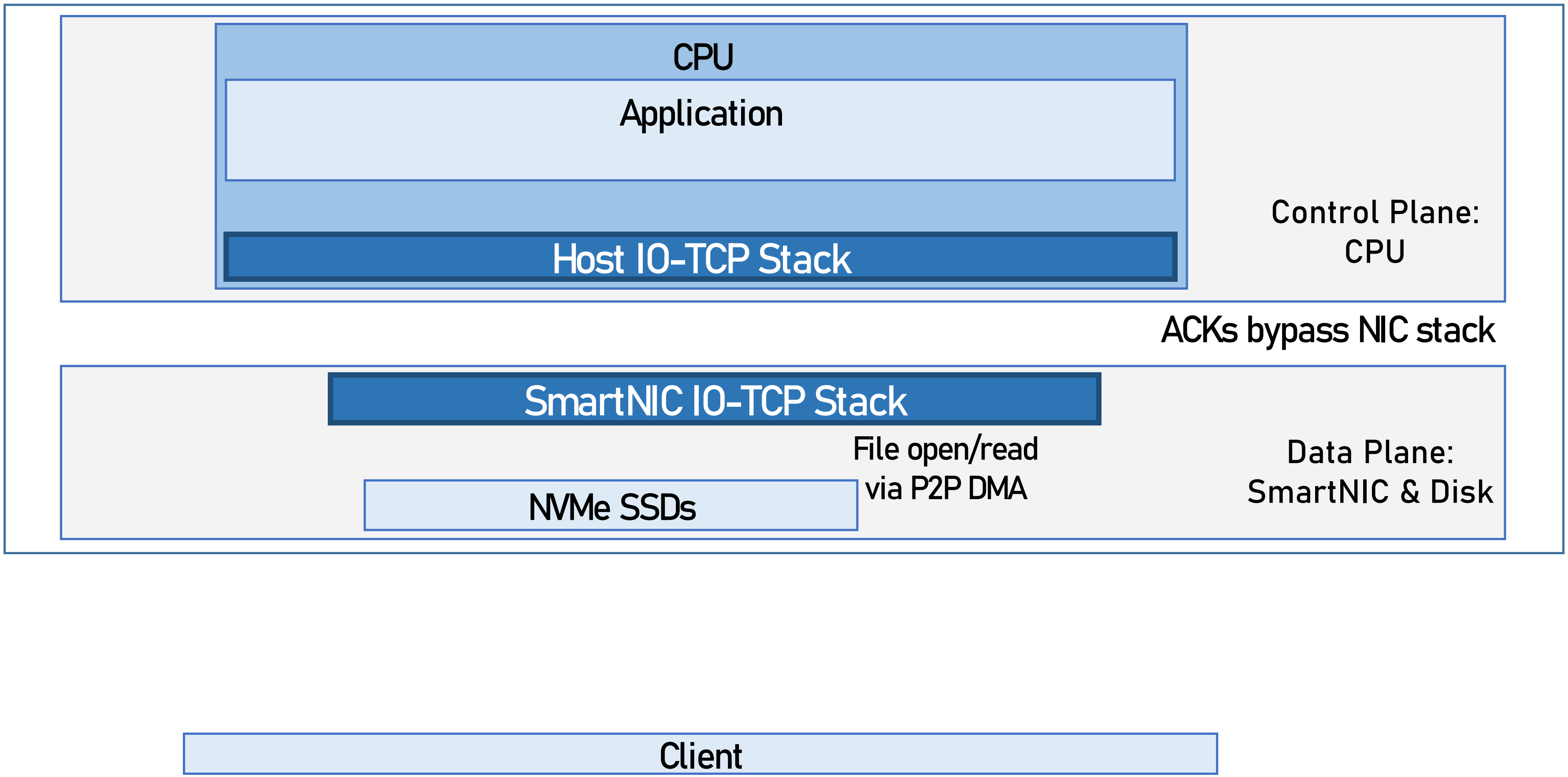
High throughput data operations

IO-TCP Overview

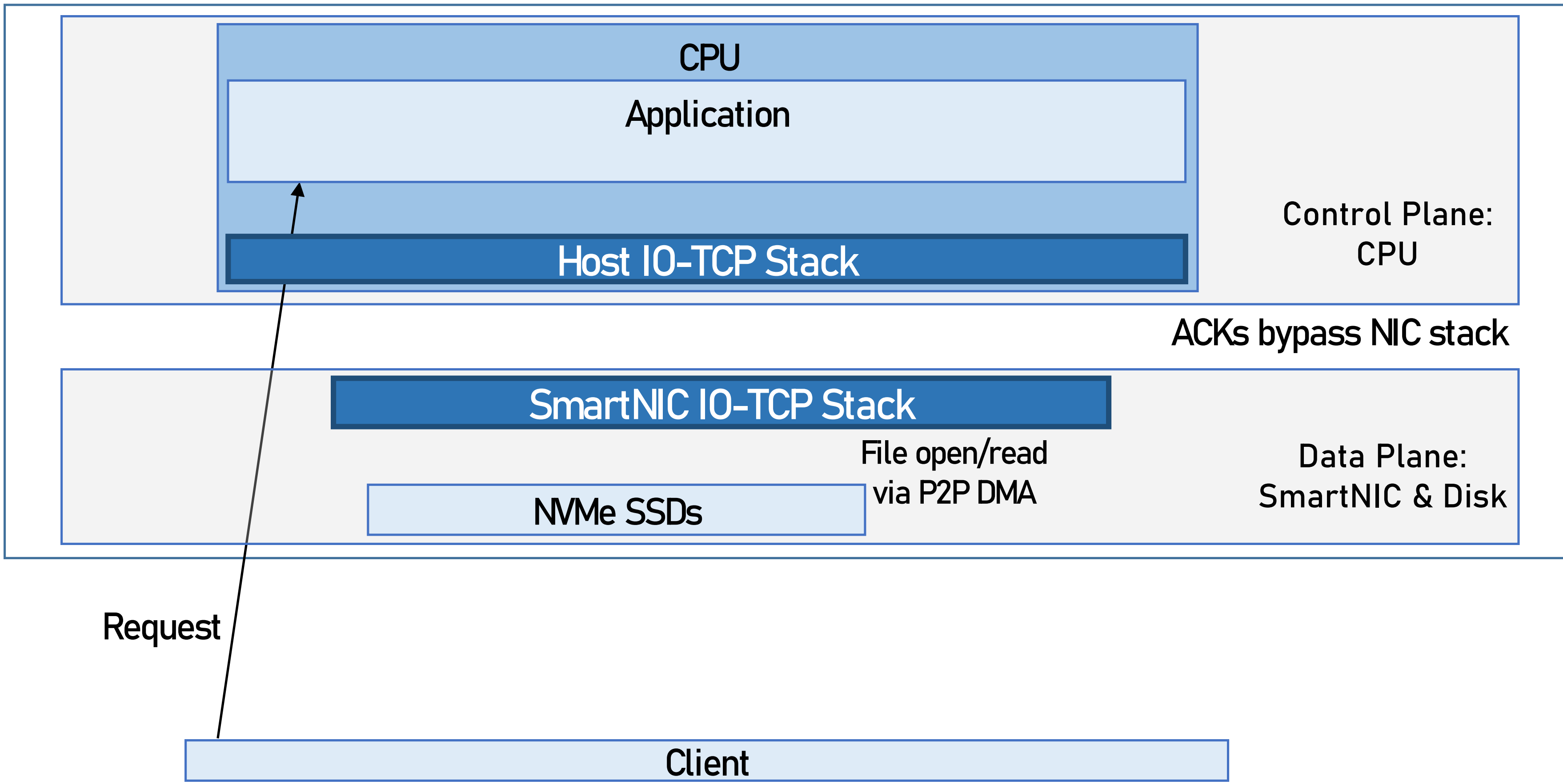


- Provides 4 offload APIs for SmartNIC execution (offload_open(), offload_fstat(), offload_close(), offload_write())
  1. Application calls offload APIs for remote execution
  2. Host sends a special command to SmartNIC for each API
  3. SmartNIC stack performs corresponding IO operations

