

# Distributed Computation in P4

Chesley Tan and Danny Qiu

# Introduction

- ❖ Network switches are becoming increasingly powerful, but they are still often highly memory-constrained.
- ❖ With the growth in programmable switches, end-users are able to take full advantage of the computing capabilities of their network switches.
- ❖ The ubiquity of network switches lends itself to distributed applications.

# Distributed Computation in P4

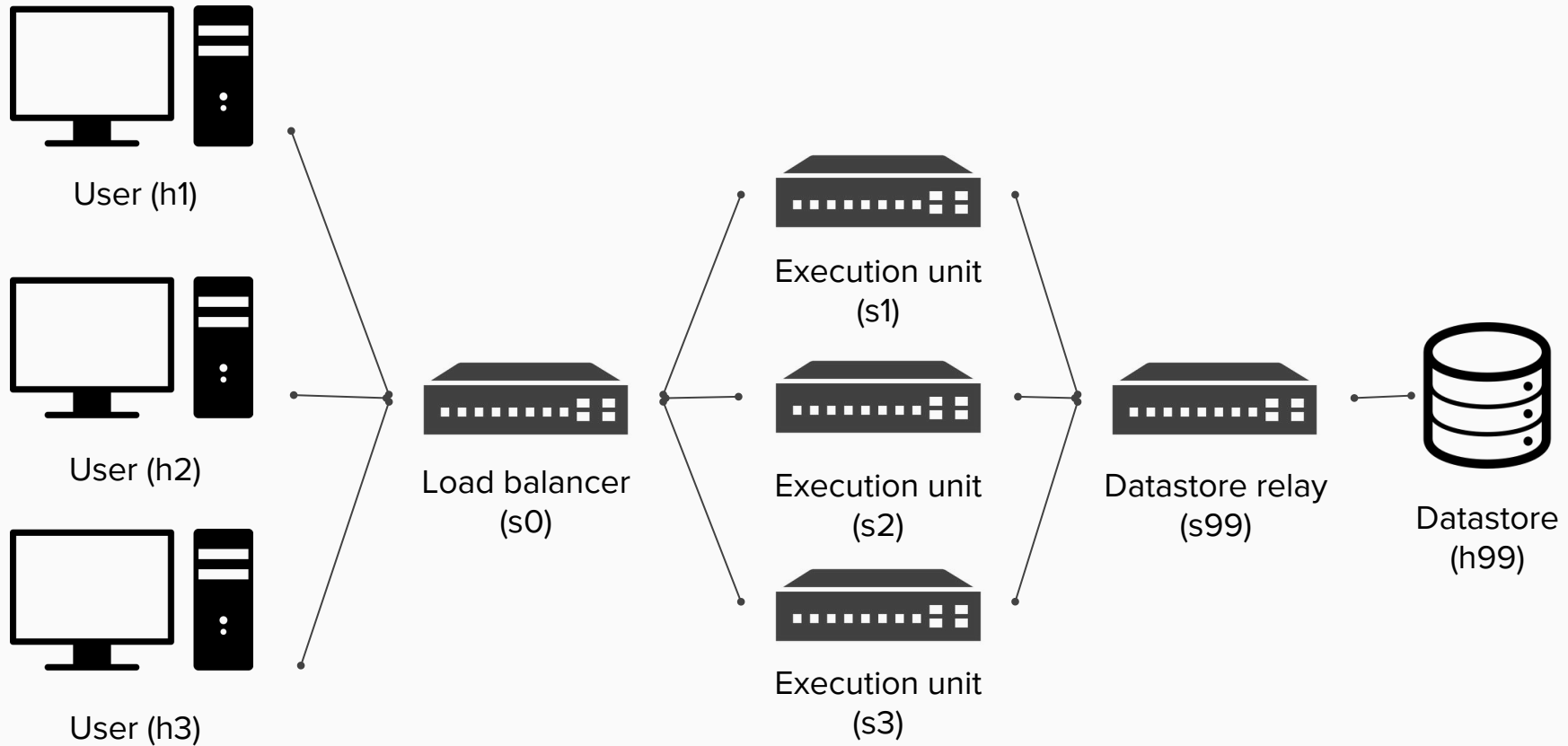
- ❖ We can model the execution of a multi-processor computer on a network of programmable switches running P4 code.
- ❖ The distributed computing system leverages an instruction-based execution model with an assembly language derived from the RISC-V ISA.
- ❖ To provide the abstraction of “unlimited” memory on the programmable switches, we use an external datastore with two-way communication with the switches.

