Making QUIC Quicker with NIC Offload

Xiangrui Yang, Lars Eggerts, Jörg Ott, Steve Uhlig, Zhigang Sun, Gianni Antichi

NUDT, NetApp, TUM, QMUL
SmartNIC to accelerate transport protocols

A TCP Offload Accelerator for 10 Gb/s Ethernet in 90-nm CMOS

Yasir Hoskote, Member, IEEE, Bradley A. Bloom, Member, IEEE, Vasantha Erraguntla, Member, IEEE, David Finan, Jason Fennessey, Greg Ruhl, James W. Tschunck, Member, IEEE, Siriram Vangala, Howard Wilson, Jianping Xu, and John Regehr, Member, IEEE

Abstract—This programmable engine is designed to offload TCP (inbound processing at wire speed for 10-Gb/s Ethernet, supporting 64-byte minimum packet size. This prototype chip employs a high-speed core and a specialized instruction set. It includes hardware support for dynamically reordering out-of-order packets. In a 90-nm CMOS process, the 8-mm² experimental chip has 400 K transistors. First silicon has been validated to be fully functional and achieves 6.4-Gb/s packet processing performance at 1.72 V and consumes 639 W.

Index Terms—Cable Ethernet, offload, packet processing, spe

Enabling Programmable Transport Protocols in Networks

Mina Tahmashi Arashloo
Princeton University
Alexey Larrov
Princeton University
Manya Ghobadi
MIT
David Walker
Princeton University

Abstract

Data-center network stacks are moving into hardware to achieve 100 Gbps data rates and beyond at low latency and low CPU utilization. However, hardwiring the network stack to the NIC to either be used directly through the socket APIs (TCP Offload Engine (TOE) or to enable RDMA (WARP)). These protocols, however, only use a small fixed set of out of the myriad of possible algorithms for reliable delivery. However, recent changes in the NIC technology and congestion control (TCP) have led to numerous optimizations such as kernel bypassing and zerocopy. Our work explores the use of programmable approaches to enhance the performance of transport protocols in the NIC to achieve higher throughput and lower latency.

10Gbps Implementation of TLS/SSL Accelerator on FPGA

Takashi Isobe, Satoshi Tsutsumi
Central Research Laboratory
Hitachi, Ltd., Japan
{takashi.isobe.fm, satoshi.tsutsumi.ba}@hitachi.com

Abstract—This paper proposes the one-chip architecture to mount all processes for TLS/SSL ciphered communication into one FPGA or ASIC, and shows the 10 Gbps implementation of low-power (25 W) TLS/SSL accelerator on 65 nm FPGA. The usage of FPGA/ASIC enables high efficient processing and low-power consumption by using parallel, optimized and pipelined processing. One-chip architecture achieves high throughput by using a switch to avoid the congestion in exchanging data between multiple processing-blocks. In this research, to reduce the circuit area in the one-chip architecture, high-efficient processing design (a parallel processing circuit shared with other processor circuits) is adopted.

Accelerating QUIC via Hardware Offloads through a Socket

Wochtman, Joanna Muniak, Manasi Deval

Accelerating QUIC via Hardware Offloads through a Socket

Takashi Isobe, Satoshi Tsutsumi
Central Research Laboratory
Hitachi, Ltd., Japan
{takashi.isobe.fm, satoshi.tsutsumi.ba}@hitachi.com

Abstract—This paper proposes the one-chip architecture to mount all processes for TLS/SSL ciphered communication into one FPGA or ASIC, and shows the 10 Gbps implementation of low-power (25 W) TLS/SSL accelerator on 65 nm FPGA. The usage of FPGA/ASIC enables high efficient processing and low-power consumption by using parallel, optimized and pipelined processing. One-chip architecture achieves high throughput by using a switch to avoid the congestion in exchanging data between multiple processing-blocks. In this research, to reduce the circuit area in the one-chip architecture, high-efficient processing design (a parallel processing circuit shared with other processor circuits) is adopted.
And the trend in QUIC...

- Understandation at IETF(v29 so far);
- Used by 4.6% of all websites (9.1% of overall traffic 2019) and growing;
- Google has pushed 42.1% of its traffic via QUIC.

