СЕТИ ПЕРЕДАЧИ И ОБРАБОТКИ ДАННЫХ

Лекция 10:
Языки программирования коммутаторов

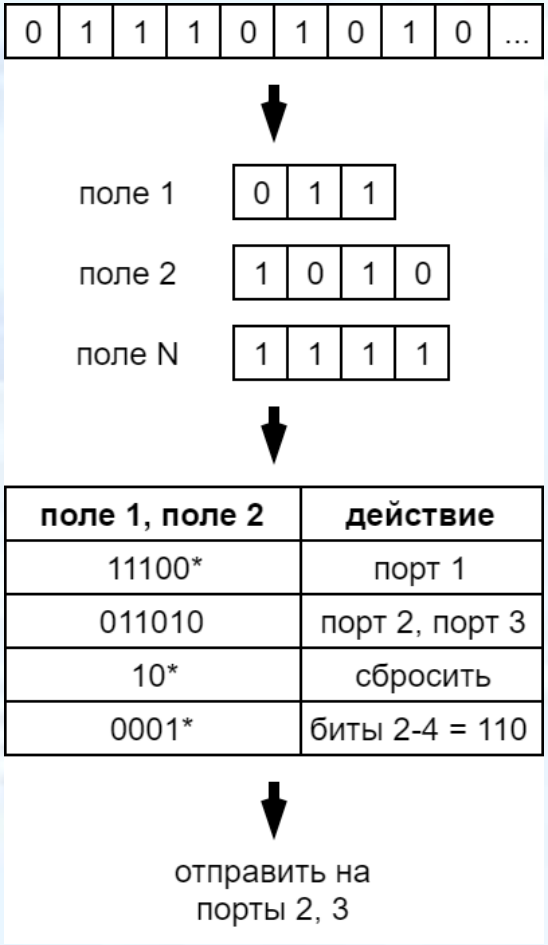
ВМК МГУ им. М.В. Ломоносова, Кафедра АСВК
Доцент, к.ф.-м.н. Волканов Д.Ю.

План лекции

- Языки программирования коммутаторов
- Язык P4
- Язык NPL

Функции СПУ

1. Получение пакета с физического уровня
2. Выделение заголовка из пакета
3. Классификация пакета – идентификация пакета по заголовку
4. Модификация заголовка пакета и принятие решения о пути следования пакета
5. Управление очередями
6. Передача пакета на физический уровень



Программируемость СПУ

- **Устройства с фиксированной функциональностью**

  Для данного класса характерна работа с фиксированным стеком протоколов и фиксированная программа обработки пакетов

- **Конфигурируемые устройства**

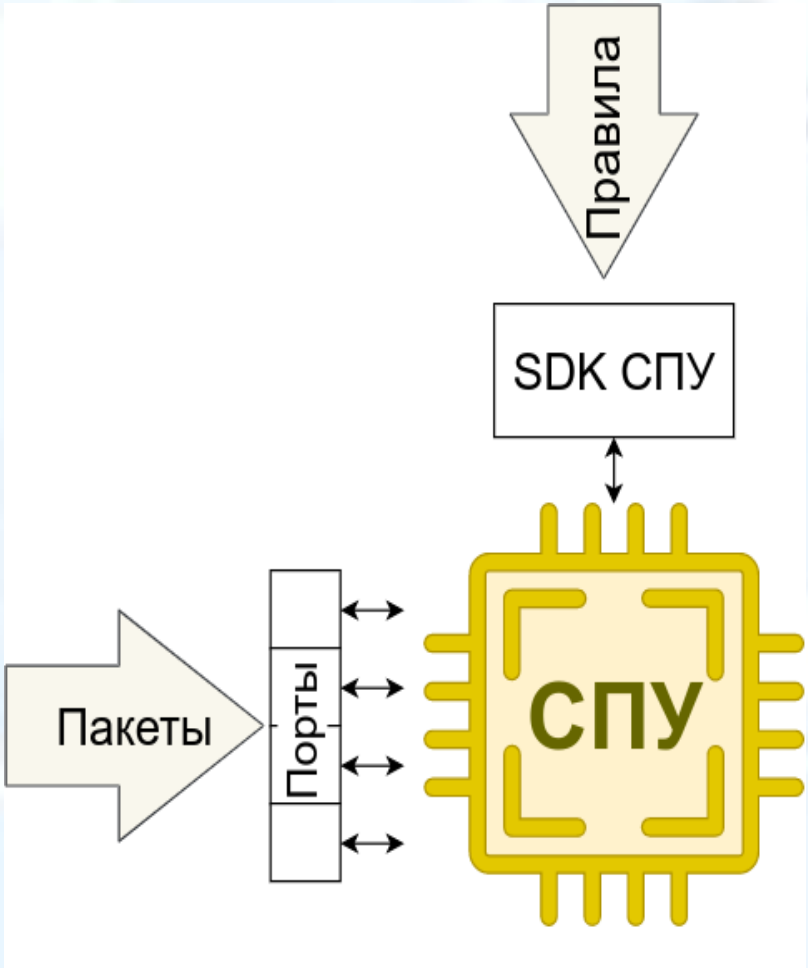
  СПУ данного класса позволяют осуществлять загрузку в устройство новой программы обработки пакетов в рамках фиксированного стека протоколов

