Моделирование и анализ информационных систем. Т. 26, № 1 (2019), с. 39–62
Modeling and Analysis of Information Systems. Vol. 26, No 1 (2019), pp. 39–62

DOI: 10.18255/1818-1015-2019-1-39-62

УДК 517.9

Об одном подходе к построению сетевого процессорного устройства

Беззубцев С. О., Васин В. В., Волканов Д. Ю., Жайлауова Ш. Р., Мирошник В. А., Скобцова Ю. А., Смелянский Р. Л.

Поступила в редакцию 10 января 2019
После доработки 12 февраля 2019
Принята к публикации 15 февраля 2019

Аннотация. В работе предложена архитектура и основные требования к сетевому процессору для OpenFlow коммутаторов программно-конфигурируемых сетей (ПКС). Представлен анализ архитектур известных сетевых процессоров – NP-5 компании EZchip (в настоящее время Mellanox) и Tofino компании Barefoot Networks. Рассмотрены достоинства и недостатки двух разных вариантов архитектур сетевого процессора: на основе конвейеров, ячейки которых представлены набором процессорных ядер общего назначения, и на основе конвейеров, ячейкам которых соответствуют ядра, специализированные под конкретные операции обработки пакета. На основе выделенного набора наиболее общих сценариев обработки пакетов предложена новая архитектура сетевого процессорного устройства (СПУ) с функционально специализированными ячейками (стадиями) конвейера. В статье представлено описание имитационной модели СПУ предложенной архитектуры. Имитационная модель построена на языке C++ с использованием открытой библиотеки SystemC. Для проведения функционального тестирования полученной модели СПУ были реализованы описанные сценарии обработки пакетов на языке C. Для оценки производительности предложенной архитектуры СПУ в ходе исследования были использованы программные средства компании KM211, а также семейство микроконтроллеров KMX32. Оценка производительности СПУ проводилась на основе имитационной модели. Получены оценки времени обработки одного пакета и средняя пропускная способность модели СПУ для каждого сценария. Эти оценки показали, что полученная скорость СПУ позволяет их использование в коммутаторах уровня распределения (агрегации).

Ключевые слова: сетевое процессорное устройство, коммутатор, компьютерные сети, программно-конфигурируемые сети, архитектура компьютера, имитационное моделирование, протокол OpenFlow

Для цитирования: Беззубцев С.О., Васин В.В., Волканов Д.Ю., Жайлауова Ш.Р., Мирошник В.А., Скобцова Ю.А., Смелянский Р.Л., "Об одном подходе к построению сетевого процессорного устройства", *Моделирование и анализ информационных систем*, **26**:1 (2019), 39–62.

Об авторах:
Беззубцев Станислав Олегович, технический специалист, orcid.org/0000-0002-2419-7632
Центр прикладных исследований компьютерных сетей,
Ленинские горы, 1, стр. 77, г. Москва, 119992 Россия, e-mail: stas.bezzubtsev@gmail.com

Васин Вячеслав Викторович, ведущий программист-разработчик, orcid.org/0000-0002-5759-3105
Центр прикладных исследований компьютерных сетей,
Ленинские горы, 1, стр. 77, г. Москва, 119992 Россия, e-mail: vvasin@arccn.ru

Волканов Дмитрий Юрьевич, канд. физ.-мат. наук, доцент, orcid.org/0000-0001-9940-5822
Московский государственный университет имени М.В. Ломоносова,
Ленинские горы, 1, г. Москва, 119991 Россия, e-mail: volkanov@lvk.cs.msu.su

Жайлауова Шынар Рустембековна, аспирант, orcid.org/0000-0001-5725-7040
Московский государственный университет имени М.В. Ломоносова,
Ленинские горы, 1, г. Москва, 119991 Россия, Россия, e-mail: szhaylau@cs.msu.ru

Мирошник Владислав Александрович, программист-разработчик, orcid.org/0000-0003-0883-0116
Московский государственный университет имени М.В. Ломоносова,
Ленинские горы, 1, г. Москва, 119991 Россия, e-mail: miroshnikv@lvk.cs.msu.su

Скобцова Юлия Александровна, студент-магистр, orcid.org/0000-0001-8351-3191
Московский государственный университет имени М.В. Ломоносова,
Ленинские горы, 1, г. Москва, 119991 Россия, e-mail: xenerizes@lvk.cs.msu.su

Смелянский Руслан Леонидович, чл.-кор. РАН, д-р физ.-мат. наук, проф., orcid.org/0000-0003-2311-4513
Московский государственный университет имени М.В. Ломоносова,
Ленинские горы, 1, г. Москва, 119991 Россия, e-mail: smel@cs.msu.su

Благодарности:
Работа выполнена при финансовой поддержке гранта РФФИ (№ 19-07-01076).

Введение

Целью данной работы являлась разработка архитектуры сетевого процессорного устройства (СПУ). Под СПУ понимаем программируемое устройство, настраиваемое под стек протоколов. Разрабатываемая архитектура должна учитывать требования к коммутаторам как для традиционных TCP/IP сетей (IP/MPLS маршрутизаторов), так и для программно-конфигурируемых сетей (ПКС) с коммутацией пакетов и позволять настройку устройства на работу в одной из вышеупомянутых видов сетей программными методами. Рассматривается проблема построения коммутаторов для ПКС сетей [1]. В контексте данной статьи наиболее существенно то, что коммутаторы в ПКС программно управляемы. А именно, рассматриваются ПКС сети с протоколом OpenFlow [2]. Такие ПКС сети предполагают, что центр управления сетью – ПКС контроллер со своими приложениями – формирует программы обработки пакетов данных для каждого коммутатора в контуре передачи данных.

В качестве устройства обработки и передачи трафика в таких коммутаторах может использоваться:

- универсальный микропроцессор;
- сетевой процессор;
- специализированная интегральная схема.

СПУ занимает промежуточное место между универсальными микропроцессорами (также называемыми центральными процессорными устройствами ЦПУ), выполняющими практически любые задачи на низкой скорости, и специализированными интегральными схемами (ASIC), разработанными для эффективного решения конкретного набора задач, объединяя в себе гибкость первых и при правильном проектировании мощность вторых.

Ниже представлена последовательность задач обработки пакета (см. Рисунок 1), выполняемых СПУ: выделение заголовка пакета, классификация и модификация. Одним из способов ускорения обработки данных является конвейеризация выполняемых над ними действий.

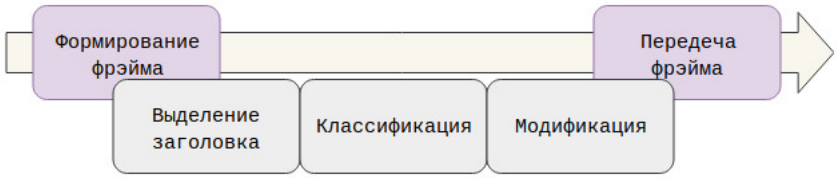

Рис. 1. Классификация задач СПУ. Белым цветом выделены задачи СПУ

Fig. 1. Taxonomy of network processing functions.
The network processor tasks are highlighted in white

К примеру, в случае классического коммутатора может быть достаточно выполнения одной цепочки действий:

- выделение заголовка;
- классификация;
- модификация.

Как будет видно из раздела 3., число цепочек обработки может возрасти.

В коммутаторах, используемых в традиционных компьютерных сетях, архитектура процессора обычно оптимизирована для решения определённого круга задач и работы по определённым протоколам. В случае программируемого СПУ в составе коммутатора, возможно "переиспользование" ячеек конвейера для разных сценариев. Это означает, что действия, выполняемые в каждой из ячеек, некоторым образом параметризуют, и в дальнейшем используют для программирования поведения ячейки. Программист может перепрограммировать работу ячейки с помощью библиотеки функций на языке общего назначения, например на языке C для сетевого процессора EZChip NP5 [5], или с помощью особого предметно-ориентированного языка описания поведения сетевого устройства, например языка P4 [6] для сетевого процессора Barefoot Tofino [7].

Современные крупные сети очень сложны, поскольку определяются множеством протоколов, конфигурациями и технологиями. Используя иерархическую модель, можно легко упорядочить все компоненты сети согласно характеристикам каждого иерархического уровня. Примером такой модели может служить трехуровневая иерархическая модель сети [4], определяющая подход к проектированию сетей и включающая в себя три логических уровня:

- уровень ядра ($>$ 10 Гбит/с);
- уровень распределения/агрегации (1 – 10 Гбит/с);
- уровень доступа ($<$ 1 Гбит/с).

