Оглавление

Введение

В связи с быстрым развитием и внедрением современных сетевых технологий а, следовательно, и увеличением объема данных, передаваемых по сети, а также с возникновением все большего количества новых сетевых протоколов прикладного уровня, в наше время возрастает необходимость в анализе сетевого трафика и его классификации. Актуальность данной задачи, также значительно растет, вследствие расширения области ее применения, в которую сейчас включены не только области управления трафиком для повышения эффективности использования каналов связи, но и системы применения политик, а также сфера информационной безопасности.

И так как задач связанных с обработкой больших объемов данных становится все больше, то, соответственно, реализация их вручную занимает огромное количество времени и ресурсов. Решением данной проблемы является, набирающее все большую популярность, в связи с широким спектром применения, машинное обучение. Оно позволяет с помощью статистики находить закономерности в данных и на их основе, с помощью алгоритмов, делать необходимые прогнозы.

Таким образом, целью данной работы является разработка программной системы анализа сетевого трафика с использованием алгоритмов машинного обучения.

1. Техническое задание

1.1 Назначение разработки и область применения

Программная система, разрабатываемая в данной работе, предназначена для классификации сетевого трафика через определение протокола прикладного уровня.

Область применения разрабатываемой системы:

1. Обеспечение информационной безопасности предприятия при пересылке документации через сетевые протоколы.

2. Обнаружения факта использования запрещенных приложений.

Данная система может использоваться на стационарных и портативных компьютерах.

1.2 Технические требования

Рассмотрим требования, предъявляемые разрабатываемой системой к ЭВМ:

1) Операционные системы Linux версии 4.x, аппаратное обеспечение ЭВМ должно поддерживать работу соответствующей ОС;

2) Устройства ввода: мышь, клавиатура;

3) Устройства вывода: монитор.

Разрабатываемая программная система анализа сетевого трафика должна классифицировать захваченный трафик и определять протокол прикладного уровня. У данной системы имеется следующий функционал:

1) Пользователь системы имеет возможность выбрать дамп сетевого трафика в формате .pcap для преобразования в таблицу признаков в формате .csv;

2) на основании полученного файла признаков производится обучение модели с помощью классификатора машинного обучения «случайный лес»;

3) С помощью графического интерфейса отображается захваченный в реальном времени трафик с предсказанным протоколом прикладного уровня.

2. Анализ технического задания

2.1 Выбор операционной системы

На начальном этапе разработки программного продукта, встает задача определения операционной системы, для которой он будет создан.

Для решения этой задачи нужно провести анализ существующих и доступных на момент разработки операционных систем. Для выбора необходимой операционной системы, необходимо принять оптимальное решение, основываясь, на совокупности факторов удовлетворяющих требованиям к разрабатываемой системе.

Рассмотрим самые распространенные на сегодняшний день операционные системы: Linux, Windows, Mac OS:

1) Linux – операционные системы, которые основаны на свободном и открытом программном обеспечении. Операционные системы на базе ядра Linux и включающих в себя библиотеки им системные программы проекта GNU, представляют собой группу Unix-подобных операционных систем. Как правило, Linux-системы распространяются в виде бесплатных готовых дистрибутивов. Дистрибутивы Linux создаются на основе одноименного ядра, библиотек и системных программ, которые разрабатываются в рамках проекта GNU. Несмотря на то, что Linux не так давно начал приобретать популярность, он уже превратился в одну из самых надежных компьютерных систем на планете. Объединяя эту надежность с нулевой стоимостью входа, в результате получаем идеальное решение для настольной платформы.

2) Windows – это наиболее популярная пользовательская операционная система производства компании Microsoft. По данным статистики, она установлена на 85% устройств. Такую популярность у пользователей можно объяснить ориентированностью на управление с помощью графического интерфейса.Windows их можно разделить на 4 группы:

- MS-DOS;
- Windows-9x;
- Windows NT;
- Windows для смартфонов;
- Windows Embedded.

3) Mac OS – это продукт компании Apple появившийся на рынке в 1984 году. Это сопутствующее ПО для выпускаемых этой корпорацией устройств. Основана на FreeBSD, исходный код закрыт. На рынке операционных систем в отличие от Linux, несмотря на общую основу, не является системой с открытым исходным кодом и занимает второе по популярности место.

Для разработки программной системы анализа сетевого трафика была выбрана операционная система Linux, так как она содержит все необходимые возможности для разработки данного программного обеспечения.

2.2 Выбор языка программирования

Следующим важным этапом разработки программного обеспечения является по выбор языка программирования, на котором будет производиться разработка всего приложения. Необходимый для решения поставленной задачи язык должен быть приспособлен для обработки больших массивов данных и иметь инструменты для применения машинного обучения. А также выбор языка зависит от навыков разработчика.

Рассмотрим наиболее подходящие под данные условия языки:

1) Python – высокоуровневый язык программирования, с минималистичным синтаксисом ядра, ориентированный на повышение производительности разработчика и читаемости кода. Стандартная библиотека Python включает большой объём полезных функций. Этот язык обладает динамической типизацией и автоматическим управлением памятью, а также поддерживает объектно-ориентированное, императивное, структурное, функциональное и аспектно-ориентированное программирование. Python является универсальным языком и применяется во многих сферах, в том числе для анализа и статистической обработки данных, например, в область его применения входят back-end разработка, разработка графических интерфейсов, веб-сайтов, разработка баз данных, научные и числовые вычисления. Пакеты, вроде pandas, scikit-learn и Tensorflow, делают Python отличным вариантом для современных приложений с машинным обучением.

2) R – язык программирования, ориентированный на статистическую обработку данных и работу с графикой. R является языком с открытым исходным кодом, поэтому в нем всегда своевременно появляются пакеты с новыми методами обработки данных, созданные активным сообществом. Огромное количество разнообразных библиотек, предназначенных для работы с различными типами баз данных и методов их обработки,

способствуют все более широкому распространению этого языка. К недостаткам следует отнести невысокую скорость работы.

3) MatLab - язык программирования, используемый в одноименном пакете прикладных программ MatLab для инженерных расчетов, включает основанные на матрицах структуры данных, широкий спектр функций, интегрированную среду разработки, объектно-ориентированные возможности и интерфейсы к программам, написанным на других языках программирования. Имеет высокую скорость работы, удобный графический интерфейс и является простым в изучении. Но, между тем, имеет и ряд недостатков. Это и неполная поддержка необходимых нам статистических функций, и слабость в части интеграции при разработке большой системы, и дороговизна лицензии.

4) Java –объектно-ориентированный язык программирования. Был призван упростить разработку приложений на C++, путём исключения из него низкоуровневых возможностей. Изначально разрабатывался для бытовой электроники, но впоследствии стал. Главная особенность Java - то, что компилируется этот язык не в машинный код, а в платформо-независимый байт-код, который может выполняться с помощью интерпретатора - виртуальной Java-машины(JVM), что делает этот язык частично платформо-независимым. Достоинством Java является то, что этот язык, строго типизированный и очень серьезен по отношению к определению типов, для приложений, работающих с большими объемами данных это бесценно.

Учитывая все рассмотренные выше характеристики языков, в качестве используемого языка для разработки программной системы был выбран Python, так как данный язык показывает высокую скорость вычислений, простой и удобный синтаксис, а также имеет все необходимые для решения задачи библиотеки.

