P4 IN NIX

Gauvain Roussel-Tarbouriech
WHAT IS P4?
“Programming Protocol-independent Packet Processors (P4) is a domain-specific language for network devices, specifying how data plane devices (switches, NICs, routers, filters, etc.) process packets.”
What is P4?

“Programming Protocol-independent Packet Processors (P4) is a domain-specific language for network devices, specifying how data plane devices (switches, NICs, routers, filters, etc.) process packets. Great, what does that mean?”
WHAT IS P4?

"Programming Protocol-independent Packet Processors (P4) is a domain-specific language for network devices, specifying how data plane devices (switches, NICs, routers, filters, etc.) process packets."

Great, what does that mean?

It's a language for hardware optimized network processing (think SIMD for network)
WHAT IS P4???
WHAT IS P4???

It roughly looks like C:
WHAT IS P4???

It roughly looks like C:

```c
parser MyParser(packet_in pkt, out headers_t hdr,
               inout meta_t meta, inout std_meta_t std_meta) {
  state start {
    pkt.extract(hdr.type);
    transition select(hdr.type.tag) {
      HOPS: parse_hops;
      STANDARD: parse_standard;
      default: accept;
    }
  }
  [...]  
```
WHAT IS P4???

It roughly looks like C:

```plaintext
/parser MyParser(packet_in pkt, out headers_t hdr,
                inout meta_t meta, inout std_meta_t std_meta) {
    state start {
        pkt.extract(hdr.type);
        transition select(hdr.type.tag) {
            HOPS: parse_hops;
            STANDARD: parse_standard;
            default: accept;
        }
    }
    [...]
}
```

...With a few oddities :)
language
Functions are replaced by parser, control, package.
Functions are replaced by *parser*, *control*, *package*.

- **parser**: Parses an incoming packet according to structs, typedefs, etc...
Functions are replaced by parser, control, package.

- **parser**: Parses an incoming packet according to structs, typedefs, etc...
- **control**: Modify a parsed packet in order to resent
Functions are replaced by **parser**, **control**, **package**.

- **parser**: Parses an incoming packet according to structs, typedefs, etc...
- **control**: Modify a parsed packet in order to resent
- **package**: Defines the binding logic between the hardware and P4 (e.g. control and data plane)
Functions are replaced by parser, control, package.

- **parser**: Parses an incoming packet according to structs, typedefs, etc...
- **control**: Modify a parsed packet in order to resent
- **package**: Defines the binding logic between the hardware and P4 (e.g. control and data plane)

Other interesting keywords such as state or tables exist but are out of scope for this talk.
Functions are replaced by parser, control, package.

- **parser**: Parses an incoming packet according to structs, typedefs, etc...
- **control**: Modify a parsed packet in order to resent
- **package**: Defines the binding logic between the hardware and P4 (e.g. control and data plane)

Other interesting keywords such as state or tables exist but are out of scope for this talk.

**LET'S MAKE A TRANSPILER!**
WHAT IS A TRANSPILER?
WHAT IS A TRANSPILER?

• Nix -> P4 translator
WHAT IS A TRANSPILER?

- Nix -> P4 translator
- P4 Compiler
WHAT IS A TRANSPILER?

- Nix -> P4 translator
  - P4 Compiler
  - Target compiler
WHAT IS A TRANSPILER?

- Nix -> P4 translator
  - P4 Compiler
  - Target compiler

What does it look like?
source = {
  include = [ "core.p4" "v1model.p4" ];

define = { "test" = "test2"; }
headers = {
  const = {
    "MAX_HOPS" = { type = "int"; value = "10"; }
   "STANDARD" = { type = "int"; value = "0"; }
   "HOPS" = { type = "int"; value = "1"; }
  }
};
header = { "type_t".content = [ { "tag" = "bit<8>"; } ];
  "hop_t".content = [ { "port" = "bit<8>"; }
     { "bos" = "bit<8>"; }
  ];
  "standard_t".content = [ { "src" = "bit<8>"; }
    { "dst" = "bit<8>"; }
  ];
};
in p4Platform.mkProgram {
  name = "test";
  src = (p4Platform.runTranspiler {
    p4Source = source;
  }));
}
WHICH WE CAN SIMPLIFY!

source = {
    include = [ "core.p4" "v1model.p4" ];

    define = { "test" = "test2"; };
    headers = {
        header = { inherit ethernet_h ipv4_no_options_h; };
        typedef = { inherit macAddr ip4Addr; };
    };

