

Памятка по языку P4

Базовые типы данных

typedef

Служит для определения дополнительного имени типа.

```
typedef bit<48> macAddr_t; // Объявление нового типа для 48-битового MAC-адреса Ethernet
typedef bit<32> ip4Addr_t; // Объявление нового типа для 32-битового адреса IPv4
```

header

Упорядоченный набор элементов (полей заголовка). Заголовок обязательно включает скрытый элемент (бит) validity. Для проверки и изменения этого бита служат функции isValid(), setValid(), setInvalid().

```
header ethernet_t {
    macAddr_t dstAddr; // Поле заголовка с MAC-адресом получателя
    macAddr_t srcAddr; // Поле заголовка с MAC-адресом отправителя
    bit<16> type; // Поле заголовка, указывающее тип кадра (EtherType)
}
```

Объявление переменных для заголовков имеет вид

```
ethernet_t ethernet; // Переменная типа ethernet_t
```

Для доступа к полям заголовков используется нотация с точкой, как показано ниже.

```
macAddr_t src = ethernet.srcAddr; // Переменная src получает MAC-адрес отправителя
```

struct

Структуры в P4 представляют собой неупорядоченные наборы элементов.

```
struct headers_t {
    ethernet_t ethernet; // Структура, содержащая 1 заголовок типа ethernet_t
}
```

Стеки заголовков

Стек заголовков представляет собой индексированный массив элементов типа header.

```
header label_t { // Объявление заголовка label_t (метка).
    bit<20> label; // Собственно метка.
    bit bos; // Флаг последней метки в стеке.
}
```

```
struct header_t { // Объявление стека меток.
    label_t[10] labels; // Стек меток.
}
```

```
header_t hdr; // Переменная стека меток.
```

```
action pop_label() { // Удаление метки из стека.
    hdr.labels.pop_front(1); // Выталкивание метки.
}
```

```
action push_label(in bit<20> label) { // Добавление метки в стек.
    hdr.labels.push_front(1); // Вталкивание метки.
    hdr.labels[0].setValid(); // Объявление метки 0 действительной.
    hdr.labels[0] = {label, 0};
}
```

Операторы и выражения

Объявление и назначение локальных метаданных

```
bit<16> tmp1; bit<16> tmp2; // Переменные tmp1 и tmp2 типа bit<16>
tmp1 = hdr.ethernet.type; // Переменная tmp1 получает значение EtherType из заголовка пакета
«Нарезка» и объединение (конкатенация) битов
```

```
tmp2 = tmp1[7:0] ++ tmp1[15:8]; // Переменная tmp2 получает значение объединения битов 0-7
// из переменной tmp1 с битами 8-15 из той же переменной.
// В данном случае это перестановка старшего и младшего байтов
```

Сложение, вычитание и приведение типов

```
tmp2 = tmp1 + tmp1 - (bit<16>)tmp1[7:0]; // Переменная tmp2 получает значение суммы tmp1 и
// старшего байта tmp1
```

Побитовые операции

```
tmp2 = (~tmp1 & tmp1) | (tmp1 ^ tmp1); // Переменная tmp2 получает значение 0, поскольку
// операции И для числа и его «отрицания», а также
// Исключительное ИЛИ с самим собой дают 0, а
// ИЛИ для двух нулей также дает 0.
tmp2 = tmp1 << 3; // Переменная tmp2 получает значение смещенных на
// три позиции влево битов переменной tmp1. Три
// младших бита будут иметь значение 0.
```

Действия

Операции на основании входных данных от плоскости управления.

```
action set_next_hop(bit<32> next_hop) {           // Установка следующего интервала пересылки
    if (next_hop == 0) {                         // Если параметр next_hop еще не задан,
        metadata.next_hop = hdr.ipv4.dst;        // в поле метаданных помещается адрес получателя
                                                // из заголовка IP в пакете.
    } else {                                     // В противном случае поле метаданных
        metadata.next_hop = next_hop;           // получает значение параметра next_hop.
    }
}
```

Операции на основании входных параметров от плоскости данных.

```
action swap_mac(inout bit<48> x,                // Перестановка MAC-адресов отправителя и получателя,
                inout bit<48> y) {             // переданных параметрами действия.
    bit<48> tmp = x;                            // Сохранение адреса x во временной переменной.
    x = y;                                     // Назначение переменной x значения переменной y.
    y = tmp;                                   // Назначение переменной y исходного значения переменной x.
}
```

Операции на основании входных параметров от плоскостей данных и управления.

```
action forward(in bit<9> p, bit<48> d) {        // Задание выходного порта и адреса получателя.
    standard_metadata.egress_spec = p;         // Указание выходного порта в метаданных по параметру.
    headers.ethernet.dstAddr = d;            // Указание адреса получателя по параметру.
}
```

Удаление заголовка из пакета (декапсуляция).

```
action decap_ip_ip() {                        // Декапсуляция внешнего заголовка IP.
    hdr.ipv4 = hdr.inner_ipv4;               // Перенос внутреннего заголовка во внешний.
    hdr.inner_ipv4.setInvalid();             // Объявление внутреннего заголовка недействительным.
}
```

Таблицы

Ниже приведен пример определения таблицы для поиска по максимальному совпадению префиксов.

```
table ipv4_lpm {                               // Таблица сопоставления по префиксам
    key = {
        hdr.ipv4.dstAddr : lpm;               // Выполняется сопоставление lpm для поиска самого длинного
                                                // префикса. Стандартные типы сопоставления - exact, ternary,
                                                // lpm.
    }
    // Задание возможных действий.
    actions = {
        ipv4_forward;                         // Пересылка IPv4.
        Drop;                                 // Отбрасывание.
        NoAction;                             // Нет действий.
    }
    // Свойства таблицы
    size = 1024;                              // Число записей
    default_action = NoAction();              // Используемое при отсутствии совпадений действие.
}
```