Данная работа организована следующим образом. В разделе 1. описаны две характерные архитектуры СПУ. Сценарии обработки пакетов представлены в разделе 2. Предлагаемая архитектура СПУ описана в разделе 3. Реализованная имитационная модель СПУ рассматривается в разделе 4. Раздел 5. содержит результаты экспериментов, проведенных с имитационной моделью. В заключение проделанной работы в шестом разделе приводится краткое описание направлений предстоящих исследований.

1. Обзор существующих архитектур СПУ

Можно выделить два основных варианта конвейерных архитектур СПУ: использование внутри ячеек конвейера большого числа простых процессорных ядер, которые в явном виде не специализированы под конкретные задачи, и, наоборот, использование ядер, специализированных под конкретные операции обработки пакета. Каждый из этих подходов имеет свои достоинства и недостатки [3].

В настоящем разделе будут рассмотрены обе архитектуры. P4-программируемая абстрактная модель коммутатора (которая может быть рассмотрена как очень общая модель конвейера СПУ) со своей реализацией в процессоре Tofino [7] (компании Barefoot Networks) является примером первого подхода с одинаковыми стадиями конвейера. В качестве примера второго подхода рассмотрим процессор NP-5 [5] (компании EZchip) с функционально специализированными стадиями. Также будут описаны особенности структуры пути обработки данных и некоторые особенности физического устройства.

В основе архитектуры сетевых процессоров EZchip NP-5 лежит сеть специализированных ядер (см. Рисунок 2).

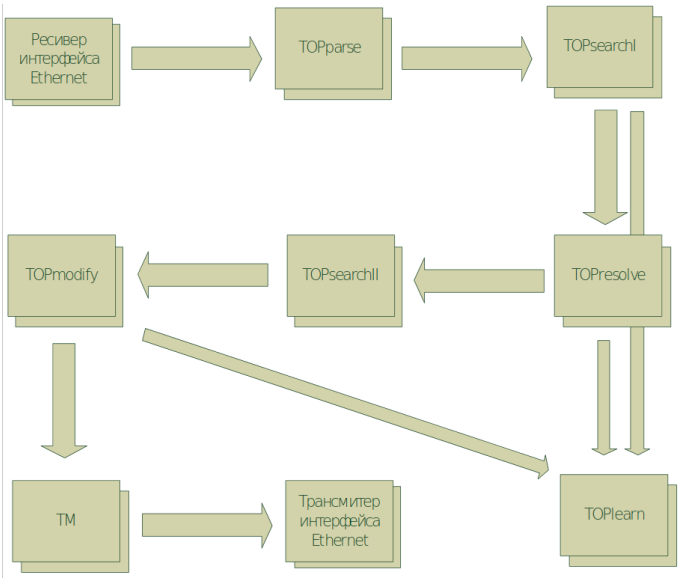

Рис. 2. Логические связи между блоками коммутатора, построенного с использованием сетевого процессора NP-5

Fig. 2. Diagram of links between the objects of the switch based on NP-5

Ядра относятся к разным классам, представители каждого из классов ориентированы на выполнение определённого набора действий: по разбору заголовков сетевых пакетов, классификации сетевых пакетов, преобразованию, управлению потоками. Каждый экземпляр микропроцессора обладает набором ячеек памяти и подключен к ограниченному набору шин данных, необходимых для выполнения определенной задачи.

По тракту обработки данных сетевого процессора NP-5 видно, что он включает в себя одну цепочку действий вида выделение заголовка – поиск – модификация. В случае, если для сценария обработки пакетов этого недостаточно, происходит «заворот» на первую ячейку конвейера, и ячейки используются повторно для реализации последующей обработки пакета. Подобный подход, несмотря на значительную гибкость, понижает скорость работы конвейера. Если для трафика, проходящего по конвейеру сетевого устройства, характерно прохождение двух таких цепочек действий, т. е. всегда используется хотя бы один заворот, скорость обработки пакетов понижается в два раза. Это происходит за счет того, что новые пакеты вынуждены ожидать своей очереди, пока ранее полученные пройдут два цикла обработки.

Модель абстрактного СПУ предложена консорциумом языка программирования сетевых устройств P4 [6]. Стоит отметить, что с точки зрения традиционных сетей подобное устройство может представлять собой и маршрутизатор в зависимости от сценария использования, для работы с которым оно запрограммировано. Предлагаемая модель связана с архитектурой PISA (Protocol Independent Switch Architecture) [6] и опирается на предположение об универсальности ячеек конвейера СПУ и возможность описания их функциональности средствами языка P4. Реализацией данной модели является сетевой процессор Tofino, созданный компанией Barefoot Networks [7].

Тракт обработки данных (см. Рисунок 3) модели абстрактного СПУ подразумевает наличие двух этапов конвейера: входной и выходной обработки, при этом эти этапы, в свою очередь, состоят из трёх программируемых ячеек: ячейка парсера, ячейка конвейера, ячейка депарсера.

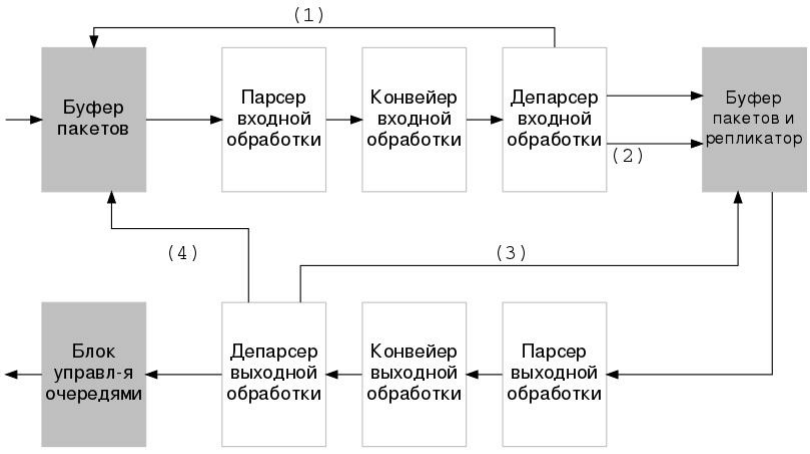

Рис. 3. Схема тракта обработки данных абстрактного СПУ P4

Fig. 3.The data path of the abstract switch model

Устройство ячеек, выполняющих буферизацию, репликацию пакетов и управление очередями, в описании архитектуры не специфицируется. Стоит отметить, что рассматриваемые ячейки и этапы являются абстракциями для разработчика программного обеспечения СПУ и могут значительно отличаться от аппаратной реализации.

Каждая из рассмотренных архитектур имеет достоинства и недостатки. Сетевой процессор NP-5 оптимизирован для использования в коммутаторе, имеющем одну цепочку вида выделение данных — классификация – модификация. В случае, если для сценария использования этого недостаточно, необходимо «завернуть» пакет на вход первой ячейки конвейера, что влечет дополнительные накладные расходы на передачу контекста пакета назад и уменьшение эффективной пропускной способности из-за увеличения времени занятости конвейера. В случае использования MPLS меток возможно большое число «заворотов» внутри конвейера, что приводит к деградации производительности. Сетевой процессор Tofino, претендующий на решение широкого круга задач, допускает большее число таких цепочек, не требуя при этом «заворота» к первой ячейке. В подходе P4 можно производить разбор любого заголовка, разработчик определяет свои структуры для заголовка, что позволяет более гибко работать при использовании MPLS. Тем не менее, здесь тоже может возникнуть проблема физического ограничения на число цепочек для сложных сценариев. Однако здесь не будет возможности сделать «заворот» – будет получена ошибка компиляции программы на P4. Также отсутствуют возможности для оптимизации для какого-либо сценария из-за универсальности ячеек и ограниченности средств языка P4.

Исследования, представленные в данной статье, заключены в попытке скомбинировать оба вышеописанных подхода, когда ячейки конвейера специализированы частично для оптимизации архитектуры с точки зрения сценариев обработки пакетов, описанных в следующем разделе.

2. Описание сценариев обработки пакетов

В ходе работы был выделен набор наиболее общих сценариев обработки пакетов, на которые должно быть рассчитано проектируемое СПУ. В данном разделе приведено описание сценариев обработки пакетов, необходимых для выполнения функционального тестирования конвейера обработки данных и его модели. Такой конвейер может быть использован для разработки семейства ПКС коммутаторов и IP/MPLS маршрутизаторов.

