CS344 – Lecture 3
P4 Toolchain for BMv2 software simulation
Basic Workflow

- **test.p4**
  - p4c-bm2-ss
  - test.json

**simple_switch_CLI**
- Program-independent CLI and Client
- TCP Socket (Thrift)
- Program-independent Control Server

**simple_switch (BMv2)**
- PRE
  - Ingress
    - Parser
    - Deparser
    - Port Interface
- Egress

**Linux Kernel**
- veth0..n
- P4 Debugger
- Packet sniffer
Step 1: P4 Program Compilation

```
$ p4c-bm2-ss -o test.json test.p4
```

```
test.p4
   \_ p4c-bm2-ss
     \_ test.json
```
Step 2: Preparing veth Interfaces

```bash
$ sudo ~/p4lang/tutorials/examples/veth_setup.sh

# ip link add name veth0 type veth peer name veth1  
# for iface in "veth0 veth1"; do  
  ip link set dev $iface up  
  sysctl net.ipv6.conf.$iface.disable_ipv6=1  
  TOE_OPTIONS="rx tx sg tso ufo gso gro lro rxvlan txvlan rxhash"  
  for TOE_OPTION in $TOE_OPTIONS; do  
    /sbin/ethtool --offload $intf "$TOE_OPTION"  
  done
done
```

---

Copyright © 2018 – P4.org
Step 3: Starting the model

```
$ sudo simple_switch --log-console --dump-packet-data 64 \ 
  -i 0@veth0 -i 1@veth2 ... \  
  test.json
```
Step 4: Starting the CLI

$ simple_switch_CLI

BMv2 CLI

Program-independent
CLI and Client

TCP Socket
(Thrift)

Program-independent
Control Server

test.p4

test.json

veth0..n

Linux Kernel

Port Interface

Parser

Deparser

PRE

Egress

Ingress
Step 5: Sending and Receiving Packets

- **scapy**
  
  ```
  p = Ethernet()/IP()/UDP()/'Payload' 
  sendp(p, iface="veth0")
  ```

- **Ethereal, etc..**

- **Wireshark, tshark, tcpdump**

- **scapy**
  ```
  sniff(iface="veth0", prn=lambda x: x.show())
  ```
Overview
NetFPGA = Networked FPGA

• A line-rate, flexible, open networking platform for teaching and research

Network Interface Card
Hardware Accelerated Linux Router
IPv4 Reference Router
Traffic Generator
Openflow Switch
More Projects
Add Your Project
NetFPGA Family of Boards


NetFPGA-10G (2010)

NetFPGA-1G-CML (2014)

NetFPGA-SUME (2014)
International Community

- Over 1,200 users, using over 3500 cards at 200 universities in over 47 countries

- Join the mailing list: cl-netfpga-sume-beta@lists.cam.ac.uk
NetFPGA board

Networking Software running on a standard PC

A hardware accelerator built with FPGA driving 1/10/100 Gb/s network links
NetFPGA consists of …

Four elements:

• NetFPGA board
• Tools + reference designs
• Contributed projects
• Community
Xilinx Virtex 7 690T

- Optimized for high-performance applications
- 690K Logic Cells
- 52 Mb RAM
- 3 PCIe Gen. 3 Hard cores
Memory Interfaces

- **DRAM:**
  2 x DDR3 SoDIMM
  1866MT/s, 4GB

- **SRAM:**
  3 x 9MB QDRII+, 500MHz
Host Interface

- PCIe Gen. 3
- x8 (only)
- Hardcore IP
Front Panel Ports

- 4 SFP+ Cages
- Directly connected to the FPGA
- Supports 10GBase-R transceivers (default)
- Also Supports 1000Base-X transceivers and direct attach cables
Expansion Interfaces

- **FMC HPC connector**
  - VITA-57 Standard
  - Supports Fabric Mezzanine Cards (FMC)
  - 10 x 12.5Gbps serial links