- **Программируемые устройства**

  Позволяют не только загружать новую программу обработки пакетов в устройство, но и определять новые протоколы передачи данных в этой программе

Что нужно для поддержки нового протокола?

- Конфигурация алгоритма обработки пакетов (разбор заголовка, структура таблиц классификации, допустимые действия над пакетом) – с помощью **языка программирования СПУ**
- Интерфейс взаимодействия с СПУ во время выполнения (изменение правил таблиц классификации)



Язык программирования СПУ

Предназначен для задания следующих параметров СПУ:

- Поддерживаемые поля заголовков пакетов
- Автомат разбора заголовка пакета
- Структура таблиц классификации:
  - Структура ключа поиска
  - Тип поиска (точное совпадение, тернарный, …)
  - Допустимые действия над пакетом
- Порядок выполнения программы

План рассмотрения языка

- Модель СПУ
- Структура программы
- Типы полей и определение заголовка пакета
- Определение автомата разбора заголовка (парсер)
- Определение действий над пакетом
- Определение структуры таблиц классификации

Будут рассматриваться предметно-ориентированные языки P4 и NPL

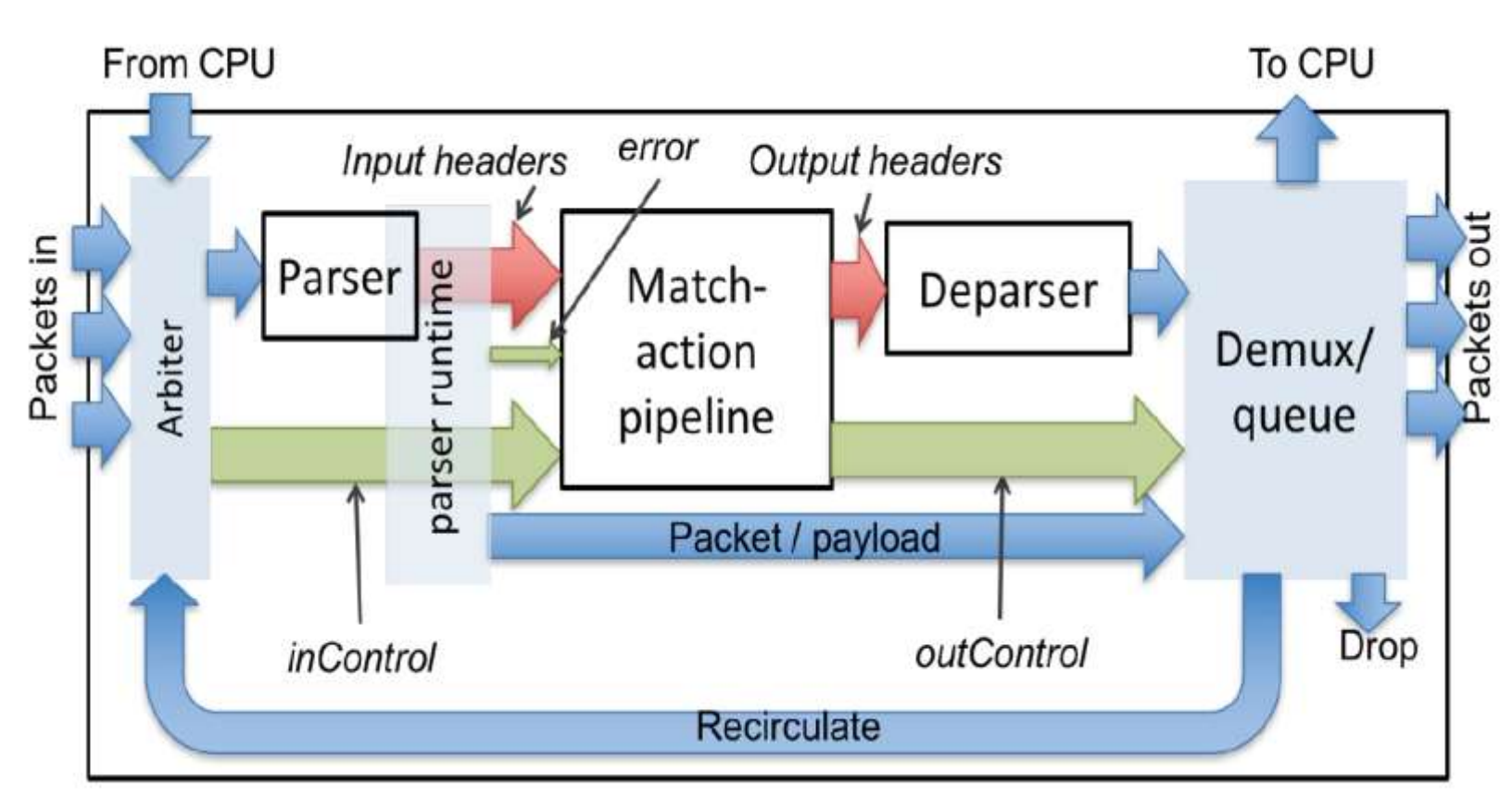
Язык P4

- 2014, Barefoot Networks, Stanford University

- «Programming Protocol-Independent Packet Processors»
- «Reconfigurability. Protocol independence. Target independence»
- Язык программирования СПУ Barefoot Tofino (13 Тбит/с)

P4 Language Consortium: P4.org
P4language on Github: https://github.com/p4lang

P4: модель СПУ



P4: программа

- Стандартная библиотека core.p4
  - Операции над входным и выходным пакетами
