Архитектура uBPF

Приведённый ниже включаемый файл (include) с переведёнными на русский язык комментариями содержит определения пакета ubpf в одноимённой модели. Файл ubpf.p4 размещается в каталоге p4include пакета p4c и доступен по ссылке.

```
/*
Copyright 2019 Orange

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

    http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.
*/

#ifndef _UBPF_MODEL_P4_
#define _UBPF_MODEL_P4_

#include <core.p4>

/*
 * Платформа uBPF в настоящее время позволяет пропускать (pass) или отбрасывать
 * (drop) пакеты. По умолчанию пакеты пропускаются. Можно использовать
 * внешнюю функцию mark_to_drop() чтобы пометить пакет для отбрасывания.
 * Функция mark_to_drop() меняет лишь состояние, скрытое от  пользовательской
 * программы P4 и вызывать её следует лишь из элемента управления pipe.
 */
extern void mark_to_drop();

/*
 * Платформа uBPF в настоящее время позволяет пропускать (pass) или отбрасывать
 * (drop) пакеты. По умолчанию пакеты пропускаются. Можно использовать
 * внешнюю функцию mark_to_pass() чтобы пометить пакет для пропускания (это
 * отменяет предшествующее действие mark_to_drop()). Функция mark_to_drop()
 * меняет лишь состояние, скрытое от  пользовательской программы P4
 * и вызывать её следует лишь из элемента управления pipe.
 */
extern void mark_to_pass();


extern Register<T, S> {
  /***
   * Объект Register создаётся вызовом его конструктора. При вызове нужно
   * указать размер Register, определяющий максимальное число записей в
   * регистре. После создания объекта Register его можно применять в
   * действиях и блоках apply. При создании Register не инициализируется.
   */
  Register(bit<32> size);

  /***
   * Метод read() считывает состояние (T) из массива регистров по заданному
   * индексу S и возвращает значение в параметре result.
   *
   * @param index Индекс элемента массива регистров для считывания. Обычно
   *              из диапазона [0, size-1].
   * @return Возвращает результат типа T (в настоящее время только bit<W>).
   *         Если индекс находится в диапазоне, result получает значение
   *         элемента массива регистров. При index >= size результат
   *         становится неопределённым и его следует игнорировать.
   */
  T read  (in S index);

  void write (in S index, in T value);
}

/*
 * Функция возвращает метку текущего времени в наносекундах.
 */
extern bit<48> ubpf_time_get_ns();

enum HashAlgorithm {
    lookup3
}
```

```
/***
 * Рассчитывает хэш-функцию от значения, заданного параметром data. В силу
 * ограничений uBPF максимальный размер data составляет bit<64>.
 *
 * Отметим, что типы всех параметров могут быть одинаковыми или разными.
 * Во втором случае их битовые размеры могут различаться.
 *
 * Результат всегда имеет размер bit<32>.
 *
 * @param D    Должен иметь тип tuple, состоящий из битовых полей (type bit<W>, int<W>)
 *             или varbit. Максимальный размер D составляет 64 бита (ограничение uBPF).
 */
extern void hash<D>(out bit<32> result, in HashAlgorithm algo, in D data);

/***
 * Расчёт контрольной суммы с помощью инкрементального обновления (RFC 1624).
 * Функция реализует расчёт контрольной суммы для 16-битовых полей.
 */
extern bit<16> csum_replace2(in bit<16> csum,  // Текущая контрольная сумма.
                             in bit<16> old,   // Старое значение поля.
                             in bit<16> new);

/***
 * Расчёт контрольной суммы с помощью инкрементального обновления (RFC 1624).
 * Функция реализует расчёт контрольной суммы для 32-битовых полей.
 */
extern bit<16> csum_replace4(in bit<16> csum,
                             in bit<32> old,
                             in bit<32> new);

/*
 * Архитектура
 *
 * Параметр M должен иметь тип struct.
 *
 * Параметр H должен иметь тип struct, а каждый элемент структуры должен
 * быть заголовком, стеком или объединением заголовков (header_union).
 */

parser parse<H, M>(packet_in packet, out H headers, inout M meta);
control pipeline<H, M>(inout H headers, inout M meta);

/*
 * В теле сборщика (deparser) допускается лишь вызов метода packet_out.emit().
 */
@deparser
control deparser<H>(packet_out b, in H headers);

package ubpf<H, M>(parse<H, M> prs,
                   pipeline<H, M> p,
                   deparser<H> dprs);

#endif /* _UBPF_MODEL_P4_ */
```

Перевод на русский язык

Николай Малых

nmalykh@protokols.ru