Yet its also a complex thus resource burning protocol.

According to Google[1], QUIC burns 3.5 times more CPU cycles than TCP&TLS.

The question in the context of QUIC is:

**Goal**: What are the *primitives in QUIC* that should be offloaded onto *SmartNICs*?
QUIC is evolving really **FAST**, 29 versions within 3 years, over 20 implemenations!
How do we choose among them?

Principle:

- Comply with the latest draft version? Yes!
- Opensource? Yes, we might need to add instrumentations.
- Same programming language while efficient? Yes!

And it's also good to compare different I/O engines! (socket, kernel-bypass...)
Next is the testbed...

- Server and client are pinned to 2 separate cores and isolated using different network namespace;

- TLEM is used to simulate different traffic scenarios (loss, delay, reorder); **better performance!**

- NIC-offload features are disabled to avoid potential interferences.

Lesson 1: I/O Engines matter A LOT

- Start with one connection:

<table>
<thead>
<tr>
<th>Table 2: Maximum throughput vs CPU usage.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Throughput</td>
</tr>
<tr>
<td>Server</td>
</tr>
<tr>
<td>Client</td>
</tr>
</tbody>
</table>

- with netmap, the overall throughput grows 10x higher compared to other QUIC impls
- around 50% for QUIC impls using posix socket
- with netmap, the core utilization of both server and client gets around 90%

Then the question is: what are the bottlenecks in different QUIC impls?
Lesson 2: Crypto engines cost 40%+ CPU cycles

Then we breakdown the CPU utilization of both server & client

In quant, the performance bottleneck (45%+) is the crypto func used for AEAD operations. While in the other 3 impls, the bottleneck (~45%) is the data copies between user/kernel.
Lesson 3: Packet reordering harms performance

Then, we introduce different levels or traffic interference on the link with TLEM

- quant
- quickly
- picoquic
- mvfst

An example:

picoquic(linear) vs picoquic(splay tree)

An unefficient reorder algorithm could be a potential performance bottleneck

Inspired by: https://github.com/private-octopus/picoquic/issues/741#issuecomment-665062732
Picoquic and Mvfst outperform Quicly of about 4x when the connections exceeds 40.

High throughput without kernel-bypass but instead relying on multiple connections.

CPU cost of each connection doesn't change much.

21 connections simultaneously;

similar to single conn scenario, packet ooo has negative effect on throughput (quicly & mvfst)

Throughput of mvfst is heavily influenced by both packet out-of-order and packet loss, could be a potential bug.
A recap to the measurement we did

• **Lesson #1**: Data copy between user/kernel space costs around **50%** CPU usage, can be avoided efficiently by kernel bypass techniques.

• **Lesson #2**: With kernel-bypass, crypto operations become the main performance bottleneck, costing **40%+** overall cycles.

• **Lesson #3**: The way dealing with packet out-of-order matters a lot to the performance when the network is in such scenarios.
So, how do we offload QUIC efficiently?

• **Guidelines:**
  1. Provide NIC-support for AEAD operations;
  2. Move packet reordering to the NIC;
  3. Keep control operations in the host CPU.

• **High-level Design:**
  • HW: AEAD engine, reorder engine
  • SW: control plane operations

CPU <-----> conn table <-----> NIC
Potential challenges?

- **Hardware/Software Synchronization**
  - a general connection table could be of great help
  - overhead of table entry updating? (AEAD keys, etc.)
  - Algorithms of determine which conn shall be offloaded?

- **Low frequency for most AEAD IP core**
  - the possibility of parallelize multi modules?
  - timing issue & resource usage on FPGA?

- **Packet reordering on FPGA**
  - HBM on Xilinx board (AU280) could be useful
  - TCAM is a perfect tool for reordering on the hardware
  - How to distinguish packet ooo from packet loss (timer shall be needed)
Limitations and ongoing work

• Didn't consider the influence of the offloading features that current NIC provides (GSO, packet pacing, etc);

• Didn't investigate some commercial QUIC implementations like msquic from Microsoft, quiche from Netflix and so on.

But we've started to patch that!

opensource @ https://github.com/Winters123/QUIC-measurement-kit
Questions?