- **QTH-DP**
  - 8 x 12.5Gbps serial links
Storage

- 128MB FLASH
- 2 x SATA connectors
- Micro-SD slot
- Enable standalone operation
Reference Switch Pipeline

- **Five stages**
  - Input port
  - Input arbitration
  - Forwarding decision and packet modification
  - Output queuing
  - Output port
- **Packet-based module interface**
- **Pluggable design**
Full System Components

Software

PCIe Bus

NetFPGA

Application

Driver

nf0

nf1

nf2

nf3

ioctl

DMA

Registers

CPU RxQ

CPU TxQ

user data path

10GE Tx

10GE Rx

Ports

AXI Lite
NetFPGA – Host Interaction

• Linux driver interfaces with hardware
  ○ Packet interface via standard Linux network stack
  ○ Register reads/writes voa ioctl system call with wrapper functions
    ■ rwaxi(int address, unsigned *data);
    ■ Eg: rwaxi(0x7d400000, &val)
NetFPGA to Host Packet Transfer

1. Packet arrives – forwarding table sends to DMA queue

2. Interrupt notifies driver of packet arrival

3. Driver sets up and initiates DMA transfer
NetFPGA to Host Packet Transfer

4. NetFPGA transfers packet via DMA

5. Interrupt signals completion of DMA

6. Driver passes packet to network stack
Host to NetFPGA Packet Transfer

1. Software sends packet via network sockets. Packet delivered to driver

2. Driver sets up and initiates DMA transfer

3. Interrupt signals completion of DMA
NetFPGA Register Access

1. Software makes ioctl call on network socket. ioctl passed to driver

2. Driver performs PCIe memory read/write
Overview
General Process for Programming a P4 Target

P4->NetFPGA tools

P4 Program ➔ P4 Compiler ➔ Target

SimpleSumeSwitch Architecture

P4 Architecture Model

Target-specific configuration binary

Load

Data Plane

Control Plane

Add/remove table entries
Extern control
Packet-in/out

CPU port

RUNTIME

NetFPGA SUME

NetFPGA tools
**P4\(\rightarrow\)NetFPGA Compilation Overview**

**P4 Program**

**Xilinx P4\(_{16}\) Compiler**

**Xilinx SDNet Tools**

**SimpleSumeSwitch Architecture**

**NetFPGA Reference Switch**

- Input Arbiter
- SimpleSume Switch
- Output Queues
  - 10GE RxQ
  - 10GE Tx
  - DMA
Packet Processing Spec.
- PX (domain specific language)
- describe function in packet-oriented terms

SDNet Compiler
- Throughput & Latency
- Resources
- Programmability

Xilinx SDNet Design Flow & Use Model

Firmware

Tailored Packet Processor
Xilinx P4 Design Flow & Use Model

- Xilinx P4_{16} Compiler
  - $ p4c-sdnet switch.p4

Verification Environment

- Top level Verilog wrapper
- Verilog Engines (Encrypted)
- System Verilog Testbench
- Lookup Engine C++ Drivers
- High level C++ Testbench
Considerations When Mapping to SDNet

- Identifying parallelism within P4 parser and control blocks
  - table lookups
  - actions
  - etc.

- P4 packet processing model
  - extract entire header from packet
  - updates apply directly to header
  - deparser re-inserts header back into packet

- SDNet packet processing model
  - stream packet through “engines”
  - modify header values in-line without removing and re-inserting
Mapping P4 Architectures to SDNet
Support for Multiple Architectures

SimpleSumeSwitch

Only Parser

➢ Pull information from packet w/o updates
SimpleSumeSwitch Architecture Model for SUME Target

- P4 used to describe parser, match-action pipeline, and deparser
Standard Metadata in SimpleSumeSwitch Architecture

/* standard sume switch metadata */
struct sume_metadata_t {
    bit<16> dma_q_size;
    bit<16> nf3_q_size;
    bit<16> nf2_q_size;
    bit<16> nf1_q_size;
    bit<16> nf0_q_size;
    bit<8> send_dig_to_cpu; // send digest_data to CPU
    bit<8> dst_port; // one-hot encoded
    bit<8> src_port; // one-hot encoded
    bit<16> pkt_len; // unsigned int
}

