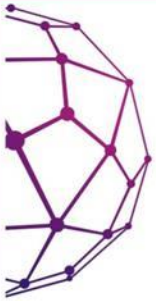


ЦЕНТР
ПРИКЛАДНЫХ
ИССЛЕДОВАНИЙ
КОМПЬЮТЕРНЫХ
СЕТЕЙ

P4: Programming Protocol-independent Packet Processors

Pat Bosshart, Dan Talayco, Glen Gibb, Martin Izzard (Barefoot Networks), Dan Daly (Intel), Nick McKeown (Stanford University), Jennifer Rexford, Cole Schlesinger, David Walker, (Princeton University), Amin Vahdat (Google), George Varghese (Microsoft Research)

Вячеслав Васин: vvasin@arccn.ru

Постановка задачи:

Version	Date	Header Fields
OF 1.0	Dec 2009	12 fields (Ethernet, TCP/IPv4)
OF 1.1	Feb 2011	15 fields (MPLS, inter-table metadata)
OF 1.2	Dec 2011	36 fields (ARP, ICMP, IPv6, etc.)
OF 1.3	Jun 2012	40 fields
OF 1.4	Oct 2013	41 fields

P4 – прототип дальнейшего совершенствования протокола OpenFlow:

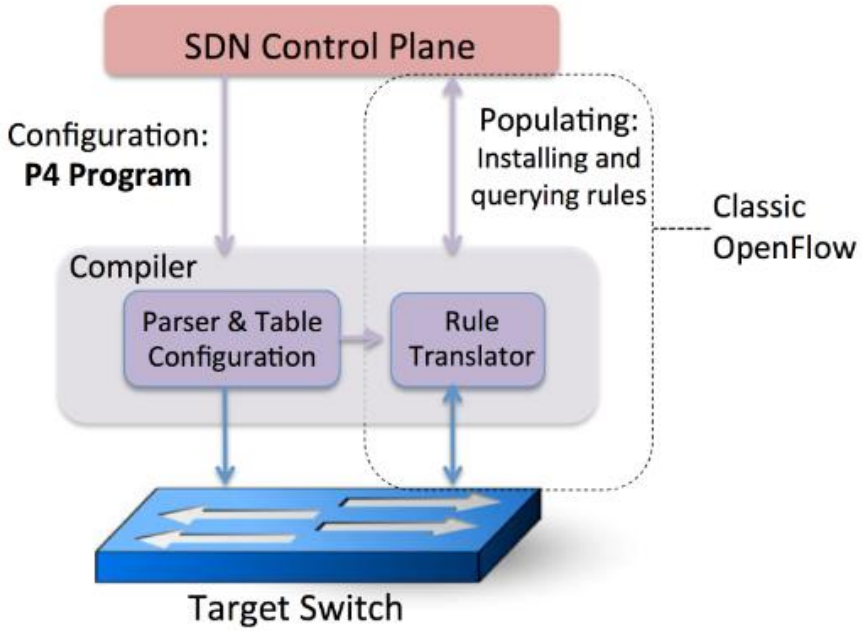
- Контроллер должен уметь менять алгоритм обработки пакетов коммутаторами.
- Коммутаторы не должны быть завязаны на обработку пакетов каких либо сетевых протоколов. Контроллер должен определять:
  - Алгоритм разбора заголовков пакетов (определение типов, имен)
  - Совокупность таблиц (match+action) для обработки этих заголовков
- Программисты должны иметь возможность описывать процесс обработки пакетов независимо от того как реально это буде реализовано в аппаратуре.
  - Компилятор должен преобразовывать платформо-независимые описания (на языке P4) в платформо-зависимые воздействия на конкретный коммутатор.

Нужен гибкий механизм определения используемых при анализе заголовков - новый API (OpenFlow2.0)

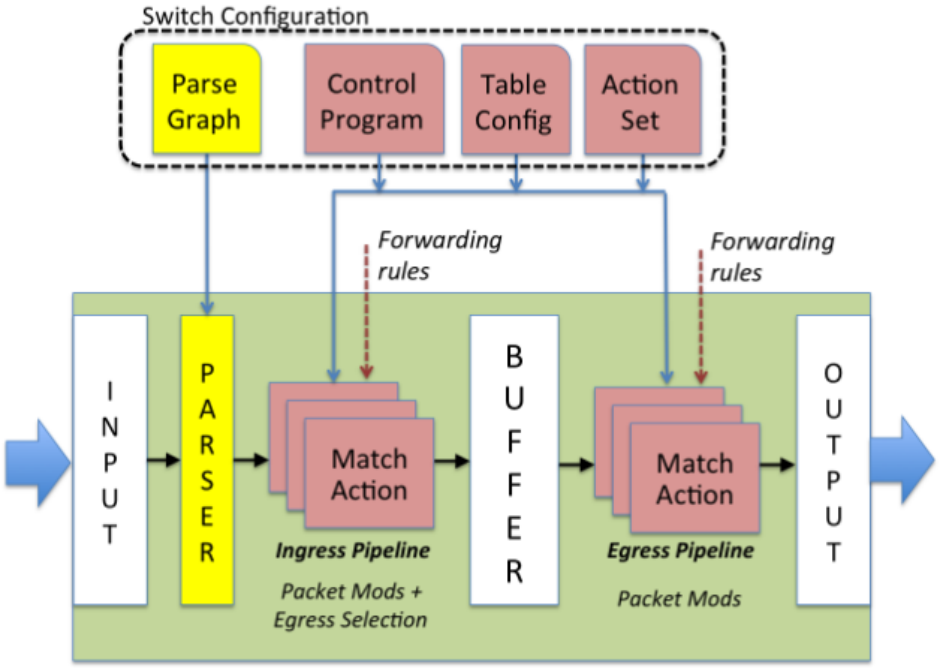
Архитектура решения:

P4 – высокоуровневый язык программирования пакетных процессоров.

- P4 – интерфейс взаимодействия контроллера и коммутатора, описывающий как коммутатору обрабатывать пакеты.
- Обработка посредством программируемого парсера (против фиксированного в OF) и множества последовательных или параллельных стадий match+action (в OF только последовательно)



P4 – язык конфигурирования коммутатора



Абстрактная модель коммутатора

P4:

Операция конфигурирования (*Configure*) в абстрактной модели:

- Программирование парсера
- Определение порядка match+action стадий
- Определение используемых на каждой стадии полей заголовков и производимых с ними действий (проверки и уменьшения для TTL, добавления поля для создания туннеля, проверки контрольной суммы и пр)

Операция заполнения (*Populate*) в абстрактной модели:

- Добавление и удаление записей в match+action таблицы

Таблицы (match+action) делятся на:

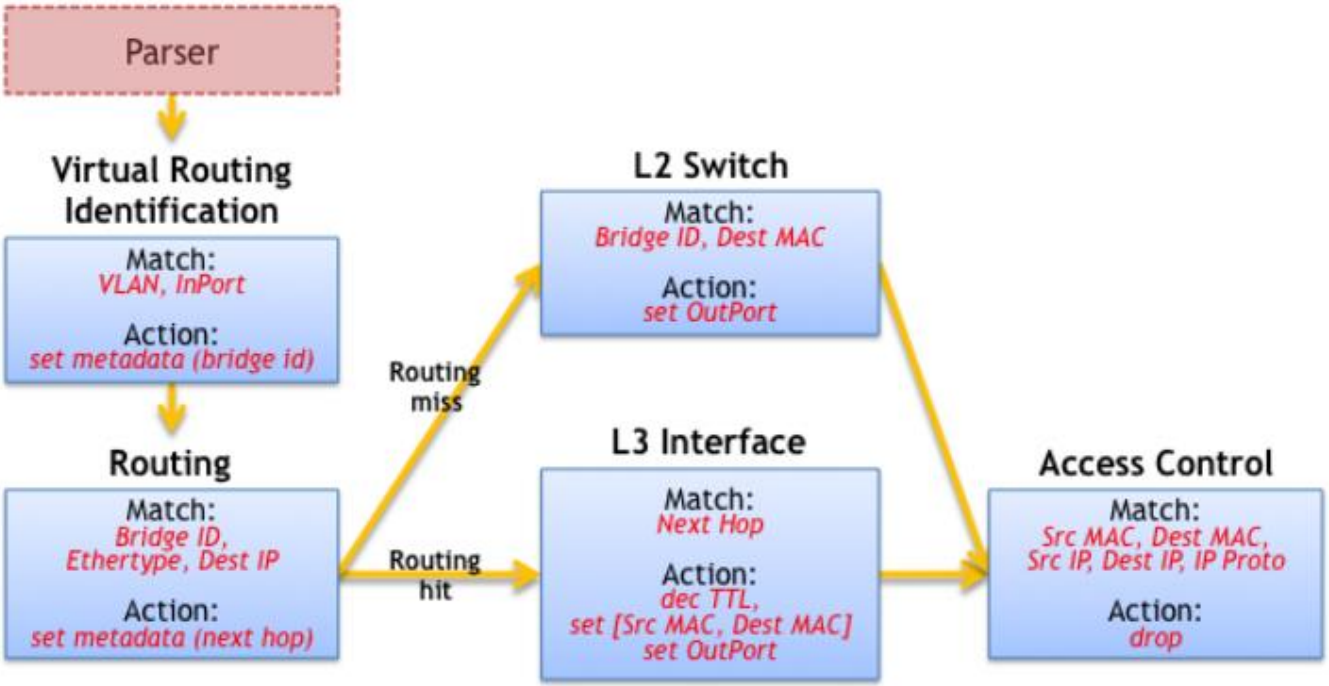
- Входящие:
  - Определяют исходящий порт + очередь.
  - Пакет может быть отправлен, реплицирован, сброшен, отправлен на контроллер.
- Исходящие:
  - Осуществляют индивидуальную модификацию заголовков (например для multicast)

P4. Описание графа зависимостей между таблицами.