Рассмотрим библиотеки языка Python, которые помогут нам в разработке программной системы анализа сетевого трафика:

1) PyQt4- инструмент для создания GUI приложений. PyQt представляет собой сочетание языка программирования Python и превосходной библиотеки Qt.

2) NumPy- библиотека с открытым исходным кодом, рассматривается как альтернатива MatLab**.**

3) Sklearn (scikit-learn)- библиотека для машинного обучения с открытым исходным кодом, с ее помощью реализовываются различные алгоритмы классификации, кластеризации и регрессии.

4)Pandas – библиотека на предназначенная для анализа и обработки данных. Строится поверх библиотеки NumPy.

5) Pcapy- модуль расширения, позволяющий программному обеспечению, написанному на Python, получать доступ к подпрограммам из библиотеки пакетов Pcap.

6) ndpiReader - анализатор трафика, использующий библиотеку nDPI. Скомпилирован под Linux версии 4.x.

7) dpkt - это модуль для быстрого и простого анализа пакетов, с определением основных протоколов TCP / IP.

2.3 Обзор существующих методов классификации сетевого трафика

Для решения задачи классификации трафика создано множество алгоритмов, которые, в свою очередь можно классифицировать по используемым в них подходам. Один из вариантов классификации этих подходов приведен на рис. 1

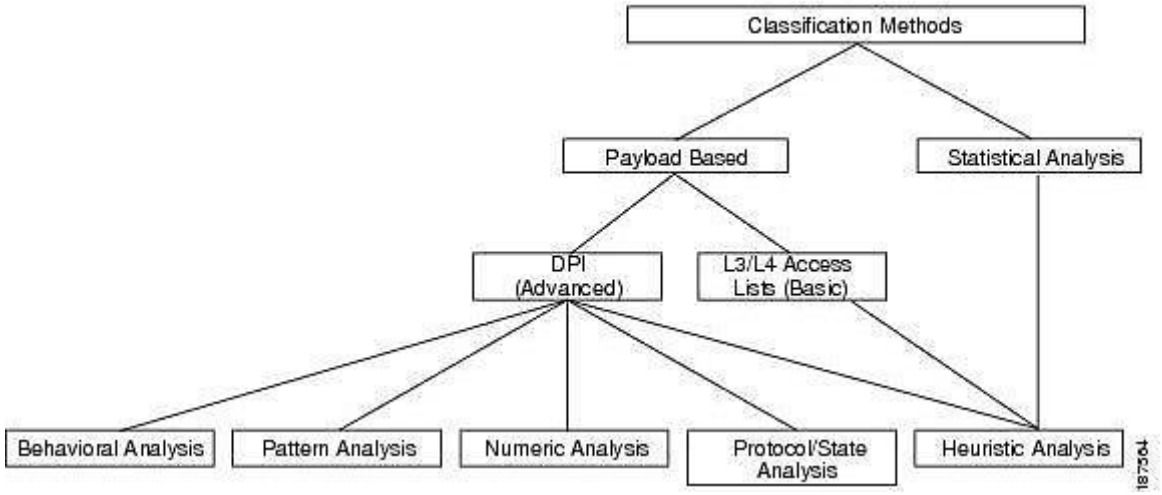


Рис. 1. Подходы к решению задачи классификации

То есть существуют два основных метода классификации трафика:

- Классификация на основе блоков данных (Payload-Based Classification). Основывается на полях с блоками данных, таких как порты OSI (отправитель, получатель или оба). Данный метод является наиболее распространенным, но не работает с зашифрованным и туннелированным трафиком.

- Классификация на основе статистического метода. (Statistical Analysis) Основывается на анализе поведения трафика (время между пакетами, время сеанса и т. п.).

При классификации на основе блоков данных выделяют универсальный подход к классификации трафика и метод глубокого анализа пакетов (DPI).

Универсальный подход к классификации трафика основывается на информации в заголовке IP-пакета – как правило, это IP-адрес, MAC-адрес, используемый протокол. Этот подход имеет ограниченные возможности, поскольку информация берется только из IP-заголовка, так же, как ограничены методы с использованием портов – ведь далеко не все приложения используют стандартные порты.

Более совершенную классификацию позволяет осуществить глубокий анализ пакетов (DPI). Системы глубокого анализа трафика позволяют классифицировать те приложения и протоколы, которые невозможно определить на IP-адресах и портах, например URL внутри пакета, содержимое сообщений мессенджеров, голосовой трафик Skype, p2p-пакеты BitTorrent. То есть DPI анализирует не только заголовки пакетов, но и полное содержимое трафика на уровнях модели OSI со второго и выше.

Основным механизмом идентификации приложений в DPI является анализ сигнатур (Signature Analysis). Каждое приложение имеет свои уникальные характеристики, которые занесены в базу данных сигнатур. Сопоставление образца из базы с анализируемым трафиком позволяет точно определить приложение или протокол. Но так как периодически появляются новые приложения, то базу данных сигнатур также необходимо обновлять для обеспечения высокой точности идентификации

Существуют несколько методов сигнатурного анализа:

- Анализ образца (Pattern analysis).
- Числовой анализ (Numerical analysis).
- Поведенческий анализ (Behavioral analysis).
- Эвристический анализ (Heuristic analysis).
- Анализ протокола/состояния (Protocol/state analysis).

2.4 Выбор подхода к решению задачи анализа сетевого трафика

Как говорилось ранее, наиболее точную классификацию имеет метод DPI. Однако, его применение при современных объемах трафика требует больших

вычислительных затрат, и соответственно получает низкую скорость проведения анализа трафика.

Следовательно, для устранения данной проблемы необходимо применить альтернативный способ решения одной из главных задач DPI – определения протокола прикладного уровня на основе применения алгоритмов машинного обучения. Этот метод будет иметь высокую скорость анализа, так как он основан на малом количестве информации, не требует оценивать и запоминать содержимое IP-пакетов, не сверяясь со списком широко известных портов, и не глядя в полезную нагрузку пакетов.

В средствах DPI, обычно, объектом классификации является поток трафика транспортного уровня – это совокупность IP-пакетов, у которых совпадает протокол транспортного уровня, а также неупорядоченная пара (ip источника, порт источника)-(ip назначения, порт назначения). В данном методе будут использоваться именно такие потоки.

Идея, которая лежит в основе предлагаемого метода, заключается в том, что разные приложения, пользующиеся разными протоколами, также генерируют потоки транспортного уровня с разными статистическими характеристиками. Если аккуратно и ёмко определить набор статистических метрик потока, то по значениям этих метрик можно будет с высокой точностью предсказывать, какое приложение сгенерировало данный поток, и, соответственно, какой протокол прикладного уровня этим потоком переносится.

На этапе принятия решения есть возможность использовать разные алгоритмы машинного обучения. Для определения наилучшего из них для решения поставленной задачи было проведено исследование с помощью облачной службы Microsoft Azure.

3. Разработка структуры программной части

3.1 Разработка общей структуры схемы

Систему анализа сетевого трафика можно представить в виде следующих модулей.

Рис.2. Структура программной системы

На вход системы подается захваченный трафик, эти данные должны подвергаться предварительной обработке. Она включает в себя следующие пункты

- Убрать полученные протоколы, с малым количеством потоков;
- Убрать неизвестные протоколы;
- Объединить трафик, полученный с одинаковых протоколов.

На следующем этапе формируются признаки, которые заносятся в базу признаковых описаний.