Сценарии обработки пакетов можно разделить на четыре группы (в скобках представлены их ссылочные названия):

1. Сценарии работы классического L2-коммутатора с обучением:
   - простейший Ethernet-коммутатор для локальной сети (L2 switch);
   - Ethernet-коммутатор для виртуальных локальных сетей, протокол 802.1Q (L2 switch + VLAN).
2. Сценарии работы L2/L3 коммутатора:

- L2-коммутатор с обучением для виртуальных локальных сетей и маршрутизацией L3-unicast трафика (L2 switch + VLAN + L3-unicast);
- L2-коммутатор с обучением для виртуальных локальных сетей и маршрутизацией L3-multicast трафика (L2 switch + VLAN + L3-multicast).

3. Сценарии работы сетевых приложений на ПКС-коммутаторе:
   - сценарий обработки L3-multicast трафика (Multicast);
   - виртуальные домены B2C (для коммутаторов с ролями AR, DR), P2P, Multipoint.
4. Сценарии агрегирования, очередизации и перенаправления трафика на ПКС-коммутаторе:
   - сценарий агрегирования каналов (LAG);
   - сценарий разбора поля Ethertype для LACP/LLDP/QoS;
   - сценарий зеркалирования трафика (Mirror);
   - передача трафика уровня управления по каналам уровня передачи данных (Inband).

Далее опишем по одному сценарию из каждой группы.

2.1. Сценарий L2 switch

Данный сценарий соответствует работе простейшего L2-коммутатора для локальной сети. Сценарий предполагает наличие таблицы, которая содержит записи вида:

$$< \text{MAC-адрес} >, < \text{номер порта} > .$$

Каждая из таких записей означает, что за данным портом находится устройство с данным MAC-адресом.

В данном сценарии необходимо выполнить два действия (см. Рисунок 4). Во-первых, определить порт, в который пакет должен быть отправлен. Во-вторых, запомнить пару (MAC-адрес отправителя, номер порта), чтобы в дальнейшем эффективнее коммутировать пакеты.

Для выполнения первой задачи проводится поиск по таблице. Ключом поиска является MAC-адрес получателя. В случае успешного завершения поиска по таблице получаем искомый номер порта, в который будет отправлен пакет. Иначе данный пакет будет отправлен во все порты, кроме порта, через который этот пакет был получен.

Для выполнения второй задачи в той же таблице проводится поиск записи:

$$< \text{MAC-адрес отправителя} >, < \text{номер порта} > .$$

Если поиск завершился успешно, то у найденной записи обнуляется параметр age_time. Иначе такая запись добавляется в таблицу, и обнуляется соответствующий ей параметр age_time.

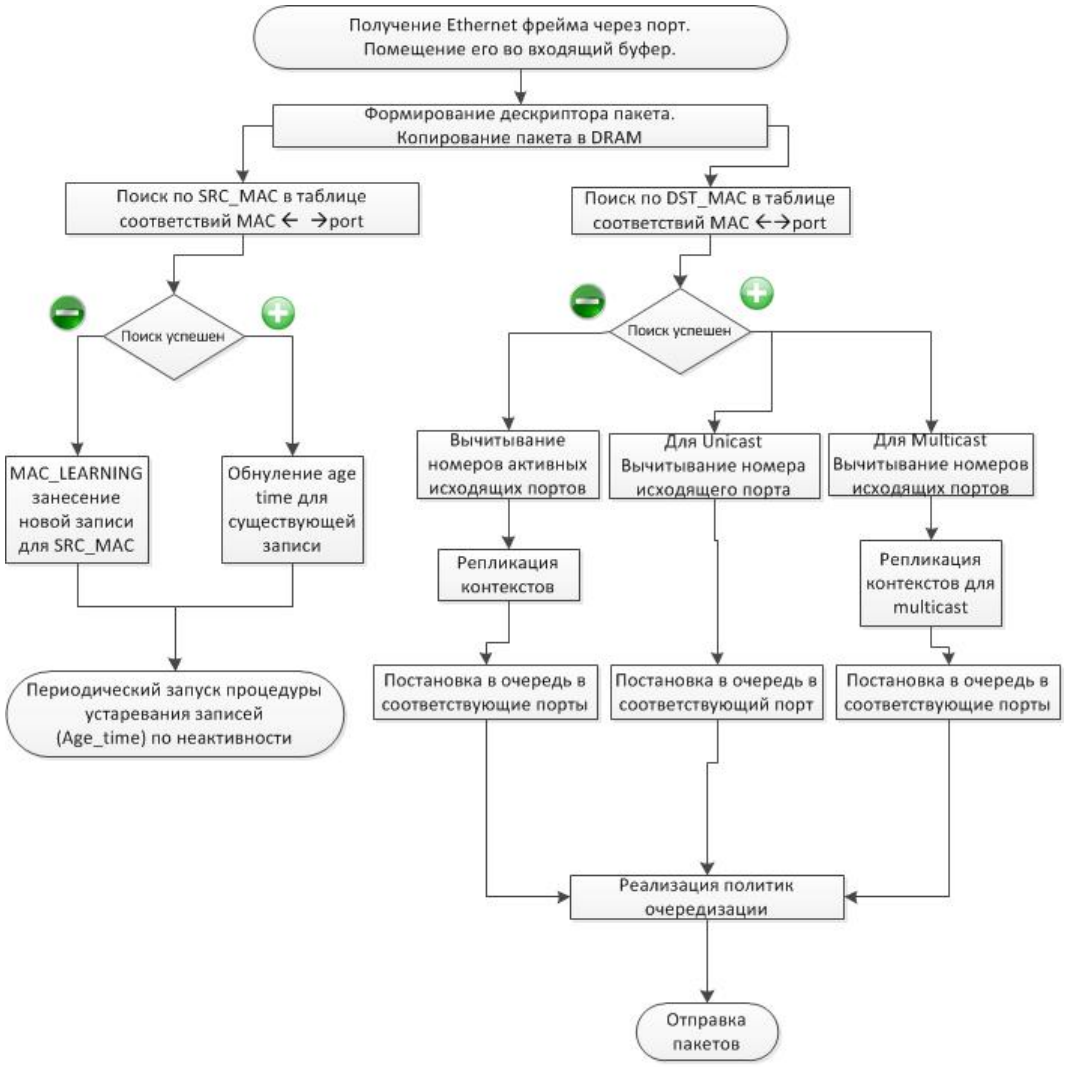

Рис. 4. Схема работы процессора для сценария L2 switch

Fig. 4. The use case scenario for L2-switch

Как правило, к сети постоянно подключаются новые устройства и отключаются старые. Поэтому на коммутаторе работает механизм поддержки актуальности записей в таблице. Для этого периодически запускается процедура устаревания записей по неактивности. Она просматривает все существующие записи, и если параметр age_time превышает заданный порог, то запись из таблицы удаляется.

2.2. Сценарий L2 switch + VLAN + L3-unicast

Данный сценарий является расширением сценария L2 switch за счет добавления дополнительной обработки VLAN-тегов, а также ветки маршрутизации пакета на уровне L3 [11], если полученный пакет необходимо маршрутизировать, а не коммутировать (см. Рисунок 5). Решение об этом принимается после присвоения пакету соответствующего номера vlan и снятия тега 802.1Q, если он присутствует в пакете.

