Leveraging FABRIC’s Hardware Resources for Programmable Networking

Nik Sultana

KNIT8 – Mar 21, 2024
Goals for today

1. Examples of using FABRIC’s programmable network hardware.

2. A “starter kit” for your own work.
Background

• Programmable networking

• Applications:
  • Cybersecurity
  • Research network infrastructure
sudo!
sudo!
Integrated computation

Operand 1: 3
Operand 2: 5

Add

(Expand)
Many Alveos on **FABRIC**

- Large geographic span
- Resource for the research community
- Non-networking and non-research applications
- High performance
- Convenient access/interface – including authn and authz, notebooks and SSH
Setup

CP = Control Plane
DP = Data Plane

Alveo node → DP → NIC → Tester node
Generate packet: “Request: 3 + 5”
Request: 3 + 5
### Ethernet ###

dst = 06:e4:2c:4e:db:c2  
src = e8:eb:d3:24:b1:23  
type = 0x1234

### P4calc ###

P = 'P'
Four = '4'
ignore = 0x0
version = 0x1
op = 43
operand_a = 3
operand_b = 5
result = 3405695742

### Raw ###

load = ''

Request: 3 + 5
Alveo node

Tester node

Request: 3 + 5

KNIT8 – N. Sultana
Alveo node

DP

CP

Tester node

NIC

Reply: 8

KNIT8 – N. Sultana
### [Ethernet]###

dst = e8:eb:d3:24:b1:23
src = 06:e4:2c:4e:db:c2
type = 0x1234

### [P4calc]###

P = 'P'
Four = '4'
ignore = 0x0
version = 0x1
op = 43
operand_a = 3
operand_b = 5
result = 8

### [Raw]###

load = 

---

Alveo node

DP

Reply: 8

Tester node
Received packet: “Reply: 8”
Alveo node

DP

NIC

y = f(x)

CP

x

y

Tester node
$y = f(x)$
Programmable Networking

• (Limited) computation in the data plane

• Rather than confine programmability to edge or control.
Programmable Networking

• (Limited) computation in the data plane

Less Expressiveness
Better Performance

• Rather than confine programmability to edge or control.
Programmable Networking in FABRIC

• BMv2 and DPDK-based: see examples from *Uni. of South Carolina*.  

• FPGAs including Alveo’s: *Northeastern + UMass*.  

• Alveo-based: *ESnet* (this talk)
SmartNIC framework

• Origins: performance requirements of large-scale and high-performance DoE networks.

• Supports P4: doesn’t require hardware-engineering skills.

• Extensible and open-source.

• Streamlined, modular workflow based on using containers.

• Builds on AMD/Xilinx toolchain. (See University Program.)
FABRIC’S Alveos:
1) What use?
2) How use?
What can we use Alveos for?

• Ongoing research
  • Network monitoring
  • Cybersecurity

• IIT Courses:
  CS595 (Applications of Programmable Networking) :
  http://www.cs.iit.edu/~nsultana1/teaching/F23CS595/

  CS542 (Computer Networking) :
  http://www.cs.iit.edu/~nsultana1/teaching/S24CS542/
Using FABRIC’s Alveos: Network Profiling

- **Goal:** Understanding the traffic composition and characteristics of FABRIC experiments. Sharing this data + mechanism with the community of testbed users.

- **Motivation:** Provide data and insight for research projects on “sui generis” network. (Unliked datacenter, telecom, etc).

- **How:** Profiler is *itself* a FABRIC experiment.
Using FABRIC’s Alveos: Network Profiling

• **Findings:** a lot to digest! Speak to Nishanth and Prajwal if you missed their demos on Tuesday.

• **Role of Alveos:** offloading sampling, filtering, truncation, and blinding to the Smart NIC from the host.
Using FABRIC’s Alveos: Cybersecurity Research

• **Goal:** Adapting “Remote Attestation” to network dataplanes.

• **Motivation:** Early detection of misconfigurations and APTs, mitigating attack surface from programmability.

• **How:** In-band carrying of verifiable evidence about network element state.
Using FABRIC’s Alveos: Cybersecurity Research

- **Prototypes and Security Properties:**
  Speak to **Alexander** and **Hyunsuk** if you missed their demos on Tuesday.

- **Role of Alveos:** “sandwiching” black box, distrusted, third-party network elements. Provide visibility of those elements’ behavior, and serializing and verifying evidence.
How do we use FABRIC’s Alveos?

• Development done outside FABRIC, Experiment done inside FABRIC.