Затем после предварительной обработки собранных данных, сопоставляя их с выбранными признаками, соответственно, получаем обученную модель. На этапе тестирования, то есть построения модели предсказаний, берутся выбранные признаки

и модель, полученная на этапе обучения, и подаются на вход классификатора, который выдает нам результат классификации трафика.

Расширенная схема выглядит, как показано на рисунке 5.

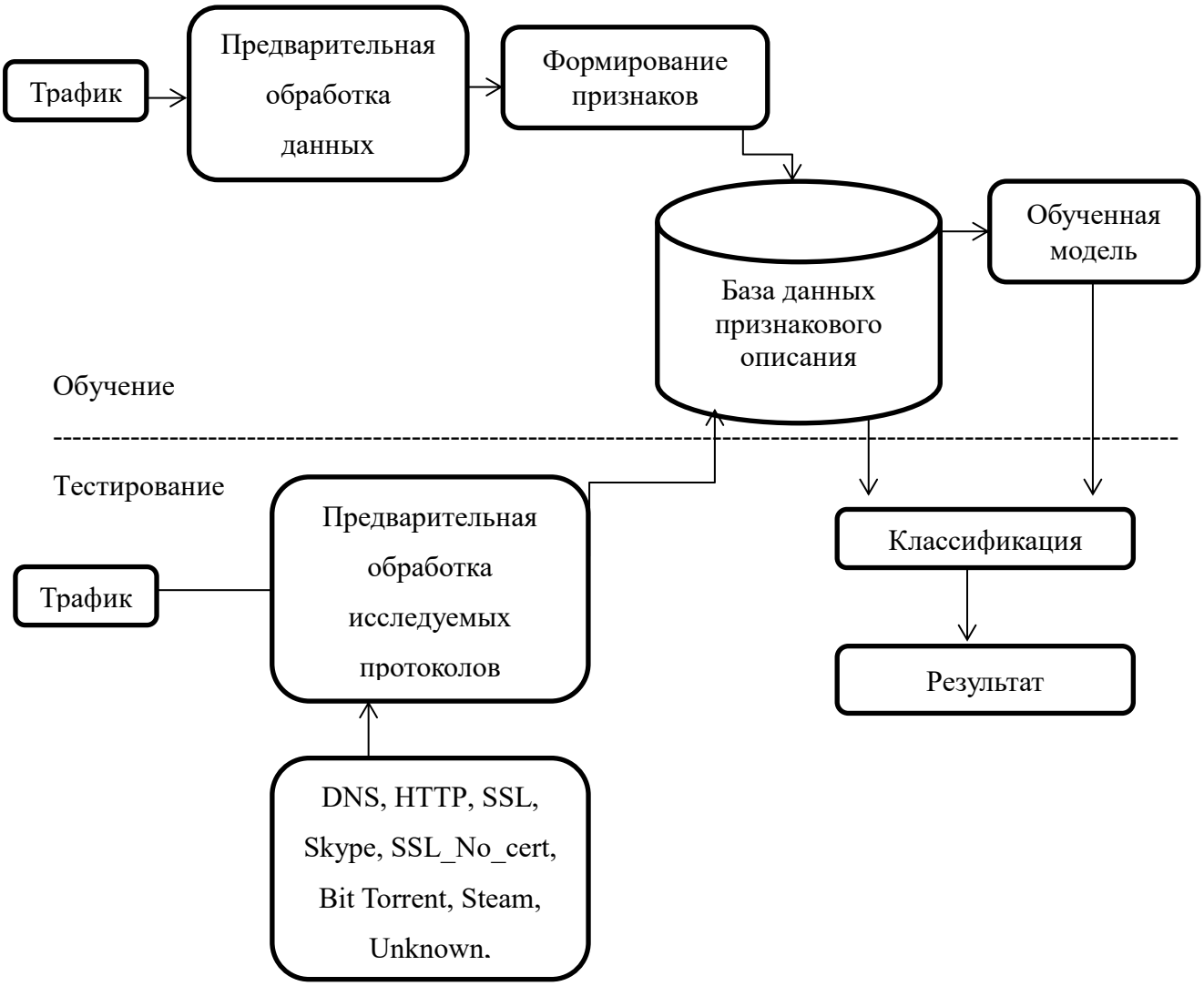


Рис.3. Расширенная структурная схема

Захват трафика производится с помощью программы Wireshark. Для обучения алгоритма было собрано около 10 Гб захваченного трафика. Wireshark – это программа, которая поддерживает разбор большого количества различных сетевых протоколов, а также предоставляет возможность сортировки и фильтрации трафика. Программа позволяет пользователю просматривать и сохранять весь проходящий по сети трафик в режиме реального времени (для этого необходимо дополнительно установить библиотеку

WinPcap (libpcap)). На рис. 2 представлена архитектура Wireshark, она состоит из двух компонентов:

1. Библиотека, посредством которой осуществляется перехват сетевого трафика (WinPcap для ОС Windows, libpcap для ОС Linux);

2. Прикладная программа, предоставляющая функции разбора протоколов и графический интерфейс для визуализации результатов анализа и взаимодействия с системой.

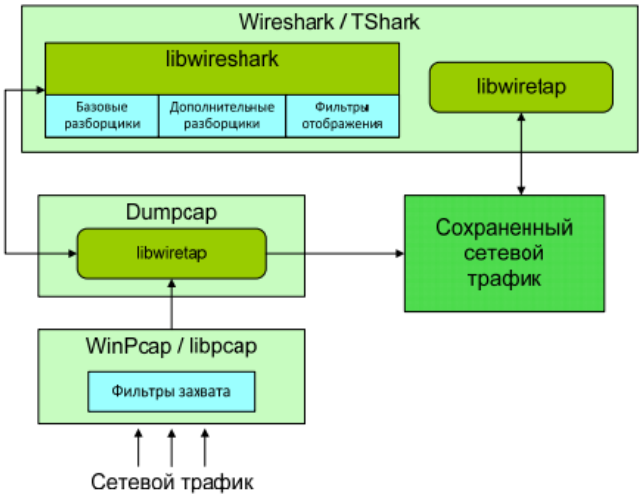


Рис. 4. Архитектура Wireshark

Но помимо самого трафика нам также нужно для каждого захваченного потока достоверно определить переносимый им протокол прикладного уровня. В этом нам помогут средства DPI, конкретно – библиотека nDPI. В комплекте с этой библиотекой поставляется пример анализатора трафика под названием ndpiReader.

Для определения протокола прикладного уровня с использованием алгоритмов машинного обучения нам необходим набор обучающих данных. Чтобы из потока трафика транспортного уровня выделить структурированные данные, необходимо обозначить статистические метрики этого потока. В данной работе используются следующие статистические метрики потока для клиента и для сервера:

- Последовательность размеров сегментов транспортного уровня;
- Последовательность размеров порций данных;

Данные последовательности, представляют собой ряды чисел, которые наилучшим образом характеризуют поток данных, и на их основе можно достаточно точно предсказать протокол прикладного уровня. Основываясь на этих рядах можно выделить следующие метрики (признаки) потока трафика:

1. Средний размер пакета со стороны клиента и со стороны сервера;
2. Средний размер порции данных со стороны клиента и со стороны сервера;
3. Стандартное отклонение размера пакета со стороны клиента и со стороны сервера;
4. Стандартное отклонение размера порции данных со стороны клиента и со стороны сервера;
5. Срединное отклонение размера пакета со стороны клиента и со стороны сервера;
6. Срединное отклонение размера порции данных со стороны клиента и со стороны сервера;
7. Среднее число пакетов на порцию данных со стороны клиента и со стороны сервера;
8. Соотношение количества переданных клиентом байт, относительно количества переданных сервером;
9. Соотношение количества переданных клиентом пакетов, относительно количества переданных пакетов сервером;
10. Размеры первого и второго сегментов транспортного уровня со стороны клиента и со стороны сервера;
11. Размеры первого и второго данных со стороны клиента и со стороны сервера;
12. Тип протокола транспортного уровня (0 — UDP, 1 — TCP).