Поток управления

Метод apply служит для вызова блоков «сопоставление-действие».

```
apply {
    // Ветвление по признаку действительности заголовка.
    if (hdr.ipv4.isValid()) {                 // Если заголовок пакета IPv4 действителен,
        ipv4_lpm.apply();                    // выполняется сопоставление по самому длинному префиксу.
    }
    // Ветвление по результатам поиска в таблице
    if (local_ip_table.apply().hit) {
        send_to_cpu();                       // Передача пакета в порт CPU
    }
    // Ветвление для выбора действия в таблице.
    switch (table1.apply().action_run) {
        action1: { table2.apply(); }
        action2: { table3.apply(); }
    }
}
```

Синтаксический анализ заголовков

Использование внешнего элемента packet_in для входного пакета.

```
extern packet_in {
    void extract<T>(out T hdr);               // Извлечение заголовка в выходную переменную T.
    void extract<T>(out T hdr, in bit<32> n); // Назначение значения выходной переменной T
                                                // (заголовок) по входной переменной n.
    T lookahead<T>();                         // Извлечение заголовка в T.
    void advance(in bit<32> n);                // Перемещение указателя текущей позиции на n.
    bit<32> length();                          // Размер пакета.
}
```

Анализ начинается с predetermined состояния start.

```
state start {
    transition parse_ethernet;               // Переход к анализу заголовка Ethernet.
}
```

Заданное пользователем состояние анализатора для определения типа пакета.

```

state parse_ethernet {
    packet.extract(hdr.ethernet);           // Извлечение заголовка Ethernet.
    transition select(hdr.ethernet.type) { // Определение типа пакета.
        0x800: parse_ipv4;                 // Анализ заголовка IP при EtherType=IP.
        default: accept;                   // Восприятие прочих пакетов без дополнительного
    }                                       // анализа.
}

```

Определения для заголовков IPv4 и IPv6.

```

header ip46_t {
    bit<4> version;           // Версия протокола IP.
    bit<4> reserved;
}

```

Анализ стека заголовков.

```

state parse_labels { // Анализ меток в заголовке
    packet.extract(hdr.labels.next); // Извлечение метки
    transition select(hdr.labels.last.bos) {
        0: parse_labels; // Цикл анализа меток.
        1: guess_labels_payload; // Последняя метка
    }
}

```

Анализ данных после стека заголовков.

```

state guess_labels_payload { // Анализ содержимого.
    transition select(packet.lookahead<ip46_t>().version) {
        4 : parse_inner_ipv4; // Пакет IPv4
        6 : parse_inner_ipv6; // Пакет IPv6
        default : parse_inner_ethernet; // Кадр Ethernet.
    }
}

```

Сборка пакетов после обработки

Внешний элемент `packet_out` используется для отправки обработанных пакетов.

```

extern packet_out {
    void emit<T>(in T hdr);
}
apply { // Включение действительных заголовков в пакет
    packet.emit(hdr.ethernet); // Отправка пакета.
}

```

Архитектура V1Model

Базовые внешние элементы архитектуры перечислены ниже.

```

extern void truncate(in bit<32> length);
extern void resubmit<T>(in T x);
extern void recirculate<T>(in T x);
enum CloneType { I2E, E2I }
extern void clone(in CloneType type, in bit<32> session);
// Элементы конвейера vlmodel.
parser Parser<H, M>(packet_in pkt,
    out H hdr,
    inout M meta,
    inout standard_metadata_t std_meta);
control VerifyChecksum<H, M>(inout H hdr, inout M meta );
control Ingress<H, M>(inout H hdr,
    inout M meta,
    inout standard_metadata_t std_meta);
control Egress<H, M>(inout H hdr,
    inout M meta,
    inout standard_metadata_t std_meta);
control ComputeChecksum<H, M>(inout H hdr, inout M meta );
control Deparser<H>( packet_out b, in H hdr);
// Коммутатор vlmodel
package V1Switch<H, M>(Parser<H, M> p,
    VerifyChecksum<H, M> vr,
    Ingress<H, M> ig,
    Egress<H, M> eg,
    ComputeChecksum<H, M> ck,
    Deparser<H> d);

```

Стандартные метаданные V1Model

```

struct standard_metadata_t {
    bit<9> ingress_port;
    bit<9> egress_spec;
    bit<9> egress_port;
    bit<32> clone_spec;
    bit<32> instance_type;
    bit<1> drop;
    bit<16> recirculate_port;
    bit<32> packet_length;
    bit<32> enq_timestamp;
    bit<19> enq_qdepth;
    bit<32> deq_timedelta;
    bit<19> deq_qdepth;
}

```

```
    bit<48> ingress_global_timestamp;
    bit<48> egress_global_timestamp;
    bit<32> lf_field_list;
    bit<16> mcast_grp;
    bit<32> resubmit_flag;
    bit<16> egress_rid;
    bit<1> checksum_error;
    bit<32> recirculate_flag;
}
```

Счетчики и регистры V1Model

Счетчики

```
counter(8192, CounterType.packets) c;
action count(bit<32> index) {
    c.count(index);        // Инкрементирование счетчика пакетов по индексу.
}
```

Регистры

```
register<bit<48>>(16384) r;
action ipg(out bit<48> ival, bit<32> x) {
    bit<48> last;
    bit<48> now;
    r.read(last, x);
    now = std_meta.ingress_global_timestamp;
    ival = now - last;
    r.write(x, now);
}
```

Николай Малых

nmalykh@protocols.ru