P4 IN NIX

Gauvain Roussel-Tarbouriech

language

WHAT IS P4?

language

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.

language

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?

language

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)

language

WHAT IS P4???

language

WHAT IS P4???

It roughly looks like C:

language

WHAT IS P4???

It roughly looks like 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;
        }
    }
[...]
```

language

WHAT IS P4???

It roughly looks like C:

```
1  parser MyParser(packet_in pkt, out headers_t hdr,
2                  inout meta_t meta, inout std_meta_t std_meta) {
3      state start {
4          pkt.extract(hdr.type);
5          transition select(hdr.type.tag) {
6              HOPS: parse_hops;
7              STANDARD: parse_standard;
8              default: accept;
9          }
10     }
11 [...]
```

...With a few oddities :)

language

language

Functions are replaced by **parser, control, package.**

language

Functions are replaced by **parser, control, package.**

- **parser:** Parses an incoming packet according to structs, typedefs, etc...

language

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

language

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)

language

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.

language

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!

transpiler

WHAT IS A TRANSPILER?

transpiler

WHAT IS A TRANSPILER?

- Nix -> P4 translator

transpiler

WHAT IS A TRANSPILER?

- Nix -> P4 translator
- P4 Compiler

transpiler

WHAT IS A TRANSPILER?

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

transpiler

WHAT IS A TRANSPILER?

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

What does it look like?

transpiler

```
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; });
  }
```

transpiler

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; });
  }
```

transpiler

THANKS TO HELPERS!

```
header = {
  "ethernet_h".content = [
    { "dstAddr" = "macAddr"; }
    { "srcAddr" = "macAddr"; }
    { "etherType" = "bit<16>"; }
  ];
};
```

transpiler

THANKS TO HELPERS!

```
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?

transpiler

WHICH IS PARSED BY THIS

```
# transpiler:
mkHeader = header:
  concatStringsSep "\n\n" (mapAttrsToList (name: value:
    (if (value.union) then "header_union " else "header ")
      + name + " {\n " +
      (concatStringsSep "\n" (flatten (imap1 (_: v:
        (mapAttrsToList (name: value: "    " + value + " " + name + ";") v)
      ) value.content))) + "\n}" ) header);

[...]
# module:
header = mkOption {
  description = ''
    The list of headers of the program.
  '';
  default = { };
  type = types.attrsOf (types.submodule {
    options = {
      union = mkOption {
        type = types.bool;
        default = false;
      };
      content = mkOption {
        type = types.listOf (types.attrsOf types.str);
        default = [ ];
      };
    };
  });
```

transpiler

WHICH IS PARSED BY THIS

```
# transpiler:
mkHeader = header:
  concatStringsSep "\n\n" (mapAttrsToList (name: value:
    (if (value.union) then "header_union " else "header ")
      + name + " {\n " +
      (concatStringsSep "\n" (flatten (imap1 (_: v:
        (mapAttrsToList (name: value: "    " + value + " " + name + ";") v)
      ) value.content))) + "\n}" ) header);

[...]
# module:
header = mkOption {
  description = ''
    The list of headers of the program.
  '';
  default = { };
  type = types.attrsOf (types.submodule {
    options = {
      union = mkOption {
        type = types.bool;
        default = false;
      };
      content = mkOption {
        type = types.listOf (types.attrsOf types.str);
        default = [ ];
      };
    };
  });
```

What does the end result looks like?

transpiler

P4 CODE:

```
1  /* This file has been auto-generated by Nix, do not edit it manually! */
2  #include <core.p4>
3  #include <v1model.p4>
4
5  #define test test2
6
7  const int HOPS = 1;
8  const int MAX_HOPS = 10;
9  const int STANDARD = 0;
10
11 typedef standard_metadata_t std_meta_t;
12
13 header standard_t {
14      bit<8> src;
15     bit<8> dst;
16 }
17
18
19 struct headers_t {
20      type_t type;
21     hop_t[MAX_HOPS] hops;
22     standard_t standard;
23 }
24
25 parser MyParser(packet_in pkt, out headers_t hdr, inout meta_t meta, inout
26     state start {
27     [...]
```

transpiler

The end result looks like this on BMV2:

```
/* 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;
}


```

transpiler

The end result looks like this on BMV2:

```
/* 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?

setup

GLAD YOU ASKED!

setup

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.*

setup

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.

setup

To use P4 you need a target, the three most common targets are:

setup

To use P4 you need a target, the three most common targets are:

- userland (eBPF, DPDK, etc)

setup

To use P4 you need a target, the three most common targets are:

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

setup

To use P4 you need a target, the three most common targets are:

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

setup

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!

setup

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

setup

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

setup

Introducing : FPGAs on Nix

(Yes, really)

setup

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)

setup

The core idea is:

setup

The core idea is:

1. Add hardware definitions to Nix for FPGAs

setup

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)

setup

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).

setup

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.

setup

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!

konami-code

QUESTIONS?

konami-code

THANK YOU!

@GovanifY

govanify.com(/resume.pdf)

gauvain@govanify.com

secure-boot

ONE LAST THING...