Design Principles for Packet Parsers

Glen Gibb[1], George Varghese[2], Mark Horowitz[1], Nick McKeown[1]

Тезисы

Все сетевые устройства должны анализировать заголовки пакетов для выбора способов обработки пакета. Коммутатор Ethernet с 64 портами 10 Гбит/с должен анализировать миллиарды пакетов каждую секунду, извлекая поля, нужные для принятия решений. Хотя анализаторы являются необходимой частью любого коммуникационного устройства, очень мало написано об устройстве анализаторов и сравнении различных вариантов устройства. Что лучше - один быстрый анализатор или несколько медленных? Сколь дорого изменение конфигурации анализатора в «полевых» условия? Как выбор анализатора влияет на размеры элементов и потребляемую мощность?

В этой статье сравниваются варианты синтаксических анализаторов для коммутаторов и маршрутизаторов, а также описан генератор синтаксических анализаторов, создающий файлы Verilog, пригодные для загрузки в устройства. Показано, что (1) анализаторы пакетов занимают 1-2% площади микросхем, а (2) возможность программирования анализаторов лишь удваивает (достаточно малые) размеры анализатора.

1. Введение

Несмотря на разнообразие сетевых устройств, каждое из них проверяет поля в заголовках пакетов для выбора применяемых к пакету действий. Например, маршрутизатор проеряет IP-адрес получателя для определения дальнейшего пути пакета (куда пересылать), а межсетевой экран сравнивает несколько полей со списками контроля доступа (ACL[3]), чтобы решить вопрос об отбрасывании пакета.

Процесс идентификации и извлечения полей заголовка называется синтаксическим анализом (parsing) и является темой данной статьи. Основной тезис работы заключается в том, что синтаксический анализ является наиболее значимым из узких мест (bottleneck) в высокоскоростных сетевых устройствах по причине сложности заголовков в пакетах и методы организации потоковых анализаторов с малыми задержками критически важны для современных высокоскоростных сетевых устройств.

Почему анализ пакетов представляет сложность? Размер и формат пакетов меняется от сети к сети и от пакета к пакету. Базовая структура включает один или множество заголовков, данные (payload) и может также включать трейлер. На каждом этапе инкапсуляции в заголовок включается идентификатор, показывающий тип данных, следующих после заголовка. На рисунке 1 приведен простой пример пакета TCP.

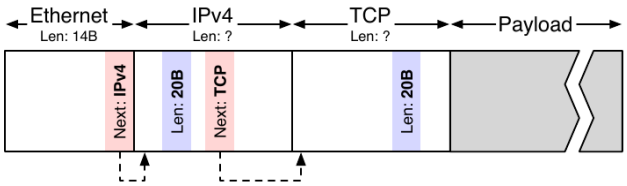


Рисунок 1. Пакет TCP.

На практике пакеты зачастую включают множество заголовков, которые содержат информацию протоколов вышележащих уровней (например, заголовки HTTP) или дополнительные данные, отсутствующие в имеющихся заголовках (например, идентификатор VLAN[4] в кампусной сети или метка MPLS[5] в магистрали Internet). Сегодня наличие множества заголовков в пакете стало нормой.

Для анализа пакета сетевое устройство идентифицирует заголовки по порядку до извлечения и обработки конкретных полей. Анализатор пакетов представляется простым, так как заранее известно, какие типы заголовков можно ожидать. Однако практическая реализация анализаторов связана с целым рядом сложностей.

1. ***Пропускная способность***. Большинство анализаторов должно работать со скоростью линии, поддерживая непрерывный поток следующих один за другим пакетов минимального размера. Канал Ethernet 10 Гбит/с может доставлять новый пакет каждые 70 нсек, а современный контроллер ASIC 64 × 40 Гбит/с в коммутаторе Ethernet должен обрабатывать пакет каждые 270 пксек.
2. ***Последовательность заголовков***. Заголовки обычно включают поле, указывающее следующий заголовок, предлагая их последовательную обработку.
3. ***Неполная информация***. Некоторые заголовки (например, MPLS) не указывают тип следующего заголовка и его приходится выводить по индексу в таблице поиска или умозрительной обработки (угадывания).
4. ***Неоднородность***. Сетевое устройство должно обрабатывать много разных форматов заголовков, появляющихся в произвольном порядке и в разных местах.
5. ***Программируемость***. Формат заголовка может измениться уже после создания анализатора в результате появления нового стандарта или использования сетевым оператором специальных заголовков для идентификации трафика в сети. Например, в последние годы были предложены или внедрены форматы PBB, VxLAN, NVGRE, STT, OTV.

Хотя синтаксический анализатор имеется в каждом сетевом устройстве, посвященных анализаторам работ опубликовано очень мало. Авторам известны лишь две публикации, непосредственно относящиеся к анализу пакетов

[1] *Stanford University.*
[2] *Microsoft Research.*
[3] *Access-control list.*
[4] *Virtual LAN [20] - виртуальная ЛВС.*
[5] *Multiprotocol Label Switching [19] - многопротокольная коммутация по меткам.*

[1, 8] и ни в одной из них не рассматриваются компромиссы между размером на кристалле, скоростью и потребляемой мощностью, при этом в обеих вносится неприемлемая для скоростных приложений задержка. Сопоставление по регулярным выражениям не применимо - анализ обрабатывает часть каждого пакета под управлением графа разбора, тогда как при сопоставлении regex сканируется весь пакет.

Цель этой статьи заключается в обучении разработчиков учету влияния устройства синтаксического анализатора на размеры, скорость и потребляемую мощность в жестко запрограммированных и настраиваемых системах. Не предлагается «идеального» решения, поскольку авторы считают, что это зависит от конкретной задачи.

При разработке анализатора возникает ряд естественных вопросов. Какую долю площади кристалла и потребляемой мощности можно отдать анализатору в вариантах одного или нескольких анализаторов на микросхему? Каким должен быть размер слова при обработке заголовков? Короткие слова требуют более высоких частот, длинные позволяют обработать несколько заголовков в один прием. Насколько программируемым должен быть анализатор: Какая гибкость нужна пользователям для добавления произвольных заголовков с полями в произвольных местах?