*_q_size – size of each output queue, measured in terms of 32-byte words, when packet starts being processed by the P4 program

src_port/dst_port – one-hot encoded

user_metadata/digest_data – structs defined by the user
Interface Naming Conventions

src / dst port fields:

x-x-x-x-x-x-x-x-x

nf0  nf1  nf2  nf3  ioctl

Application

Driver

DMA

Registers

AXI Lite

Ports

user data path

CPU RxQ  CPU TxQ

10GE Tx  10GE Rx

Copyright © 2018 – P4.org
#include <core.p4>
#include <sume_switch.p4>

/******* CONSTANTS *******/
#define IPV4_TYPE 0x0800

/******* TYPES *******/
typedef bit<48> EthAddr_t;
header Ethernet_h {...}
struct Parsed_packet {...}
struct user_metadata_t {...}
struct digest_data_t {...}

/******* EXTERN FUNCTIONS *******/
extern void const_reg_rw(...);

/******* PARSERS and CONTROLS *******/
parser TopParser(...) {...}
control TopPipe(...) {...}
control TopDeparser(...) {...}

/******* FULL PACKAGE *******/
SimpleSumeSwitch(TopParser(), TopPipe(), TopDeparser()) main;
P4 → NetFPGA Extern Function library

• Implement platform specific functions
  • Black box to P4 program

• Implemented in HDL

• Stateless – reinitialized for each packet

• Stateful – keep state between packets

• Xilinx Annotations
  • @Xilinx_MaxLatency() – maximum number of clock cycles an extern function needs to complete
  • @Xilinx_ControlWidth() – size in bits of the address space to allocate to an extern function
Stateless vs. stateful operations

Stateless operation: \( \text{pkt.f4} = \text{pkt.f1} + \text{pkt.f2} - \text{pkt.f3} \)

Can pipeline stateless operations
Stateless vs. stateful operations

Stateful operation: \( x = x + 1 \)

\( X = 0 \)

\( \text{pkt.tmp} = x \)
\( \text{tmp} = 0 \)
\( \text{pkt.tmp} ++ \)
\( \text{tmp} = 1 \)
\( x = \text{pkt.tmp} \)

X should be 2, not 1!
Stateless vs. stateful operations

Stateful operation: \( x = x + 1 \)

\[ X++ \]

Cannot pipeline, need atomic operation in h/w
P4→NetFPGA Extern Function library

- HDL modules invoked from within P4 programs
- Stateful Atoms [1]

<table>
<thead>
<tr>
<th>Atom</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R/W</td>
<td>Read or write state</td>
</tr>
<tr>
<td>RAW</td>
<td>Read, add to, or overwrite state</td>
</tr>
<tr>
<td>PRAW</td>
<td>Predicated version of RAW</td>
</tr>
<tr>
<td>ifElseRAW</td>
<td>Two RAWs, one each for when predicate is true or false</td>
</tr>
<tr>
<td>Sub</td>
<td>IfElseRAW with stateful subtraction capability</td>
</tr>
</tbody>
</table>

- Stateless Externs

<table>
<thead>
<tr>
<th>Atom</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IP Checksum</td>
<td>Given an IP header, compute IP checksum</td>
</tr>
<tr>
<td>LRC</td>
<td>Longitudinal redundancy check, simple hash function</td>
</tr>
<tr>
<td>timestamp</td>
<td>Generate timestamp (granularity of 5 ns)</td>
</tr>
</tbody>
</table>

- Add your own!