3.2 Анализ алгоритмов классификации данных

При выборе алгоритма машинного обучения, нельзя однозначно точно заранее определить какой классификатор для поставленной задачи будет лучшим, на эффективность алгоритмов влияет множество факторов, например, объем и структура исследуемых данных. Следовательно, необходимо проверять эффективность каждого в каждом конкретном случае на тестовом наборе данных и затем выбирать лучший вариант. В решение этой задачи и помогает Microsoft Azure ML.

3.2.1 Обзор платформы Microsoft Azure Machine Learning

Azure ML — это облачная служба аналитики, которая позволяет быстро создавать и развертывать прогнозные модели в качестве решений аналитики. Для этого она содержит все необходимое, от большой библиотеки алгоритмов до студии для создания моделей и удобных функций развертывания моделей в виде веб-служб. Так же обладает возможностью быстро создавать, тестировать и вводить в эксплуатацию прогнозные модели, управлять ими, а также включать элементы исходного кода на R и Python. Сервис машинного обучения Azure Machine Learning в настоящее время находится в предварительном публичном тестировании и доступном каждому.

Поскольку Azure Machine Learning является облачным решением, то нет необходимости в использовании локальной среды разработки, весь необходимый для реализации функционал доступен через веб-интерфейс.

Azure ML предлагает множество алгоритмов для классификации задач, включая Multiclass Decision Forests, Decision Jungles, Logistic Regression, Neural Networks и TwoClass Support Vector Machine. Так же есть возможность применять статистические функции, такие как вычисление отклонений и оценки корреляции. Предоставляется функционал для визуализации данных, построения графиков и диаграмм.

Общая схема проведенного эксперимента изображена на рис.3

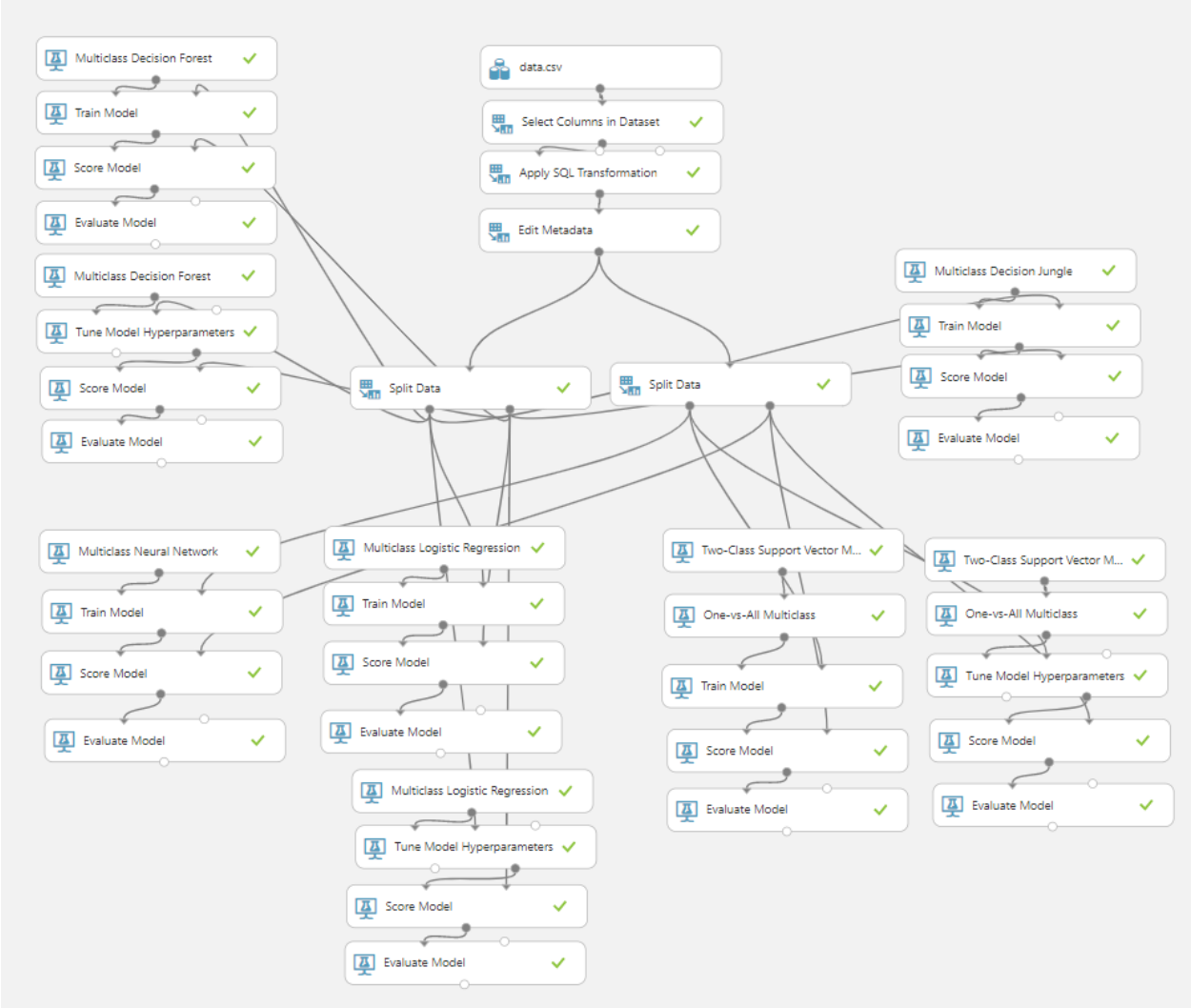
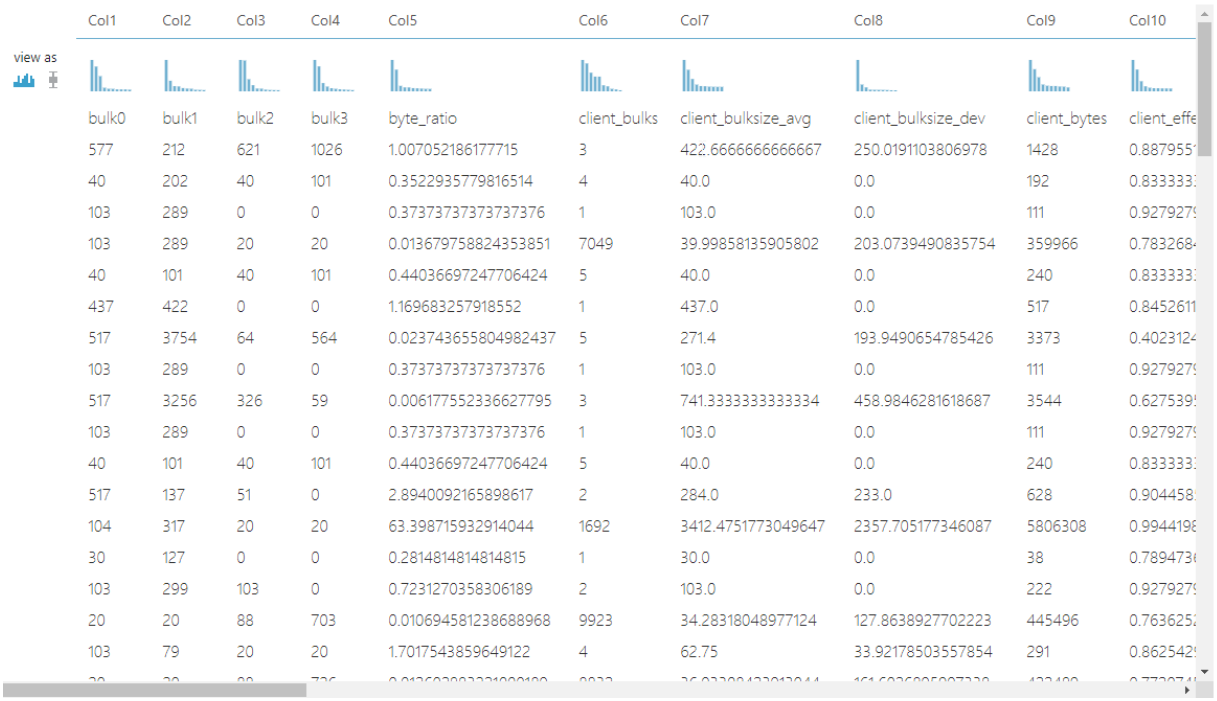