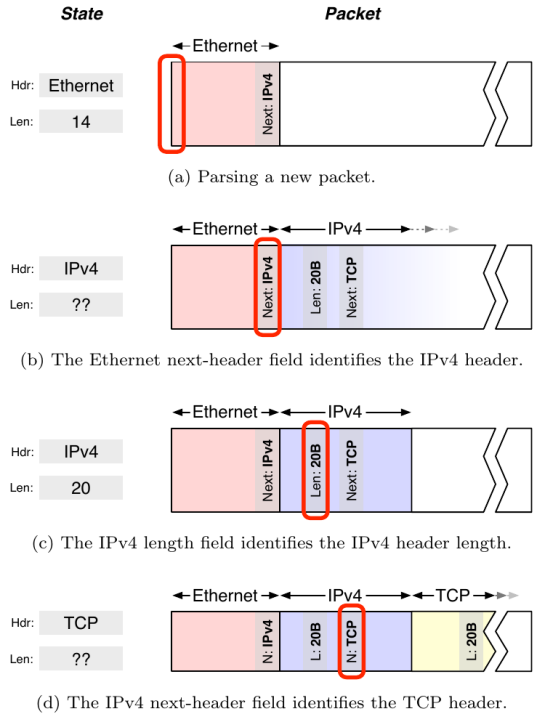
Рассмотрим эти вопросы по порядку. Сначала описывается процесс анализа (2. Синтаксический анализ) и вводится понятие графа анализа для представления последовательности заголовков и описания конечного автомата анализа (3. Граф анализа). Затем рассматривается устройство фиксированных и программируемых анализаторов (4. Устройство анализатора) и детали генерации записей в таблицах программируемого анализатора (5. Генерация таблицы анализа). В заключение представлены принципы устройства анализаторов на примерах (6. Принципы устройства). Варианты устройства были подготовлены с использованием подготовленного инструмента, который на основе заданного графа анализа создает анализатор, параметризованный по ряду характеристик. Было сгенерировано более 500 различных анализаторов на основе библиотеки TSMC 45 nm ASIC. Для сравнения каждый анализатор настраивался на обработку пакетов в Ethernet ASIC 64 × 10 Гбит/с.

В целом эта статья:

- фиксирует проблему устройства потоковых анализаторов пакетов (4.5. Работа в потоковом режиме);
- выделяет сходства и различия с декодированием инструкций в процессорах (4.6. Сравнение с декодированием инструкций);
- описывает новую методологию устройства потоковых анализаторов пакетов для оптимизации прощади, скорости и потребляемой мощности (6.1. Генератор анализаторов); в работе [8], напротив, описана реализация FPGA и теоретическая модель, не дающая представления об основных компромиссах между площадью и потребляемой мощностью в ASIC;
- описывает многочисленные результаты, включая коммутаторы 64 × 10 Гбит/с, являющиеся важными компонентами современных сетей, и указывает принципы построения таких коммутаторов (6.2. Фиксированный анализатор, 6.3. Программируемый анализатор);
- показывает, что программируемый анализатор занимает 1–2% площади (6.3. Программируемый анализатор);
- показывает увеличение стоимости программируемого анализатора вдвое (6.3. Программируемый анализатор).

2. Синтаксический анализ

Синтаксическим анализом называют процесс идентификации и извлечения полей для обработки на последующих этапах. Анализ по своей природе является последовательным - каждый заголовок идентируется предшествующим, что требует последовательного распознавания. Отдельный заголовок не содержит достаточной информации для однозначного определения его типа. Поля следующих заголовков показаны на рисунке 1 для Ethernet и IP - заголовок Ethernet указывает, что за ним следует заголовок IP.



(a) Parsing a new packet.

(b) The Ethernet next-header field identifies the IPv4 header.

(c) The IPv4 length field identifies the IPv4 header length.

(d) The IPv4 next-header field identifies the TCP header.

Рисунок 2. Процесс анализа.

Процесс анализа показан на рисунке 2. Большой прямоугольник обозначает анализируемый пакет, а меньшие прямоугольники с закруглением - место текущей обработки. Состояние анализатора отслеживает тип и размер текущего заголовка.

Обработка начинается с «головы» пакета (2a). Исходный тип пакета обычно не меняется для данной сети и известен анализатору (например, Ethernet). Анализатор знает структуру каждого типа заголовков, что позволяет ему найти поле (поля), указывающие размер данного заголовка и тип следующего.

Анализатор читает поле следующего заголовка как IPv4 (2b). Размер заголовка IPv4 не является постоянным - поле размера включено в заголовок, но изначально неизвестно анализатору.

Считывается размер заголовка IPv4 и состояние анализатора соответствующим образом обновляется (2c). Размер указывает положение следующего заголовка и должен быть определен до начала работы с тем.

Процесс повторяется, пока не будут обработаны все заголовки. Извлечение полей выполняется в параллель с идентификацией полей и на рисунке не указано для простоты.

3. Граф анализа

Граф анализа указывает последовательность заголовков, распознаваемых коммутатором или присутствующих в сети. Графы анализа являются ориентированными ациклическими графами, где вершины представляют типы заголовков, а ребра задают порядок. Примеры графов анализа представлены на рисунке 3.

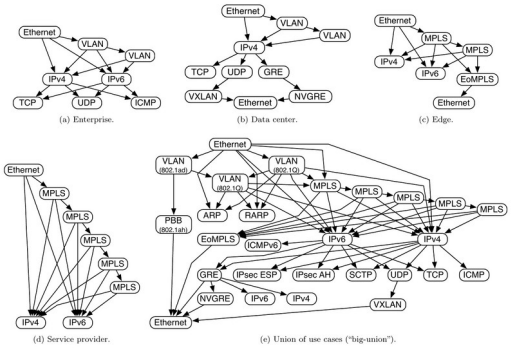


Рисунок 3. Примеры графов анализа.

Граф анализа является конечным автоматом (машиной состояний), который последовательно распознает заголовки в пакете. Начиная с корневого узла переходы (смена) состояний выполняются в соответствии со значениями поля следующего заголовка (next-header). Путь через граф соответствует последовательности заголовков.

Граф анализа (и конечный автомат) в анализаторе может быть фиксированным (жестко заданным) или программируемым. Фиксированный граф создается в процессе разработки и не может быть изменен после изготовления, тогда как программируемый граф анализа задается и может изменяться в процессе работы.

Традиционные анализаторы используют фиксированный граф. Для поддержки различных возможных вариантов реальный граф устройства представляет собой объединение (union) графов для разных возможных случаев. На рисунке 3e показан пример графа анализа в коммутаторе - это объединение графов для отдельых случаев, включая показанные на рисунках 3a-d. Объединение включает 28 узлов и 677 путей. Данное объединение в статье называется big-union.

4. Устройство анализатора

В этом разделе описано базовое устройство анализатора. Сначала рассматривается модель абстрактного анализатора, описываются фиксированные и программируемые анализаторы. Затем более подробно рассмотрены требования и приведено сравнение с декодированием инструкций в процессорах.

4.1. Модель абстрактного анализатора

Напомним, что анализатор идентифицирует и извлекает поля из пакета. Извлеченные поля служат для поиска в таблице пересылки на последующих этапах обработки в коммутаторе. Все входные поля для поиска должны быть доступны до начала поиска. Поля извлекаются по мере обработки заголовков, что требует их буферизации[1] до того, как станут доступны все требуемые для поиска поля.