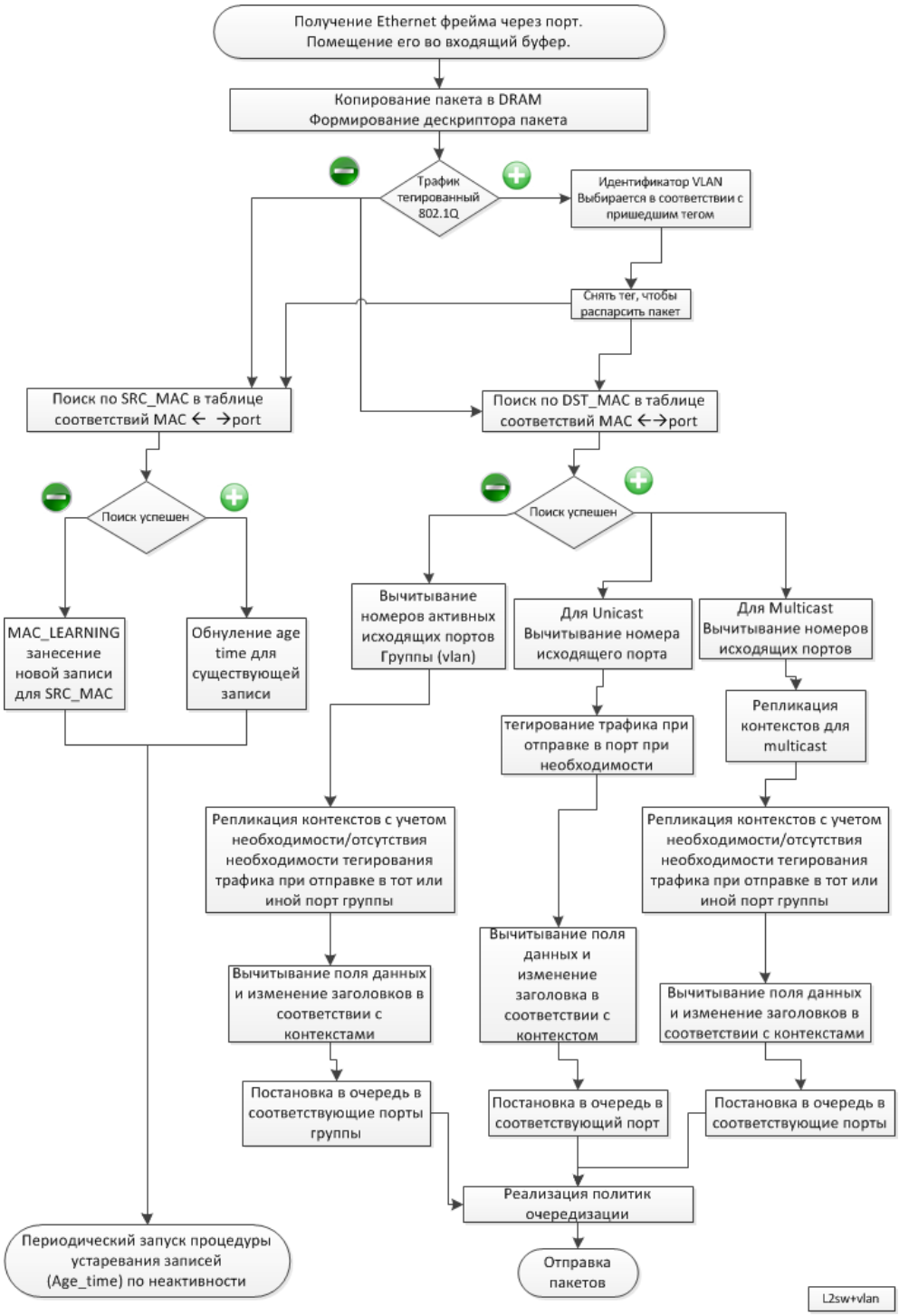

Рис. 5. Схема работы процессора для сценария L2 switch + VLAN + L3-unicast

Fig. 5. The use case for L2 switch + VLAN + L3-unicast

Для этого производится поиск по таблице, содержащей MAC-адреса получателей широковещательного трафика, которая заполняется при получении коммутатором IGMP запросов определенного типа. Если MAC-адрес получателя из принятого пакета был найден в таблице, то принимается решение о маршрутизации пакета. Ина-

че выполнение сценария идет по ветке коммутации, где все действия совпадают с описанным выше сценарием.

Далее рассчитывается контрольная сумма CRC заголовка L3. Если полученное значение не совпадает со значением поля контрольной суммы пакета, то он сбрасывается. Затем поле TTL уменьшается на единицу, и если оно стало равно нулю, то пакет сбрасывается. Далее необходимо определить, в какие порты отправить данный пакет. Для этого есть специальная таблица Multicast маршрутизации, которая содержит записи вида:

$$< \text{IP-адрес получателя} >, < \text{IP-адрес отправителя} >,$$

$$< \text{номер входящего интерфейса} >, < \{\text{список исходящих интерфейсов}\} > .$$

Ключом поиска является IP-адрес получателя, содержащийся в заголовке пакета. Если поиск был неуспешен, пакет сбрасывается. Иначе рассчитывается новая контрольная сумма L3, так как ранее поле TTL было уменьшено, и в L2 заголовке данного пакета MAC-адрес отправителя заменяется на MAC-адрес исходящего L3 интерфейса коммутатора. Затем происходит репликация контекстов для отправки в каждый порт с учетом необходимости или отсутствия необходимости тегирования заголовка пакета.

2.3. Сценарий B2C-AR

Сценарии виртуального домена B2C отвечают за предоставление отказоустойчивого сервиса клиенту. Клиент идентифицируется по MAC-адресу и VLAN-тегу. Для клиента, подключенного к коммутатору, реализующему сценарий B2C-DR, предоставляется сервис (например, доступ в Интернет) через коммутатор, реализующий сценарий B2C-AR. Коммутатор AR выбирает работающий порт по VLAN трафика клиента, и передает трафик на него. Этот сценарий требует одной таблицы, которая содержит поля:

$$< \text{номер порта} >, < \text{номер vlan} > .$$

Поиск по таблице ведется по принципу полного совпадения полей. В случае неудачи пакет передается на обработку по другой схеме (см. Рисунок 6).

Если поиск завершается успешно, то далее проводится поиск по групповой таблице. Группа в этом сценарии — множество портов, из которого по специальному алгоритму выбирается один порт, куда будет отправлен пакет. Алгоритм состоит в том, что сначала проверяется порт, указанный первым в списке соответствующей группы. Если он находится в рабочем состоянии, пакет отправляется на этот порт. Если он в неработоспособном состоянии, то проверяется следующий по списку порт. Эти действия продолжаются до тех пор, пока не будет найден работающий порт. Если все порты оказываются в нерабочем состоянии, то пакет сбрасывается.

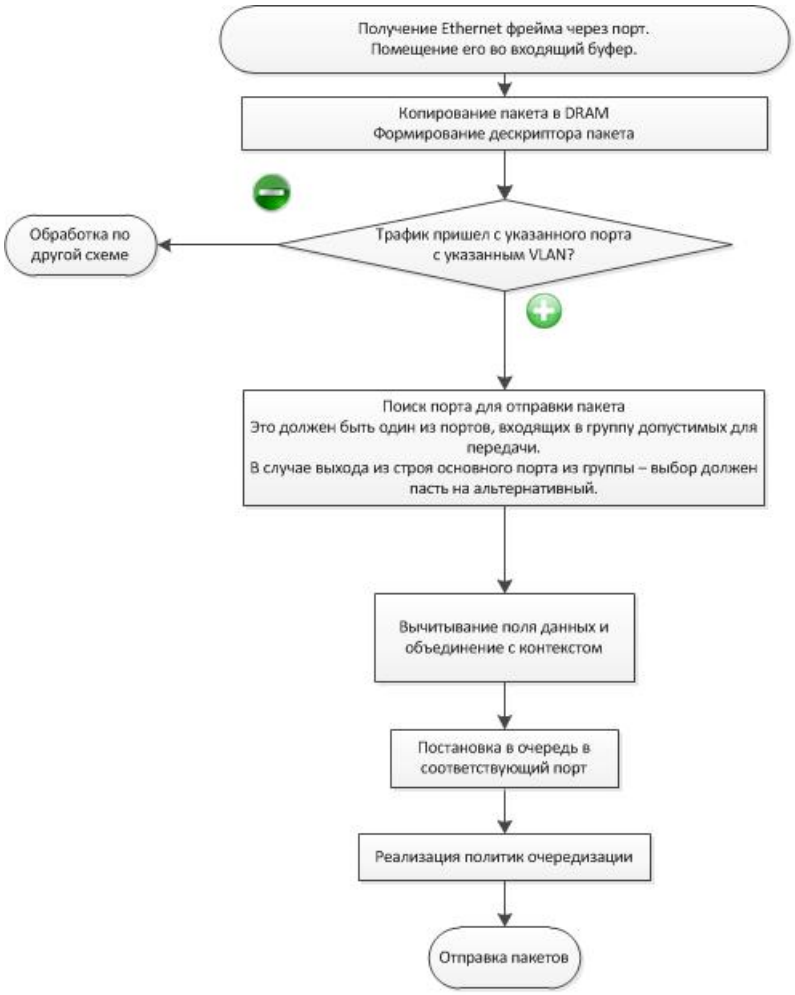

Рис. 6. Схема работы процессора для сценария B2C-AR

Fig. 6. The use case scenario for B2C-AR

2.4. Сценарий Mirror

Сценарий Mirror реализует зеркалирование трафика на коммутаторе программно-конфигурируемой сети. Для данного сценария достаточно одной таблицы, которая содержит поля:

$$< \text{номер порта} >, < \text{s\_vlan} >, < \text{c\_vlan} >, < \text{набор действий} > .$$

Обработка пакета происходит следующим образом (см. Рисунок 7). Сначала происходит проверка наличия тега 802.1Q в заголовке пакета. Если он есть, то значение vlan запоминается в контексте как s_vlan, а тег снимается. Далее проверяется наличие второго тега и происходят аналогичные действия, а значение vlan запоминается как c_vlan. В результате заголовок пакета не содержит тегов, и все нужные значения записаны в контекст. На следующем шаге просматривается описанная выше таблица, в которой ищутся полные совпадения по полям. В случае успеха производится копирование пакета, и к данной копии добавляются теги при их наличии в

изначальном пакете, а затем она отправляется на нужный порт. Тело оригинального пакета передается на обработку по другой схеме.