Рис.5. Схема эксперимента

Data.csv- набор ранее собранных и преобразованных данных.



Col1	Col2	Col3	Col4	Col5	Col6	Col7	Col8	Col9	Col10
bulk0	bulk1	bulk2	bulk3	byte_ratio	client_bulks	client_bulksize_avg	client_bulksize_dev	client_bytes	client_effe
577	212	621	1026	1.007052186177715	3	422.6666666666667	250.0191103806978	1428	0.887955
40	202	40	101	0.3522935779816514	4	40.0	0.0	192	0.833333
103	289	0	0	0.37373737373737376	1	103.0	0.0	111	0.927927
103	289	20	20	0.013679758824353851	7049	39.99858135905802	203.0739490835754	359966	0.783268
40	101	40	101	0.44036697247706424	5	40.0	0.0	240	0.833333
437	422	0	0	1.169683257918552	1	437.0	0.0	517	0.845261
517	3754	64	564	0.023743655804982437	5	271.4	193.9490654785426	3373	0.402312
103	289	0	0	0.37373737373737376	1	103.0	0.0	111	0.927927
517	3256	326	59	0.006177552336627795	3	741.3333333333334	458.9846281618687	3544	0.627539
103	289	0	0	0.37373737373737376	1	103.0	0.0	111	0.927927
40	101	40	101	0.44036697247706424	5	40.0	0.0	240	0.833333
517	137	51	0	2.8940092165898617	2	284.0	233.0	628	0.904458
104	317	20	20	63.398715932914044	1692	3412.4751773049647	2357.705177346087	5806308	0.994419
30	127	0	0	0.2814814814814815	1	30.0	0.0	38	0.789473
103	299	103	0	0.7231270358306189	2	103.0	0.0	222	0.927927
20	20	88	703	0.010694581238688968	9923	34.28318048977124	127.8638927702223	445496	0.763625
103	79	20	20	1.7017543859649122	4	62.75	33.92178503557854	291	0.862542

Рис.6. Табличное представление собраных данных

Split Data – разделяет данные на две части в пропорции 0.75 –для обучающей выборки, 0.25 –для тестовой выборки (В схеме использовались две Split Data для наглядности).

Train Model – обучает модель, на вход принимает один из представленных классификаторов и обучающую выборку, а выдает обученную модель. Так же необходимо указывать параметр, который будет предсказываться – в данной задаче это proto, то есть протокол.

Tune model Hyperparametrs – у каждого классификатора есть параметры настройки. Если для каждого параметра одно значение, используется Train Model, а если несколько, то данный элемент, который прогоняет все комбинации параметров, находит лучший и обучается по нему модель.

Score Model – предсказывает значение, на вход принимает обученную модель и тестовую выборку, иа на выход выдает результат.

Evaluate Model – отображается полученные данные от Score model.

3.2.2 Алгоритмы классификации в Microsoft Azure ML

Multiclass Decision Forests- алгоритм принятия решений является ансамблевым

методом обучения, применяется для задач классификации, регрессии и кластеризации.

Основная идея заключается в использовании большого ансамбля решающих деревьев, каждое из которых само по себе даёт очень невысокое качество классификации, но за счёт их большого количества результат получается хорошим. Деревья решений, в целом являются непараметрическими моделями, то есть они поддерживают данные с различным распределением. В каждом дереве для каждого класса выполняется последовательность простых тестов, увеличивая уровни древовидной структуры до достижения конечного узла (решения).

Деревья решений имеют много преимуществ:

- Они могут представлять нелинейные границы решения.
- Они эффективны в вычислениях и использовании памяти во время обучения и прогнозирования.
- Они выполняют комплексный выбор и классификацию объектов.
- Они устойчивы при наличии шумных особенностей.

Decision Jungles – расширение лесов принятия решений. Решающие джунгли состоят из множества ациклических графов, ориентированных на принятие решений (DAG). В отличие от обычных деревьев решений, которые допускают только один путь к каждому узлу, DAG в джунглях принятия решений допускает несколько путей от корня до каждого листа.

Решающие джунгли имеют следующие преимущества:

- Позволяя объединять ветви дерева, группа DAG для принятия решений, как правило, имеет меньший объем памяти и лучшую производительность обобщения, чем дерево решений, хотя и за счет некоторого более длительного времени обучения.
- Решающие джунгли - это непараметрические модели, которые могут представлять нелинейные границы решения.
- Они выполняют интегрированный выбор и классификацию функций и устойчивы при наличии шумных функций.

Logistic Regression- статистический алгоритм, который используется для прогнозирования вероятности исхода и особенно популярен для задач классификации. Алгоритм прогнозирует вероятность возникновения события путем подгонки данных к логистической функции. Стандартная логистическая регрессия

является биномиальной и предполагает два выходных класса. Мультиклассовая логистическая регрессия предполагает три или более выходных классов.

Neural Networks- набор взаимосвязанных слоев. Входные данные являются первым слоем и связаны с выходным слоем ациклическим графом, состоящим из взвешенных ребер и узлов.

Между входным и выходным слоями вы можете вставить несколько скрытых слоев. Большинство прогнозирующих задач можно легко выполнить только с одним или несколькими скрытыми слоями. Однако недавние исследования показали, что глубокие нейронные сети (DNN) со многими слоями могут быть очень эффективными в сложных задачах, таких как распознавание изображений или речи. Последовательные слои используются для моделирования возрастающих уровней семантической глубины.

Связь между входами и выходами изучается из обучения нейронной сети на входных данных. Направление графика идет от входов через скрытый слой и к выходному слою. Все узлы в слое связаны взвешенными ребрами с узлами в следующем слое.

Чтобы вычислить выходные данные сети для определенного входа, значение вычисляется на каждом узле в скрытых слоях и в выходном слое. Значение устанавливается путем вычисления взвешенной суммы значений узлов из предыдущего слоя. Затем к этой взвешенной сумме применяется функция активации.