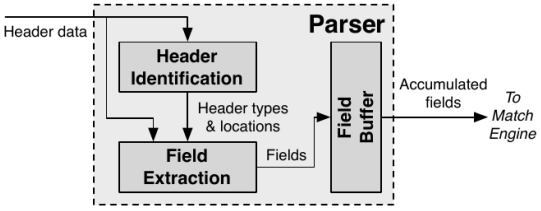


Рисунок 4. Абстрактный анализатор.

На рисунке 4 представлена абстрактная модель анализатора, включающая модули идентификации заголовка (header identification), извлечения полей (field extraction) и буферизации (field buffer). Данные заголовка попадают в анализатор и передаются модулям идентификации заголовков и извлечения полей. Модуль идентификации распознает заголовки и информирует модуль извлечения о типе и местоположении заголовка. Модуль извлечения считывает поля и

[1] *Неочевидное требование. Пока поля не подвергаются редактированию, можно хранить указатели на них в метаданных пакета. Это позволит сэкономить на операциях копирования и обеспечит экономию памяти. При этом можно использовать «двойные указатели - основной на буфер всего пакета и смещения от начала буфера для полей. Прим. перев.*

отправляет их в модуль буферизации. Этот модуль собирает извлеченные поля, передавая их последующим этапам обработки после извлечения всех нужных полей.

Идентификация заголовка
Модуль идентификации заголовков реализует конечный автомат графа анализа (3. Граф анализа). Алгоритм 1 показывает процесс анализа, определяющий тип и размещение каждого заголовка.

Алгоритм 1. Определение типа и размера заголовка
```
procedure IdentifyHeaders(pkt)
       hdr = initialType
       pos = 0
       while hdr ≠ DONE do
              NotifyFieldExtraction(hdr, pos)
              len = GetHdrLen(pkt, hdr, pos)
              hdr = GetNextHdrType(pkt, hdr, pos)
              pos = pos + len
       end while
end procedure
```

Извлечение поля
Процесс извлечения поля прост и выполняется в соответствии с типом и местоположением поля, представленным модулю, а также списком полей для каждого типа заголовков. Алгоритм 2 показывает этот процесс.

Алгоритм 2. Извлечение полей
```
procedure ExtractFields(pkt, hdr, hdrPos)
       fields = GetFieldList(hdr)
       for (fieldPos, fieldLen) = fields do
              Extract(pkt, hdrPos + fieldPos, fieldLen)
       end for
end procedure
```
Извлечение полей может выполняться параллельно идентификации заголовков - извлечение происходит из найденных заголовков, а идентификация выполняется последовательно. Извлечение отдельных полей после определения типа и положения заголовка также может происходить параллельно.

Буфер полей
Буфер полей аккумулирует извлеченные значения до выполнения поиска в таблице коммутатора. Извлеченные поля выводятся буфером как «широкий» вектор битов, поскольку поиск в таблице выполняется одновременно по всем полям. Вывод в форме вектора битов требует реализации буфера в форме широкого массива регистров. На каждый регистр требуется один мультиплексор (MUX) для выбора полей, возвращенных блоком извлечения.

4.2. Фиксированный анализатор
Фиксированный анализатор работает с одним графом анализа, выбранным при разработке и представляющим собой объединение графов для каждого варианта, который коммутатор должен поддерживать. Логика в фиксированном графе оптимизирована для выбранного графа анализа.

Описанное здесь устройство анализатора не отражает в точности анализаторы реальных устройств, но обесепчивает качественное представление. Как показано в параграфе 6.2. Фиксированный анализатор, размер фиксированного анализатора на кристалле существенно меньше размера буфера полей. Размер буфера определяется графом анализа и для параллельной передачи всех извлеченных полей в модули сопоставления буфер должен включать массив регистров и мультиплексоров. Отсутствие гибкости в устройстве буфера предполагает, что его размер не зависит существенно от устройства анализатора.

4.2.1. Идентификация заголовка
Модуль идентификации заголовков, показанный на рисунке 5, состоит из четырех элементов: конечный автомат (state machine), буфер, процессоры заголовков и элемент упорядочивания (sequence resolution).

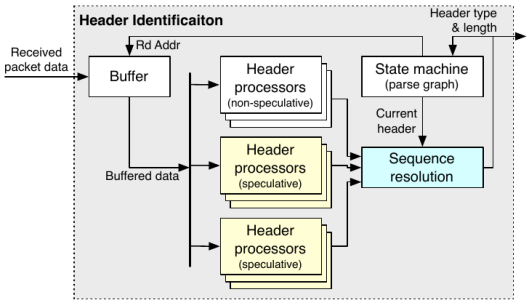


Рисунок 5. Модуль идентификации заголовка.

Конечный автомат реализует выбранный граф анализа, а буфер сохраняет принятый пакет до идентификации. Имеется один или несколько процессоров заголовков (для каждого поддерживаемого анализатором типа), читающих поля размера и следующего заголовка для данного типа заголовка, указывающие размер заголовка и тип следующего.

Блок идентификации заголовков распознает один заголовок за цикл и содержит один процессор, специфичный для заголовка, на каждый распознаваемый тип, а элемент упорядочивания является простым мультиплексором (MUX). Мультиплексор выбирает выход процессора, соответствующий данному типу заголовка. Перехода конечного автомата определяются выходом MUX.

За счет наличия нескольких экземпляров некоторых процессоров можно идентифицировать несколько заголовков в одном цикле. Каждый уникальный экземпляр процессора заголовков обрабатывает данные со своим смещением в буфере. Например, тег VLAN имеет размер 4 байта, а наличие двух процессоров VLAN позволяет обработать 2 заголовка VLAN в одном цикле. Первый процессор VLAN обрабатывает данные со смещением 0, второй - со смещением 4.

В начале цикла анализатор знает лишь тип заголовка со смещением 0, а обработка остальных является умозрительной. В конце цикла анализа элемент упорядочивания определяет, из какого «умозрительного» процессора использовать результаты. Вывод из процессора со смещением 0 указывает использование первого «умозрительного» процессора, а вывод этого процессора указывает второй «умозрительный» процессор и т. д.

4.2.2. Извлечение полей

Модуль извлечения полей показан на рисунке 6. буфер хранит данные в ожидании идентификации заголовка. Поля, извлекаемые для каждого типа заголовка, хранятся в табице. Простой конечный автомат управляет процессом извлечения - ожидание идентификации заголовка, поиск идентифицированного типа в таблице извлечения и извлечение нужных полей из пакета.

