Разработка алгоритма поиска в таблице классификации сетевого процессора

Документация

Основное задание:

При выполнении основного задания мною был разработан алгоритма поиска по таблице MAC-VLAN, хранящейся в бинарном дереве поиска. Для этого был реализована соответствующая структура данных в виде класса BinarySearchTree, в котором были написаны функции вставки и поиска, а также их рекурсивные версии. Основной файл задания – binary_search_tree.py Помимо структуры бинарного дерева в нём также реализованы функции загрузки таблицы классификации и классификации пакетов по портам соответственно. В файлах create_pcap[i].pcap хранятся пакеты, которые мы загружаем в файл input_packets В соответствующих файлах с расширением .json хранятся таблицы классификации, загружающиеся в основной код.

О работе алгоритма классификации:

Читаем файл с нашими пакетами, из которых извлекаем уровни IP и UDP при наличии. Далее проходим по нашему бинарном дереву до того момента, пока не найдём соответствующие dst_mac и vlan. Если данному ключу соответствует порт, определяем пакет на данный порт, иначе – выводим сообщение об ошибке и сбрасываем пакет в файл dropped.pcap
Важной частью логики работы программы является тот факт, что если на порт не должно отправиться ни одного пакета(в т.ч. если в файл сброшенных пакетов не было таковых) , то на этот порт мы посылаем соответствующее сообщение с текстом по широковещательному каналу. Далее выводим количество пакетов на каждом порте и количество сброшенных пакетов.

Запуск программы:

В основной части программы binary_search_tree есть 5 строчек, загружающих разные таблицы классификации. Каждую можно запустить, предварительно скомпилировав файл, создающий пакеты(create_pcap[i]). Пять файлов, создающих тестовое покрытие данной части задания, содержат в себе:

1) 5 пакетов, где каждый отправляется на свой порт
2) 5 пакетов, где никакой не отправляется на порт из-за несоответствующих vlan
3) 10 пакетов, где каждому порту соответствует по 2 пакета
4) 10 пакетов, где на порт отправляется соответствующее число пакетов(1 или 2)
5) 15 пакетов, случайно выбранных

Каждому выходному порту соответствует файл output_port[i]. Результаты классификации пакетов наблюдаются именно в них и в файле dropped. Таблицы классификации mac_vlan[i] созданы в соответствии с образцом, приложенным к файлу с заданием.

Дополнительное задание:

В рамках дополнительного задания мною был реализован алгоритм поиска по наибольшему совпадению префикса в таблице классификации FIB, произведено тестирование производительности двух описанных структур: бинарного дерева поиска и B-дерева(реализованного в дополнительной части задания), а также создано тестовое покрытие с помощью библиотеки языка Python pytest.

Класс ClassificationTables:

Реализует непосредственно B-дерево для каждой из таблиц классификации(FIB, ARP, MAC-VLAN), а также методы для работы с ними, такие как: вставка элемента и поиск *ключа. Загрузка таблиц также производится через этот класс.*

О работе алгоритма классификации:

В данной части задания алгоритм классификации пакетов по портам претерпел значительные изменения: добавились дополнительные таблицы классификации, а также изменился основной алгоритм поиска совпадения. Сначала мы проверяем существование таблицы классификации fib, если она существует – ищем в неё соответствие к поданному на вход IP. Если нет совпадения или таблица не была загружена, выводим соответствующие сообщения и сбрасываем пакет в первом случае. Далее те же действия совершаем с таблицей ARP: по полученному из таблицы FIB IP_next_hop пытаемся найти MAC узла, куда нужно отправить пакет. Далее таблица MAC-VLAN, получив на вход MAC пакета сопоставляет его с vlan и ищет порт, соответствующий данной комбинации mac-vlan. *Важно:* алгоритм устойчив к отсутствию какой-либо из таблиц: при отсутствии fib будут использованы значения по умолчанию, при отсутствии таблицы arp будет использоваться mac адрес «прибытия» пакета (dst_mac), если же отсутствует таблица mac-vlan, то все пакеты будет сброшены, т.к. у нас фактически не созданы порты, готовые принимать пакеты. Создание пустых пакетов и сброс пакетов аналогичны версии основного задания.

О тестировании алгоритма:

На этапе «ручного» тестирования(об автоматическом – далее) было создано 2 теста, на которых мы проверяем работоспособность программы. В каждом из тестов создаётся по 5 пакетов. Первый тест проверяет корректность работы при отсутствии какой-либо из таблиц(или более), а второй тест проверяет непосредственно наш алгоритм поиска по наибольшей длине префикса: мы отправляем один и тот же пакет, но задаём разные значения поля prefix_len, соответственно для вычисления ip адреса следующего узла берётся тот пакет, где наибольшая длина префикса. Поочерёдный запуск двух этих тестов аналогичен варианту основного задания, в main части программы закомментированы две тройки подключения таблиц классификации, а также в папке находятся pcap файлы, создающие пакеты.

О тестировании производительности:

По заданию мне также было необходимо протестировать производительность работы двух реализованных структур. Для этого в классе описана также функция test_performance. В

ней мы генерируем случайные тестовые данные для ip адреса(нам неважно, какой он будет), а далее засекаем время вставки/поиска в бинарном дереве поиска и в B-дереве соответственно. По результатам можно заметить, что B-дерево быстрее работает при поиске элемента, но в разы медленнее при вставке ключа в дерево. Это объясняется тем, что B-дерево имеет намного более сложную структуру и при вставке ключа нужно изменять большое количество других узлов, лежащих в дереве.

Система автоматического тестирования:

Для моей задачи была реализована система автоматического тестирования при помощи встроенной библиотеки pytest. Она включает в себя: тесты поиска по таблицам классификации(по одному на каждую таблицу) и тест итоговой классификации пакетов. Две фикстуры pytest отвечают за генерацию тестовых данных и инициализацию таблиц классификации соответственно. Далее каждая из четырёх функций проверяет, верен ли результат работы программы, и при успешном прохождении теста, выводит результат на экран. В папке две версии программы: b_tree_no_pytest и b_tree_pytest. В первой реализация автоматического тестирования отсутствует, соответственно можно использовать наши 2 теста для «ручной» проверки. В версии с pytest 4 пройденных теста – показатель правильной работы программы.

Названия файлов аналогичны основному заданию + файлы таблиц классификации fib.json и arp.json