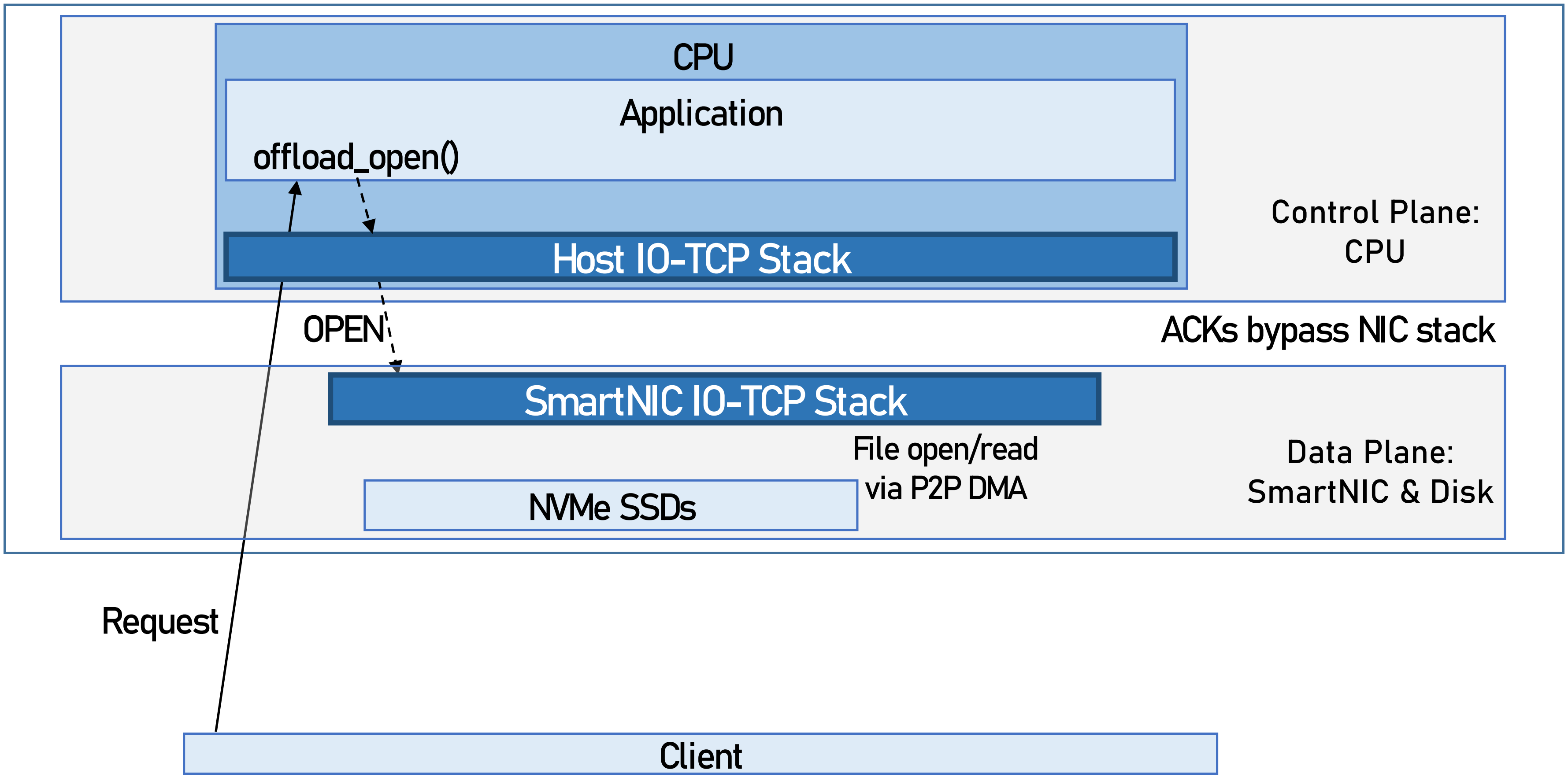
IO-TCP based Web Server Workflow



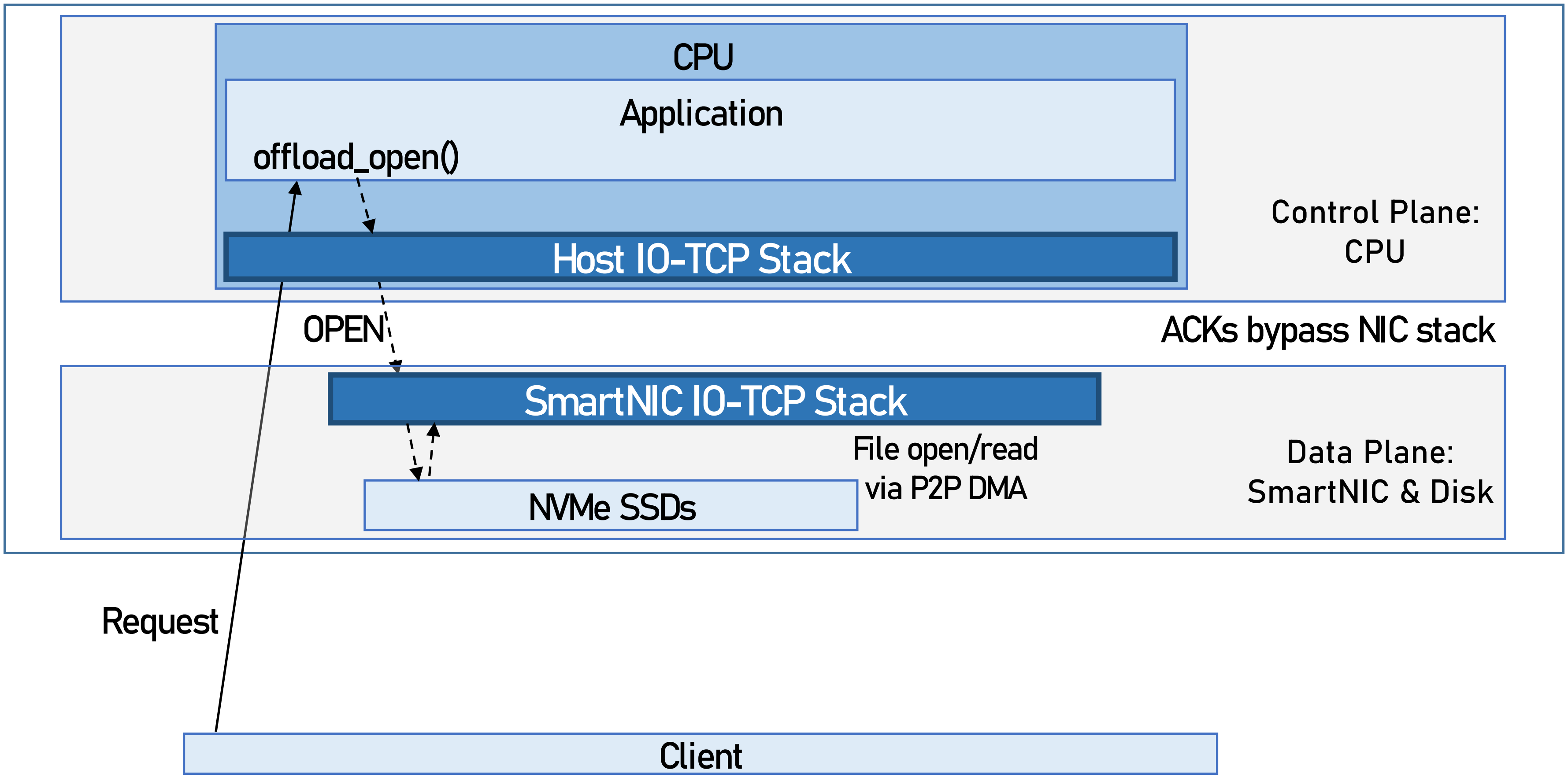
IO-TCP based Web Server Workflow



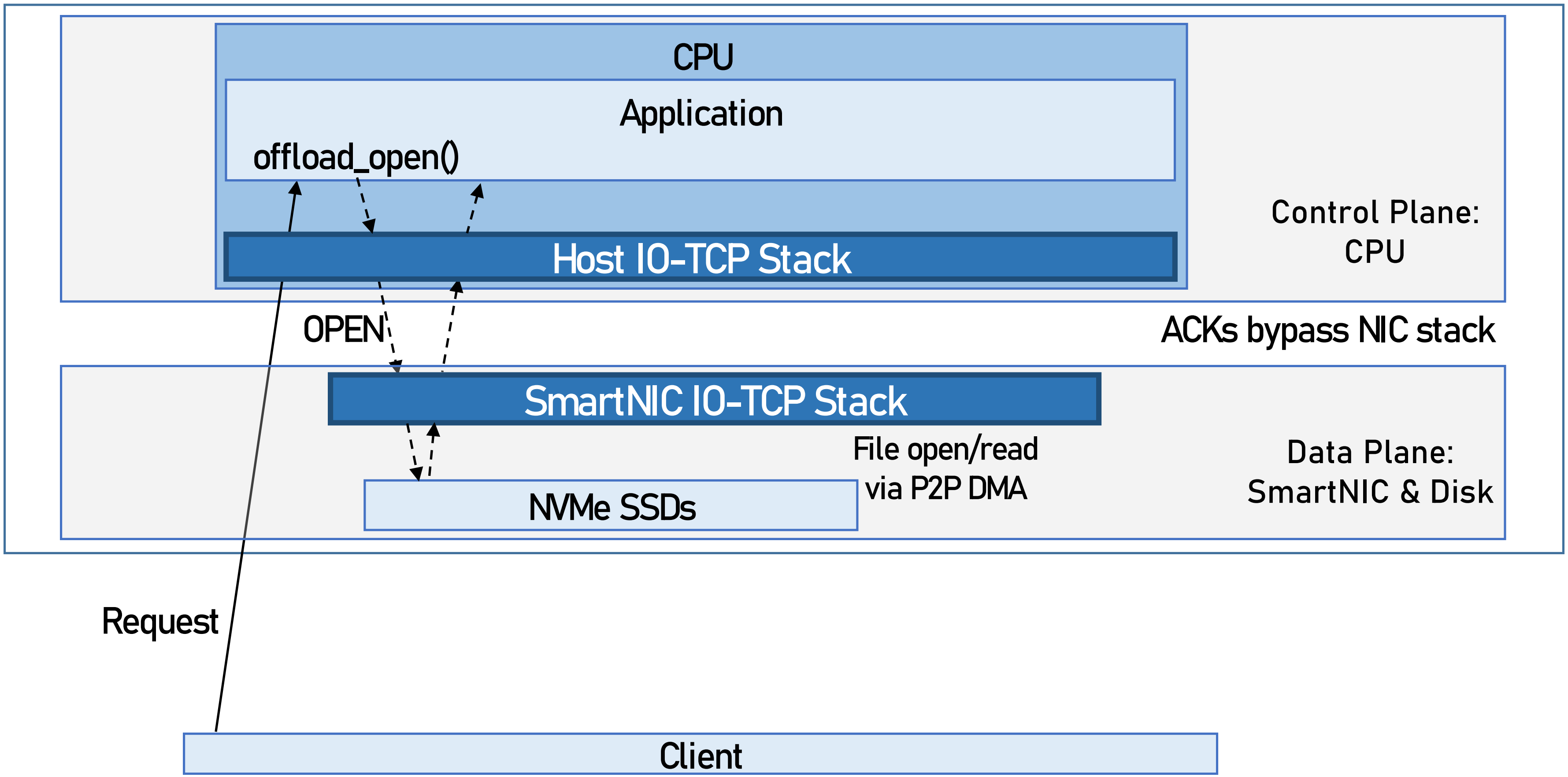
IO-TCP based Web Server Workflow



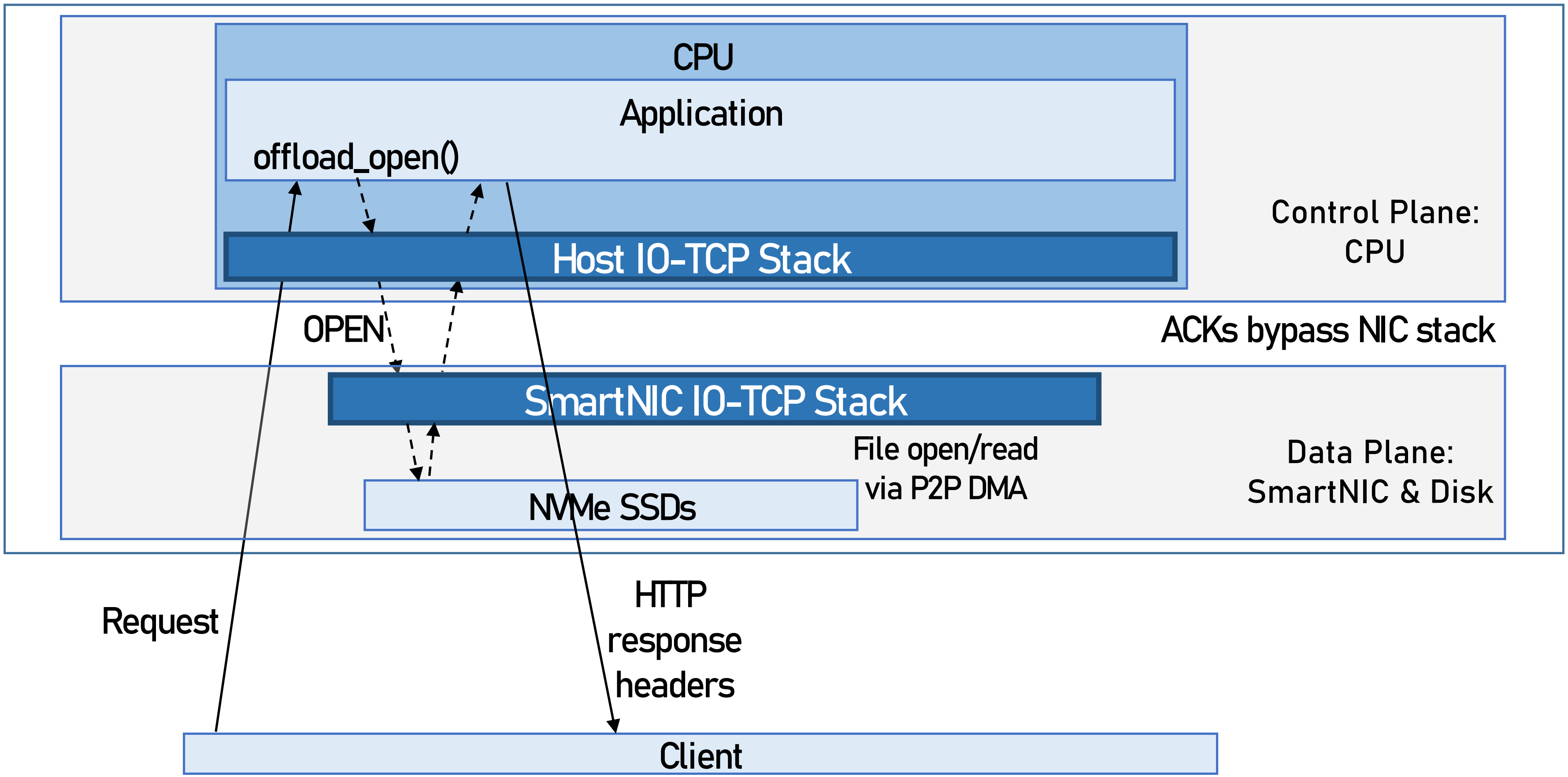
IO-TCP based Web Server Workflow



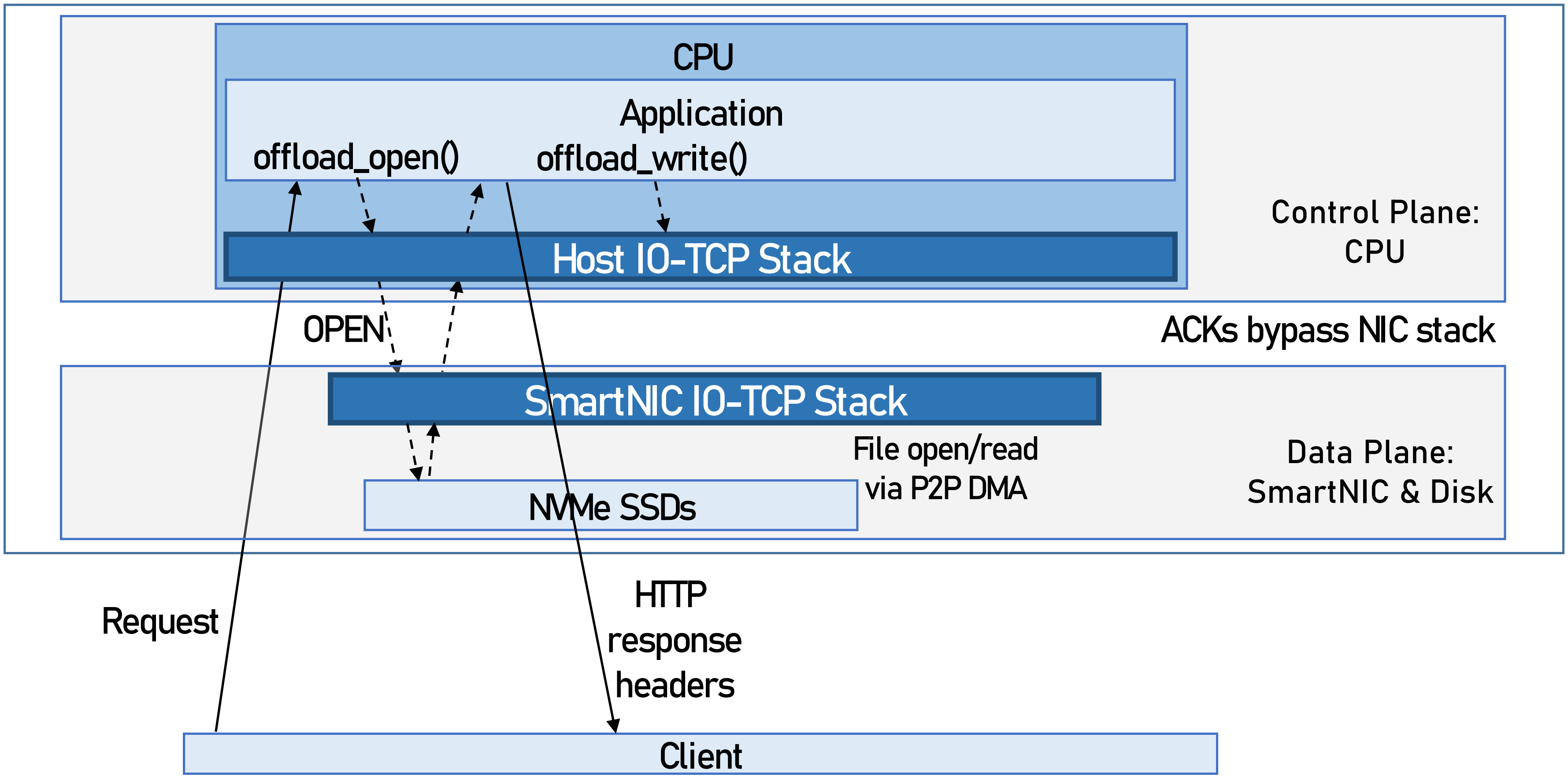
IO-TCP based Web Server Workflow



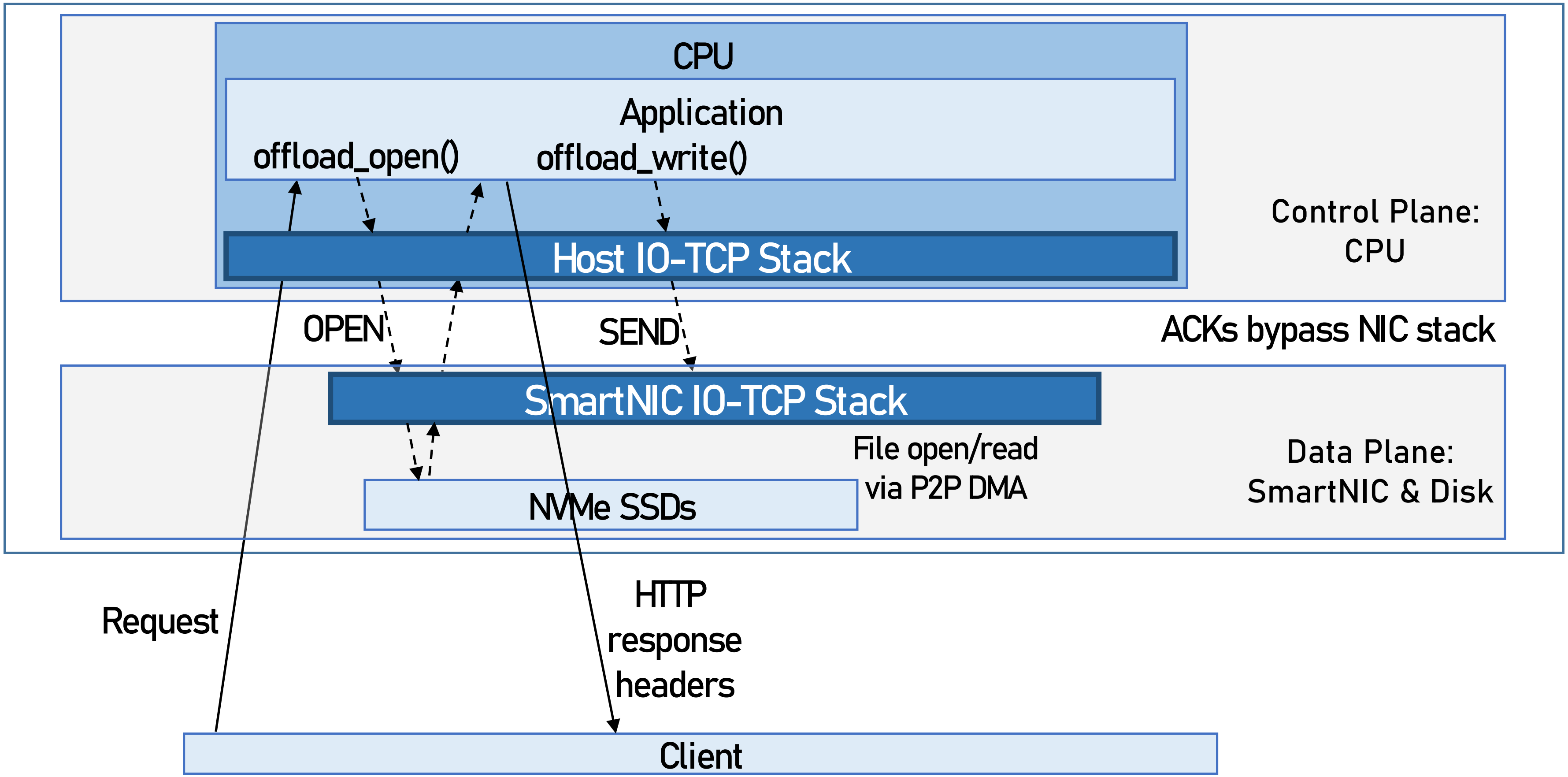
IO-TCP based Web Server Workflow



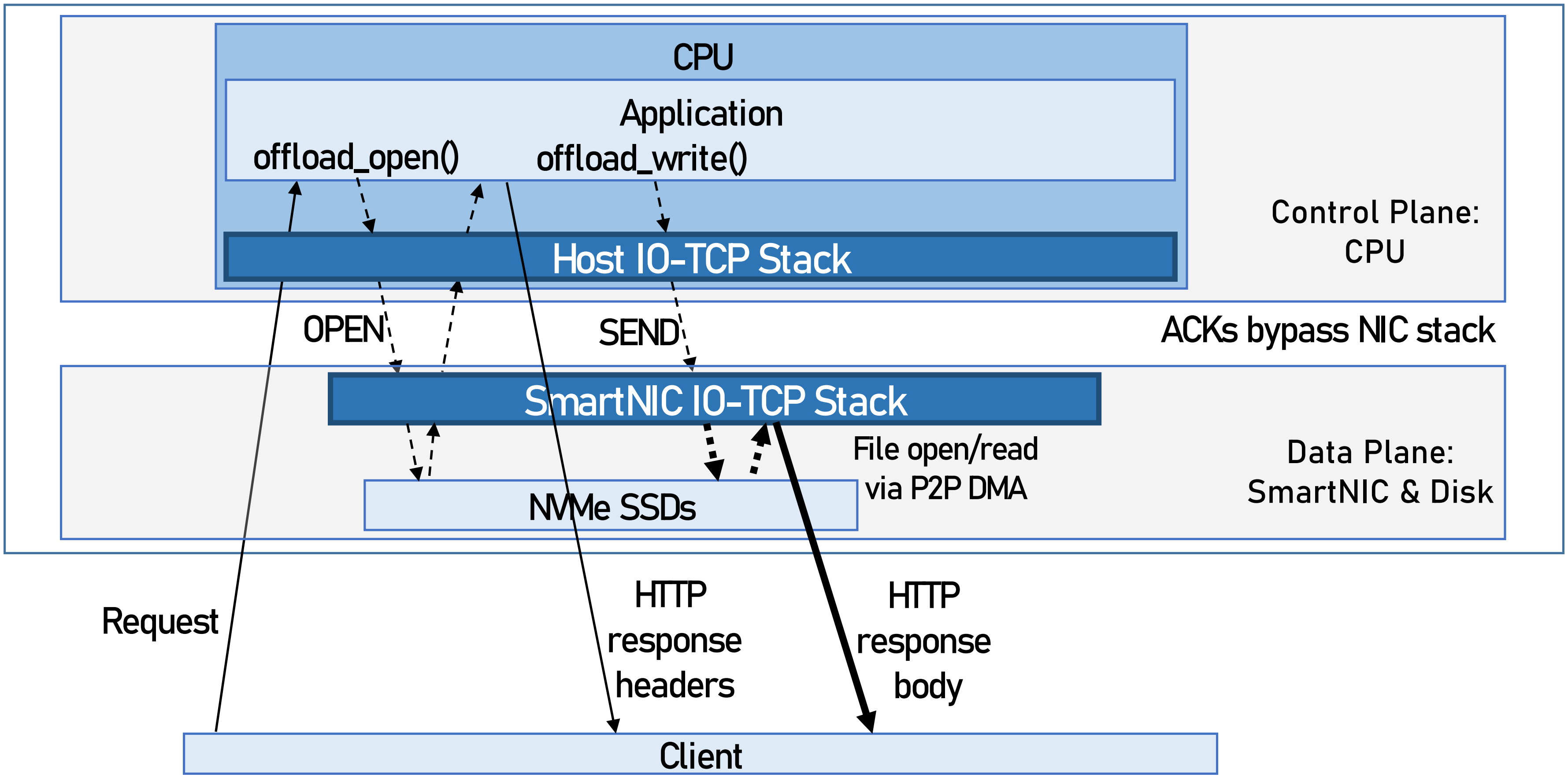
IO-TCP based Web Server Workflow



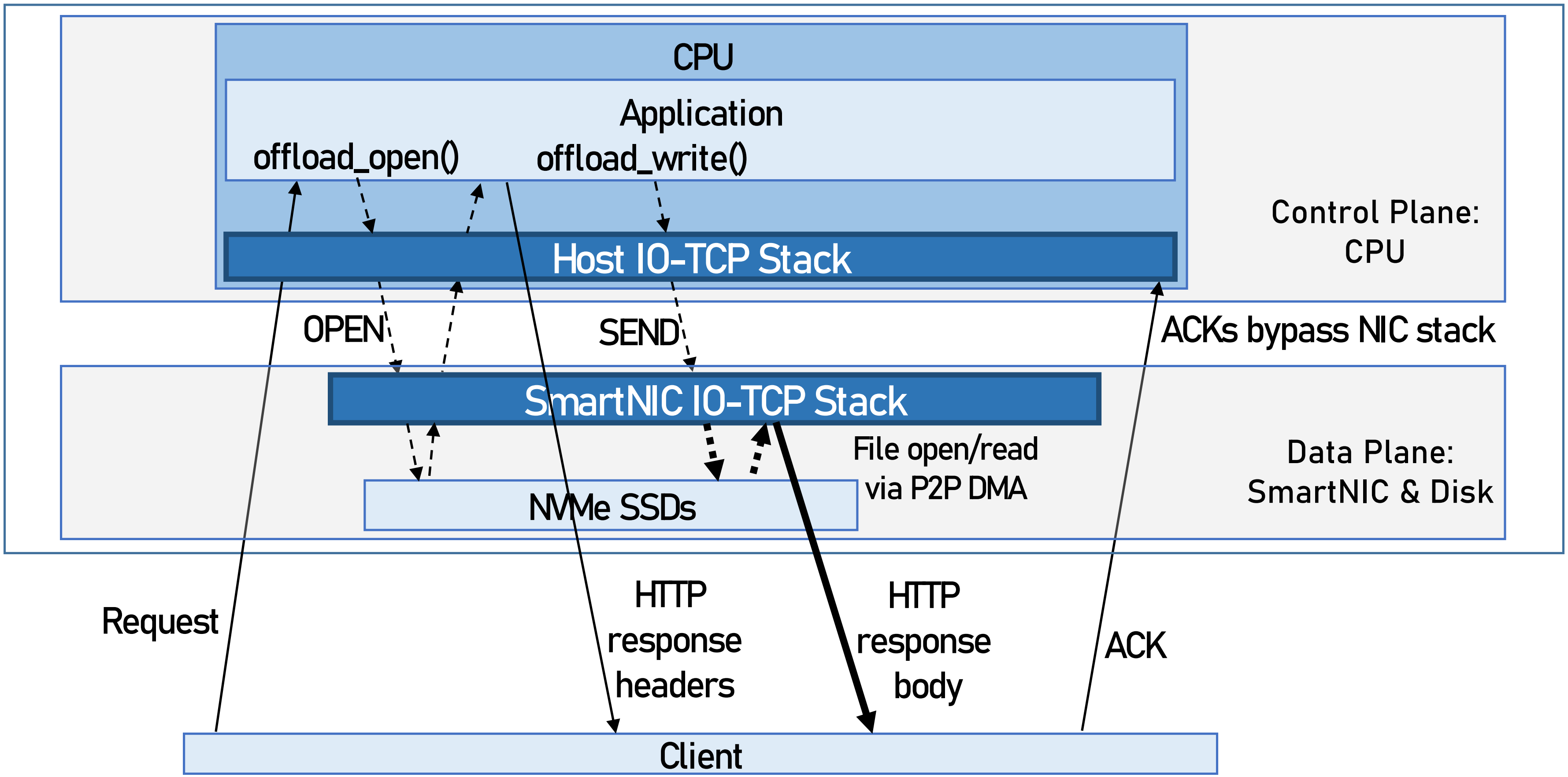