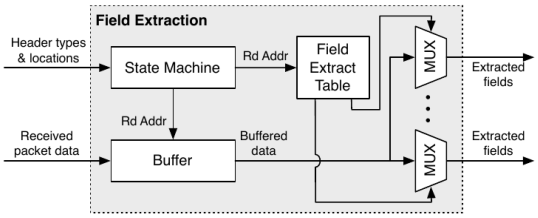


Рисунок 6. Модуль извлечения поля.

4.3. Программируемый анализатор

Программируемый анализатор использует граф анализа, задаваемый в процессе работы. Здесь выбран подход на основе таблицы состояний для простоты понимания и реализации. Таблицы состояний легко реализуются в RAM и/или TCAM. Адресуемая по содержимому память (CAM[1]) - это ассоциативная память, оптимизированная для поиска, что позволяет выполнять поиск по всей памяти параллеотно для каждого входного значения. В двоичной CAM сопоставляется каждый бит, а в троичной (TCAM) не имеющие значения (don't care) биты пропускаются при сравнении.

Представленную выше абстрактную модель анализатора легко изменить для включения программируемых элементов состояния, как показано на рисунке 7. Основным отличием является добавление блоков TCAM и RAM. В TCAM хранятся последовательности битов для идентификации заголовков, а в RAM - информаци о следующем состоянии, биты для извлечения и прочие данные, нужные при анализе. Модули идентификации заголовков и извлечения полей больше не содержат жесткой логики для определенных заголовков, используя взамен эти блоки памяти.

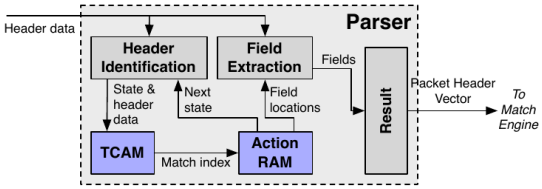


Рисунок 7. Программируемый анализатор.

Модуль идентификации заголовков упрощен по сравнению с фиксированным анализатором, как можно видеть из рисунка 8. Он содержит логику отслеживания состояний и буфер, а вся логика, относящаяся к заголовкам, перенесена в TCAM и RAM. Текущее состояние и часть буферов передаются в TCAM, откуда возвращается первая совпадающая запись. Передаваемые в TCAM байты могут быть непрерывным блоком данных или разрозненными элементами из разных мест, а общий размер может составлять от 1 байта до всего пакета. Запись RAM, соответствующая найденной в TCAM записи, считывается и указывает следующее состояние для модуля идентификации заголовков, а также совпадающие заголовки. Запись RAM может также задавать такие данные как число байтов для перемещения (к следующему заголовку) и подмножество байтов для следующего извлечения.

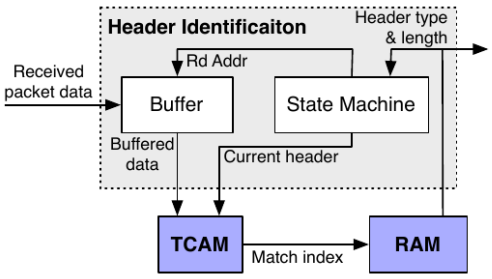


Рисунок 8. Идентификация заголовка.

Модуль извлечения полей почти не отличается от фиксированного анализатора, лишь таблица с местоположениями полей перенесена из модуля в RAM.

4.4. Требования производительности

Анализатор должен работать со скоростью линии для худших вариантов картины трафика в приложениях (например, коммутатор Ethernet). Невозможность анализа со скоростью линии приведет к потере пакетов при переполнении входного буфера.

[1]*Content-addressable memory.*

Один экземпляр анализатора может оказаться неспособным работать со скоростю линии - например при тактовой частоте 1 ГГц в коммутаторе 64 × 10 Гбит/с коммутатор должен обрабатывать в среднем 640 битов за один такт. Параллельная работа нескольких экземпляров анализатора позволяет обеспечить требуемую производительность, если каждый экземпляр анализатора обрабатывает свой пакет.

Для многих современных коммутаторов предъявляются очень высокие требования в части сквозной задержки и все элементы коммутатора, включая синтаксический анализатор, должны работать с минимальными задержками. Это предполагает анализ пакетов в процессе приема («на лету») вместо буферизации всего пакета (или заголовков) до начала анализа.

4.5. Работа в потоковом режиме

Анализаторы можно разделить на потоковые (streaming) и прочие (non-streaming). Не поддерживающий потоки анализатор получает весь комплект заголовков до начала анализа, тогда как потоковый ведет анализ непосредственно в процессе приема пакета. Примером анализатора без поточной обработки может служить Kangaroo [8].

Не работающие с потокм анализаторы вносят задержку, связанную с ожиданием приема всей цепочки заголовков, что делает их непригодными для высокоскоростных систем с малыми задержками. Например, буферизация 125 байтов заголовков при скорости 1 Гбит/с сносит задержку в 1 мксек, что уже является проблемой для приложений ЦОД. Преимуществом анализаторов с буферизацией является возможность работать в процессе анализа с любой частью заголовков, что существенно упрощает процесс.

Потоковые анализаторы снижают задержку за счет обработки заголовков на лету, однако доступ к полям ограничен небольшим окном свежепринятых данных. Разработчик должен обеспечить анализ каждого заголовка, прежде чем он уйдет за пределы окна.

Фиксированные и программируемые анализаторы могут работать в потоковом и буферизуемом режиме. Далее рассматриваются потоковые реализации.

4.6. Сравнение с декодированием инструкций

Анализ пакетов похож на декодирование инструкций в современных процессорах CISC[1] [15], когда каждая инструкция CISC преобразуется в одну или множество микроопераций в стиле RISC[2] (μops). Оба эти процесса являются двухэтапными с последовательной и распараллеливаемой (non-serial) фазами. Фазами анализа являются идентификация заголовков и извлечение полей, фазами декодирования инструкций - определение размера (ILD[3]) и декодирование (ID[4]) инструкции. Система кодирования инструкций проще заголовков и не требуется декодировать размер, что позволяет использовать одну операцию определения размера для любой инструкции. Процесс ILD является последовательным, размер инструкции определяет начало следующей. ID определяет тип каждой инструкции, извлекает поля (операнды) и выдает соответствующую последовательность μops. Множество инструкций может обрабатываться параллельно, поскольку их стартовые позиции известны.

Несмотря на сходство, процессы анализа заголовков и декодирования инструкций имеют существенные различия. Типы заголовков неоднородны и форматы их значительно разнообразней форматов инструкций. Тип заголовка задается предшествующим заголовком, а не кодируется в нем самом.

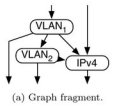
5. Генерация таблицы анализа

Рассмотрение программируемых анализаторов не будет полным без обсуждения генерации записей таблиц анализа. В этом параграфе описаны эти записи и представлен алгоритм и эвристика минимизации числа таких записей.