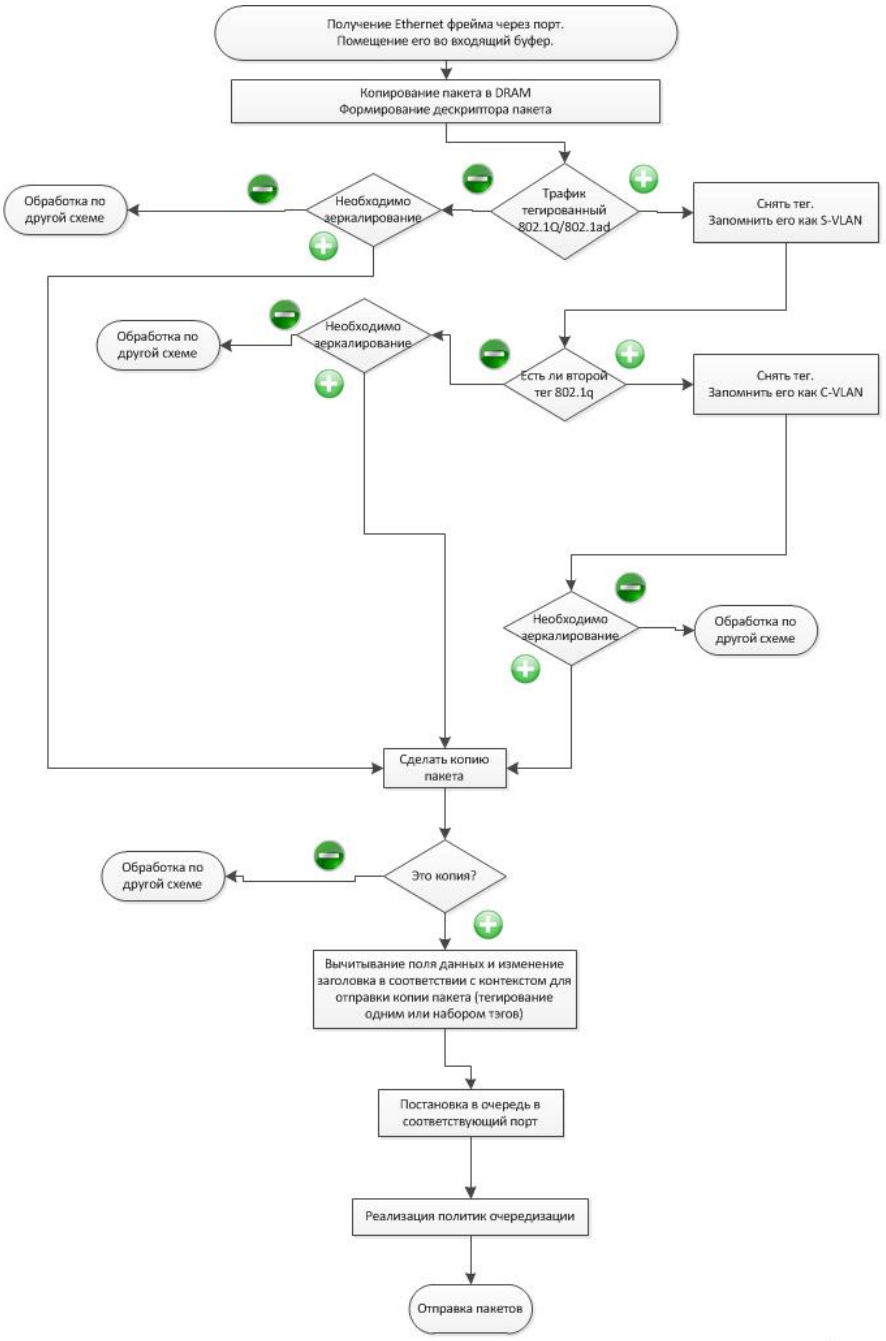

Рис. 7. Схема работы процессора для сценария Mirror

Fig. 7. The use case scenario for Mirror

3. Описание предлагаемой архитектуры

В данном разделе представлено описание архитектуры СПУ, основанного на конвейере обработки данных с функционально специализированными ячейками (см. Рисунок 8).

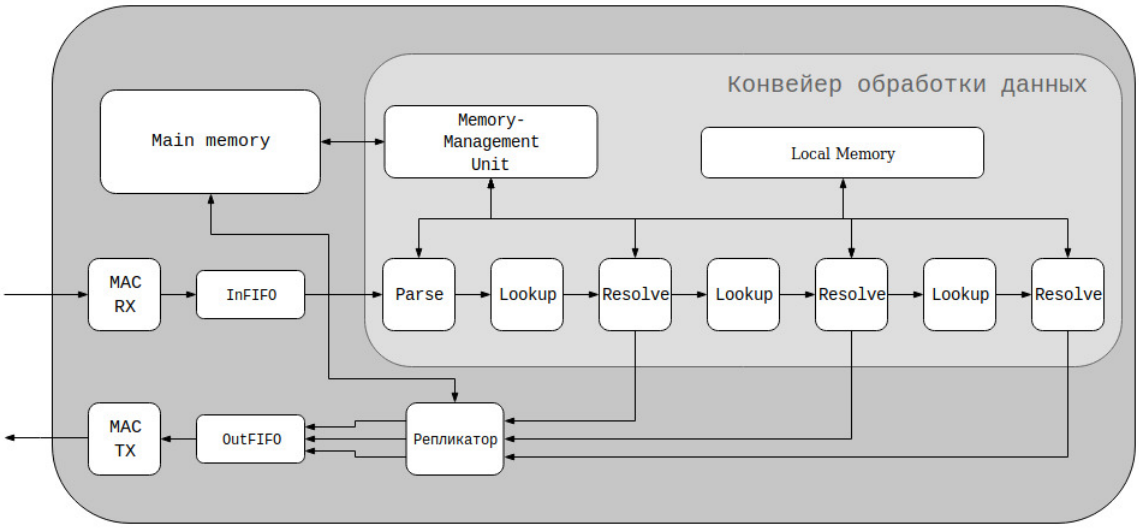

Рис. 8. Описание предлагаемой архитектуры

Fig. 8. Proposed NPU architecture

Основными блоками предлагаемого СПУ являются следующие:

- MAC_RX – ресивер интерфейса Ethernet.
- InFifo/OutFifo – буферизирующие устройства входных/выходных пакетов.
- Конвейер обработки данных - конвейер с семью ячейками, состоящий из ячеек следующих типов:
  - Parse – считывание данных заголовка пакета, необходимых для классификации пакета.
  - Lookup – поиск подходящей записи в одной из таблиц потоков (классификация).
  - Resolve – обработка результата поиска: выполнение модификаций заголовка и принятие решения о завершении обработки пакета либо о её продолжении.
- Resolve – обработка результата поиска: выполнение модификаций заголовка и принятие решения о завершении обработки пакета либо о её продолжении.
- Memory-Management unit – блок управления доступом к основной памяти.
- Main Memory – основная память, предназначенная для хранения тел пакетов.

- Local Memory - внутренняя память, предназначенная для хранения контекстов, таблиц маршрутизации.
- Репликатор – блок, выполняющий репликацию пакетов.
- MAC_TX – трансмиттер интерфейса Ethernet.

При прохождении конвейера обработки данных с пакетом ассоциируется контекст, который далее передается блоками друг другу и модифицируется ими. Таким образом, фактически по тракту перемещается только контекст пакета. Контекст состоит из заголовка пакета и/или сопутствующих метаданных (как стандартных специфицированных, так и заданных разработчиком). Состав контекста, т.е. наборы полей входных и выходных метаданных, для различных блоков отличается. Любая копия пакета обрабатывается как отдельный пакет со своим собственным контекстом.