  - Стандартные типы ошибок, вспомогательные функции…
- Описание модели СПУ
  - Интерфейсы блоков (`parser`, `control`)
  - Структура модели (`package`)
  - Описание контекстов (входных и выходных переменных) блоков
  - Интерфейсы специальных функций СПУ
- Файл программы
  - Директивы включения вспомогательных файлов
  - Определение пакета и пользовательских контекстов
  - Определение функциональности блоков модели (`parser`, `control`)
  - Структура программы (`package`)

P4: описание модели СПУ

```
extern packet_in {
    void extract<T>(out T hdr);
    bit<32> length();
    ...
}
extern packet_out {
    void emit<T>(in T hdr);
}
```

core.p4

```
struct std_metadata_t {
    bit<9>  ingress_port;
    bit<9>  egress_port;
}
```

```
parser Parser<H, M>(packet_in b, out H parsedHeaders, out M userMetadata);
control Pipe<H, M>(inout H headers, in error parseError,
                   inout std_metadata_t stdMetadata, inout M userMetadata);
control Deparser<H, M>(in H outputHeaders, packet_out b, in M userMeta);

package Switch<H, M>(Parser<H, M> p, Pipe<H, M> ma, Deparser<H, M> d);
```

P4: структура программы

```
•# include <core.p4>
•# include "switch_model.p4"

•struct parsed_packet { ... }
•struct user_metadata { ... }

•parser ParserImpl(packet_in b, ..., out
 user_metadata_t um) {...}
•control PipeImpl(inout parsed_packet p, ...,
•                 inout user_metadata_t um) {...}
•control DeparserImpl(in parsed_packet p, ...,
•                     in user_metadata_t um) {...}

•SwitchImpl(ParserImpl(), PipeImpl(),
 DeparserImpl()) main;
```