На 1 стадии на (высоком) уровне описывается процесс обработки пакетов на языке P4

На 2 стадии компилятор транслирует P4 в Table Dependency Graph (TDG)

На 3 стадии TDG приводится в соответствие с особенностями коммутатора



L2/L3 Table Dependency Graph (TDG)

P4. Пример

Описание формата заголовка

```
header ethernet {
  fields {
     dst_addr : 48; // width in bits
     src_addr : 48;
     ethertype : 16;
  }
}
header vlan {
  fields {
      pcp : 3;
      cfi : 1;
      vid : 12;
      ethertype : 16;
  }
}
```

```
header mTag {
   fields {
      up1 : 8;
      up2 : 8;
      down1 : 8;
      down2 : 8;
      ethertype : 16;
   }
}
```

P4. Пример

Описание парсера (определение заголовков и их последовательностей)

```
parser start {
    ethernet;
}
parser ethernet {
    switch(ethertype) {
        case 0x8100: vlan;
        case 0x9100: vlan;
        case 0x800: ipv4;
    }
}
parser vlan {
    switch(ethertype) {
        case 0xaaaa: mTag;
        case 0x800: ipv4;
    }
}
```

```
parser mTag {
    switch(ethertype) {
        case 0x800: ipv4;
    }
}
```

P4. Пример

Описание Таблиц

```
table mTag_table {
    reads {
        ethernet.dst_addr : exact;
        vlan.vid : exact;
    }
    actions {
         add_mTag;
    }
    max_size : 20000;
}
```

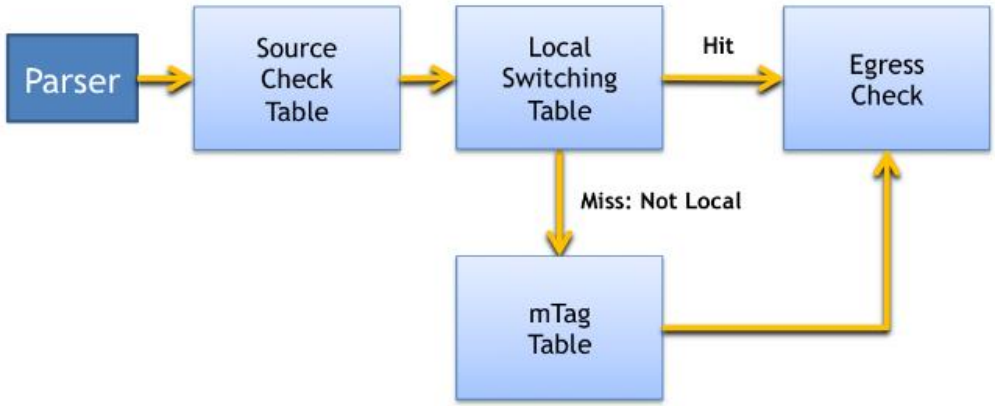
```
table source_check {
    reads {
        mtag : valid; // Was mtag parsed?
        metadata.ingress_port : exact;
    }
    actions { // Each table entry specifies *one* action
        // If inappropriate mTag, send to CPU
        fault_to_cpu;
        // If mtag found, strip and record in metadata
        strip_mtag;
        // Otherwise, allow the packet to continue
        pass;
     }
     max_size : 64; // One rule per port
}
```

P4. Пример

Описание Действий (set_field, copy_field, add_header, Remove_header. Increment, checksum)

```
action add_mTag(up1, up2, down1, down2, egr_spec) {
    add_header(mTag);
    // Copy VLAN ethertype to mTag
    copy_field(mTag.ethertype, vlan.ethertype);
    // Set VLAN's ethertype to signal mTag
    set_field(vlan.ethertype, 0xaaaa);
    set_field(mTag.up1, up1);
    set_field(mTag.up2, up2);
    set_field(mTag.down1, down1);
    set_field(mTag.down2, down2);
    // Set the destination egress port as well
    set_field(metadata.egress_spec, egr_spec);
}
```

P4. Пример



Описание Table Dependency Graph

```
control main() {
  // Verify mTag state and port are
consistent
  table(source_check);
  // If no error from source_check, continue
  if (!defined(metadata.ingress_error)) {
    // Attempt to switch to end hosts
    table(local_switching);
    if (!defined(metadata.egress_spec)) {
      // Not a known local host; try mtagging
       table(mTag_table);
     }
     // Check for unknown egress state or
     // bad retagging with mTag.
     table(egress_check);
  }
}
```

P4. Пример

Компиляция с учетом:

- программных коммутаторов
- аппаратных коммутаторов с RAM и TCAM
- коммутаторов поддерживающих параллельные таблицы
- коммутаторов поддерживающих применение действий только по окончании прохода конвейера
- коммутаторы с малым количеством таблиц
- прочее

Выводы

- Протокол должен позволять настраивать коммутаторы на лету “in field”.
- Программист не должен беспокоиться о деталях аппаратной реализации.
- Предложен язык конфигурации.
- Коммутаторы должны стать гибче.

Спасибо!

Вячеслав Васин: vvasin@arccn.ru