Блок MAC_RX отвечает за формирование фреймов. Преобразование фрейма включает в себя конвертацию битов в данные пакета посредством группировки битов в логические единицы (фреймы, ячейки и пакеты). В зависимости от используемого протокола логические единицы определяются как фреймы, ячейки и пакеты. Блок MAC_TX является зеркальным отображением блока MAC_RX, то есть выполняет все функции блока MAC_RX с точностью до наоборот. Он отвечает за «расщепление» фрейма на биты и дальнейшую передачу последовательностей битов на физический интерфейс. Блоки InFIFO/OutFIFO представляют собой логические устройства, осуществляющие буферизацию пакетов, помещая их во входную и выходную очереди соответственно.

Конвейер обработки данных состоит из семи функционально специализированных ячеек. Все ячейки можно разбить на три типа: Parse, Lookup и Resolve. Каждый из этих типов ячеек предназначен для определенных действий:

1. Тип ячейки Parse – анализ заголовка пакета, необходимого для дальнейшей классификации пакета, запись данных в контекст, передача контекста в следующую ячейку.
2. Тип ячейки Lookup – классификация пакета по данным из контекста (поиск в одной из таблиц), запись результата поиска в контекст пакета и передача в следующую ячейку.
3. Тип ячейки Resolve – в случае успешного поиска выполнение преобразования контекста в соответствии с записью таблицы и передача контекста в следующую ячейку, если обработка еще не завершена, либо обработчику очередей, если обработка завершена и указан(ы) порт(ы) для отправки. В случае неудачного поиска выполняется действие по умолчанию (например, отправка на порт контроллера).

Блок Memory-Management Unit (MMU) отвечает за размещение тела пакета в основную память. Являясь посредником между конвейером и основной памятью, блок MMU соединен с некоторыми ячейками конвейера, однако для каждой такой ячейки доступен не весь интерфейс блока MMU. Этот интерфейс состоит из следующих действий: выделение буфера для тела пакета, получение указателя на этот

буфер с целью записи или чтения, размещение тела пакета в буфере, освобождение буфера. Ниже приведены действия, выполнимые MMU для каждой ячейки:

- Типу ячейки Parse доступны операции:
  - выделение буфера для тела пакета;
  - получение указателя на буфер для записи;
  - запись – размещение тела пакета в буфере;
  - освобождение буфера;
- Типы ячейки Lookup не имеют доступа к основной памяти.
- Типы ячейки Resolve не имеют доступа к основной памяти.
- Репликатору доступны операции:
  - получения указателя на буфер для чтения;
  - считывание пакета из буфера;
  - освобождение буфера.

Получив контекст пакета, предназначенный для отправки на порт, репликатор анализирует его. Репликатор считывает весь пакет из основной памяти и освобождает память из-под считанного пакета. Если необходимо копирование пакета, например, для широковещательной или мультивещательной рассылки, создается указанное количество копий с соответствующими изменениями заголовка. Вся необходимая репликатору информация содержится в контексте пакета. Результирующие пакеты передаются выходному буферизирующему устройству.

4. Описание модели

4.1. Описание имитационной модели

Предлагаемая имитационная модель работы СПУ выполняет следующие действия:

- передача поступившего пакета сетевому процессору;
- запись поступившего пакета в локальную память;
- обработка заголовка пакета;
- в зависимости от результата обработки принятие решения о дальнейшей отправке пакета на соответствующие порты коммутатора;
- проверка работы сетевого процессора.

Рассматриваемая модель состоит из трех компонентов:

1. поставщика (provider);

2. проверяемого устройства (DUT, Device Under Test) – СПУ;

3. потребителя (consumer).

Все три компонента модели связаны между собой каналами, организованными по принципу FIFO (см. Рисунок 9).

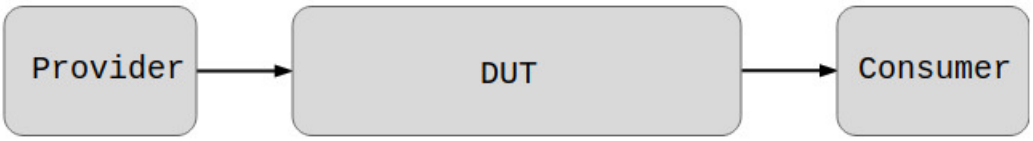

Рис. 9. Основные компоненты модели сетевого процессора

Fig. 9. Main components of the network processor model

Поставщик передает пакеты на обработку сетевому процессору, имитируя поступление пакетов на порты коммутатора. Поставщик каждый такт с начала симуляции поставляет пакет и номер порта коммутатора, на который поступил пакет, и отправляет его СПУ. СПУ, получив пакет и номер порта, начинает его обработку. Потребитель, в свою очередь, ответственен за проверку результатов работы СПУ. Потребитель связан каналами с каждым из этапов обработки пакета, по которым передается информация о том, на каком этапе завершилась обработка пакета.

4.2. Реализация модели

Имитационная модель сетевого процессора реализована на языках C/C++ с использованием открытой C++ библиотеки SystemC [9]. Базовым элементом в SystemC является модуль, включающий в себя процессы и другие модули. Модель в SystemC состоит из нескольких модулей, которые общаются друг с другом через порты. В нашем случае модель состоит из трех основных модулей (блоков): Provider, DUT и Consumer. Каждый блок оснащен портами, с помощью которых происходит взаимодействие между блоками. В построенной модели порты каждого блока делятся на два типа: порты, по которым блок получает данные, и порты, через которые блок отсылает данные. Коммуникация между блоками осуществляется с помощью библиотеки SystemC TLM. Блоки соединены друг с другом однонаправленными каналами из библиотеки TLM.

Блок Provider посылает блоку DUT каждый такт сообщение, содержащее пакет и номер порта, на который якобы пришел пакет. Блок DUT начинает свою работу, как только сообщение поступило на его входящий порт. Блок Consumer аналогично блоку DUT начинает работу, как только получает пакет на входящие порты. Во время моделирования процессы, соответствующие блокам, выполняются асинхронно по отношению друг к другу. В представленной модели процессы прерываются только в двух случаях: задержки, моделирующей время работы отдельного блока, и задержки, связанной с ожиданием поступления сообщения на входные порты блока.

Типы ячеек конвейера и остальные объекты представлены в виде C++ классов, экземпляры которых, в свою очередь, являются полями класса DUT.

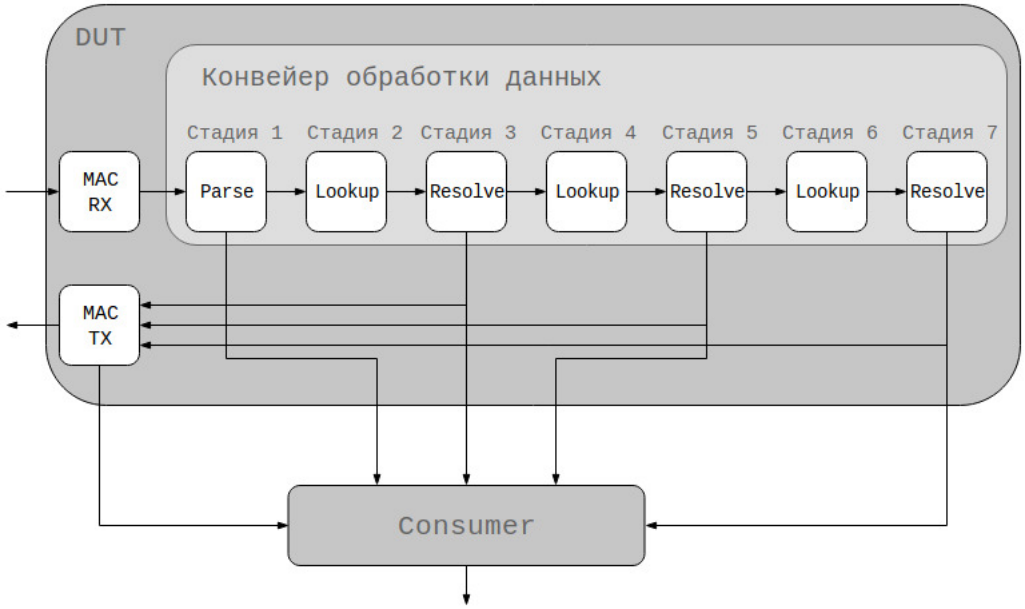

Рис. 10. Схема каналов, по которым передаются сообщения о сбросе пакета

Fig. 10. Notification channels between consumer and NPU units in implemented simulation model

На Рисунке 10 представлены внутренние связи блока DUT, а также связи между блокоми DUT и Consumer. Связи между блокоми MMU и DUT опущены.