Adding Custom Externs

1. Implement verilog extern module

2. Add entry to $SUME_SDNET/bin/extern_data.py

   • No need to modify and existing code
   • AXI Lite control interface module auto generated
Packet processing pseudo code:

```plaintext
count[NUM_ENTRIES];

if (pkt.hdr.reset == 1):
    count[pkt.hdr.index] = 0
else:
    count[pkt.hdr.index]++
```
Using Atom Externs in P4 – Resetting Counter

```c
#define REG_READ 0
#define REG_WRITE 1
#define REG_ADD 2

// count register
@Xilinx_MaxLatency(64)
@Xilinx_ControlWidth(3)
extern void count(in bit<3> index, in bit<32> newVal,
in bit<32> incVal, in bit<8> opCode,
out bit<32> result);

bit<16> index = pkt.hdr.index;
bit<32> newVal; bit<32> incVal; bit<8> opCode;
if(pkt.hdr.reset == 1) {
    newVal = 0;
    incVal = 0; // not used
    opCode = REG_WRITE;
} else {
    newVal = 0; // not used
    incVal = 1;
    opCode = REG_ADD;
}

bit<32> result; // the new value stored in count reg
count_reg_raw(index, newVal, incVal, opCode, result);
```
API & Interactive CLI Tool Generation

• Both Python API and C API
  • Manipulate tables and stateful elements in P4 switch
  • Used by control-plane program

• CLI tool
  • Useful debugging feature
  • Query various compile-time information
  • Interact directly with tables and stateful externs in at run time
P4→NetFPGA Workflow

1. Write P4 program
2. Write externs
3. Write python gen_testdata.py script
4. Compile to Verilog / generate API & CLI tools
5. Run simulations
6. Build bitstream
7. Check implementation results
8. Test the hardware

All of your effort will go here
Debugging P4 Programs

- SDNet HDL Simulation
- SDNet C++ simulation
  - Verbose packet processing info
  - Output PCAP file
- Full SUME HDL simulation
- Custom Python Model
Assignment 1: Switch as a Calculator
Switch as a Calculator

- **Supported Operations**
  - ADD – add two operands
  - SUBTRACT – subtract two operands
  - ADD_REG – add operand to current value in the register
  - SET_REG – overwrite the current value in the register
  - LOOKUP – Lookup the given key in the table

```c
header Calc_h {
  bit<32> op1;
  bit<8> opCode;
  bit<32> op2;
  bit<32> result;
}
```
Switch as a Calculator

User PC

DST: MAC1
SRC: MAC2
Type: CALC_TYPE

Ethernet

Calc

op1: 1
opCode: ADD
op2: 2
result: 0
Payload...

NetFPGA SUME
Switch as a Calculator

User PC

NetFPGA SUME

Ethernet

DST: MAC1
SRC: MAC2
Type: CALC_TYPE

Calc

op1: 1
opCode: ADD
op2: 2
result: 0 3

Payload...
Switch as a Calculator

User PC

NetFPGA SUME

Ethernet

DST: MAC2
SRC: MAC1
Type: CALC_TYPE

Calc

op1: 1
opCode: ADD
op2: 2
result: 3

Payload...
Switch Calc Operations

**ADD**
\[
\text{result} = \text{op1} + \text{op2}
\]

**SUB**
\[
\text{result: } \text{op1} - \text{op2}
\]

**ADD_REG**
\[
\text{result} = \text{op2} + \text{const[op1]}
\]

**SET_REG**
\[
\text{const[op1]} = \text{op2}
\]

**LOOKUP**
\[
\begin{array}{|c|c|}
\hline
\text{key} & \text{val} \\
\hline
0 & 1 \\
1 & 16 \\
2 & 16^2 \\
3 & 16^3 \\
\hline
\end{array}
\]

result: val
Research topics
Examples of ongoing P4 Research Topics

- **P4 Infrastructure**
  - Programmable scheduling
  - Programmable target architectures
  - PacketMod

- **Data-plane Programs**
  - In-band network telemetry
  - Congestion control
  - Load balancing

- **Networking-Offloading Applications**
  - Aggregation for MapReduce applications
  - Key-value caching
  - Consensus
Programmable Scheduling

Why scheduler is not programmable … so far

- Plenty of scheduling algorithms, but no consensus on right abstractions. Contrast to:
  - Parse graphs for parsing
  - Match-Action tables for forwarding

- Scheduler has tight timing requirements
  - One decision every few ns
What does the Scheduler do?