One-vs-All Multiclass- Сам классификатор не имеет параметров. Параметры имеет Two-Class Support Vector Machine, который идет на вход классификатору. Этот модуль полезен для создания моделей, которые предсказывают три или более возможных результата, когда результат зависит от непрерывных или категориальных переменных предиктора. Этот метод также позволяет использовать методы двоичной классификации для проблем, требующих нескольких выходных классов. В то время как некоторые алгоритмы классификации допускают использование более двух классов по своему замыслу, другие ограничивают возможные результаты одним из двух значений (двоичной или двухклассовой моделью). Однако даже двоичные алгоритмы классификации могут быть адаптированы для задач классификации нескольких классов, используя различные стратегии.

Этот модуль реализует метод «One-vs-All» , в котором двоичная модель создается для каждого из нескольких выходных классов. Каждая из этих бинарных моделей для отдельных классов оценивается с точки зрения ее дополнения (всех других классов в

модели), как если бы это была проблема бинарной классификации. Затем прогнозирование выполняется путем запуска этих двоичных классификаторов и выбора прогноза с наибольшим доверительным счетом. По сути, создается множество отдельных моделей, а затем результаты объединяются для создания единой модели, которая прогнозирует все классы. Таким образом, любой двоичный классификатор может использоваться в качестве основы для модели «One-vs-All».

Two-Class Support Vector Machine- это хорошо изученный класс контролируемых методов обучения. Эта конкретная реализация подходит для прогнозирования двух возможных результатов, основанных на непрерывных или категориальных переменных.

Two-Class Support Vector Machine являются одними из первых алгоритмов машинного обучения, и модели SVM использовались во многих приложениях, от поиска информации до классификации текста и изображений. SVM могут использоваться как для задач классификации, так и для регрессии.

Эта модель SVM является контролируемой моделью обучения, которая требует помеченных данных. В процессе обучения алгоритм анализирует входные данные и распознает шаблоны в многомерном пространстве признаков, называемом *гиперплоскостью*. Все входные примеры представлены как точки в этом пространстве, и сопоставлены с выходными категориями таким образом, что категории делятся на максимально широкие и устраняют разрыв, насколько это возможно.

Для прогнозирования алгоритм SVM назначает новые примеры в одну или другую категорию, отображая их в том же пространстве.

В разрабатываемой программной системе будет использоваться алгоритм из Multiclass Decision Forests, а именно Random Forest.Так как в результатах нашего исследования этот классификатор показал хорошую точность классификации, и потому что из-за достаточно большого количества признаков, нам нужен алгоритм, который слабо чувствителен к шумам и корреляции метрик.

3.3 Алгоритм машинного обучения « Random Forest»

Random Forest **(**случайный лес**)** — это множество решающих деревьев. В задаче регрессии их ответы усредняются, в задаче классификации принимается решение голосованием по большинству. Все деревья строятся независимо по следующей схеме:

- Выбирается подвыборка обучающей выборки – по ней строится дерево (для каждого дерева — своя подвыборка).
- Для построения каждого расщепления в дереве просматриваем случайные признаки (для каждого нового расщепления — свои признаки).
- Выбираются наилучшие признаки и расщепление по нему (по заранее заданному критерию). Дерево строится, как правило, до исчерпания выборки (пока в листьях не останутся представители только одного класса), но в современных реализациях есть параметры, которые ограничивают глубину дерева, число объектов в листьях и число объектов в подвыборке, при котором проводится расщепление.

Основная идея была описана при рассмотрении Multiclass Decision Forests. Общий алгоритм работы классификатора «случайный лес»

Дана обучающая выборка размерности N, размерность множества признаков M и параметр m (обычно m = sqrt(M)).

1) Генерируется случайная выборка с повторением размером N из обучающей выборки

2) На ее основании строится дерево принятия решений, которое классифицирует примеры из этой выборки. В ходе создания очередного узла дерева выбирается признак, на основе которого производится разбиение, не из всего множества M признаков, а лишь из m случайно выбранных. Выбор наилучшего из этих m признаков может осуществляться различными способами.

3) Построение дерева продолжается до полного исчерпания выборки. Описанные выше шаги, повторяются для заданного количества деревьев дерева. Как говорилось ранее, в задачах классификации решение принимается голосованием по большинству, то есть каждое дерево комитета относит классифицируемый объект к одному из классов, и побеждает класс, за который проголосовало наибольшее число деревьев. Оптимальное число деревьев подбирается таким образом, чтобы минимизировать ошибку классификатора на тестовой выборке.

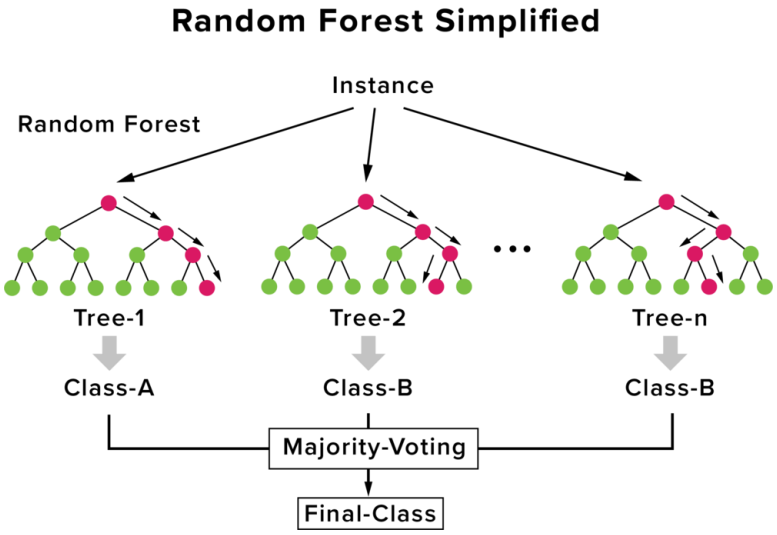


Рис. 7. Алгоритм Random Forest

Теперь рассмотрим подробнее описание данного алгоритма. Начать необходимо с описания его составляющих, а именно деревьев принятия решений – это основной структурный элемент леса. Алгоритм случайного леса характеризуется отличиями и особенностями построения деревьев, входящими в него.

Алгоритм построения бинарного дерева решений работает по схеме жадного алгоритма: на каждой шаге для входного подмножества обучающего множества строится такое разбиение пространства гиперплоскостью, которое минимизировало бы среднюю меру неоднородности двух полученных подмножеств. Данная процедура выполняется рекурсивно для каждого полученного подмножества до тех пор, пока не будут достигнуты критерии остановки.
При каждом делении все объекты делятся на две более мелкие группы, т.е. рассматриваемая в каждом из узлов задача разбивается на две более мелкие подзадачи.

Заданием максимального числа объектов в вершине-листе дерева устанавливается один из возможных критериев останова для алгоритма. Таким образом, можно качественно классифицировать рассматриваемую выборку объектов при помощи всего одного дерева решений. Однако в реальных задачах часто встречаются погрешности в измерениях и объекты-выбросы (ошибки), которые серьезно портят качество классификации одним конкретным деревом решений. Поэтому данный алгоритм необходимо реализовывать для множества деревьев. Соответственно, чем больше количество и глубина деревьев, тем выше точность классификации, но увеличивается и время работы алгоритма.

3.4 Параметры оценки эффективности классификатора.