В модели используются динамические библиотеки, содержащие реализацию всех ячеек типов Parse, Lookup и Resolve для каждого сценария. При запуске модели указывается, какой сценарий будет промоделирован. Выбор сценария осуществляется выбором реализаций ячеек типов Parse, Lookup и Resolve из динамической библиотеки. Также во время запуска в качестве опций задаются следующие параметры: период тактового сигнала, количество тактов, в течение которого происходит моделирование, набор задержек для каждой ячейки. Если данные параметры при запуске не указываются, то при моделировании используются значения этих параметров по умолчанию.

4.3. Реализация сценариев обработки пакетов

Сценарии обработки пакетов, представленные в разделе 2., реализованы на языке C. Код для каждого сценария состоит из функций обработки пакета, соответствующих ячейкам конвейера и реализованных с использованием следующего интерфейса (API) модели СПУ:

- библиотека функций и структур для работы с памятью (такие, как memcpy или memcmp);
- библиотека функций, имитирующих получение и передачу пакетов с физического интерфейса на коммутатор;

- библиотека функций, вычисляющих контрольную сумму L3-заголовка и хэш-функций для адресов;
- библиотека контекстов;
- функции, обрабатывающие результаты поиска и принимающие решение о дальнейших действиях, таких, как
  - удаление, добавление или замена тега;
  - модификация полей пакета;
  - копирование пакета;
  - размещение пакета в буфер коммутатора;
  - отправка контекста на следующую ячейку конвейера;
  - сброс пакета.

5. Оценка производительности

Данный раздел содержит результаты оценки производительности рассматриваемой архитектуры СПУ.

5.1. Экспериментальный трафик

При проведении тестовых прогонов учитывался опыт нагрузочного и функционального тестирования, проводимого в ПАО Ростелеком.

Распределение трафика при тестировании:

1. Трафик групповой адресации (сценарий Multicast): 400 потоков (с разной пропускной способностью, то есть с разным качеством) давали 1.5 Гбит/с.
2. B2C трафик (сценарий B2C-AR B2C-DR): 100 потоков при идентификации их по внешнему VLAN давали 3.5 Гбит/с (реально порядка 10000 потоков).
3. B2B трафик (сценарий P2P): 50 потоков при идентификации их по внешнему VLAN потоков давали 4 Гбит/с (реально порядка 2500 потоков).
4. B2B трафик (сценарий MP): 1000 потоков с разными MAC давали 1 Гбит/с.
5. Inband трафик (сценарий INBAND): до 1000 пакетов в секунду на сети из 5 коммутаторов (до 200 пакетов/с – обмен между контроллером и CPU коммутатора).
6. LLDP трафик (сценарий LLDP): на каждый интерфейс каждого коммутатора по 1 пакету в секунду (тоже своего рода обмен между контроллером и коммутаторами).

Кроме того, в ходе данного исследования был адаптирован и использован набор программных средств, разработанных компанией KM211. Также рассматривалась возможность использования семейства микроконтроллеров KMX32 (спроектированных компанией KM211).

5.2. Методика получения оценок

Для измерения времени работы процессора с применением процессоров семейства KMX32 применялась следующая методика:

1. Измерение времени работы функций, вызываемых на каждой из ячеек, для каждого теста сценария на эмуляторе процессора KMX32.

2. Запуск имитационной модели сетевого процессора с каждым из тестов. В качестве исполняемых ячейками функций на вход модели подаются адреса из динамической библиотеки функций сценария. В качестве задержек работы каждой из ячеек указываются значения, полученные на шаге 1.

3. Измерение итогового времени работы на каждой из ячеек с учётом моделируемых задержек.

Измерение пессимистичной оценки пропускной способности конвейера представленной архитектуры производилось для пакетов размером 80 байт (в худшем случае). В каждом сценарии выделялось узкое место, т.е. ячейка конвейера, требующая наибольшего времени для обработки контекста. Далее подсчитывалось число пакетов, обработанных за одну секунду, и умножалось на размер пакета (в худшем случае). Полученное значение пропускной способности для каждого сценария является искомой средней пропускной способностью.

5.3. Результаты оценок

В данном разделе представлены результаты оценки производительности, проведенной по вышеописанной методике, для каждого сценария (см. Таблица 1 – Таблица 8), описанного в разделе 2.

Таблица 1. Время обработки пакета для сценария: a) B2C-DR, b) Mirror

Table 1. Packet processing time for a) B2C-DR and b) Mirror scenarios

a)

Test	**RX**	**P**	**L**	**R**	**TX**
1	6495	311	175	94	
2	6549	313	696	1388	2540
3	6551	309	1932	3922	7255
4	6547	309	1445	1388	2539
5	6547	309	1172	1388	2539
6	5945	278	1335	94	

b)

Test	**RX**	**P**	**L**	**R**	**TX**
1	6146	195	353	3043	1794
2	6488	1583	298	4453	1874
3	6858	3195	243	5891	1954
4	6148	192	253	51	

Для сценариев, представленных в Таблицах 3 – 8, в некоторых тестах происходит отправка нескольких пакетов.

В Таблицах 1 – 2 столбцы соответствуют ячейкам конвейера обработки данных, ресиверу (блок MAC_RX) и трансмиттеру (MAC_TX), а строки – тестам, запущенным на предложенной модели конвейера.

Таблица 2. Время обработки пакета для сценария Multicast

Table 2. Packet processing time for Multicast scenario

Test	**RX**	**P**	**L1**	**R1**	**L2**	**R2**	**TX**
1	6165	261	523	110	208	5895	3530
2	6219	231	259	1340			1778
3	6251	263	336	55			

Таблица 3. Время обработки пакета для сценария Inband

Table 3. Packet processing time for Inband scenario

Test	**RX**	**P**	**L**	**R**	**TX**
1	3687.2	480.8	1328	846	1194
2	6281.8	474	1292	1310	1834
3	6275	474	1160	48	
4					
packet 1	3739.2	3.8	1658	846	1194
packet 2	3694	448	1997	847	1193

Таблица 4. Время обработки пакета для сценария B2C-AR

Table 4. Packet processing time for B2C-AR scenario

Test	**RX**	**P**	**L**	**R**	**TX**
Failover					
packet 1	6495	224	358	1369	2546
packet 2	6549	196	358	1368	2540
packet 3	6500	200	451	1368	2540
packet 4	6496	200	544	1368	2540
No match					
packet 1	6504	226	146	48	
packet 2	6551	226	146	48	

В полях таблиц представлено время, затраченное блоком, соответствующим названию столбца, на обработку одного пакета при выполнении теста, указанного в строке. Результаты представлены в наносекундах. Пустые поля в таблице свидетельствуют о том, что соответствующий блок не был задействован при обработке пакета.

На основе результатов экспериментов, представленных в Таблицах 1 – 8, для каждого сценария был произведен расчёт пессимистичной оценки пропускной способности СПУ. Таблица 9 содержит результаты для различных сценариев и вариантов построения СПУ.

Таблица 5. Время обработки пакета для сценария P2P

Table 5. Packet processing time for P2P scenario

Test	**RX**	**P**	**L**	**R**	**TX**
Failover					
packet 1	6495	261	722	1466	2521
packet 2	6549	232	723	1465	2515
packet 3	6500	236	1156	1466	2514
packet 4	6496	237	1589	1466	2514
No match					
packet 1	6504	263	1679	48	
packet 2	6551	262	165	48	
packet 3	6551	262	165	48	

Таблица 6. Время обработки пакета для сценария LACP/LLDP/QoS

Table 6. Packet processing time for LACP/LLDP/QoS scenario

Test	**RX**	**P**	**L**	**R**	**TX**
Tag					
packet 1	6795.2	3221.8	1	1153	
packet 2	6508.8	1644	1	1146	
Ethernet	6221	223	1	1146	
LACP/LLDP					
packet 1	6302	181	1	1289	2515
packet 2	6302	161	1	1289	2514

Таблица 7. Время обработки пакета для сценария LAG

Table 7. Packet processing time for LAG scenario

Test	**RX**	**P**	**L**	**R**	**TX**
1	6197	62	167	48	182
2					
packet 1	6190	68	994	1293	1785
packet 2	6175	62	1036	1294	1784
packet 3	6173	62	1107	1294	1784
packet 4	6170	62	1330	1319	1785

Данные результаты показывают, что в пессимистичном варианте при трафике, состоящем преимущественно из пакетов небольшого размера, такой вариант архитектуры не достигает требуемых показателей. Таким образом, для достижения необходимых показателей нужна доработка имеющихся процессорных ядер.