Decides:

- In what order are packets sent?
  - Ex: FCFS, Priorities, WFQ
- At what time are packets sent?

Key observation:

- For many algorithms, the relative order in which packets are sent does not change with future arrivals
  - i.e. scheduling order can be determined before enqueue
PIFO

- PIFO - proposed abstraction that can be used to implement many scheduling algorithms
- Packets are pushed into an arbitrary location based on computed rank

```
f = flow(pkt)
p.tmp = T[f] + p.len
...
...
p.rank = 2 * p.tmp
```

Rank Computation

PIFO scheduler

(fixed logic)
PIFO Tree

Hierarchical Packet Fair Queuing

root

Red (0.5)

a (0.99)
b (0.01)

Blue (0.5)

x (0.5)
y (0.5)

PIFO-root (WFQ on Red & Blue)

PIFO-Red (WFQ on a & b)

PIFO-Blue (WFQ on x & y)
PIFO Remarks

- Very limited scheduling in modern switching chips
  - Deficit Round Robin, traffic shaping, strict priorities
- Scheduling algorithms that can be implemented with PIFO
  - Weighted Fair Queueing, Token Bucket Filtering, Hierarchical Packet Fair Queueing, Least-Slack Time-First, the Rate Controlled Service Disciplines, and fine-grained priority scheduling (e.g., Shortest Job First)
- PIFO cannot implement algorithms that require
  - Changing the scheduling order of all packets of a flow
  - Output rate limiting
- PIFO implementation feasibility?
Programmable Target Architectures

Observations:

- Current P4 expectation: target architectures are *fixed*, specified in English
- FPGAs can support many different architectures

Idea:

- Extend P4 to allow description of target architectures
  - More precise definition than English description
- Generate implementation on FPGA
- Easily integrate custom modules
- Explore performance tradeoffs of different architectures
Many Possible Architectures…

SimpleSumeSwitch

Parser → M/A → Deparser → Output Queues

V1 Model

Parser → M/A → TM → M/A → Deparser → Output Queues

Portable Switch Architecture

Parser → M/A → Deparser → TM → Parser → M/A → Deparser → Output Queues
Many Possible Architectures…

Custom Traffic Manager

Programmable Packet Generator
package SimpleSumeSwitch<H, M, D> {
    Parser<H, M, D> TopParser,
    Pipe<H, M, D> TopPipe,
    Deparser<H, M, D> TopDeparser) {

    // Top level I/O
    packet_in instream;
    inout sume_metadata_t sume_metadata;
    out D digest_data;
    packet_out outstream;

    // Connectivity of the architecture
    connections {
        // TopParser input connections
        TopParser.b = instream;
        TopParser.sume_metadata = sume_metadata;

        // TopPipe <-- TopParser
        TopPipe.p = TopParser.p;
        TopPipe.user_metadata = TopParser.user_metadata;
        TopPipe.digest_data = TopParser.digest_data;
        TopPipe.sume_metadata = TopParser.sume_metadata;

        // TopDeparser <-- TopPipe
        TopDeparser.p = TopPipe.p;
        TopDeparser.user_metadata = TopPipe.user_metadata;
        TopDeparser.digest_data = TopPipe.digest_data;
        TopDeparser.sume_metadata = TopPipe.sume_metadata;

        // TopDeparser output connections
        digest_data = TopDeparser.digest_data;
        sume_metadata = TopDeparser.sume_metadata;
        outstream = TopDeparser.b;
    }
}
Workflow

- Two Actors: (1) **Target Architecture Designer**, (2) P4 Programmer

  **Provides:**
  - P4+ architecture declaration

  **Implements:**
  - non-P4 elements
  - externs in target architecture

- Someone who is more familiar with FPGA development
Workflow

- Two Actors: (1) Target Architecture Designer, (2) P4 Programmer

Implementation of P4 elements

<table>
<thead>
<tr>
<th>Compile to PX</th>
</tr>
</thead>
<tbody>
<tr>
<td>PX subsystems</td>
</tr>
</tbody>
</table>