5.1. Структура таблицы

Таблица анализа является таблицей переходов состояний, т. е. определяет смены состояний конечного автомата. Таблица содеррит таблец текущего состояния (state), входные значения, следующее состояние и выходные значения. Строки таблицы задают переход к следующему состоянию на основе текущего и комбинации входных значений. Для каждого действительного (возможного) состояния и комбинации входных значений в таблице имеется одна строка.

Таблица содержит столбцы для текущего заголовка (current header), его размера (current header length), искомых значений (lookup value), типа следующего заголовка (next header), и смещения поиска в следующем цикле. Каждое ребро графа анализа задает смену состояния, поэтому каждому ребру должна соответствовать запись в таблице.На рисунке 9a показан граф анализа, а на рисунке 9b - соответствующие записи таблицы.



(a) Graph fragment.

Curr. Hdr.	Lookup Val.	Next Hdr.	Hdr. Len	Next lookup offset
$VLAN_1$	0x0800	IPv4	4	0
$VLAN_1$	0x8100	$VLAN_2$	4	2
$VLAN_2$	0x0800	IPv4	4	0

(b) Corresponding parse table entries.

Рисунок 9. Записи таблицы анализа.

Вместо использования всего пакета в качестве входных данных для таблицы анализа достаточно полей, указывающих размер заголовка и тип следующего заголовка, что существенно снижает размер таблицы. Например, для заголовка IPv4 достаточно 4 байтов со смещением 20). Смещение указывает поля, служащие входными данными для таблицы анализа.

5.2. Генерация таблиц

Число записей в таблице можно снизить путем кодирования в одной записи нескольких переходов и, следовательно, идентификации множества заголовков. Фрагмент графа с рисунка 9a можно представить иначе (рисунок 10a) и новая

[1] *Complex Instruction Set Computing - архитектура процессора с полным набором команд.*
[2] *Restricted (Reduced) Instruction Set Computing - архитектура процессора с ограниченным (упрощенным) набором команд.*
[3] *Instruction length decode - декодирование размера инстукции (команды).*
[4] *Instruction decode - декодирование инструкции.*

таблица будет иметь на 1 запись меньше. Снижение числа записей в таблице обеспечивает преимущество, поскольку от размера таблицы зависит занимаемая на кристалле площадь (6.3. Программируемый анализатор).

Объединение нескольких переходов в одну запись таблицы можно рассматривать как кластеризацию или слияние узлов. Кластер, соответствующий первой записи на рисунке 10a, показан на рисунке 10b.

Алгоритм генерации таблиц

Кластеризация графов является сложной задачей сетевой обработки [5, p. 209]Ю для решения которой имеется множество приближений [3, 4, 7], но они плохо подходят. Kozanitis с соавторами [8] представили алгоритм динамического программирования для снижения числа записей в таблицах Kangaroo (анализатор с промежуточной буферизацией), но этот алгоритм не подходит для потокового анализа «на лету». Алгоритм предполагает возможность доступа к данным в любой части заголовка, но потоковая модель обеспечивает возможность работы лишь с небольшим окном данных.

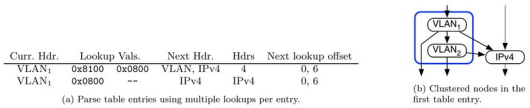


Рисунок 10. Кластер узлов.

Авторы разработали на основе алгоритма Kangaroo свой вариант, который подходит для потокового анализа. Входными данными алгоритма служат ориентированный ациклический граф G = (V, E), максимальное число искомых згачений (k), требуемая скорость (B) в битах за цикл и размер окна w. Алгоритм кластеризует узлы G так, что (1) для каждого кластера требуется не более k операций поиска, (2) выполняемых в окне w, (3) все пути требуют в среднем обработки не менее B битов на цикл и (4) число кластеров минимизировано. Использующий динамическое программирование алгоритм представлен уравнением (1).

$$OPT(n,b,o) = \min_{c \in Clusters(n,o)} \left(Entries(c) + \sum_{j \in Successor(c)} OPT(j, B + (b - W(c,j)), NewOffset(c,j,o)) \right) \quad (1)$$

Функция OPT(n, b, o) возвращает число записей таблицы, требуемых для субграфа от узла n в окне со смещением o и требуемой скоростью обработки b. Clusters(n, o) указывает все действительные кластеры, начинающиеся с узла n в окне со смещением o. Смещение окна определяет число заголовков после n, попадающих в это окно. Clusters использует число операций поиска k и размер окна w для ограничения размера кластеров. Функция Entries(c) возвращает число записей таблицы, требуемых для кластера c. Successor(c) указывает все узлы, достижимые из кластера c через одно ребро. На рисунке 11a показан кластер и соответствующие преемники (successor), а на рисунке 11b показано влияние окна w на формирование кластера.

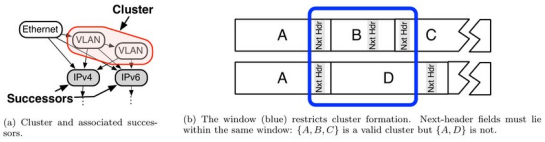


Рисунок 11. Формирование кластера.

Рекурсивные вызовы OPT указывают число записей таблицы, требуемых для каждого узла-преемника. Значения b и o требуется обновлять. Обновленное значение b отражает число битов, потребляемых в цикле анализа, где обрабатывается c. Обновленное смещение o отражает объем данных, которые были получены и потреблены в процессе обработки c.

Этот алгоритм эквивалентен Kangaroo, если в качестве w задан максимальный размер последовательности заголовков. В этом случае алгоритм может использовать любой байт из области заголовков в течение любого цикла обработки.

Улучшенная генерация таблиц

Описанный выше алгоритм (и Kangaroo) корректно идентифицирует минимальный набор записей таблицы лишь для графа анализа, являющегося деревом. Алгоритм независимо обрабатывает каждый субграф, что ведет к генерации разных наборов записей для некоторых перекрывающихся секций субграфов. На рисунке 12a показан фрагмент графа анализа, в котором субграфы из узлов C и K имеют совпадающие узлы F и G. Генерация разных наборов кластеров для перекрывающихся узлов ведет к ненужному росту числа записей. На рисунке 12b показан другой вариант кластеризации где для области наложения генерируется общий кластер.

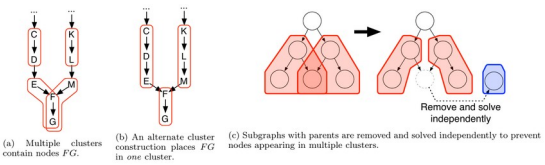


Рисунок 12. Улучшение кластера.