Таблица 8. Время обработки пакета для сценария L2 switch + VLAN

Table 8. Packet processing time for L2 switch + VLAN scenario

Test	**RX**	**P**	**L1**	**R1**	**L2**	**R2**	**L3**	**R3**	**TX**
1									
packet 1	6209	328	205	188	477	817	936	11926	12830
packet 2	6474	441	193	186	477	843	735	4793	4714
2	6529	437	193	187	476	843	873	6457	8771
3	6235	326	212	188	476	843	880	3293	8570
4	6295	354	205	188	477	843	894	3293	8570
5									
packet 1	6525	439	193	70					
packet 2	6528	441	193	70					
6	6240	358	205	188	476	843	839	1709	4713

Таблица 9. Оценка пропускной способности СПУ для рассмотренных сценариев

Table 9. NPU pipeline throughput for considered scenarios

Сценарии	**1 конвейер × 1 порт, Мбит/с**	**2 конвейера × 1 порт, Мбит/с**	**1 конвейер × 24 порта, Гбит/с**	**2 конвейера × 24 порта, Гбит/с**
B2C-AR	97.7	195.4	2.3	4.7
B2C-DR	97.7	195.4	2.3	4.7
Inband	101.9	203.8	2.4	4.5
LACP/ LLDP/ QoS	94.2	188.4	2.2	4.5
LAG	103.3	206.6	2.5	5
Mirror	93.3	186.6	2.2	4.5
Multicast	102.3	204.6	2.5	4.9
P2P	97.7	195.4	2.3	4.7
L2 swtich VLAN	98	196	2.3	4.7

Заключение

Целью данной работы было изучение различных подходов к разработке архитектуры сетевого процессора и предложение подхода к созданию СПУ. Были изучены архитектуры конвейеров СПУ EZChip NP-5 и Barefoot Tofino. Учитывая их достоинства и недостатки, была предложена архитектура конвейера СПУ, ориентированная на работу со сценариями, описанными в разделе 2. В работе предложена архитектура сетевого процессорного устройства на основе разработанного конвейера. Для тестирования сетевого процессора на наборе сценариев обработки пакета была разработана его имитационная модель на языке C ++. В качестве ядер процессора

в модели использовался процессор KMX32, разработанный компанией KM211. Затем было проведено экспериментальное исследование, результаты которого были использованы для оценки времени выполнения каждой ячейки конвейера.

Результаты исследований показали, что скорость СПУ позволяет применить их в коммутаторах уровня распределения в трехуровневой иерархической сети. Для использования СПУ в коммутаторах ядра сети требуется дальнейшее совершенствование предлагаемой архитектуры. Это возможно достичь двумя способами. Первый из них заключается в усовершенствовании существующих процессорных ядер с точки зрения добавления специализированных программируемых устройств для обработки сетевого трафика. Другой способ – создать такое специализированное программируемое устройство с нуля.

Список литературы / References

[1] Смелянский Р. Л., “Программно-конфигурируемые сети”, *Открытые системы. СУБД*, **9** (2012), 15–26; [Smeliansky R.L., “Software Defined Network”, *Open Systems. DBMS*, **9** (2012), 15–26, (in Russian).]

[2] Смелянский Р. Л., “Технологии реализации программно конфигурируемых сетей: Overlay vs OpenFlow”, *Журнал сетевых решений LAN*, 2014, №4, 53–55; [Smeliansky R.L., “Tekhnologii realizacii programmno konfiguriruemyh setej: Overlay vs OpenFlow”, *Zhurnal setevyh reshenij LAN*, 2014, №4, 53–55, (in Russian).]

[3] Kornaros G., *Multi-core embedded systems*, Boca Raton, FL: CRC Press, 2010.

[4] Cisco Networking Academy, *Connecting Networks Companion Guide*, Cisco Press, 2014.

[5] *EZchip NP-5 Product Brief*, http://www.ezchip.com.

[6] Bosshart P., et al., “P4: Programming protocol-independent packet processors”, *ACM SIGCOMM Computer Communication Review*, **44**:3 (2014), 87–95.

[7] “Tofino: World’s fastest P4-programmable Ethernet switch ASICs”, *Barefoot*, https://barefootnetworks.com/products/brief-tofino/.

[8] Kaushalram A., Budiu M., Kim C., *Data-plane stateful processing units in packet processing pipelines*, US Patent App 14864088, 2017, http://www.freepatentsonline.com/y2017/0093987.html.

[9] Accellera Standarts: SystemC, http://www.accellera.org/downloads/standards/systemc.

[10] “Семейство микроконтроллеров KMX32”, http://km211.ru; [“KMX32 family microcontrollers”, http://km211.com, (in Russian).]

[11] Петров И. С., Смелянский Р. Л., “Минимизация группового трафика и обеспечение его отказоустойчивости в программно-конфигурируемых сетях”, *Известия Российской академии наук. Теория и системы управления*, 2018, №3, 64–75; in English: Petrov I. S., Smeliansky R. L., “Minimization of Multicast Traffic and Ensuring Its Fault Tolerance in Software-Defined Networks”, *Journal of Computer and Systems Sciences International*, **57**:3 (2018), 407–419.

Bezzubtsev S. O., Vasin V. V., Volkanov D. Yu., Zhailauova Sh. R., Miroshnik V. A., Skobtsova Yu .A., Smeliansky R. L., "An Approach to the Construction of a Network Processing Unit", *Modeling and Analysis of Information Systems*, **26**:1 (2019), 39–62.

DOI: 10.18255/1818-1015-2019-1-39-62

Abstract. The paper proposes the architecture and basic requirements for a network processor for OpenFlow switches of software-defined networks. An analysis of the architectures of well-known network processors is presented – NP-5 from EZchip (now Mellanox) and Tofino from Barefoot Networks. The advantages and disadvantages of two different versions of network processor architectures are considered: pipeline-based architecture, the stages of which are represented by a set of general-purpose processor cores, and pipeline-based architecture whose stages correspond to cores specialized for specific packet processing operations. Based on a dedicated set of the most common use case scenarios, a new architecture of the network processor unit (NPU) with functionally specialized pipeline stages was proposed. The article presents a description of the simulation model of the NPU of the proposed architecture. The simulation model of the network processor is implemented in C ++ languages using SystemC, the open-source C++ library. For the functional testing of the obtained NPU model, the described use case scenarios were implemented in C. In order to evaluate the performance of the proposed NPU architecture a set of software products developed by KM211 company and the KMX32 family of microcontrollers were used. Evaluation of NPU performance was made on the basis of a simulation model. Estimates of the processing time of one packet and the average throughput of the NPU model for each scenario are obtained.

Keywords: network processor, network processing unit, switch, computer networks, SDN, computer architecture, simulation modeling, Open Flow protocol

On the authors:
Stanislav O. Bezzubtsev, technical expert, orcid.org/0000-0002-2419-7632
Applied Research Center for Computer Networks,
1, bd. 77 Leninskie Gory, Moscow, 119992 Russia, e-mail: stas.bezzubtsev@gmail.com

Vyacheslav V. Vasin, senior software developer, orcid.org/0000-0002-5759-3105
Applied Research Center for Computer Networks,
1, bd. 77 Leninskie Gory, Moscow, 119992 Russia, e-mail: vvasin@arccn.ru

Dmitry Yu. Volkanov, PhD, assistant professor, orcid.org/0000-0001-9940-5822
Lomonosov Moscow State University,
GSP-1, Leninskie Gory, Moscow, 119991, Russia, e-mail: volkanov@lvk.cs.msu.su

Shynar R. Zhailauova, PhD student, orcid.org/0000-0001-5725-7040
Lomonosov Moscow State University,
GSP-1, Leninskie Gory, Moscow, 119991, Russia, e-mail: szhaylau@cs.msu.ru

Vladislav A. Miroshnik, software developer, orcid.org/0000-0003-0883-0116
Lomonosov Moscow State University,
GSP-1, Leninskie Gory, Moscow, 119991, Russia, e-mail: miroshnikv@lvk.cs.msu.su

Yuliya A. Skobtsova, M.A. student, orcid.org/0000-0001-8351-3191
Lomonosov Moscow State University,
1 Leninskie Gory str., Moscow, Russia, e-mail: xenerizes@lvk.cs.msu.su

Ruslan L. Smeliansky, Corresponding Member of Russian Academy of Sciences, professor, doctor of sciences,
orcid.org/0000-0003-2311-4513
Lomonosov Moscow State University,
GSP-1, Leninskie Gory, Moscow, 119991, Russia, e-mail: smel@cs.msu.su

Acknowledgments:
This work was supported by the Russian Foundation for Basic Research, Grant No 19-07-01076.