P4: операторы

- Присваивание
- Вызов функции
- Условный оператор, оператор switch
- Пустой оператор
- Область видимости {}
- Операторы, специфичные для блока `parser`
- Операторы, специфичные для блока `control`
- Оператор возврата из функции

P4: переменные

- Разрешается создавать переменные и константы типов: **`bool`**, **`error`**, **`bit`**, **`int`**, **`varbit`**, **`struct`**, **`tuple`**, типов заголовков, определенных пользователем типов в областях:
  - В области видимости {}
  - В состоянии парсера (область **`state`**)
  - В теле действия **`action`**
  - В областях **`parser`** и **`control`**
- Значения переменной не сохраняются между вызовами блока, где она определена

P4: функции

- Помимо специальных функций СПУ **extern**, объявленных в описании модели СПУ, могут быть определены функции:
  - только в глобальной области видимости
  - все параметры должны быть объявлены **in/out/inout**

```
bit<32> add(in bit<32> left, in bit<32> right) {
    return left + right;
}
```

P4: типы полей заголовка

Для определения заголовка используются ключевые слова:

- **header** – заголовок, допустимые поля:
  - **bit<N>, int<N>, varbit<N>**
  - **enum**
  - **bool**
  - **struct** (с ограничениями)
- **header_union** – хранит заголовок одного из перечисленных типов
- **struct** – для группировки заголовков

```
header Mpls_h {
    bit<20> label;
    bit<3> tc;
    bit bos;
    bit<8> ttl;
}
enum bit<16> EtherType {
    VLAN = 0x8100,
    QINQ = 0x9100,
    MPLS = 0x8847,
    ...
}
header_union IP_h {
    IPv4_h v4;
    IPv6_h v6;
}
struct parsed_headers {
    ...
    Mpls_h[3] mpls;
}
```

P4: пример описания заголовка

```
header ethernet_t {
    bit<48> dstAddr;
    bit<48> srcAddr;
    bit<16> etherType;
}

struct headers {
  ethernet_t ethernet;
  ipv4_t     ipv4;
}
```

```
header ipv4_t {
    bit<4>  version;
    bit<4>  ihl;
    bit<8>  diffserv;
    bit<16> totalLen;
    bit<16> identification;
    bit<3>  flags;
    bit<13> fragOffset;
    bit<8>  ttl;
    bit<8>  protocol;
    bit<16> hdrChecksum;
    bit<32> srcAddr;
    bit<32> dstAddr;
}
```

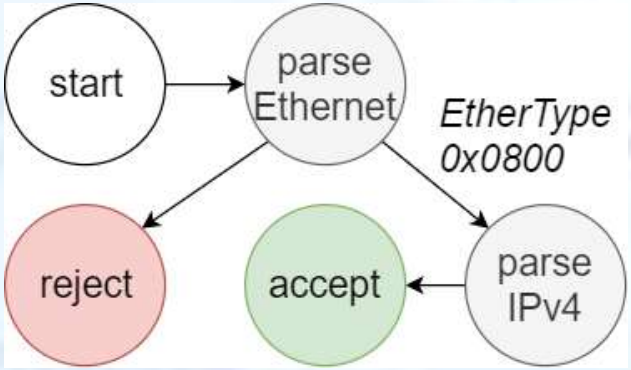
P4: парсер заголовка пакета

- Блок типа **`parser`** определяет конечный автомат разбора заголовка пакета. Возможно определить несколько парсеров, которые могут быть вызваны из основного.
  - **`start`** – начальное состояние, определяется программистом
  - **`accept`** – конечное состояние: разбор завершен успешно
  - **`reject`** – конечное состояние: разобрать заголовок не удалось

  - **`transition`** – оператор перехода к следующему состоянию
  - **`select`** – оператор выбора следующего состояния по значению переменной
  - **`verify`** – вызов функции проверки условия; если условие не выполнено, произойдет переход в состояние `reject`