# Assembly

- ❖ Register-register arithmetic instructions
  - ADD, SUB, AND, OR, XOR, SLT, SLTU, SRA, SRL, SLL, MUL
- ❖ Register-immediate arithmetic instructions
  - ADDI, ANDI, ORI, XORI, SLTI, SLTIU, SRAI, SRLI, SLLI, LUI, AUIPC
- ❖ Memory instructions
  - LW, SW
- ❖ Unconditional jump instructions
  - JAL, JR, JALR
- ❖ Conditional branch instructions
  - BEQ, BNE, BLT, BGE, BLTU, BGEU



# System Architecture



# Properties

- ❖ The system allows us to take advantage of the computing power offered by programmable switches to execute arbitrary programs.
- ❖ The system allows programmable switches to leverage “unlimited” memory using the datastore.
- ❖ The computation system naturally simulates multi-programming, with the packet scheduler acting as a thread scheduler.

# Progress

- ❖ Fully implemented `~\_(\ツ)\_/~`
- ❖ Tested using unit tests and (memoized) fibonacci program

# In Action:

## Memoized Fibonacci!

```
assembly-switch
P4 Tutorial 2018-06-01 [Running]
[[11:31:13]] root@p4: /media/ef_CS6114_Network_Programming_Languages/assembly-switch (master x) > ./fibonacci.py
WARNING: No route found for IPv6 destination :: (no default route?)
0000 00 00 FF 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....E.
0010 00 E4 00 01 00 00 40 BF 65 8A 0A 00 00 01 0A FF .....@.e.....
0020 FF 00 00 00 03 E8 00 00 00 00 00 00 00 00 00 00 .....
0030 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0040 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0050 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0060 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0070 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0080 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0090 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00a0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00b0 0F E3 00 0F E3 03 00 03 04 E3 03 00 00 0F 02 FF .....@.....
00c0 02 33 00 03 13 00 10 03 35 02 A2 D0 63 00 73 .....C..s
00d0 0E 33 00 03 13 00 0E 03 35 00 12 82 35 02 5F .....C..s
00e0 0F 33 00 0F A0 23 FE 5F F0 6F 00 03 05 13 FF FF .....@.._..0.....
00f0 FF FF .....
### Ethernet ###
dst = 00:00:00:00:00:00
src = 00:00:00:00:00:01
type = 0x800
### IP ###
version = 4L
ihl = 5L
tos = 0x0
len = 228
id = 1
flags =
frag = 0L
ttl = 64
proto = 143
chksum = 0x658a
src = 10.0.0.1
dst = 10.255.255.0
Options \
### ProgramMetadata ###
max_steps = 1000L
### Registers ###
r0 = 0
r1 = 0
r2 = 0
r3 = 0
r4 = 0
r5 = 0
r6 = 0
r7 = 0
r8 = 0
r9 = 0
r10 = 0
r11 = 0
r12 = 0
r13 = 0
r14 = 0
r15 = 0
r16 = 0
r17 = 0
r18 = 0
r19 = 0
r20 = 0
r21 = 0
r22 = 0
r23 = 0
r24 = 0
r25 = 0
r26 = 0
r27 = 0
r28 = 0
r29 = 0
r30 = 0
r31 = 0

assembly-switch
[[11:31:14]] root@p4: /media/ef_CS6114_Network_Programming_Languages/assembly-switch (master x) > ./receive.py
WARNING: No route found for IPv6 destination :: (no default route?)
sniffing on hi-eth0
0000 00 00 00 00 00 01 00 00 FF 00 00 00 08 00 45 00 .....E.
0010 00 E4 00 01 00 00 40 BF 65 8A 0A FF FF 00 0A 00 .....@.e.....
0020 00 01 00 00 03 E8 00 00 00 00 00 00 00 00 00 00 .....
0030 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0040 00 15 00 00 00 22 00 00 00 00 00 00 00 00 00 .....
0050 00 15 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0060 00 30 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0070 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0080 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0090 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00a0 00 04 00 00 00 20 00 90 05 13 00 40 0F 13 02 8F .....@.....
00b0 0F E3 00 0F A3 03 00 03 04 E3 03 00 00 0F 00 00 .....C..s
00c0 02 33 00 00 03 13 00 10 03 35 02 A2 D0 63 00 73 .....C..s
00d0 0E 33 00 03 13 00 0E 03 35 00 12 82 35 02 5F .....C..s
00e0 0F 33 00 0F A0 23 FE 5F F0 6F 00 03 05 13 FF FF .....@.._..0.....
00f0 FF FF .....
### Ethernet ###
dst = 00:00:00:00:00:01
src = 00:00:00:00:00:00
type = 0x800
### IP ###
version = 4L
ihl = 5L
tos = 0x0
len = 228
id = 1
flags =
frag = 0L
ttl = 64
proto = 143
chksum = 0x658a
src = 10.255.255.0
dst = 10.0.0.1
Options \
### ProgramMetadata ###
max_steps = 1000L
### Registers ###
r0 = 0
r1 = 0
r2 = 0
r3 = 0
r4 = 0
r5 = 8
r6 = 21
r7 = 34
r8 = 0
r9 = 0
r10 = 21
r11 = 0
r12 = 0
r13 = 0
r14 = 0
r15 = 0
r16 = 0
r17 = 0
r18 = 0
r19 = 0
r20 = 0
r21 = 0
r22 = 0
r23 = 0
r24 = 0
r25 = 0
r26 = 0
r27 = 0
r28 = 34
r29 = 0
r30 = 4
r31 = 32
```