• Development:
  • The programming language(s)
  • The toolchain(s)

• Experiment: allocating and using Alveos on FABRIC.
Plan

1. The development process using ESnet’s SmartNIC framework.
2. Allocating the necessary resources on FABRIC, including an Alveo.
3. How to deploy a program on an Alveo (that’s in FABRIC).
4. Running and testing the program.
5. Diagnosis.

What we won’t see: Recompiling the program running in the FPGA. (The compilation takes too long for this session.)
SmartNIC framework
Programming

• Header specification and *parsing*.

• Specification of program logic.

• Specification of *tables* and *actions*.
Programming

1. Header specification and parsing.

```cpp
header p4calc_t {
    bit<8> p;
    bit<8> four;
    bit<6> ver;
    bit<10> op;
    bit<32> operand_a;
    bit<32> operand_b;
    bit<32> res;
}

struct headers {
    ethernet_t ethernet;
    p4calc_t p4calc;
}
```
Programming

2. Specification of program logic.

```cpp
header p4calc_t {
    bit<8> p;
    bit<8> four;
    bit<6> ver;
    bit<10> op;
    bit<32> operand_a;
    bit<32> operand_b;
    bit<32> res;
}

struct headers {
    ethernet_t ethernet;
    p4calc_t p4calc;
}

const bit<16> P4CALC_ETYPE = 0x1234;
const bit<8> P4CALC_P = 0x50; // 'P'
const bit<8> P4CALC_4 = 0x34; // '4'
const bit<16> P4CALC_VER = 0x0001; // v0.1
const bit<8> P4CALC_PLUS = 0x2b; // '+'
const bit<8> P4CALC_MINUS = 0x2d; // '-'
const bit<8> P4CALC_AND = 0x26; // '&'
const bit<8> P4CALC_OR = 0x7c; // '|' 
const bit<8> P4CALC_CARET = 0x5e; // '^'

if (hdr.p4calc.isValid()) {
    if (hdr.p4calc.p == P4CALC_P &&
        hdr.p4calc.four == P4CALC_4 &&
        hdr.p4calc.ver == P4CALC_VER)
        calculate.apply();
    } else operation_drop();
```
3. Specification of *tables* and *actions*.

```p4
table calculate {
    key = { hdr.p4calc.op : exact; }
    actions = {
        operation_add;
        operation_sub;
        operation_and;
        operation_or;
        operation_xor;
        operation_drop;
    }
    size = 1024;
    default_action = operation_drop();
}
```
Programming

3. Specification of *tables* and *actions*.

```p4
table calculate {
    key = { hdr.p4calc.op : exact; }
    actions = {
        operation_add;
        operation_sub;
        operation_and;
        operation_or;
        operation_xor;
        operation_drop;
    } size = 1024;
    default_action = operation_drop();
}

action operation_add() {
    send_back(hdr.p4calc.operand_a + hdr.p4calc.operand_b);
}

action send_back(bit<32> result) {
    bit<48> tmp;
    hdr.p4calc.res = result;
    tmp = hdr.ethernet.dstAddr;
    hdr.ethernet.dstAddr = hdr.ethernet.srcAddr;
    hdr.ethernet.srcAddr = tmp;
    sn_meta.egress_port = sn_meta.ingress_port;
}
```
Running Alveo experiments

• Need “Component.FPGA” permission.

• Creating a FABRIC slice as normal.

• Configuring:
  • Hosts
  • Alveo card
  • Program’s tables
Creating a slice

Include a ‘FPGA_Xilinx_U280’ component as part of a node in your slice.
Configuring the hosts

1. Setting up host and software dependencies.
2. Obtaining container images used by ESnet’s framework.
3. Uploading scripts that support this experiment.
Configuring the Alveo card

1. Creating the configuration container
2. Configuration through “sn-cli” tool: sw, qdma, cmac
Configuring the program’s tables

• The `sn-p4-cli` tool.

• Adding, removing, clearing rules – cannot `inspect` rules.