Для оценки качества алгоритмов классификации данных, можно использовать различные критерии и показатели. В основном они базируются на полноте (recall) и точности (precision), которые в свою очередь являются базисом для таких метрик как F-мера или R-Precision и основываются на отношении ошибок первого и второго рода.

Ошибкой первого рода называется случай, когда классификатор определил нормальный пакет как ошибочный, то есть произошла «ложная тревога».

Ошибкой второго рода называется случай, когда классификатор определяет ошибочный пакет, как нормальный, и подобные ошибка имеет высокий приоритет минимизации.

Точность (precision) системы в пределах класса – это доля объектов действительно принадлежащих данному классу относительно всех объектов, которые система отнесла к этому классу.

Полнота (recall) системы – это доля найденных классификатором объектов принадлежащих классу относительно всех документов этого класса в тестовой выборке.

F-мера –гармоническое среднее между полнотой и точностью.

Эти значения легко рассчитать на основании таблицы контингентности, которая составляется для каждого класса отдельно

Категория i		Экспертная оценка	
		Положительная	Отрицательная
Оценка системы	Положительная	TP	FP
	Отрицательная	FN	TN

Табл. 1. Таблица контингентости

В таблице содержится информация, сколько раз система приняла верное и сколько раз неверное решение по документам заданного класса. А именно:

- TP - истинно-положительное решение;
- TN - истинно-отрицательное решение;
- FP- ложноположительное решение (ошибка первого рода);
- FN - ложноотрицательное решение (ошибка второго рода).

Тогда точность, полнота и F-мера можно выразить следующими выражениями:

- Precision = $\frac{TP}{TP+FP}$;
- Recall = $\frac{TP}{TP+FN}$;
- F= $2 * \frac{Precision*Recall}{Precision+Recall}$;

4. Разработка программных средств

В данной работе, для анализа сетевого трафика, разработаны три основных модуля. Эти модули представлены на блок схеме (Рис. 8).

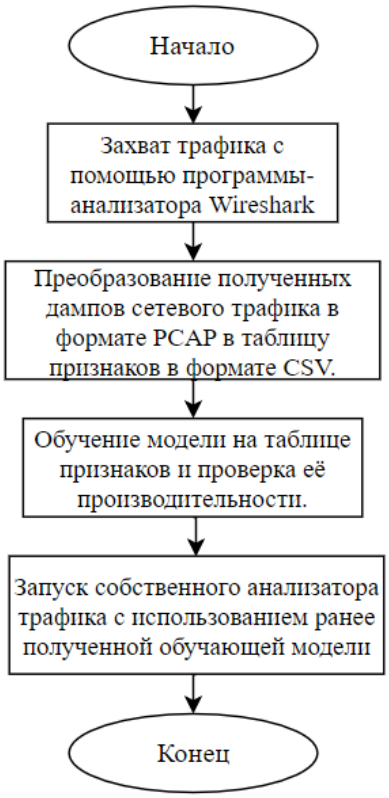


Рис. 8. Блок-схема системы анализа сетевого трафика

Далее необходимо рассмотреть подробнее каждый модуль и основные функции, входящие в него.

4.1 Модуль формирования признакового описания

Первый модуль, с которого начинается работа с разработанной системой, называется pcaptocsv.py.

Интерфейс модуля:

pcaptocsv.py [-o OUTPUT] [-s N] file [file …]

file [file …]** - один или несколько PCAP-файлов

-o OUTPUT** - имя выходного CSV-файла (по умолчанию «flows.csv»)

-s N** - строить таблицу признаков только по первым N сегментам транспортного уровня.

Данный модуль на вход принимает один или несколько pcap файлов, по которым строит таблицу признаков в виде файла csv, для удобного табличного представления.

Основные функции модуля:

main() – основная функция, с которой начинается выполнение модуля. Анализирует все pcap файлы, указанные в аргументах при запуске, разбирает каждый из них и просматривает их потоки. После всех действий, сохраняется признаки в csv файл.

```
Def main():
    parser = argparse.ArgumentParser()
    parser.add_argument("file", nargs="+", help="pcap file")
    parser.add_argument("-o", "–output", help="output csv file", default="flows.csv")
    parser.add_argument("-s", "–strip", help="leave only first N 27thernet27", metavar = "N", default=0, type=int)
    args = parser.parse_args()
    flows = {feature: [] for feature in FEATURES}
    for pcapfile in args.file:
        if len(args.file) > 1:
            print pcapfile
        for proto, subproto, flow in parse_flows(pcapfile):
            stats = forge_flow_stats(flow, args.strip)
            if stats:
                stats.update({"proto": proto, "subproto": subproto})
                for feature in FEATURES:
                    flows[feature].append(stats[feature])
    data = ps.DataFrame(flows)
    data.to_csv(args.output, index=False)
```

ip_from_string() – функция преобразования символьного представления ip адреса в четырехбайтную строку. На вход получает ip адрес в виде строки («192.168.1.65»). Возвращает строку из 4 байт.

```
Def ip_from_string(ips):
    return "".join(chr(int(n)) for n in ips.split("."))
```

parse_flows() – функция разбиения pcap файла на потоки транспортного уровня и определения прикладного протокола каждого потока. Принимает путь до обрабатываемого pcap файла. Возвращает списки прикладных протоколов, дополнительных прикладных протоколов и данные, для каждого потока.

Forge_flow_stats() – функция расчета статистических метрик потока. Принимает список Ethernet-фреймов, а также количество первых фреймов, по которым строить таблицу признаков. Возвращает Словарь, в котором ключи – названия метрик, значения – значения этих метрик. Если в потоке нет хотя бы двух порций данных, возвращает None. Полный исходный код данного модуля можно посмотреть в Приложении А.

4.2 Модуль обучения модели

Следующий модуль системы, это model.py.

Интерфейс модуля:

model.py [-o FILE] [-r SEED] file

file** - таблица признаков в формате CSV

-r SEED** - семя генератора случайных чисел

-o FILE** - имя выходного файла, куда будет сохранена обученная модель.

Данный модуль на основе полученного, из предыдущего модуля, файла признаков, производит обучение модели с помощью классификатора машинного обучения «случайный лес».

Основные функции модуля:

main() – основная функция, с которой начинается выполнение модуля. Производит обработку данных, обучение модели и её сохранение в файл mdl.

```
Parser = argparse.ArgumentParser()
   parser.add_argument(“file”, help=”csv file”)
   parser.add_argument(“-o”, “–output”, help=”output model into file”, metavar=”FILE”)
   parser.add_argument(“-r”, “–random”, help=”random seed value”, metavar=”SEED”, type=int)
   args = parser.parse_args()
   data = ps.read_csv(args.file)
   data, scaler, labeler = preprocess(data, args.random)
   clusters = split_data(data)
   for I, data_train in enumerate(clusters):
      print(“\n\n*** FOLD: {} ***\n”.format(i+1))
      data_test = ps.concat([c for c in clusters if c is not data_train])
      model = train_model(data_train, args.random)
      score_model(model, data_test, labeler)
   if args.output:
```

```
    pickle.dump((model, scaler, labeler), open(args.output, “wb”))
    print “\nModule successfully written to file ‘{}’.”.format(args.output)
```