IO-TCP based Web Server Workflow



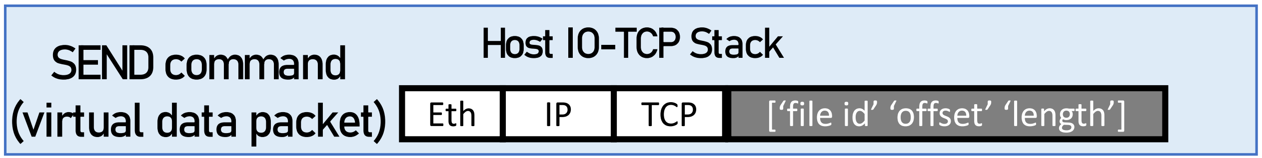
IO-TCP based Web Server Workflow



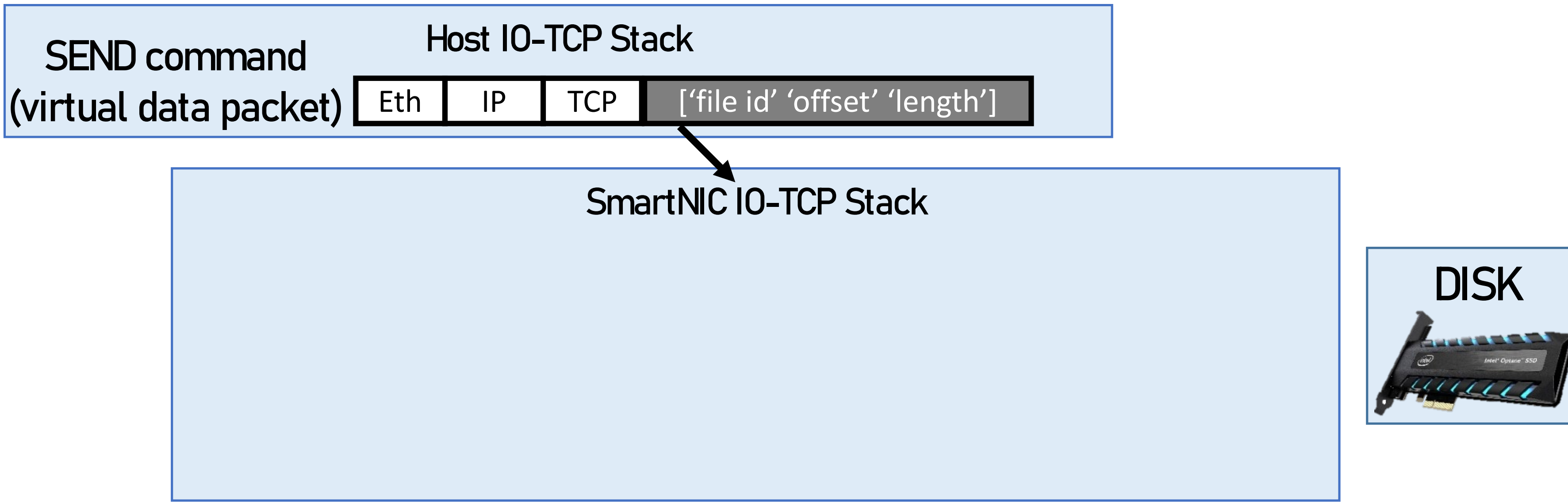
IO-TCP based Web Server Workflow



Key Operation for Offloading: SEND Command



Key Operation for Offloading: SEND Command



Key Operation for Offloading: SEND Command

Host IO-TCP Stack

SEND command
(virtual data packet)

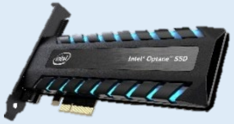
Eth	IP	TCP	['file id' 'offset' 'length']

SmartNIC IO-TCP Stack

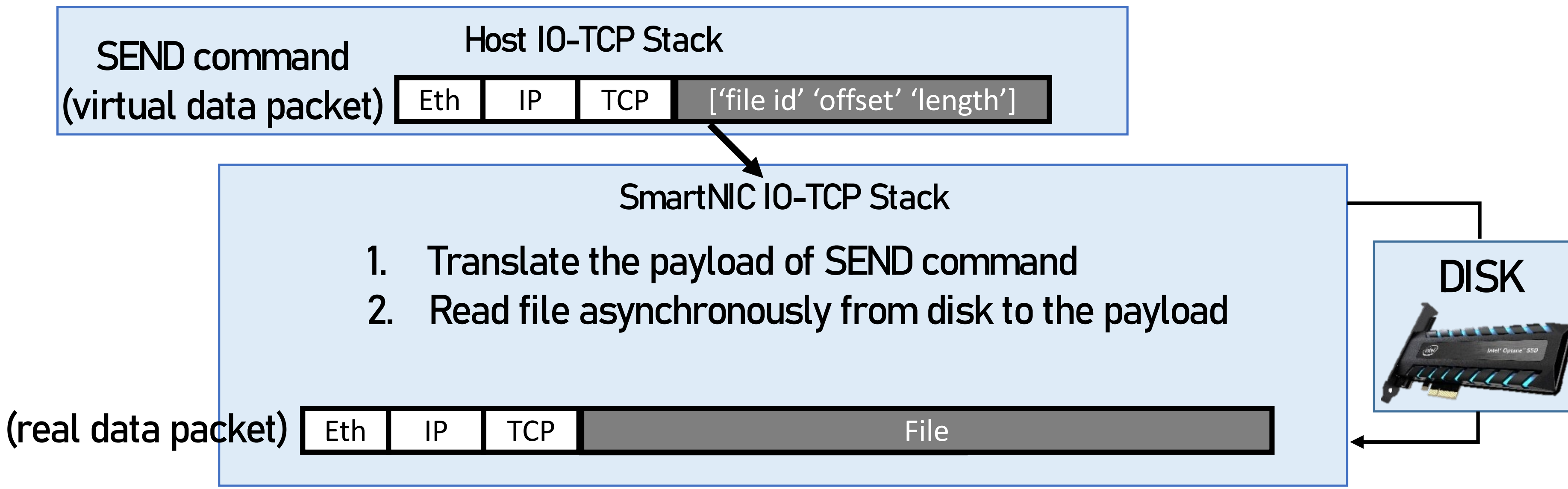