P4: пример парсера

```
parser ParserImpl(packet_in packet,
                  out headers hdr){
  state start {
    transition parse_ethernet;
  }
  state parse_ethernet {
    packet.extract(hdr.ethernet);
    transition select(hdr.ethernet.etherType){
      0x0800 :  parse_ipv4;
      default: reject;
    }
  }
  state parse_ipv4 {
    packet.extract(hdr.ipv4);
    transition accept;
  }
}
```




```
extern packet_in {
  void extract<T>(out T hdr);
  ...
}
```

```
struct headers {
  ethernet_t ethernet;
  ipv4_t     ipv4;
}
```

P4: блок control

- Предназначен для классификации и модификации разобранного заголовка пакета
- Включает определение:
  - действий над пакетом (**action**), структуры таблиц классификации, локальные переменные – в области блока **control**
  - логики обработки пакетов, в т.ч. обращений к таблицам классификации – в области **apply**

```
control Pipeline(...) {
  // Определение типов:
  // таблицы, действия;
  // объявление переменных
  action change_vlan(vid_t
 vid) {...}

  table t {...}

  apply {
    // Объявление переменных,
    // обращение к таблицам,
    // другие операторы...
    t.apply();
  }
```

P4: определение таблиц классификации

Для определения таблицы используется ключевое слово **table**

Структура таблицы:

- **key** – ключ поиска и тип поиска (`exact`, `ternary`, `lpm`)
- **actions** – типы действий в таблице
- **default_action** – действие по умолчанию (опционально)
- **size** – ограничение размера (опционально)
- **const entries** – предопределенные правила в таблице (опционально)

```
table vlan {
  key = {
    hdr.dot1q.vid       : exact;
    stdMeta.egress_port: exact;
  }
  actions = { change_vlan;
              remove_vlan;
              NoAction; }

  default_action = NoAction;

  size = 1024;

  const entries = {
      (100, 24) : remove_vlan();
      (101, 24) :
 change_vlan(102);
  }
}
```

P4: пример определения таблиц классификации

```
control PipeImpl(inout header hdr, inout std_metadata_t
stdMetadata, ...) {
  action set_vlan(vid_t vlan_id) { hdr.dot1q.vid = vlan_id; }
  action broadcast() { stdMeta.mcast_grp = BROADCAST_GRP; }
  action forward(port_t port) { stdMeta.egress_port = port; }

  table access {
    key = { stdMetadata.ingress_port: exact; }
    actions = { set_vlan; }
  }
  table dst_mac {
    key = { hdr.ethernet.dstAddr: exact; }
    actions = { broadcast; forward; }
    default_action = broadcast;
  }
  apply {
    if (hdr.dot1q.is_tagged) { access.apply(); }
    dst_mac.apply();
  }
}
```

P4: депарсер

- P4 не определяет специальных языковых конструкций для депарсера; депарсер определяется как блок типа **`control`** с как минимум одним параметром типа **`packet_out`**.