# Future work

## ❖ Limitations

### ➤ Reliability

- The network is unreliable and packet buffers could fill up or packets may be corrupted. Dropping packets will cause the program to fail to completely execute.
- A simple, but inefficient workaround is to re-transmit packets after a timeout when a response has not been received, but partial execution can leave memory in an unpredictable state.

# Future work

## ❖ Limitations

### ➤ Program size limitation

- The entire program must be stored within the headers of the packet. Our system enforces a limit of 300 instructions per program to satisfy the maximum ethernet packet size constraint.
- Several potential solutions: separating large programs into chunks, executing programs as sequences of basic blocks, storing the program assembly in memory and executing them via interpretation.

## ❖ Custom extensions to assembly language

- ### ➤ The assembly language can be extended with custom instructions, such as for retrieving information about the state of the switch or the network.

# Future work

- ❖ Failure reporting
  - Loads/stores with non word-aligned addresses cause the program execution to abort, but no failure indication is sent back to the user.

# Finding bugs in P4c (secretly what this class is about)

p4lang / p4c

Watch

73

Star

152

Fork

95

Code

Issues 71

Pull requests 10

Projects 1

Wiki

Insights

## Header stack last field generates incorrect JSON for BMv2

Edit

New issue

#1607

Open

dannyqiu opened this issue 3 days ago · 3 comments



dannyqiu commented 3 days ago • edited

+ 😊 ...

This issue is similar to #1409

Accessing a `hdr.stack.last` as an rvalue outside of a parser select transition generates:

```
{
  "type" : "field",
  "value" : ["stack", "last"]
}
```

but should actually generate:

```
{
  "type" : "stack_field",
  "value" : ["stack", "last"]
}
```

Assignees

No one assigned

Labels

bug

fixed

Projects

None yet

Milestone

No milestone

Notifications

Unsubscribe

You're receiving notifications

Questions?