1. Translate the payload of SEND command

Eth	IP	TCP	['file id' 'offset' 'length']

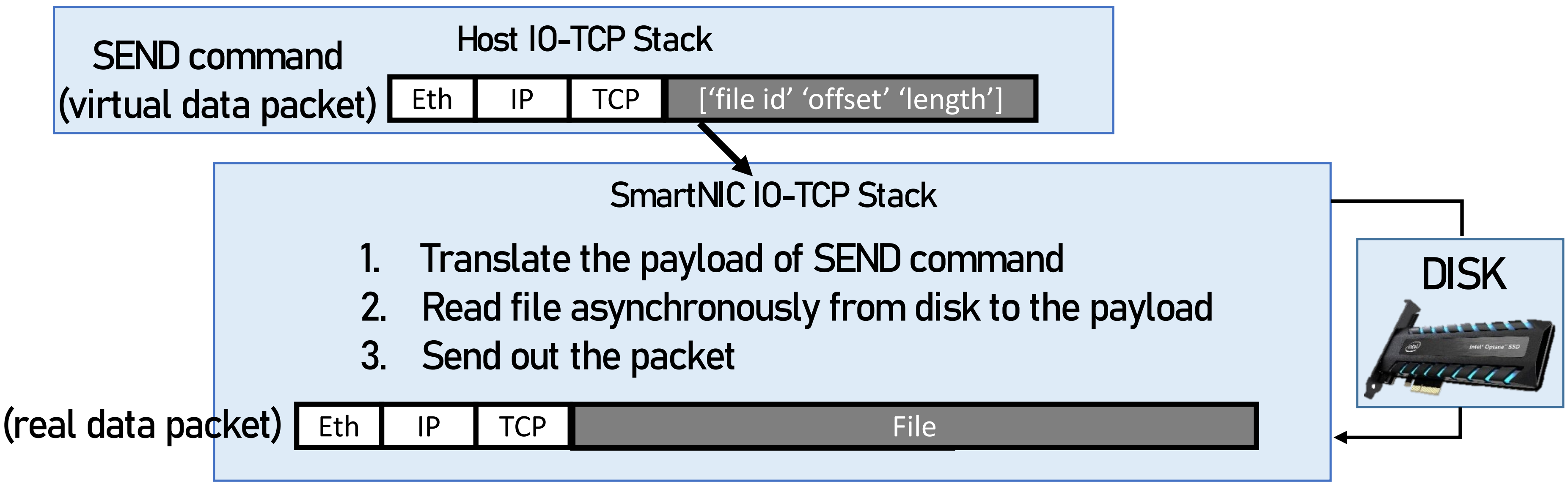
DISK



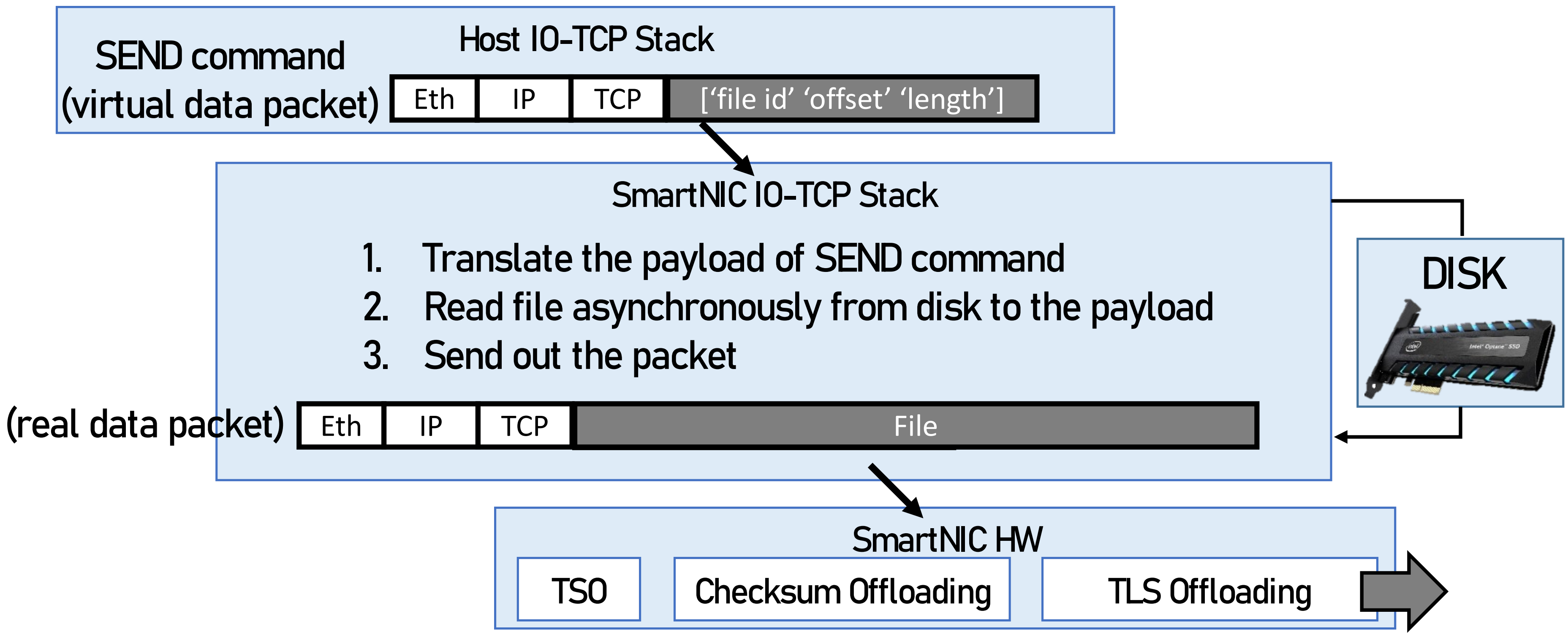
Key Operation for Offloading: SEND Command



Key Operation for Offloading: SEND Command



Key Operation for Offloading: SEND Command



IO-TCP Challenges

Challenges

How to calculate accurate *packet RTT*?

How to deal with *retransmission*?

IO-TCP Challenges

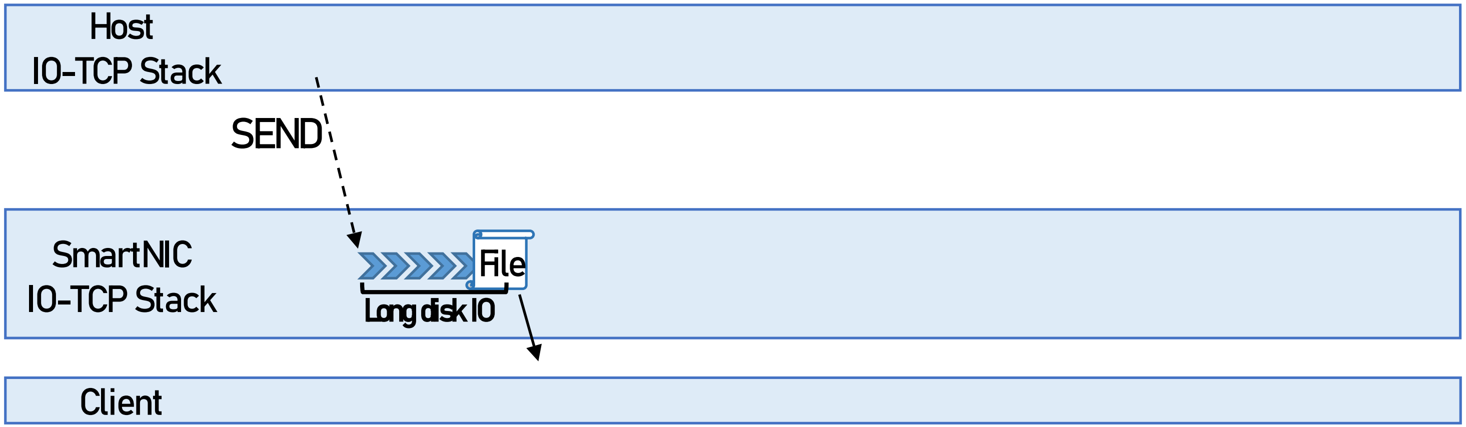
Challenges

~~How to calculate accurate *packet RTT*?~~

How to deal with *retransmission*?