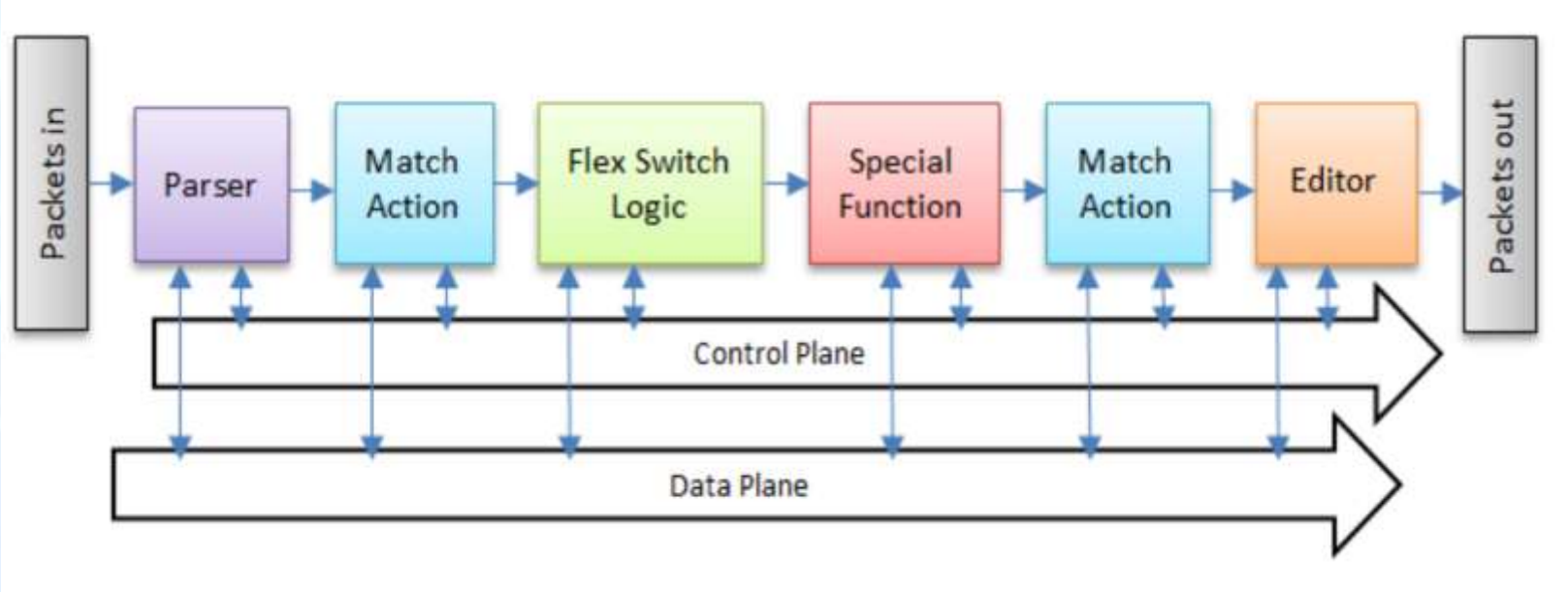
```
control DeparserImpl (packet_out packet, in headers_t hdr) {
  apply {
    packet.emit(hdr.ethernet);
    if (hdr.dot1q.is_tagged)  { packet.emit(hdr.dot1q); }
  }
}
```

Язык NPL

- 2019, Broadcom
- «Network Programming Language»
- «Open, High-Level language for developing feature-rich solutions for programmable networking platforms»
- Язык программирования СПУ Broadcom Trident 4 (13 Тбит/с)

NPL: https://nplang.org/
Nplang on Github: https://github.com/nplang

NPL: модель СПУ



NPL: структура программы

- Директивы включения файлов стандартной библиотеки, содержащих объявления специальных функций СПУ
- Описание пользовательских типов `enum`
- Описание структур данных, проходящих по логической шине (контекстов)
- Описание структуры заголовка пакета
- Описание парсера заголовка пакета
- Описание структуры таблиц классификации
- Пользовательские процедуры
- Основная программа (main)

NPL: операторы

- Условный оператор, оператор switch
- Оператор присваивания
- Вызов встроенной процедуры
- Вызов специальной функции СПУ
- Вызов пользовательской процедуры
- Область видимости {}

NPL: переменные

- В языке NPL в качестве переменных в блоках рассматриваются только:
  - заголовки пакетов
  - контексты, передаваемые по логической шине между блоками

NPL: контексты логической шины

- Предназначены для передачи данных между блоками
- Могут быть определены оверлеи (в блоке **overlays**), позволяющие обращаться к битовым подстрокам определенных полей
- Допустимые типы полей: **bit**, **bit**[], **varbit**, **struct**

```
struct control_bus_t
{
  fields {
    bit timestamp_enable;
    bit[10] ingress_port_num;
  }
}
bus control_bus_t control_id;
```

NPL: программа

- В программе разрешены вызовы следующих встроенных процедур:
  - Парсер
    `parse_begin(<node name>), parse_continue(<node name>)`
  - Поиск в таблице
    `<table>.lookup(<lookup_num>)`
  - Модификация пакета
    `add_header, delete_header, replace_header_field, create_checksum…`

```
program mim_main () {
    parse_begin (ethernet);
    port.lookup (0);
    iif.lookup (0);
    my_station.lookup (0);
    isid.lookup (0);

 mim_isid_switch_logic1();
    ...
    if (cmd_bus.do_l3) {
        l3_host.lookup (0);
    }
    ...
    l3_switch_logic1();
    ...
    next_hop.lookup (0);
    do_packet_edits();
}
```