P4+ architecture description

<table>
<thead>
<tr>
<th>Compile to PX</th>
</tr>
</thead>
<tbody>
<tr>
<td>Partial PX System</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Complete PX System</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Compile to Verilog</th>
</tr>
</thead>
<tbody>
<tr>
<td>HDL switch design</td>
</tr>
</tbody>
</table>
In-band Network Telemetry (INT)

1. Which path did my packet take?

   "I visited: switch 1 @ 780ns, switch 9 @ 1.3us, switch 12 @ 2.4us"

2. Which rules did my packet follow?

   "In switch 1, I followed rules 75 and 250. In switch 9, rules 3 and 80"
In-band Network Telemetry (INT)

3. How long did my packet queue at each switch?

“Delay: 100ns, 200ns, 19740ns”

4. Who did my packet share the queue with?
In-band Network Telemetry (INT)

1. Which path did my packet take?
2. Which rules did my packet follow?
3. How long did my packet queue at each switch?
4. Who did my packet share the queue with?

No need to add a single additional packet!
Congestion Control

Reactive Congestion Control

- Adjust Flow Rate
- Measure Congestion

- No use of explicit information about traffic matrix
- Can only react and move in right direction
- Reactive techniques are slow to converge (10s-100s of RTTs)
- Typical flows will finish in just a few RTTs as we move towards higher link speeds

Fraction of Total Flows in Bing Workload

- Small (1-10KB)
- Medium (10KB-1MB)
- Large (1MB-100MB)

<table>
<thead>
<tr>
<th>Typical Flow Completion Times</th>
</tr>
</thead>
<tbody>
<tr>
<td>10 Gb/s</td>
</tr>
<tr>
<td>40 Gb/s</td>
</tr>
<tr>
<td>100 Gb/s</td>
</tr>
</tbody>
</table>
Proactive Congestion Control

- Proactive techniques converge much more quickly than reactive.
- Faster convergence times lead to lower flow completion times.
An example proactive scheme

Switch Computation

N = 1 flow
C = 10Gb/s
Fair share = C/N = 1 Gb/s

Control Packet

Per-flow state:
• BW demand
• Current bottleneck
Per-link state

Sending host adjusts sending rate
An example proactive scheme

Switch Computation

N = 2 flow
C = 10 Gb/s
Fair share = C/N = 5 Gb/s
Proactive Algorithm in P4

L2 Forwarding Logic

Set low priority

Set high priority

Read/update link state

Compute fair share rate

Set bandwidth demand

Priority Output Queues

not ctrl pkt

is ctrl pkt
In-Network Computation

• Programmable data plane hardware → opportunity to reconsider division of computation

• What kinds of computation should be delegated to network?

• Network computations are constrained:
  ○ Limited memory size (10’s of MB of SRAM)
  ○ Limited set of actions (simple arithmetic, hashing, table lookups)
  ○ Few operations per packet (10’s of ns to process each packet)

• Goals:
  ○ Reduce: application runtime, load on servers, network congestion
  ○ Increase: application scalability

In-Network Aggregation

- Aggregate data at intermediate network nodes to reduce network traffic
- Simple arithmetic operations at switches
- Widely applicable to many distributed applications
  - Machine learning training
  - Graph analytics
  - MapReduce applications
In-Network Aggregation

- Network controller is informed of MapReduce job
  - Configures switches in aggregation tree to perform aggregation
- Significant network traffic reduction \(\rightarrow\) reduced run time
- How to make robust to loss? Encryption?

**Aggregation Tree**

![Aggregation Tree Diagram]

**Reduction Results**

- Data volume
- Reduce time (UDP baseline)
- # packets (TCP baseline)

---

Copyright © 2018 – P4.org
The P4 Language Consortium

- http://p4.org
- Consortium of academic and industry members
- Open source, evolving, domain-specific language
- Permissive Apache license, code on GitHub today
- Membership is free: contributions are welcome
- Independent, set up as a California nonprofit