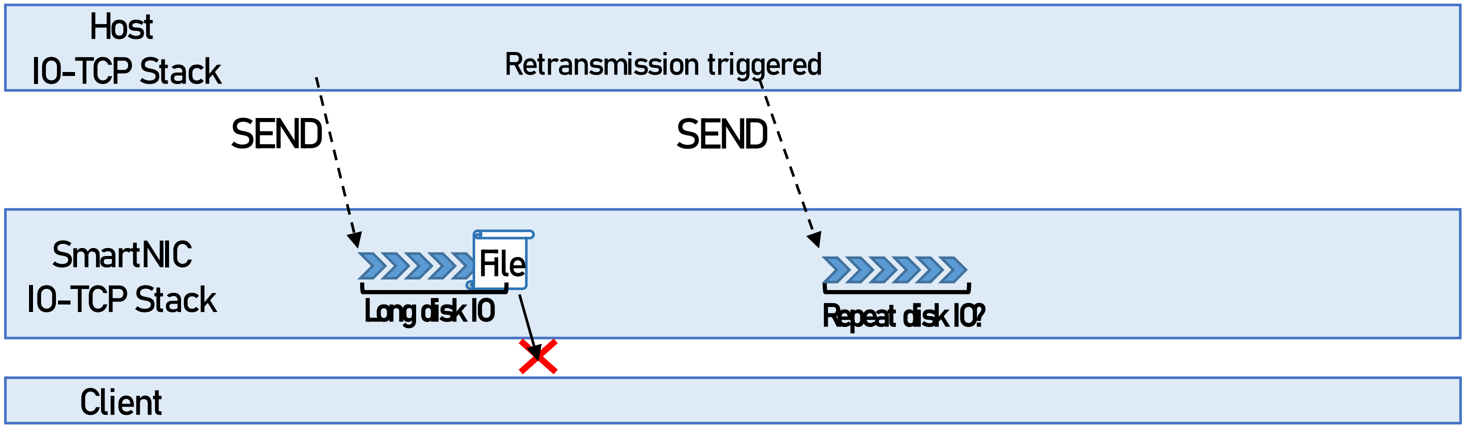
More details in the paper

How to Handle Retransmission?



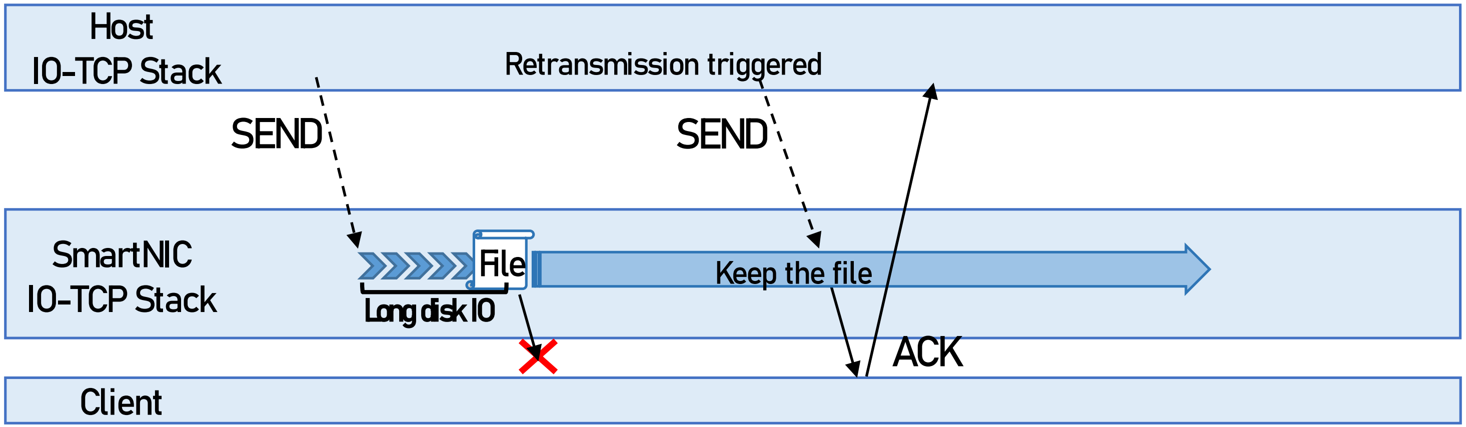
How to Handle Retransmission?

- Re-reading the disk for retransmission could be slow!



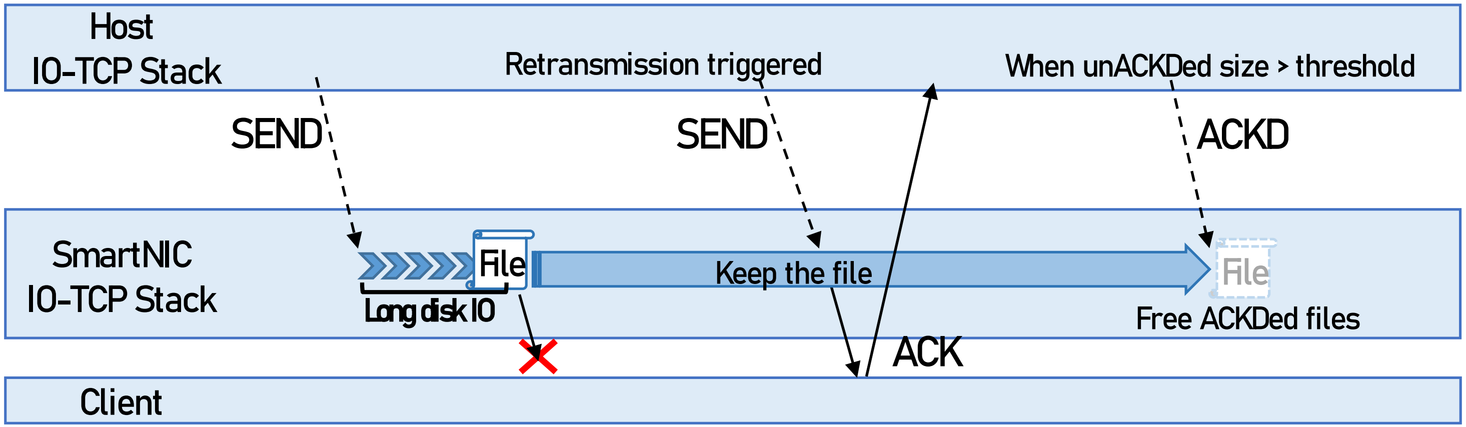
How to Handle Retransmission?

- Re-reading the disk for retransmission could be slow!
- Our approach
  - Keep the data on NIC memory until the data is confirmed to be delivered (ACK)
  - Problem: only Host receives all the ACKs (for control logic)



How to Handle Retransmission?

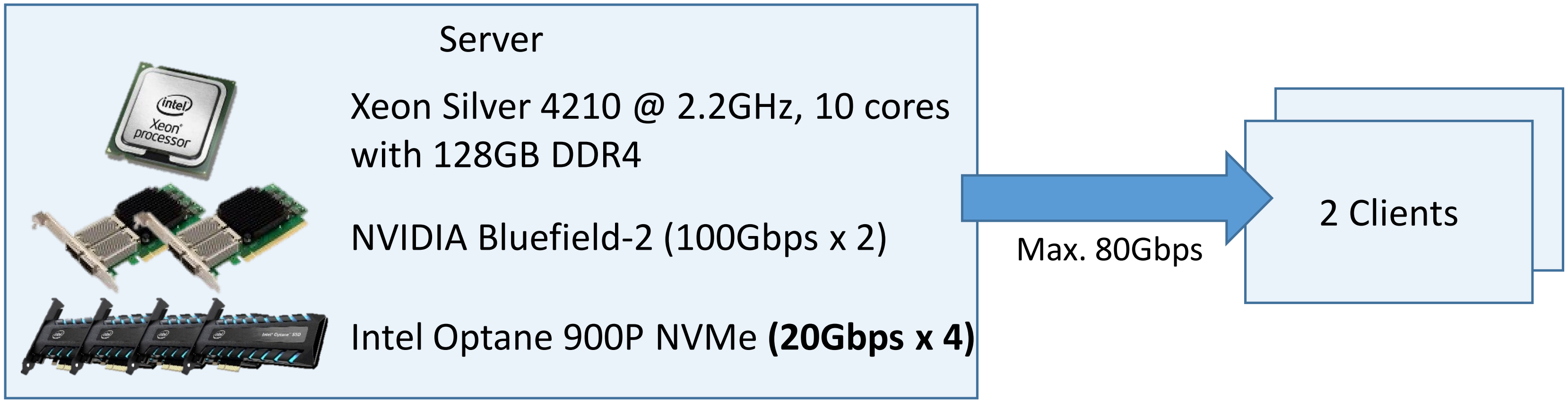
- Re-reading the disk for retransmission could be slow!
- Our approach
  - Keep the data on NIC memory until the data is confirmed to be delivered (ACK)
  - Problem: only Host receives all the ACKs (for control logic)
  - Solution: periodic notification of **ACK**nowledge**D**(ACKD) sequence numbers (Host → NIC)
    - Required memory size <= window size (e.g., 375MB for 100Gbps NIC with 30ms of average RTT)



Implementation

- Host stack: extended mTCP to support NIC offload
  - 1,793 lines of code modification on mTCP
- NIC stack: based on NVIDIA Bluefield2 SmartNIC
  - 1,853 lines of C code
  - We implement TSO, scatter-gather IO, and TLS crypto offload
- Easy to port existing apps
  - open(), fstat() and close() → offload_open(), offload_fstat() and offload_close()
  - write() → offload_write()
  - Porting Lighttpd server to IO-TCP: **~10 lines of code modification**

Experiment Setup

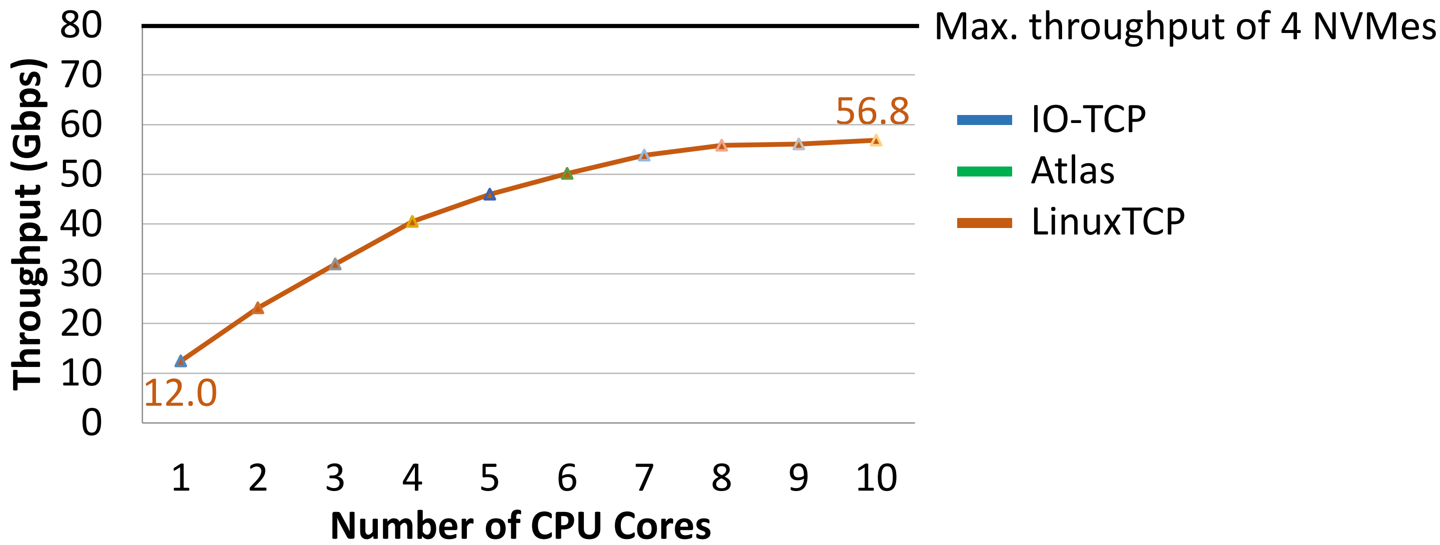


- **Baselines**
  - Lighttpd on Linux TCP with sendfile() (Kernel version: 4.14)
  - Atlas: webserver on kernel-bypass TCP stack with raw disk access [1]
    - Buffer-cache-free design
    - FreeBSD 1.10 & Chelsio 100Gbps NIC

[1] "Disk|Crypt|Net: rethinking the stack for high-performance video streaming." SIGCOMM, 2017.