• By changing table entries we’ll swap `addition` and `xor`. 
Clearing the table

```
ubuntun@nic-node:~$ sudo python3 calc.py '3 + 1'
```
Clearing the table

```
ubuntu@nic-node:~$ sudo python3 calc.py '3 + 1'
4
ubuntu@nic-node:~$ sudo python3 calc.py '3 - 1'
2
ubuntu@nic-node:~$ sudo python3 calc.py '3 ^ 1'
2
ubuntu@nic-node:~$ sudo python3 calc.py '3 & 1'
1
ubuntu@nic-node:~$ sudo python3 calc.py '3 | 1'
3
ubuntu@nic-node:~$ sudo python3 calc.py '3 | 1'
Didn't receive response
ubuntu@nic-node:~$ sudo python3 calc.py '3 & 1'
Didn't receive response
ubuntu@nic-node:~$ sudo python3 calc.py '3 & 1'
Didn't receive response
ubuntu@nic-node:~$
```
Swapping rules

```
ubuntu@nic-node:~$ sudo python3 calc.py '3 + 1'
4
ubuntu@nic-node:~$ sudo python3 calc.py '3 - 1'
2
ubuntu@nic-node:~$ sudo python3 calc.py '3 & 1'
2
ubuntu@nic-node:~$ sudo python3 calc.py '3 | 1'
3
ubuntu@nic-node:~$ sudo python3 calc.py '3 | 1'
3
```

Didn't receive response
```
ubuntu@nic-node:~$ sudo python3 calc.py '3 & 1'
Didn't receive response
ubuntu@nic-node:~$ sudo python3 calc.py '3 & 1'
Didn't receive response
ubuntu@nic-node:~$ sudo python3 calc.py '3 + 1'
4
ubuntu@nic-node:~$ sudo python3 calc.py '3 ^ 1'
4
ubuntu@nic-node:~$ sudo python3 calc.py '3 ^ 1'
4
```

Didn't receive response
```
ubuntu@nic-node:~$ sudo python3 calc.py '3 + 1'
Didn't receive response
ubuntu@nic-node:~$ sudo python3 calc.py '3 - 1'
Didn't receive response
ubuntu@nic-node:~$ sudo python3 calc.py '3 - 1'
```
First we set some parameters for the slice, such as node names and where we want to site the slice.

```python
alveo_node_name='alveo-node'
nic_node_name='nic-node'

site = <e.g., 'HAWI'>

slice_name=f'P4 calculator on {site}'

esnet_smartnic_dpkd_docker = "wget -q http://.../esnet_smartnic-dpkd-docker.tgz"
esnet_xilinx_labtools_docker = "wget -q http://.../esnet_xilinx-labtools-docker.tgz"

artifact_origin = "http://.../artifacts.au280.p4_only.0.zip_CALC"
```

The site will have two nodes. One node controls an Alveo card. The other node is used to send traffic to the Alveo.
Plan

1. The development process using ESnet’s SmartNIC framework.
2. Allocating the necessary resources on FABRIC, including an Alveo.
3. How to deploy a program on an Alveo (that’s in FABRIC).
4. Running and testing the program.
5. Diagnosis.

*What we won’t see:* Recompiling the program running in the FPGA. (The compilation takes too long for this session.)
Running + Testing the program

The program is already running.

Testing: let’s examine the script that produces and interprets experiment traffic.
Test script

class P4calc(Packet):
    name = "P4calc"
    fields_desc = [
        StrFixedLenField("P", "P", length=1),
        StrFixedLenField("Four", "4", length=1),
        XByteField("version", 0x01),
        StrFixedLenField("op", "+", length=1),
        IntField("operand_a", 0),
        IntField("operand_b", 0),
        IntField("result", 0xCAFECAFE)
    ]

pkt = Ether(dst='00:04:00:00:00:00', type=0x1234) / P4calc(op=ts[1].value,
operand_a=int(ts[0].value),
operand_b=int(ts[2].value))
Diagnosis

- Using `sn-cli` to read counter values.
- Counters provide insight about internal packet loss.
Thank you

• **Researchers who use(d) FABRIC at Illinois Tech:**

• **Community:**
  Ilya Baldin (JLAB), Peter Bengough (ESnet), Gordon Brebner (AMD), Dale Carder (ESnet), Mert Cevik (RENCI), Zongming Fei (UKY), Jim Griffioen (UKY), Yatish Kumar (ESnet), Tom Lehman (Virnao), Inder Monga (ESnet), Chris Neely (AMD), Anita Nikolich (UIUC), Phil Porras (SRI), Paul Ruth (RENCI), Stacey Sheldon (ESnet), Komal Thareja (RENCI), Xi Yang (ESnet), Vinod Yegneswaran (SRI), and the P4 community.
How to get started

• FABRIC Forums: https://learn.fabric-testbed.net/forums/forum/all-about-fpgas/
How to get started

• FABRIC Forums: https://learn.fabric-testbed.net/forums/forum/all-about-fpgas/

• This tutorial’s “starter kit” has been added to our documentation site: http://packetfilters.cs.iit.edu/esnet-smartnic-tutorial/