    [...]
};

in

p4Platform.mkProgram {
    name = "test";
    src = (p4Platform.runTranspiler {
        p4Source = source; });
}
THANKS TO HELPERS!

```javascript
1 header = {
2   "ethernet_h".content = [
3     { "dstAddr" = "macAddr"; }
4     { "srcAddr" = "macAddr"; }
5     { "etherType" = "bit<16>"; }
6   ];
7  };
```
What does the end result looks like?
WHICH IS PARSED BY THIS

```
1 # transpiler:
2 mkHeader = header:
3   concatStringsSep "\n\n" (mapAttrsToList (name: value:
4     (if (value.union) then "header_union " else "header ")
5       + name + " {\n " +
6       (concatStringsSep "\n" (flatten (imap1 (_: v:
7         (mapAttrsToList (name: value: "    " + value + " " + name + ";") v)
8       ) value.content)))) + "\n}" ) header);

[...]

10 # module:
11 header = mkOption {
12   description = ''
13   The list of headers of the program.
14   '',
15   default = { },
16   type = types.attrsOf (types.submodule {
17     options = {
18       union = mkOption {
19         type = types.bool;
20         default = false;
21       },
22       content = mkOption {
23         type = types.listOf (types attrsOf types.str);
24         default = [ ];
25       },
26     },
27   });
28 ```
transpiler

WHICH IS PARSED BY THIS

What does the end result looks like?
/* This file has been auto-generated by Nix, do not edit it manually! */
#include <core.p4>
#include <v1model.p4>

#define test test2

const int HOPS = 1;
const int MAX_HOPS = 10;
const int STANDARD = 0;

typedef standard_metadata_t std_meta_t;

header standard_t {
    bit<8> src;
    bit<8> dst;
}

struct headers_t {
    type_t type;
    hop_t[MAX_HOPS] hops;
    standard_t standard;
}

parser MyParser(packet_in pkt, out headers_t hdr, inout meta_t meta, inout state start {
    [...]
The end result looks like this on BMV2:

```c
/* This file has been auto-generated by Nix, do not edit it manually! */
#include <core.p4>
#include <v1model.p4>

#define test test2

const int HOPS = 1;
const int MAX_HOPS = 10;
const int STANDARD = 0;

typedef standard_metadata_t std_meta_t;

header standard_t {
    bit<8> src;
    bit<8> dst;
}
```
The end result looks like this on BMV2:

```c
/* This file has been auto-generated by Nix, do not edit it manually! */
#include <core.p4>
#include <v1model.p4>

#define test test2

const int HOPS = 1;
const int MAX_HOPS = 10;
const int STANDARD = 0;

typedef standard_metadata_t std_meta_t;

header standard_t {
    bit<8> src;
    bit<8> dst;
}
```

But what is BMV2?
GLAD YOU ASKED!
GLAD YOU ASKED!

“The simple_switch architecture is the de-facto architecture for most users, as it is roughly equivalent to the "abstract switch model" described in the P4_14 spec."
GLAD YOU ASKED!

“The simple_switch architecture is the de-facto architecture for most users, as it is roughly equivalent to the "abstract switch model" described in the P4_14 spec.

Basically an interface for hardware targeting the switch.
To use P4 you need a target, the three most common targets are:
To use P4 you need a target, the three most common targets are:

- userland (eBPF, DPDK, etc)
To use P4 you need a target, the three most common targets are:

- userland (eBPF, DPDK, etc)
- hardware (FPGAs, custom ASIC)
To use P4 you need a target, the three most common targets are:

- userland (eBPF, DPDK, etc)
- hardware (FPGAs, custom ASIC)
- emulated (BMV2)
To use P4 you need a target, the three most common targets are:

- userland (eBPF, DPDK, etc)
- hardware (FPGAs, custom ASIC)
- emulated (BMV2)

Obviously, those need some kind of interface!
To use P4 you need a target, the three most common targets are:

- userland (eBPF, DPDK, etc)
- hardware (FPGAs, custom ASIC)
- emulated (BMV2)

Obviously, those need some kind of interface!

The Abstract Switch Interface is usually used, with a few per-device changes
To use P4 you need a target, the three most common targets are:

- userland (eBPF, DPDK, etc)
- hardware (FPGAs, custom ASIC)
- emulated (BMV2)

Obviously, those need some kind of interface!

The Abstract Switch Interface is usually used, with a few per-device changes

This also needs changes to the transpiler!
setup

Introducing : FPGAs on Nix
Introducing: FPGAs on Nix
(Yes, really)
Introducing: FPGAs on Nix
(Yes, really)

(I forgot to take the picture before going to FOSDEM so imagine an FPGA sitting on a computer, with USB and ethernet plugged in)
The core idea is:
The core idea is:

1. Add hardware definitions to Nix for FPGAs
The core idea is:

1. Add hardware definitions to Nix for FPGAs
2. Add an auto-reload/deploy mechanism through USB or custom interfaces (e.g. JTAG)
The core idea is:

1. Add hardware definitions to Nix for FPGAs
2. Add an auto-reload/deploy mechanism through USB or custom interfaces (e.g. JTAG)
3. Add a data plane mechanism for feeding data from the host (further off).
The core idea is:

1. Add hardware definitions to Nix for FPGAs
2. Add an auto-reload/deploy mechanism through USB or custom interfaces (e.g. JTAG)
3. Add a data plane mechanism for feeding data from the host (further off).

All of this is a work-in-progress for now.
The core idea is:

1. Add hardware definitions to Nix for FPGAs
2. Add an auto-reload/deploy mechanism through USB or custom interfaces (e.g. JTAG)
3. Add a data plane mechanism for feeding data from the host (further off).

All of this is a work-in-progress for now.

But software P4 works!
QUESTIONS?
THANK YOU!

@GovanifY

govanify.com(/resume.pdf)
gauvain@govanify.com
secure-boot

ONE LAST THING...