IO-TCP Performance – Plaintext

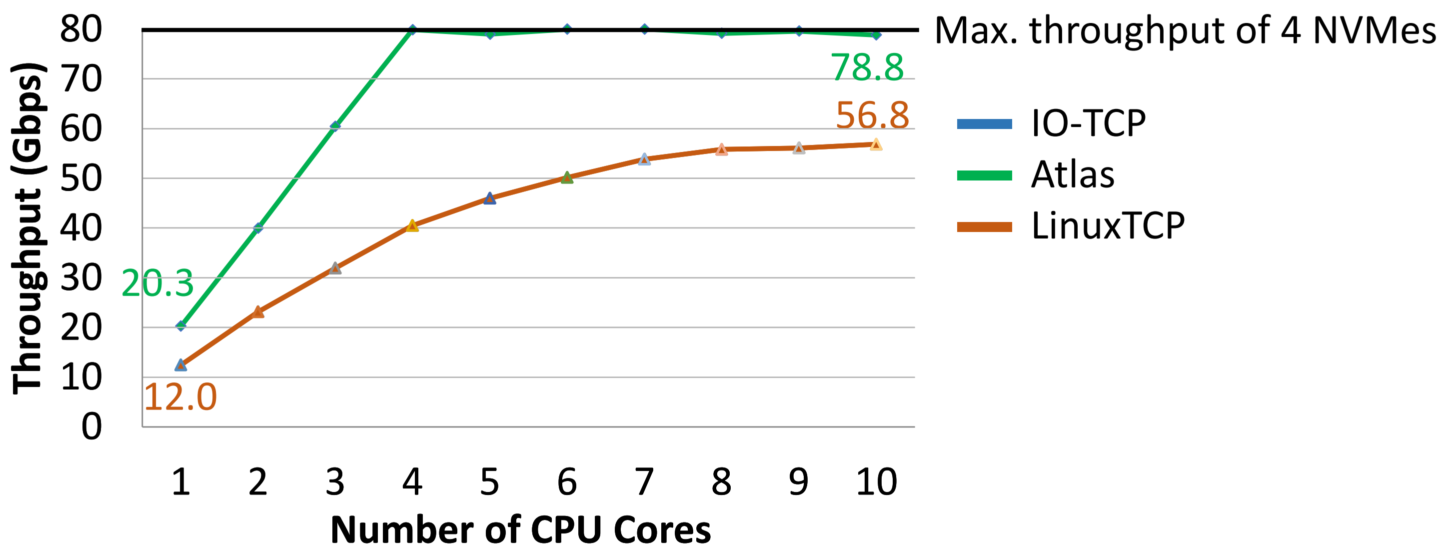
- 500KB video file chunks (disk bound)
- Lighttpd ported to IO-TCP



[1] "Disk|Crypt|Net: rethinking the stack for high-performance video streaming." SIGCOMM, 2017.

IO-TCP Performance - Plaintext

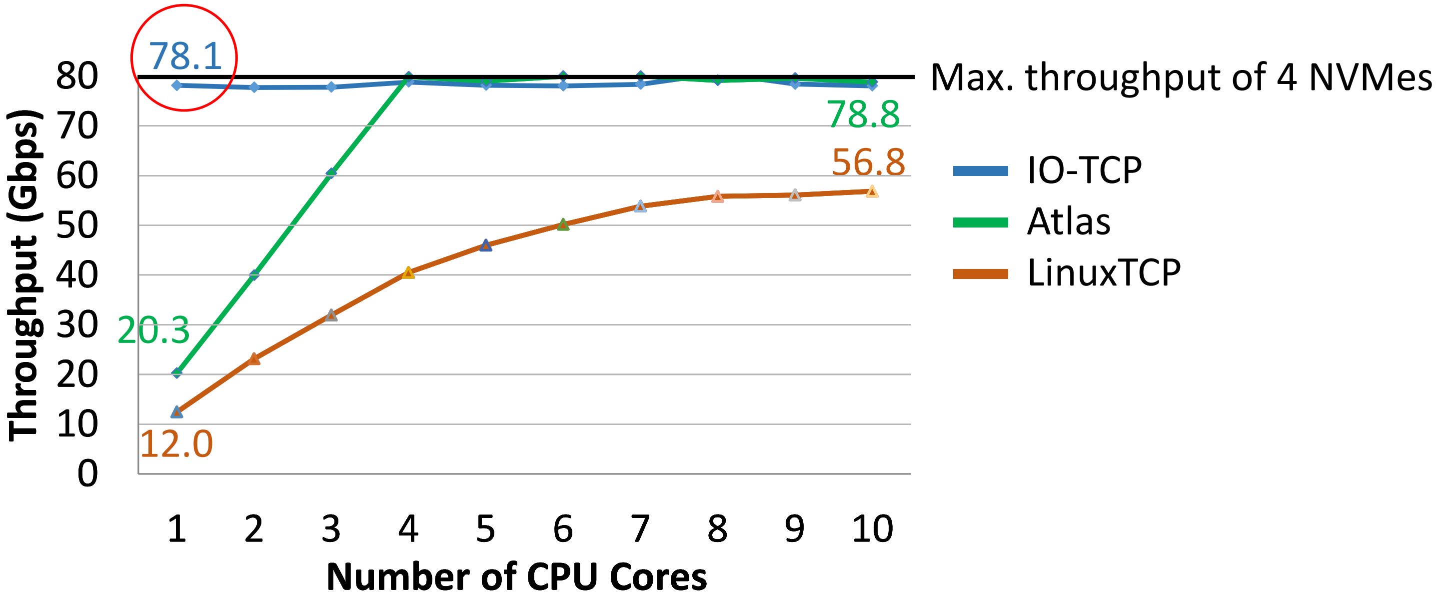
- 500KB video file chunks (disk bound)
- Lighttpd ported to IO-TCP



[1] "Disk|Crypt|Net: rethinking the stack for high-performance video streaming." SIGCOMM, 2017.

IO-TCP Performance – Plaintext

- 500KB video file chunks (disk bound)
- Lighttpd ported to IO-TCP

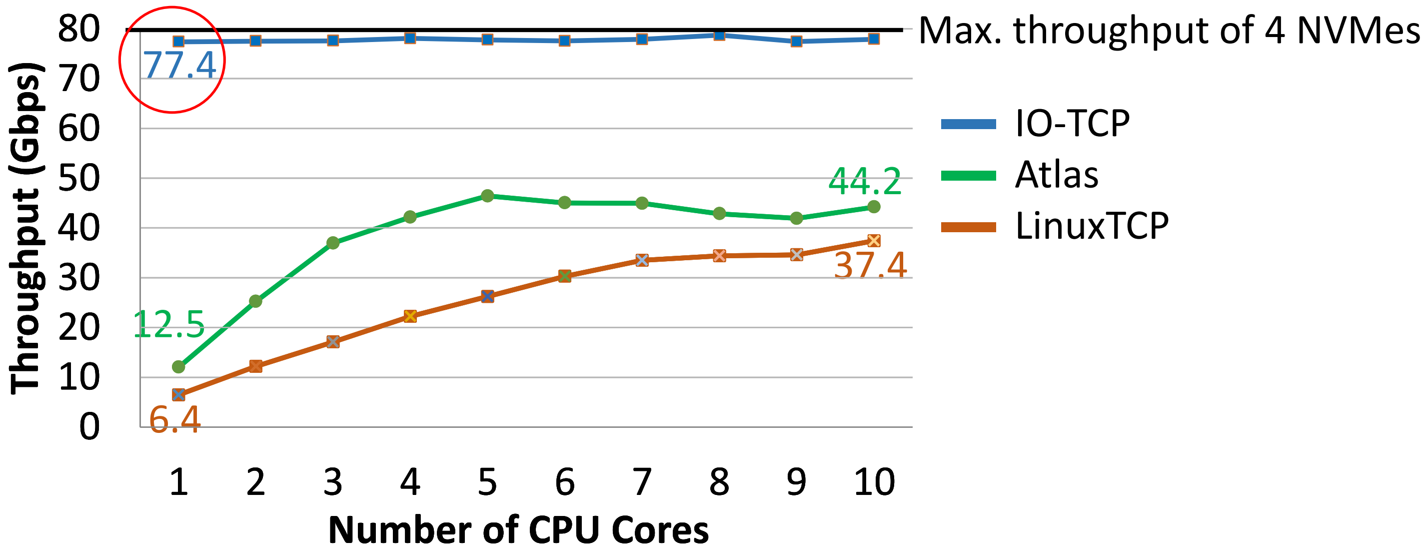


IO-TCP achieves Full BW of 4 NVMe disks with a single CPU core

[1] "Disk|Crypt|Net: rethinking the stack for high-performance video streaming." SIGCOMM, 2017.

IO-TCP Performance – TLS

- 500KB video file chunks (disk bound)
- Lighttpd ported to IO-TCP
- Cipher mode: AES-GCM 256



IO-TCP still reaches the max throughput for TLS traffic

[1] “Disk|Crypt|Net: rethinking the stack for high-performance video streaming.” SIGCOMM, 2017.

Source of Performance Improvement

Separation of Control plane / Data plane

- No main memory read/write for IO

- No CPU cache eviction by DDIO

Source of Performance Improvement

Separation of **Control plane** / **Data plane**
- No main memory read/write for IO
- No CPU cache eviction by DDIO

Mitigated ***Cache & Memory contention***
for **Control plane**
Separation achieves **27% lower LLC miss rate**

Faster control plane
IPC of the control path improves by **58%**

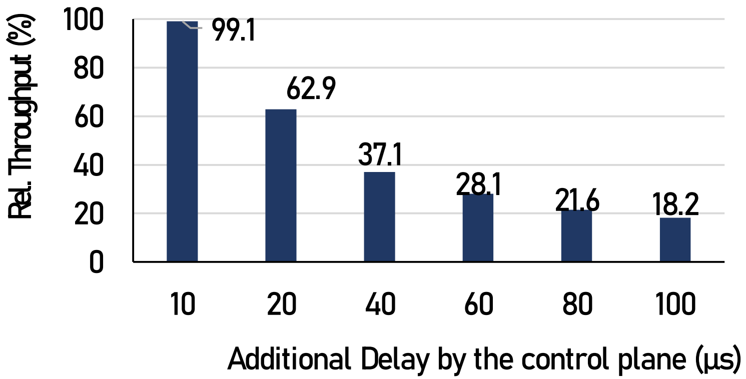
Source of Performance Improvement

Separation of **Control plane** / **Data plane**
- No main memory read/write for IO
- No CPU cache eviction by DDIO

Mitigated ***Cache & Memory contention***
for **Control plane**
Separation achieves **27% lower LLC miss rate**

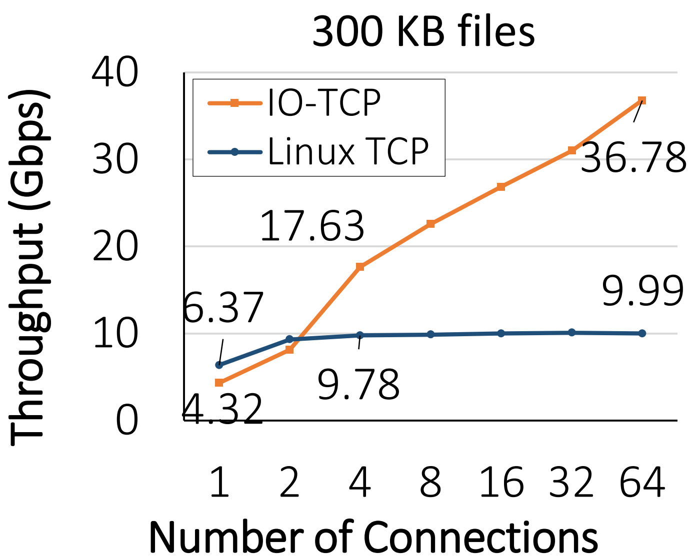
Faster control plane
IPC of the control path improves by **58%**

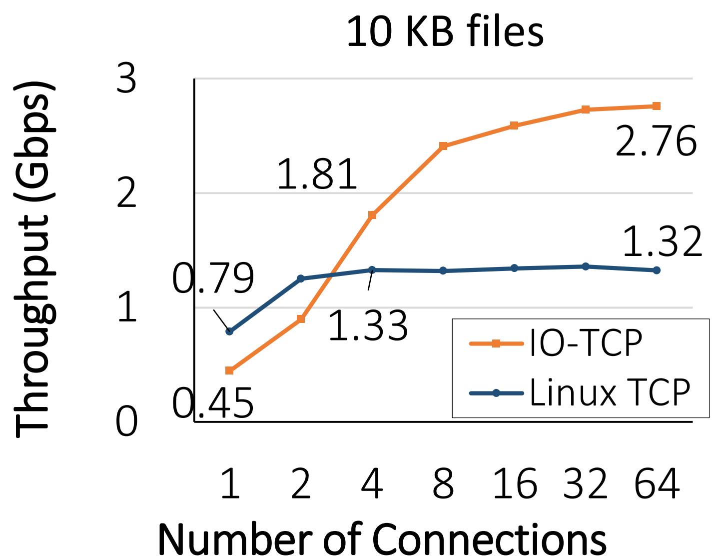
Shorter e2e RTT → Larger window size
→ Overall Throughput Improvement