NPL: процедуры и функции

В NPL разрешены подпрограммы, не имеющие параметров и не возвращающие значений

- `function l3_switch_logic() { … }`

Специальные функции СПУ имеют параметры (обязательно **in**/**out**)

В качестве типа параметра может использоваться тип **const** (целые числа и значения **enum**)

```
extern packet_drop(in bit[1] trigger, in const val, in const code);

// ...

enum drop_reason {
  NO_DROP = 0,
  MEMBERSHIP_DROP = 1,
  TTL_DROP = 2
}

// ...

packet_drop(drop_bus.disable_drop, drop_reason.TTL_DROP, 5);
```

NPL: типы полей заголовка

Заголовок – это структура, содержащая:

- поля (**fields**)
- опционально – оверлеи (**overlays**), позволяющие обращаться к битовым подстрокам полей

Допустимые типы полей заголовков:

bit, **bit**[], **varbit**[] (требует указания метода подсчета длины заголовка)

```
struct ipv4_t {
  fields {
    bit[4] version;
    bit[4] hdr_len;
    bit[8] tos;
    bit[32] sa;
    bit[32] da;
    varbit[320] option;
  }
  header_length_exp:
 hdr_len*4;
  overlays {
    dscp: tos[5:0];
    ecn : tos[7:6];
  }
}
```

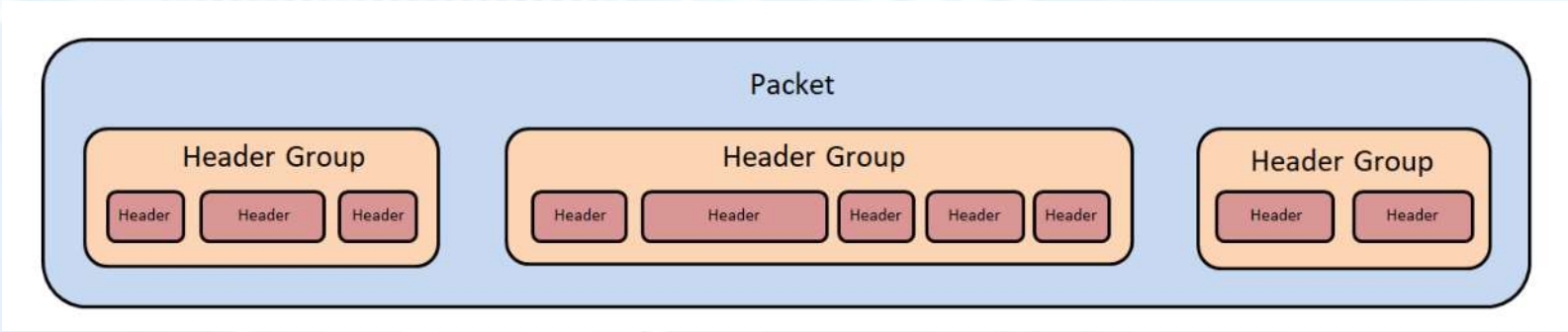
NPL: определение заголовка

Для определения заголовка необходимо:

- Описать структуру заголовков
- Сгруппировать заголовки (описать группы заголовков)
- Сгруппировать группы заголовков (описать структуру «пакета»)

С каждым заголовком будет ассоциировано однобитное поле `_PRESENT`

Разрешено определять массив заголовков **`struct`**`[]`



NPL: пример описания заголовка

```
struct ethernet_t {
  fields {
    bit[48] dstAddr;
    bit[48] srcAddr;
    bit[16] etherType;
  }
}

struct l2_group_t {
  fields {
    ethernet_t ethernet;
  }
}

struct pkt_t {
  fields {
    l2_group_t l2;
    l3_group_t l3;
  }
}

packet pkt_t pkt;
```

```
struct ipv4_t {
  fields {
    bit[4]  version;
    bit[4]  ihl;
    bit[8]  diffserv;
    bit[16] totalLen;
    bit[16] identification;
    bit[3]  flags;
    bit[13] fragOffset;
    bit[8]  ttl;
    bit[8]  protocol;
    bit[16] hdrChecksum;
    bit[32] srcAddr;
    bit[32] dstAddr;
  }
}

struct l3_group_t {
  fields {
    ipv4_t ipv4;
  }
}
```