Неоптимальные решения возникают лишь в случаях, когда к узлу ведет множество входных ребер. Эвристический подход для улучшения удаляет субграф S со множеством входных ребер из графа G и находит S независимо. Затем находится решение для графа G − S и результаты объединяются. Объединенное решение сравнивается с предыдущим и сохраняется, если ребер в нем меньше. Сравнение должно выполняться, поскольку иногда число ребер может возрастать. Процесс повторяется для каждого субграфа с множеством входных ребер. На рисунке 12c показан этот процесс для одного субграфа.

6. Принципы устройства

Разработчиков интересует ряд вопросов, связанных с синтаксическими анализаторами. Нужен один быстрый анализатор или несколько более медленных? Сколько битов должен обрабатывать анализатор за один цикл? Какова стоимость включения конкретного заголовка в граф анализа?

В этом параграфе рассматриваются важные вопросы выбора устройства анализатора и приведены принципы, определяющие выбор параметров. Сначала рассматривается генератор синтаксических анализаторов, который позволяет исследовать разные варианты, затем описаны принципы проектирования, определенные при исследовании вариантов.

Каждый из представленных анализаторов имеет пропускную способность 640 Гбит/с, если явно не указано иное. Эта производительность соответствует современным микросхемам коммутации с 64 портами 10 Гбит/с [2, 6, 10, 11]. Для достижения такой производительности нужно множество экземпляров анализатора.

6.1. Генератор анализаторов

Тщательное исследование пространства проектирования требует анализа и сравнения множества экземпляров синтаксических анализаторов. Для этого был разраюотан генератор, создающий уникальные экземпляры анализаторов по представленным графам анализа и параметрам. Генератор был собран на основе генератора микросхем Genesis [14] - инструмента для генерации вариантов микросхем с использованием «шаблонов» архитектуры и набора конфигурационных параметров приложения. Шаблоны создаются на основе Verilog и Perl, где Perl служит для генерации кода Verilog.

Генератор позволяет моделировать фиксированные и программируемые анализаторы, описанные выше. Параметрами генератора служат граф анализа, разрядность обработки, тип (фиксированный или программируемый), размер буфера полей и размер программируемой памяти TCAM/RAM. Выводом генератора являются файлы Verilog. Результаты по площади и потребляемой мощности были получены с помощью Synopsys Design Compiler G-2012.06 и библиотеки TSMC 45 нм.

Генератор доступен для загрузки по ссылке http: //yuba.stanford.edu/~grg/parser.html[1].

Работа генератора

Для фиксированного анализатора важны два параметра - граф анализа и разрядность обработки. Граф задается в текстовом файле с описанием каждого типа заголовков, содержащим имена и размеры полей, указание извлекаемых полей, отображение значений полей на тип следующего заголовка, а для заголовков переменного размера - отображение, указывающее способ определения размера по значениям полей. Описание заголовка IPv4 приведено на рисунке 13.

```
ipv4 {
        fields {
        version       : 4,
        ihl           : 4,
        diffserv      : 8 : extract,
        totalLen      : 16,
        identification : 16,
        flags         : 3 : extract,
        fragOffset    : 13,
        ttl           : 8 : extract,
        protocol      : 8 : extract,
        hdrChecksum   : 16,
        srcAddr       : 32 : extract,
        dstAddr       : 32 : extract,
        options       : *,
    }
    next_header = map(fragOffset, protocol) {
        1 : icmp,
        6 : tcp,
        17 : udp,
    }
    length = ihl * 4 * 8
    max_length = 256
}
```

Рисунок 13. Заголовок IPv4.

процессоры для конкретных заголовков (4.2.1. Идентификация заголовка) создаются генератором для каждого типа заголовка. Процессоры достаточно просты - они просто извлекают и отображают поля поиска, которые задают размер и тип следующего заголовка. Извлечение полей поиска выполняется путем подсчета слов от начала заголовка, тип заголовка и размер определяются со поставлением с набором шаблонов.

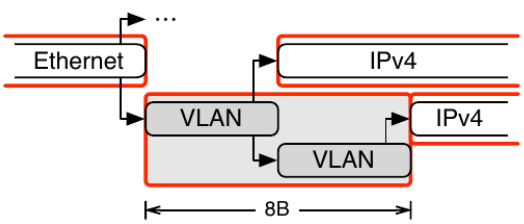


Рисунок 14. Области обработки графа.

Генератор делит граф анализа с использованием заданной разрядности обработки, как показано на рисунке 14. каждая область показывает заголовки, которые могут быть видны в одном цикле. В затененной на рисунке области может поместиться 1 или 2 тега VLAN. Процессоры заголовков инициируются с нужным смещением для каждой из указанных областей.

[1] *В настоящее время исходный код генератора перенесен на Github - https://github.com/grg/parser-gen.*

Обработка заголовка может быть отложена генератором до следующей области с целью минимизации сдвигов по заголовку. На рисунке 14 затененная область может включать 4 первых байта верхнего заголовка IPv4, однако в этом случае анализатору потребуется два процессора IPv4 - один для смещения 0 (VLAN → IPv4), другой для 4 (VLAN → VLAN → IPv4).

Таблица извлечения полей (4.2.2. Извлечение полей) генерируется с использованием помеченных для чтения полей (в описании графа анализа). Запись таблицы для заголовка IPv4 будет указывать извлечение байтов 1, 6, 8, 9 и 12–19. Размер буфера полей задается с учетом всех извлекаемых полей.

Для программируемого анализатора нужно указать разрядность обработки, размер таблицы анализа, размер окна, а также число и размер поиска в таблице анализа. Граф анализа не нужен как параметр генератора для программируемого анализатора, поскольку этот граф задается в процесмсе работы.

Параметры используются генератором для определения размеров и числа компонент. Например, размер окна определяет размер входного буфера в блоке идентификации заголовков и число входов мультиплексора, применяемых при извлечении полей для поиска в таблице анализа. Аналогично, число входов таблицы анализа определяет число мультиплексоров, нужных для извлечения входных данных. В отличие от фиксированного анализатора, программируемый не включает логики, относящейся к конкретному графу анализа.

Генератор не создает TCAM и RAM для таблицы состояний анализа. Для этого нужен генератор от производителя, учитывающий используемую технологию. Вместо этого генератор создает «фиктивную» (не синтезируемую) память для имитации.

Тестовый стенд служит для проверки создаваемых генератором анализаторов. Граф анализа используется для создания входных пакетов и проверки векторов выходных заголовков. Разрядность обработки определяет «ширину» ввода байтовых последовательностей пакета.

6.2. Фиксированный анализатор

Исследование показало, что лишь небольшое число вариантов оказывает заметное влияние на устройство анализатора - для большинства вариантов площадь и потребляемая анализатором мощность различались слабо.

