Disk | Crypt | Net

High-performance video streaming

Ilias Marinos, Robert Watson (Cambridge),
Mark Handley (UCL),
Randall Stewart (Netflix)
Modern Video Streaming

- Just lots of HTTP requests for video chunks.
- Client picks chunks to adapt rate.
- Server is pretty dumb – just has to go fast.
- HTTP/1.1 persistent connections.
- TLS becoming important (95% of Youtube traffic).

- More than 50% of US Internet traffic.
- Important to make good use of expensive hardware. How fast can you go?
New iPlayer setup, Dec 2015:

• nginx on Linux, 24 cores on two Intel Xeon E5-2680v3 processors, 512 GB DDR4 RAM, 8.6TB RAID array of SSDs.

• 20Gb/s per server.  → Can we improve performance?
Case study: Netflix

• FreeBSD, but tweaked.
  – Asynchronous `sendfile()`
    • Non-blocking zero copy from disk buffer cache to Net.
  – VM scaling
    • Fake NUMA domains to avoid lock contention.
    • Proactive cleanup of disk buffer cache.
  – RSS-assisted LRO.
    • Sort incoming packets to buckets based on 5-tuple hash to optimize LRO engine efficacy.
Let's Do Some Experiments

- 8-core Haswell server, 2x40GbE NICs, 128GB RAM, 4x Intel P3700 NVMe disks
- Linux Clients.
- Synthetic workload, middlebox for realistic RTT.

![Diagram showing the setup with Streamer, middlebox, 40GbE switch, and Client]
Unencrypted video streaming workload

Data NOT in disk buffer cache

CPU utilization doubles when fetching from disk (~350% -> ~700%)

Data comes from disk buffer cache

Conclusions
- Netflix improvements **good**
- CPU utilization is a **problem**
Encryption

Problem:

Sendfile:
- Zero copy from disk buffer cache.

TLS:
- Different encrypted stream per user.
- Kernel is unaware of TLS.

Sendfile and TLS are fundamentally incompatible!

- Conventional TLS stack gave Netflix 20 -> 8.5Gb/s
- Netflix implemented in-kernel TLS support for sendfile!

sendfile() NOT zero-copy anymore!
Encrypted video streaming workload

- CPU is saturated.
- Memory read throughput $\sim 3x$ more than network throughput!
- Performance loss ($\sim 30\%$) when content fetched from SSDs.
What’s happening?

The stack is too asynchronous. Data keeps getting flushed from the LLC, and re-loaded. System is bottlenecked on memory.
Production Netflix Workload

- 192GB for buffer cache, but only 10% hit ratio.
- Streamers bottlenecked in memory bandwidth.

✓ Modern NVMe SSDs have low latency & high throughput.
✓ Modern Intel CPUs DMA directly to L3 cache.

Can we eliminate the disk buffer cache completely, and fetch everything from the SSDs on-demand?
To achieve this, we must:
- Fetch on demand from the SSD when TCP needs data.
- As soon as the SSD returns data, process it to completion and DMA it to the NIC.
Solution Outline

1. A TCP ACK arrives, freeing up congestion window.
2. Trigger stack to request more data from SSDs to fill that congestion window.

Conventional OS stack NOT suitable:
- Highly asynchronous; storage and network stack are loosely coupled -- relies on VFS & Buffer Cache.
- Introduces overheads related to abstraction layers (VFS, POSIX etc), redundant memory copies and domain transitions (user<->kernel).
The Atlas Streaming Stack

Atlas: a complete user-space stack

- TCP/IP stack based on modified version of *Sandstorm* (*SIGCOMM’14*) and *netmap* (*ATC’12*).
- Storage handled using *diskmap* (no buffer cache, no sophisticated FS).
- *Lockless*, full zero-copy stack from disk<->NIC.
- Tight pipeline to reduce asynchrony, and ideally save memory bandwidth (w/ DDIO).
**Diskmap Architecture**

*Diskmap*: a kernel-bypass I/O framework for NVMe disks

![Diagram of Diskmap architecture](image)

- **SQ**: Sends requests to the controller.
- **CQ**: Receives responses from the controller.
- **DMA**: Direct Memory Access for efficient data transfer.
- **libnvme**: Library for NVMe operations.
- **buffers**: Memory-mapped data structures.
- **nvme0-1**, **nvme0-2**: NVMe disks.
- **I/O MMU**: Input/Output Memory Management Unit.
- **PCIe NVMe Disk**: PCIe-based NVMe storage device.

---

**Explanation**

The Diskmap architecture bypasses the kernel I/O framework to provide direct access to NVMe disks. This is achieved through memory-mapped buffers, which allow applications and libraries to interface directly with the hardware. The DMA mechanism facilitates efficient data transfer without involving the kernel, thus improving performance.

---

**Key Points**

- **User space** accesses memory-mapped buffers.
- **Kernel space** receives DMA requests from user space.
- **Admin qpairs** handle inter-process communication between user and kernel.

---

**Technical Details**

- Diskmap uses a combination of SQ/CQ pairs for request handling.
- DMA transfers data directly between memory and the controller.
- libnvme provides a library interface for NVMe operations.
- The architecture supports multiple NVMe disks (nvme0-1, nvme0-2).

---

**Applications**

- High-performance computing.
- Big data analytics.
- Cloud storage services.
- Real-time data processing.

---

**Benefits**

- Reduced latency due to kernel bypass.
- Increased data throughput.
- Lower overhead for I/O operations.

---

**Limitations**

- Requires specific hardware support for DMA.
- Limited by software and driver compatibility.
- More complex setup compared to traditional I/O frameworks.

---

**Further Reading**

- [Diskmap: Kernel-Bypass I/O Framework](https://example.com/diskmap).
- NVMe specification: [nvme.org](https://nvme.org).
- DMA mechanisms: [Intel](https://developer.intel.com).
- Memory-mapped files: [UNIX](https://man7.org).
The Atlas Execution Pipeline

Network Interface Card (NIC):
- RX (Receive)
- TX (Transmit)

TCP/IP Stack:
- User space
- Webserver
- libnmio
- libnvme

NVMe Disk:
- SQ (Send Queue)
- CQ (Completion Queue)
- Buffers

Kernel:
- Buffers

Diagram illustrating the flow of data from user space to kernel and back.
Atlas vs. Netflix, Unencrypted Content

15% better throughput than Netflix when cache hit ratio is low.

Netflix needs 8 cores, Atlas only needs 4

Almost no CPU stalls: data in LLC when we want it.
Atlas vs. Netflix, Encrypted Content

When cache hit ratio is low, 50% more throughput using half the cores.

Almost half the memory reads for each packet sent.
Netmap doesn’t provide a low-delay fine-grained way to communicate DMA completions. Can’t reuse buffers fast enough (no LIFO stack), and this contributes to some extra cache pressure.
Summary

• Netflix addressed all the low-hanging fruit
  – Very fast, but now **bottlenecked on memory**
• Atlas is a specialized stack
  – Puts SSD directly in TCP control loop
  – Immediately processes disk reads to completion and transmits.
  – 50% throughput improvement with encrypted content, close to 50% reduction in memory reads
• Netflix inspired by Atlas
  – Now experimenting with how to directly trigger encryption off of disk DMA completions in their FreeBSD stack.