NPL: разбор заголовка пакета

Язык NPL позволяет описать конечный автомат разбора заголовка пакета. Для разбора используются следующие конструкции:

`parser_node` – состояние автомата разбора заголовка

- **`root_node`** – бит начального состояния
- **`next_node`** – переход к следующему состоянию
- **`switch`** – выбор следующего состояния по значению переменной
- **`extract_fields`**`(packet.header)` – записать значения полей в разобранный пакет
- **`end_node`** – бит конечного состояния

`parse_break`/**`parse_continue`** – операторы, позволяющие прервать и продолжить разбор заголовка

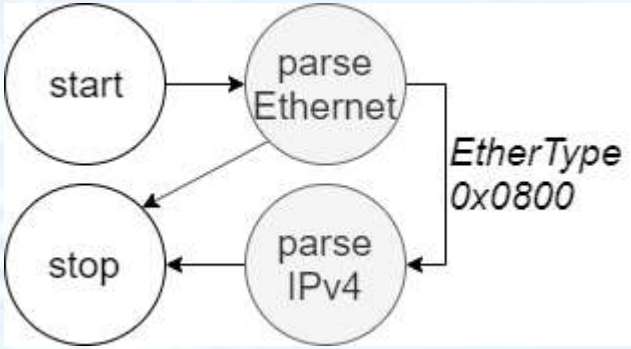
NPL: пример парсера

```
parser_node start {
  root_node: 1;
  next_node parse_ethernet;
}

parser_node parse_ethernet {
  extract_fields(pkt.l2.ethernet);
  switch(pkt.l2.ethernet.etherType){
    0x0800 : { next_node parse_ipv4 }
    default: { next_node stop };
  }
}

parser_node parse_ipv4 {
  extract_fields(pkt.l3.ipv4);
  next_node stop;
}
```

```
parser_node stop {
  end_node: 1;
}
```



NPL: определение таблиц классификации

Язык NPL предлагает для определения таблиц примитив logical_table

- **table_type** – тип таблицы (index, tcam, hash, alpm)
- **minsize, maxsize** – ограничения числа правил
- **keys** – структура ключа поиска
- **fields** – результат поиска в данной таблице
- **key_construct** – функция, определяет порядок «сборки» ключа поиска
- **fields_assign** – функция, определяет порядок действий по результатам поиска

```
logical_table mac_table {
  table_type : hash;
  minsize    : 64;
  maxsize    : 64;
  keys { bit[48] macda; }
  fields { bit[16] port; }
  key_construct() {
    if (_LOOKUP0==1) {
      macda =
ing_pkt.l2_grp.ethernet.da;
    }
    if (_LOOKUP1==1) {
      macda =
ing_pkt.l2_grp.ethernet.sa;
    }
  }
  fields_assign() {
    if (_LOOKUP0==1) {
      obj_bus.dst = port; }
    if (_LOOKUP1==1) {
      temp_bus.src_port = port; }
  }
}
```

P4 versus NPL

	P4	NPL
Модель СПУ	Позволяет определить свою модель СПУ	Модель фиксирована
Структура программы	Частично фиксирована	Не фиксирована
Поддерживаемые поля заголовков	bit, int, varbit, enum, bool, struct	bit, varbit
Заголовок пакета	Поля заголовков, заголовки можно группировать	Иерархия: заголовок, группа заголовков, пакет
Парсер	Конечный автомат. Есть результат: успех или неудача	Конечный автомат. Может быть прерван и возобновлен
Действия над пакетом	Отдельные подпрограммы	Единая программа обработки результата поиска
Ключ поиска в таблице	Набор полей, для каждого указывается тип поиска	Функция сборки ключа и тип таблицы

Спасибо за внимание!