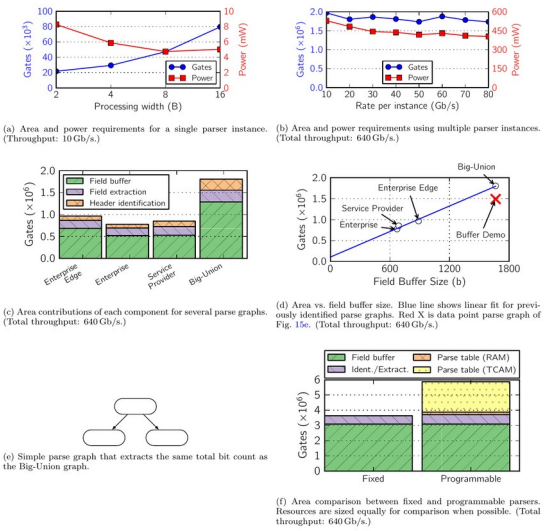
Разрядность обработки увеличивает площадь, но снижает энергопотребление.

Пропускная способность одного экземпляра анализатора определяется выражением $r = w \times f$, где w - разрядность (ширина) обработки, а f - тактовая частота. Для фиксированной пропускной способности $w \propto 1/f$.

На рисунке 15a показана зависимость прощади и потребяемой мощности одного анализатора от разрядности при скорости 10 Гбит/с. Повышение разрядности увеличивает занимаемую площадь за счет размещения дополнительных ресурсов, но энергопотребление снижается за счет уменьшения тактовой частоты (причем быстрее, чем растет объем логики[1]). Потребность в дополнительных ресурсах обусловлена двумя причинами. Во-первых, для более широкой шины требуется больше линий, регистров, мультиплексоров и т. п. Во-вторых, в области обработки (окне) могут появляться дополнительные заголовки (6.1. Генератор анализаторов), для которых потребуются дополнительные экземпляры процессоров заголовков.

Используйте меньше быстрых процессоров при агрегировании экземпляров анализатора.

На рисунке 15b показана зависимость прощади и потребяемой мощности экземпляров анализатора от их скорости при агрегировании до пропускной способности 640 Гбит/с. Можно видеть незначительный выигрыш в энергопотреблении при небольшом числе быстрых анализаторов. Общая площадь при этом почти не меняется. Скорость одного экземпляра анализатора невозможно увеличивать бесконечно и при достижении некого порога начинается рост площади и потребляемой мощности (не показано на графике).



(a) Area and power requirements for a single parser instance. (Throughput: 10 Gb/s.)
(b) Area and power requirements using multiple parser instances. (Total throughput: 640 Gb/s.)
(c) Area contributions of each component for several parse graphs. (Total throughput: 640 Gb/s.)
(d) Area vs. field buffer size. Blue line shows linear fit for previously identified parse graphs. Red X is data point parse graph of Fig. 15c. (Total throughput: 640 Gb/s.)
(e) Simple parse graph that extracts the same total bit count as the Big-Union graph.
(f) Area comparison between fixed and programmable parsers. Resources are sized equally for comparison when possible. (Total throughput: 640 Gb/s.)

Рисунок 15. Влияние устройства на площадь и потребляемую мощность.

Основную площадь занимают буферы полей.

На рисунке 15c показаны площади, занимаеые функциональными блоками для нескольких графов анализа. Буферы полей во всех случаях доминируют, занимая примерно 2/3 площади. Некоторая гибкость обеспечивается за счет создания буфера в виде массива регистров для параллельной передачи данных нисходящим элементам (4.1. Модель абстрактного анализатора).

[1] *Заявление странное, поскольку приведенный график показывает иное. Прим. перев.*

Число извлекаемых анализатором битов определяет площадь (при фиксированной разрядности).

На рисунке 15d приведена зависимость площади от числа извлекаемых битов для нескольких графов анализа. Практически все точки ложатся на одну прямую. Красным крестиком указана точка для простого графа анализа, приведенного на рисунке 15e. Этот граф включает 3 узла, но преобразуется в такое число битов, как и граф big-union. Точка лежит слегка ниже общей прямой, поскольку в этом случае проще идентификация заголовка и извлечение полей.

Фактически этот принцип является следствием предыдущего - число извлекаемых битов определяет размер буфера полей, который доминирует в занимаемой площади.

6.3. Программируемый анализатор

К программируемым анализаторам применимы описанные выше принципы, которые дополняются указанными здесь.

Влияния таблицы состояний и буфера полей имеют одинаковый порядок.

На рисунке 15f показано сравнение площадей для фиксированного (с графом big-union) и программируемого анализатора. Оба анализатора включают буферы полей размером 4 Кбит, а программируемый анализатор имеет 256 × 40 бит TCAM (запись содержит 8 бит состояния и 2 × 16 бит полей заголовков). На рисунке видно, что программруемый анализатор занимает почти вдвое большую площадь.

Важно отметить, что размер фиксированного анализатора приведен с учетом выбранного графа, а размер программируемого должен учитывать все ожидаемые графы анализа. При использовании программируемого анализатора с простым графом может остаться много свободных ресурсов. Например, граф сети предприятия требует лишь 672 бита из буфера полей 4 Кбит. Буфера полей 4 Кбит и 256 × 40 бит TCAM более чем достаточно для реализации всех проверяемых графов анализа. Их общий размер вдвое превышает размер, требуемый для big-union.

На рисунке 15f показана одна точка, но сравнение по таблицам состояний анализатора и размерам буфера полей показывает, что программируемый анализатор занимает в 1,5 - 3 раза большую площадт по сравнению с фиксированным (при разумном размере таблиц и буферов).

Анализатор занимает незначительную часть кристалла микросхемы коммутатора. Фиксированный анализатор на рисунке 15f занимает 2,6 мм^2, а программируемый - 4,4 мм^2 при технологии 45 нм. Площадь современного коммутатора 64 × 10 Гбит/с составляет 200−400 мм^2[1].

Минимизация входных данных поиска в таблице анализа.

Рост числа входных элементов при поиске в таблица анализа позволяет идентифицировать больше заголовков за один цикл, потенциально снижая общее число записей в таблице. Однако исключение каждой записи в таблице просмотра оплачивается ценой дополнительных операций поиска, независимо от числа, требуемого записью.

В таблице 1 показано требуемое число записей и общий размер таблицы для разного числа входных элементов по 16 бит для графа анализа big-union. Общее число записей незначительно снижается при переходе от 1 к 3 операциям поиска, но общий размер таблицы существенно растет. Число просмотров таблицы следует минимизировать для снижения общей площади анализатора, поскольку эта таблица является одним из основных потребителей площади.

Таблица 1. TCAM.