IO-TCP Overhead Evaluation

- Overhead factors
  - Host-NIC communication overhead
  - Performance limit of Arm-based subsystem on BF2
  - ➢ The fewer connections would be advantageous to CPU-only approach (Linux TCP)





Summary

- BIG Trend: IO device advancement outpaces the rate of CPU capacity growth

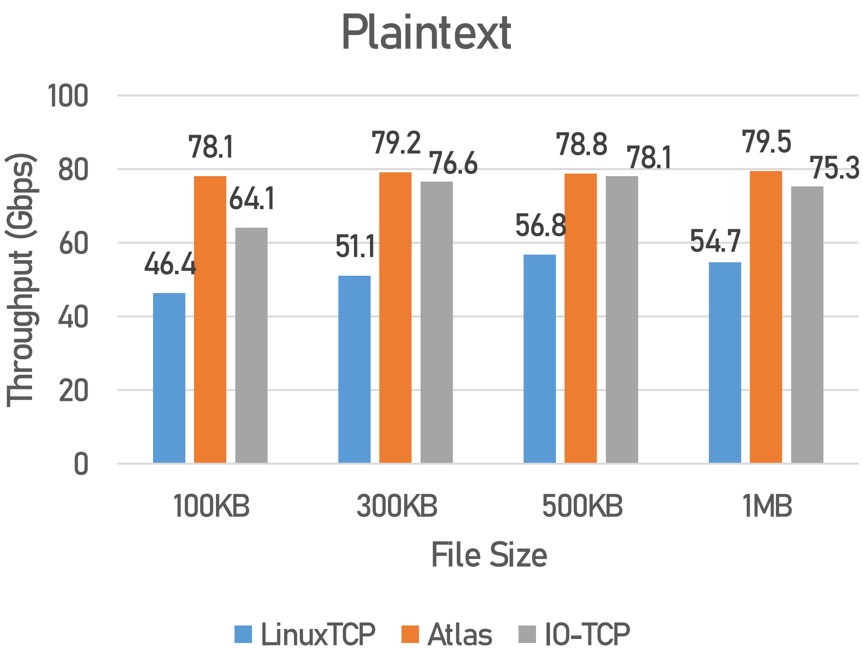
- IO-TCP: a split TCP stack architecture for a content delivery system
  - CPU host stack carries out the control plane functionalities of a TCP stack
  - NIC stack serves as data plane of a TCP stack

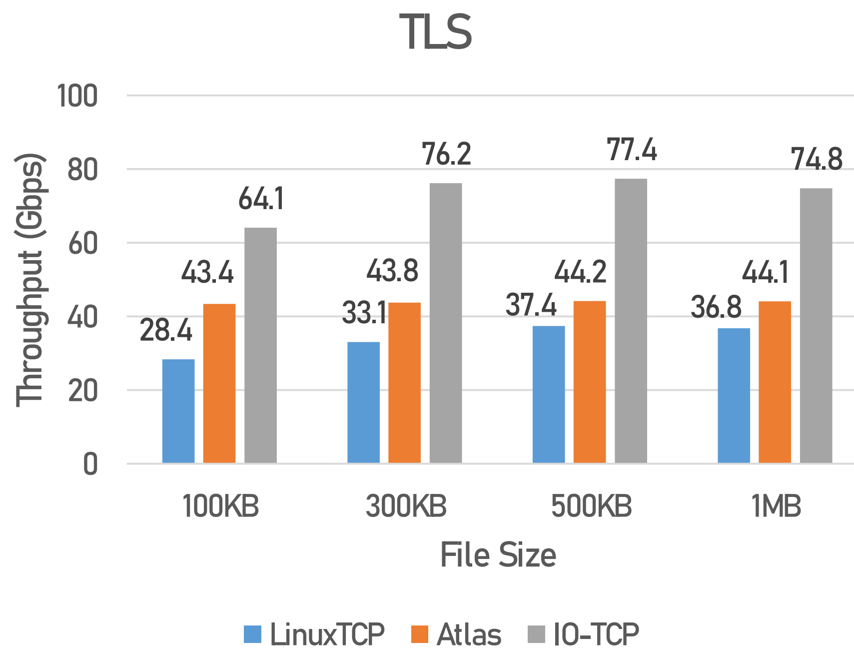
- IO-TCP achieves full bandwidth of 4 NVMe disks with a single CPU core
  - Current bottleneck lies in SmartNIC memory bandwidth
  - SmartNIC with higher memory BW will improve the throughput even more
    - *Bluefield-3* will achieve 140Gbps per NIC

- QUIC-based CDN can adopt our separated stack design as well

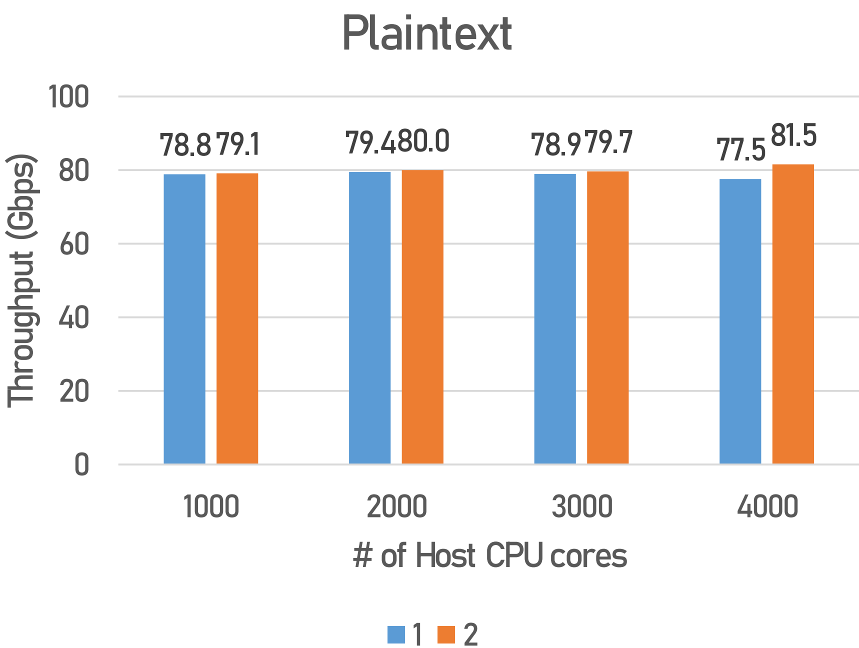
Thank you!

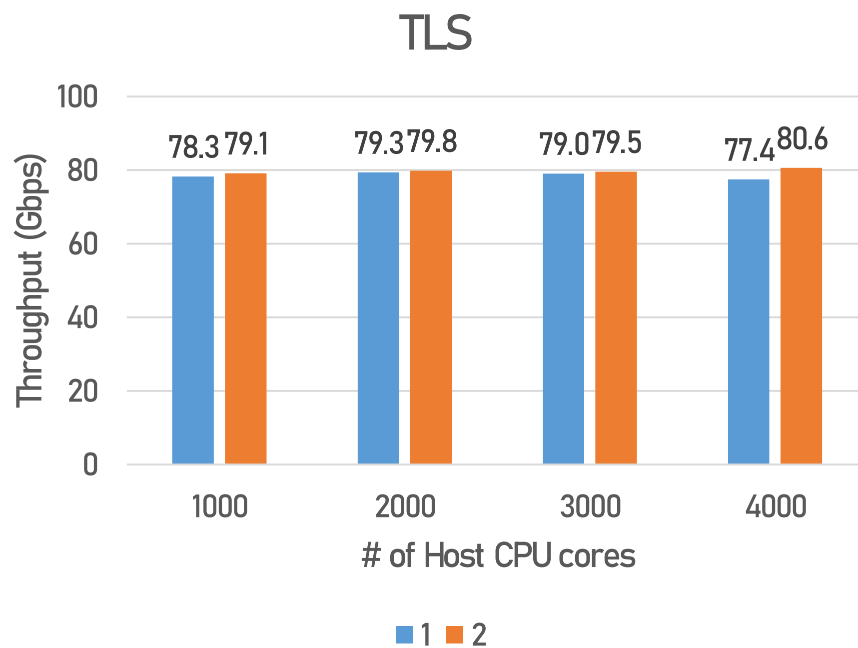
IO-TCP Performance – Varying File Sizes





IO-TCP Performance – Varying Number of Connections





Linux TCP vs. IO-TCP

Lighttpd setup	Throughput (Gbps)
Linux TCP on Bluefield-2 only	11.98
Linux TCP on Bluefiend-2 and 1 CPU core	22.02
IO-TCP on Bluefield-2 and 1 CPU core	44.13

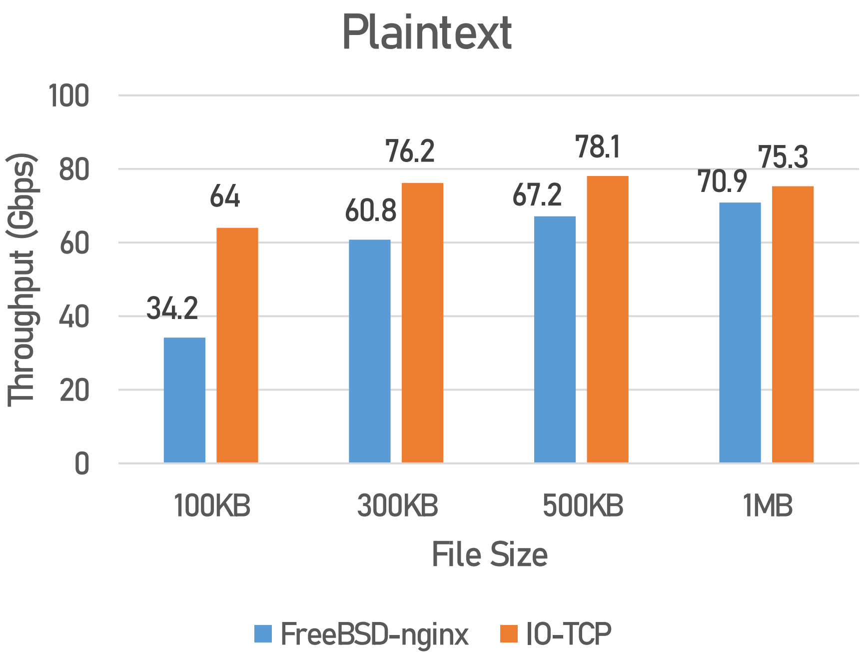
TCP Fairness

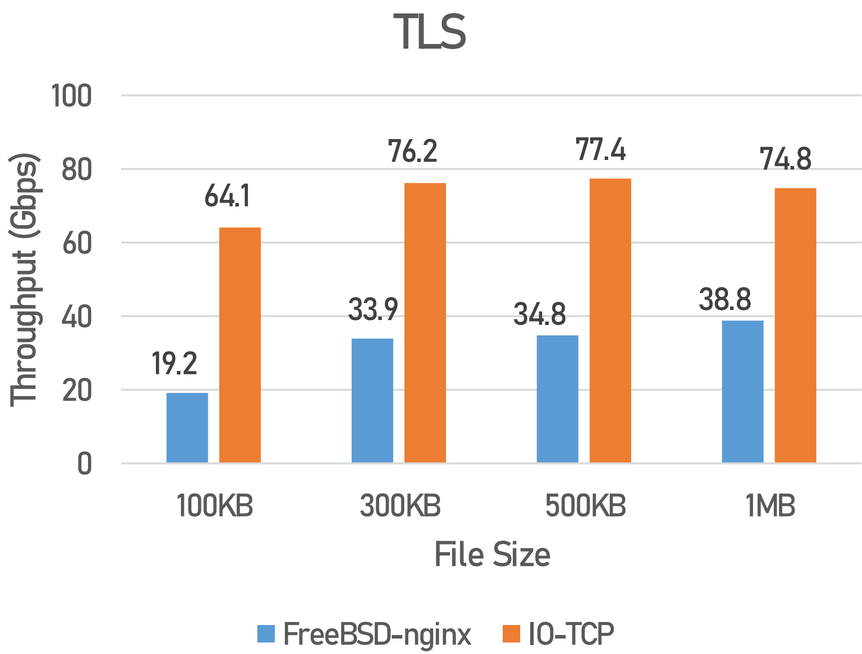
- Jain's fairness index with varying number of connections
  - IO-TCP: 0.91~0.97
  - Linux TCP: 0.90~0.97