Число входных элементов	***Число записей***	***Разрядность (бит)***	***Размер (бит)***
1	113	24	2712
2	105	40	4200
3	99	56	5544
4	102	72	7344

В этом примере число записей в таблице поиска увеличивается, когда число просмотров превышает 3. Это вызвано эвристикой снижения числа записей в таблице, которая рассматривает по очереди каждый субграф с несколькими входными ребрами. Решение удалить субграф может повлиять на поиск следующего субграфа. В этом случае последовательность выбора при трех поисках на цикл работает лучше, чем выбор при 4 поисках за цикл.

Исследование показало, что два поиска значений по 16 бит обеспечивают хороший баланс размера таблицы состояний анализа и поддержки скорости линии для широкого диапазона заголовков. Все используемые сегодня основные заголовки имеют размер не менее 4 байтов, а большинство кратны этому размеру. Большинство 4-байтовых заголовков включает лишь одно значение для поиска, что позволяет идентифицировать два заголовка по 4 байта за один цикл. В будущем не предполагается большого числа более коротких заголовков, поскольку в них не разместить достаточный объем информации.

7. Смежные работы

Kangaroo [8] представляет собой программируемый анализатор с разбором множества заголовков в одном цикле. Kangaroo буферизует все данные заголовка перед анализом, что вносит задержку, которая слишком велик для современных коммутаторов. Attig [1] представил язык для описания последовательностей заголовков вместе с анализатором на основе FPGA и компилятором. В микросхемах коммутаторов используются микросхемы ASIC, а не FPGA, что предявляет иные требования к устройству анализаторов. Однако таких работ на сегодняшний день не известно. Neither work explores design trade-offs or extract general parser design principles.

Много написано об аппаратном ускорении поиска по регулярным выражениям (например, [12,16,17]) а анализаторах уровня приложений (например, [13, 18]). Синтаксический анализ - это исследование небольшой части пакета, описываемой графом анализа, а при сопоставлении с регулярным выражением выполняется просмотр всех байтов пакета. Различия в областях поиска, обнаруживаемых элементах и требованиях к производительности ведут к разным подходам в проектировании. Анализ на прикладных уровнях достаточно часто включает сопоставление с регулярными выражениями.

Производительность программных анализаторов можно повысить за счет использования упрощенного быстрого пути и полного медленного пути [9]. Быстрый путь обслуживает большую часть входных данных, а по медленному идут лишь

[1] *Информация получена из разговора с сотрудником производителя микросхем.*

редко встречающиеся входные данные. Однако это не применимо к аппаратным синтаксическим анализаторам, поскольку коммутаторы должны гарантировать производительность линии для худших вариантов картины трафика (программные анализаторы таких гарантий не дают).

8. Заключение

Целью этой статьи было понимание устройства синтаксических анализаторов. Для решения задачи был создан генератор, способный создавать программируемые и фиксированные анализаторы, которые послужили для изучения более 500 разных анализаторов. После изучения этих анализаторов был сформулирован ряд принципов проектирования с учетом компромиссных требования.

Исследование показало, что основную площадь, используемую анализатором на кристалле, занимает буфер полей для фиксированных анализаторов и комбинация такого буфера с таблицей анализа для программируемых анализаторов. Множество анализаторов можно объединить для повышения пропускной способности и применение небольшого числа скоростных анализаторов обеспечивает преимущество по потребляемой мощности (без преимущества по площади). Программируемость анализаторов стоит недорого - это увеличивает площадь самого анализатора вдвое, но площадь кристалла лишь на 1 - 2%.

9. Литература[1]

[1] M. Attig and G. Brebner. 400 Gb/s Programmable Packet Parsing on a Single FPGA. In Proc. ANCS '11, pages 12–23, 2011.

[2] Broadcom Trident II Ethernet Switch Series. http://www.broadcom.com/products/Switching/Enterprise/BCM56850-Series.

[3] G. Even, J. Naor, S. Rao, and B. Schieber. Fast approximate graph partitioning algorithms. SIAM Journal on Computing, 28(6):2187–2214, 1999.

[4] C. M. Fiduccia and R. M. Mattheyses. A linear-time heuristic for improving network partitions. In Proc. DAC '82, pages 175–181, 1982.

[5] M. R. Garey and D. S. Johnson. Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman, 1979.

[6] Intel Ethernet Switch Silicon FM6000. http://www.intel.com/content/dam/www/public/us/en/documents/white-papers/ethernet-switch-fm6000-sdn-paper.pdf[2].

[7] B. W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. Bell Systems Technical Journal, (49):291âĂŞ–307, 1970.

[8] C. Kozanitis, J. Huber, S. Singh, and G. Varghese. Leaping Multiple Headers in a Single Bound: Wire-Speed Parsing Using the Kangaroo System. In Proc. INFOCOM 2010, pages 1–9, Mar. 2010.

[9] S. Kumar. Acceleration of Network Processing Algorithms. PhD thesis, Washington University, 2008.

[10] Marvell Prestera CX. https://origin-www.marvell.com/switching/prestera-cx/[3].

[11] Mellanox SwitxhX-2. http://www.mellanox.com/sdn/[4].

[12] J. Moscola, Y. Cho, and J. Lockwood. A Scalable Hybrid Regular Expression Pattern Matcher. In Proc. FCCM '06., pages 337–338, 2006.

[13] J. Moscola, Y. Cho, and J. Lockwood. Hardware-Accelerated Parser for Extraction of Metadata in Semantic Network Content. In IEEE Aerospace Conference '07, pages 1–8, 2007.

[14] O. Shacham, O. Azizi, M. Wachs, W. Qadeer, Z. Asgar, K. Kelley, J. Stevenson, S. Richardson, M. Horowitz, B. Lee, A. Solomatnikov, and A. Firoozshahian. Rethinking Digital Design: Why Design Must Change[5]. IEEE Micro, 30(6):9–24, Nov.-Dec. 2010.

[15] J. P. Shen and M. Lipasti. Modern Processor Design: Fundamentals of Superscalar Processors. McGraw-Hill, 2005.

[16] J. Van Lunteren. High-Performance Pattern-Matching for Intrusion Detection. In Proc. INFOCOM '06, pages 1–13, 2006.

Перевод на русский язык

Николай Малых

nmalykh@protocols.ru

[1] *Большая часть ссылок на доступные в сети документы добавлена при переводе.*
[2] *Приведенная в статье ссылка не работает. Можно воспользоваться другой. Прим. перев.*
[3] *Приведенная в статье ссылка не работает. Можно воспользоваться другой. Прим. перев.*
[4] *Приведенная в статье ссылка не работает. Можно воспользоваться другой. Прим. перев.*
[5] *Описание и код генератора Genesis доступны по ссылке. Прим. перев.*