User-level TCP Stacks vs. IO-TCP

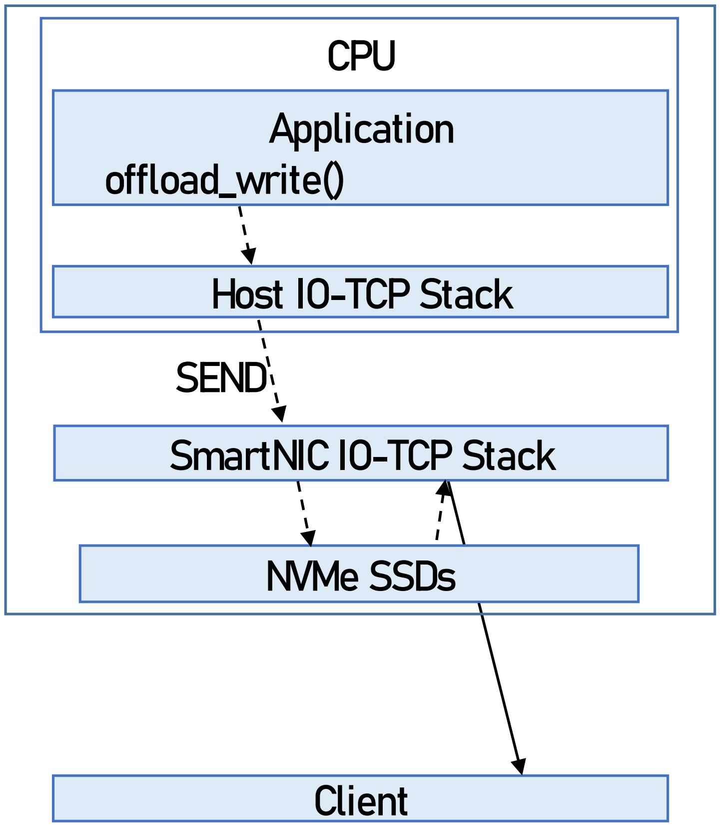
- Throughput with 500KB file delivery
  - TAS: 9.0 Gbps
  - mTCP: 21.4 Gbps
  - F-Stack: 36.0 Gbps
  - Linux TCP: 56.8 Gbps
- These stacks are not optimized for large-file content delivery
  - Optimized for small messages
  - Lacks of an implementation for sendfile() and a support for TSO

Asynchronous sendfile() on FreeBSD vs. IO-TCP



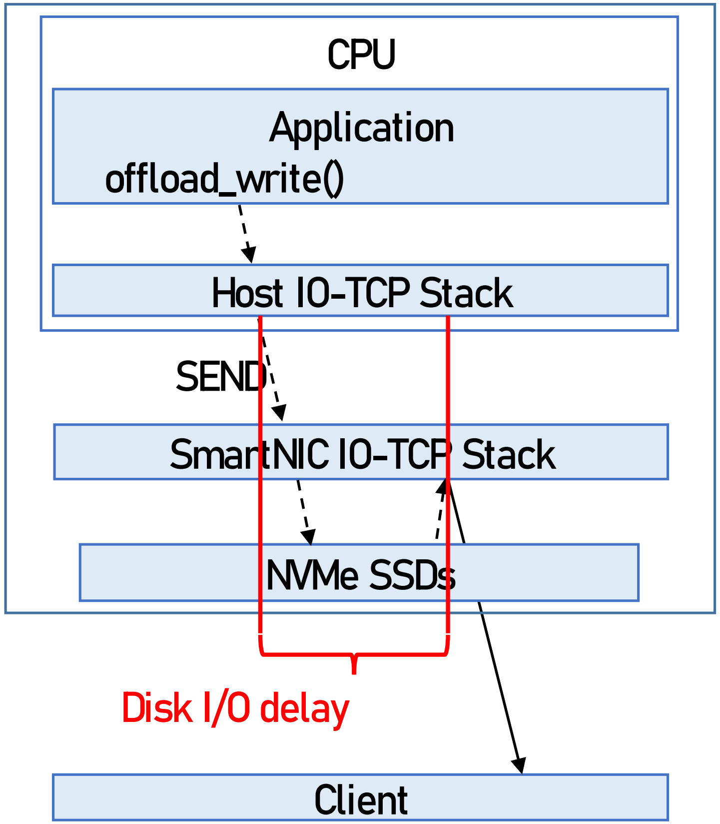


Retransmission Timer & RTT Measurement



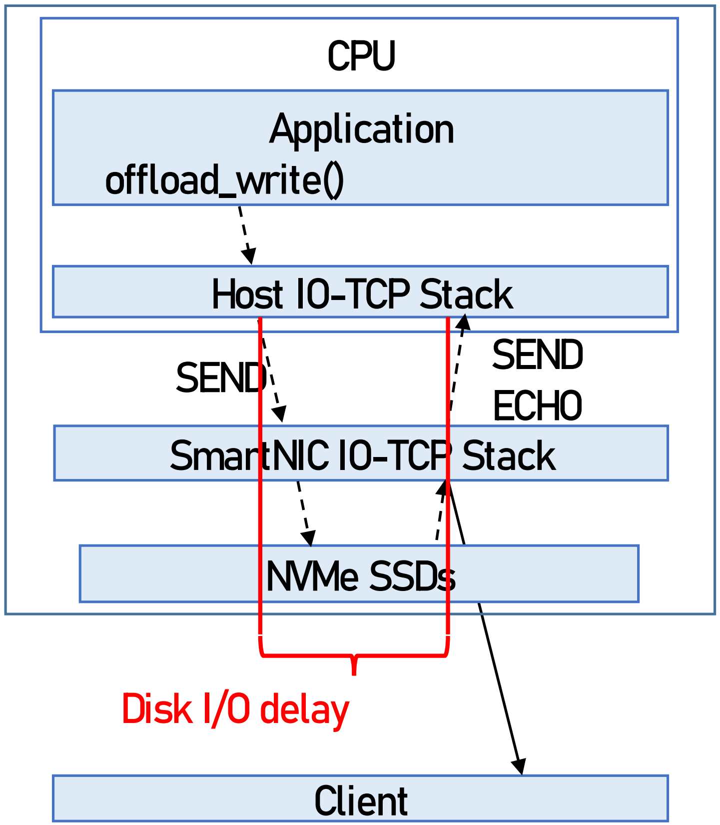
- Retransmission timer at the host stack

Retransmission Timer & RTT Measurement



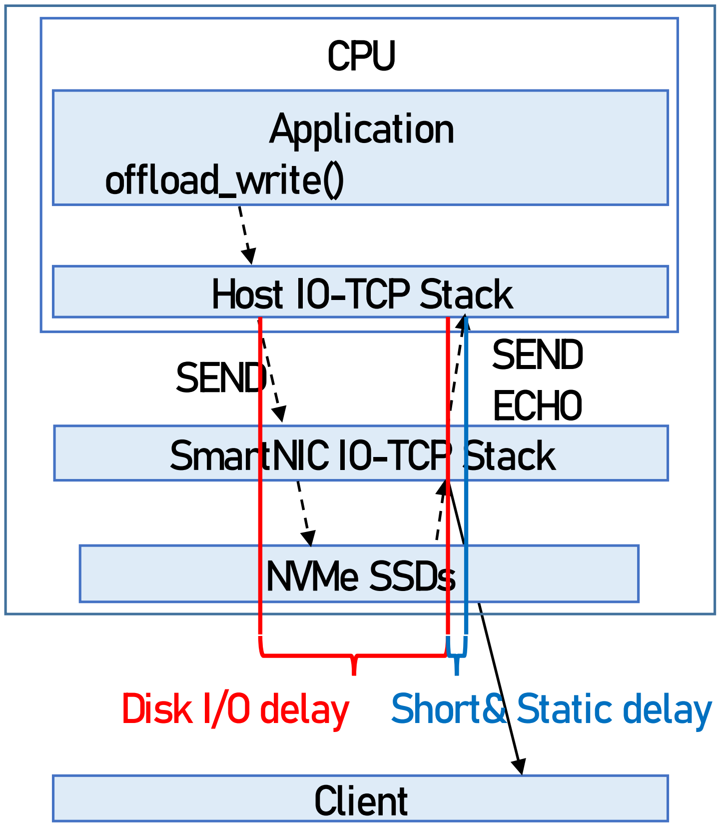
- Retransmission timer at the host stack
  - Problem: disk access delay is added
    - Up to a few ms when backlogged

Retransmission Timer & RTT Measurement



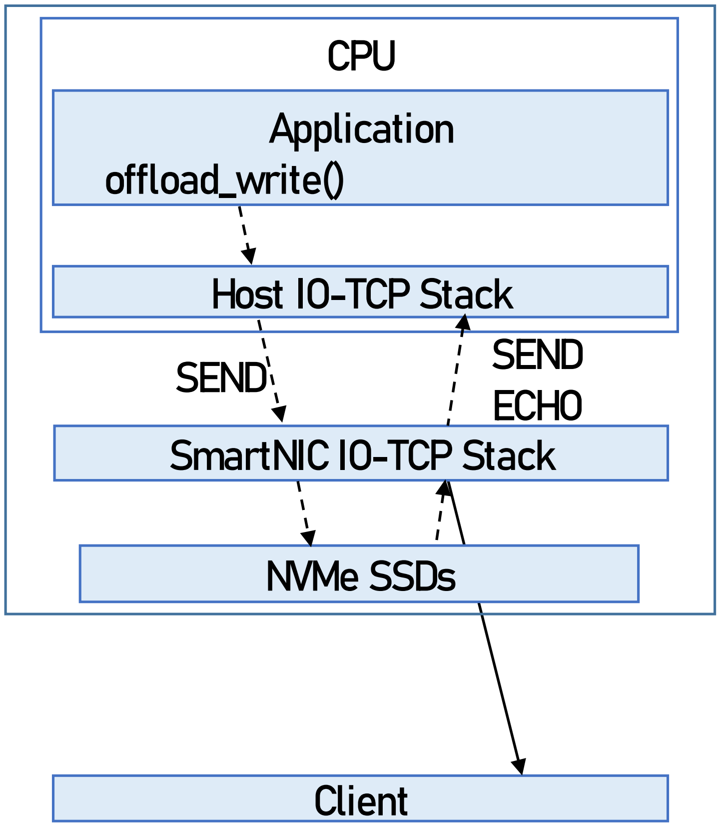
- Retransmission timer at the host stack
  - Problem: disk access delay is added
    - Up to a few ms when backlogged
  - Solution: SEND ECHO packets to the host

Retransmission Timer & RTT Measurement



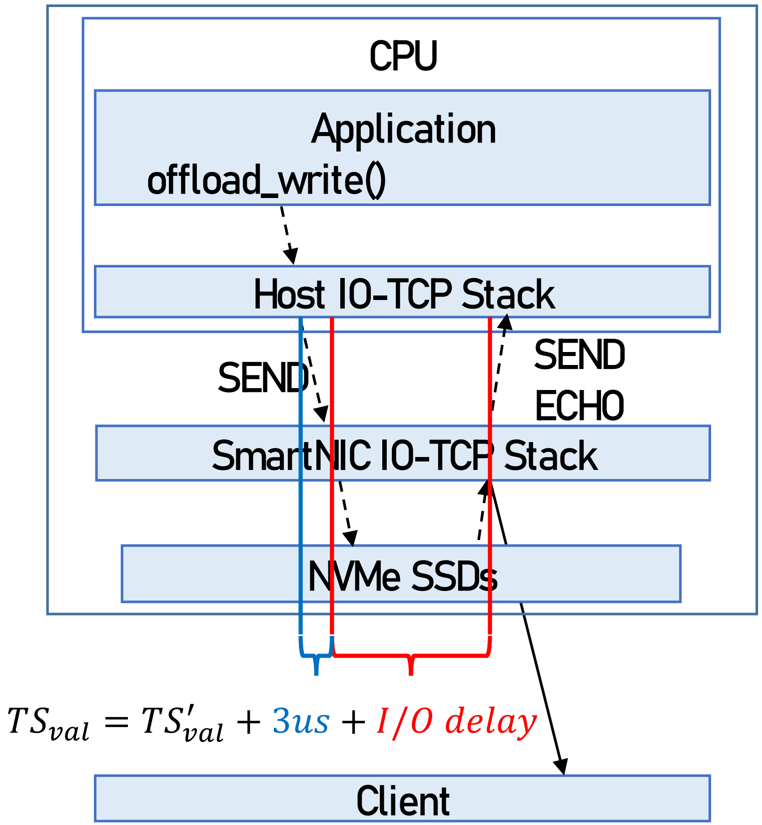
- Retransmission timer at the host stack
  - Problem: disk access delay is added
    - Up to a few ms when backlogged
  - Solution: SEND ECHO packets to the host
    - Short & fixed delay (~3us in our setup)
    - Negligible overhead: a SEND for 10s~100s MTUs

Retransmission Timer & RTT Measurement



- Retransmission timer at the host stack
  - Problem: disk access delay is added
    - Up to a few ms when backlogged
  - Solution: SEND ECHO packet to the host
    - Short and static delay (3us in our setup)
    - Negligible overhead: a SEND for 10s~100s MTU
- RTT measurement with Timestamp option

Retransmission Timer & RTT Measurement



- Retransmission timer at the host stack
  - Problem: disk access delay is added
    - Up to a few ms when backlogged
  - Solution: SEND ECHO packet to the host
    - Short and static delay (3us in our setup)
    - Negligible overhead: a SEND for 10s~100s MTU
- RTT measurement with Timestamp option
  - Add the static delay(3us) and the I/O delay to the TS_{val}

CPU as a Bottleneck for NVMe

- With **<u>multiple NVMe disks</u>**, CPU **can** be a bottleneck
  - Intel Xeon Silver 4210 (2.20GHz)
  - 6x Intel Optane 900P
  - Simple fio
  - A single CPU core cannot even support **2 disks for 4K BS**