Preprocess() - функция осуществляет предварительную очистку данных, масштабирование и маркировку. Принимает таблицу признаков и семя генератора псевдослучайных чисел. Возвращает обработанную таблицу признаков, StandardScaler, LabelEncoder.

```
Def preprocess(data, seed=None):
  drop_protos = [“Unknown”, “Unencryped_Jabber”, “NTP”, “Apple”]
  replace_protos = [(“SSL_No_Cert”, “SSL”)]
  data = data[~data[“proto”].isin(drop_protos)]
  for old_proto, new_proto in replace_protos:
    data = data.replace(old_proto, new_proto)
  if seed:
    np.random.seed(seed)
    data = data.iloc[np.random.permutation(len(data))]
  scaler = StandardScaler()
  labeler = LabelEncoder()
  X = scaler.fit_transform(data.drop([“proto”, “subproto”], axis=1))
  y = labeler.fit_transform(data[“proto”])
  cols = [col for col in data.columns if col not in (“proto”, “subproto”)]
  return ps.concat([ps.DataFrame(X, columns=cols),
        ps.DataFrame({“proto”: y})], axis=1), scaler, labeler
```

split_data() – функция разделить таблицу признаков на три части так, чтобы каждый протокол одинаково присутствовал в каждой части. Принимает таблицу признаков. Возвращает список из трех таблиц признаков, каждой из которых равен 1/3 от изначального.

```
Def split_data(data):
  proto_clusters = [data[data[“proto”] == proto] for proto in data[“proto”].unique()]
  clusters = [[], [], []]
  for cluster in proto_clusters:
    split_index = len(cluster)//3
    for I in range(3):
```

```
        clusters[i].append(
            cluster.iloc[i*split_index : (i+1)*split_index])
    return [ps.concat(clus) for clus in clusters]
```

train_model() – функция обучения модели на таблице признаков. Принимает таблицу признаков для обучающей выборки, семя генератора псевдослучайных чисел. Возвращает обученную модель «Случайного леса».

```
Def train_model(data_train, seed=None):
    X_train = data_train.drop([“proto”], axis=1)
    y_train = data_train[“proto”]
    model = RandomForestClassifier(27, “entropy”, 9, random_state=seed)
    model.fit(X_train, y_train)
    return model
```

score_model() – функция оценивает производительность модели, выведя в стандартный вывод три таблицы: важности признаков, значения полноты и точности для каждого класса, реальные и предсказанные классы. Принимает обученную модель, таблицу признаков для тестовой выборки, LabelEncoder данной выборки. Ничего не возвращает, а только выводит результат тестирования производительности в консоль.

```
Def score_model(model, data_test, labeler):
    X_test = data_test.drop([“proto”], axis=1)
    y_test = data_test[“proto”]
    y_predicted = model.predict(X_test)

    true_labels = labeler.inverse_transform(y_test)
    predicted_labels = labeler.inverse_transform(y_predicted)

    print feature_importances_report(model, X_test.columns)
    print “\n”, classification_report(true_labels, predicted_labels)
    print cross_class_report(true_labels, predicted_labels)
```

cross_class_report() – функция составления таблицы реальных и предсказанных классов. Принимает numpy-массив реальных меток классов, numpy-массив предсказанных меток классов. Возвращает таблицу.

```
Def cross_class_report(y, p):
    classes = np.unique(y)
    res = ps.DataFrame({“y”: y, “p”: p}, index=None)
    table = ps.DataFrame(index=classes, columns=classes)
    for true_cls in classes:
        tmp = res[res[“y”] == true_cls]
        for pred_cls in classes:
            table[pred_cls][true_cls] = len(tmp[tmp[“p”] == pred_cls])
    return table
```

feature_importances_report() – функция составляет отчёт о важности признаков. Принимает обученную модель, список текстовых наименований признаков. Возвращает отчет в виде строки.

```
Def feature_importances_report(model, columns):
    imp = {col: imp for imp, col
        in zip(model.feature_importances_, columns)}
    assert len(imp) == len(columns)
    return “\n”.join(“{} {:.4f}”.format(str(col).ljust(25), imp)
        for col, imp in sorted(imp.items(), key=(lambda x: -x[1])))
```

Полный исходный код данного модуля можно посмотреть в Приложении Б.

4.3 Модуль графического приложения визуализации захвата трафика в реальном времени

Данное приложение состоит еще из нескольких модулей, а также имеет графический интерфейс, написанный с использованием PyQt4.

Main.py – точка входа в приложение.

```
Import sys
from PyQt4 import QtGui
from dialogs.mainwindow import MainWindow
def main():
    app = QtGui.Qapplication(sys.argv)
    window = MainWindow()
    window.show()
    sys.exit(app.exec_())
if __name__ == «__main__»:
    main()
```

mainwindow.py – модуль, отвечающий за взаимодействие пользователя с графическим интерфейсом (начало и остановка захвата трафика, работа с таблицей отображения потоков, и обработки двойного нажатия).

Некоторые функции:

startCapture() – функция начала захвата трафика выбранного сетевого интерфейса.

```
Def startCapture(self):
    all_devs = sorted(pcapy.findalldevs())
    dev, ok = QtGui.QinputDialog.getItem(self, u"Device selection", u"Select device:", all_devs, 0, False)
    if not ok: return
    self.model = TrafficModel(self.prediction_models, self)
    self.ui.tableView.setModel(self.model)
    self.ui.tableView.resizeColumnsToContents()
    self.ui.action_start.setEnabled(False)
    self.ui.action_stop.setEnabled(True)
    self.thread = CapThread(str(dev))
    self.thread.framesReceived.connect(self.addFrames)
    self.thread.start()
```

stopCapture() – функция остановки захвата трафика.

```
Def stopCapture(self):
    self.ui.action_start.setEnabled(True)
    self.ui.action_stop.setEnabled(False)
    self.thread.terminate()
    self.thread = None
```

addFrames() – функция добавления новой записи в таблицу потоков.

```
Def addFrames(self, raws):
    flows = {}
    for raw in raws:
       eth = dpkt.ethernet.Ethernet(raw)
       if eth.type != dpkt.ethernet.ETH_TYPE_IP:
          continue
       ip = eth.data
       if ip.p == dpkt.ip.IP_PROTO_TCP:
          proto = "TCP"
       elif ip.p == dpkt.ip.IP_PROTO_UDP:
          proto = "UDP"
       else:
          continue
       seg = ip.data
       flow_key = (frozenset([(ip.src, seg.sport), (ip.dst, seg.dport)]), proto)
```

```
            if flow_key not in flows:
                flows[flow_key] = []
            flows[flow_key].append(eth)
        self.model.addFlows(flows)
        self.ui.tableView.resizeColumnsToContents()
```

ui_mainwindow.py – модуль конфигерации графического интерфейса.

Capthread.py.- модуль запуска потока для асинхронной работы.

```
From PyQt4.QtCore import Qthread, pyqtSignal


import pcapy, time


class CapThread(Qthread):
    framesReceived = pyqtSignal(object)


    def __init__(self, dev, parent=None):
        super(CapThread, self).__init__(parent)
        self.dev = dev
        self.exiting = False


    def __del__(self):
        self.exiting = True
        self.wait()


    def run(self):
        pcap = pcapy.open_live(self.dev, 65536, False, 100)
        pcap.setnonblock(1)
        while not self.exiting:
            frames = [1]
            while frames[-1]:
                try:
                    frames.append(pcap.next()[1])
                except Exception, e:
                    break
```

```
            if len(frames) > 2:
                self.framesReceived.emit(frames[1:-1])
            time.sleep(0.05)
```

util.py – модуль содержащий функции для обработки захваченного трафика.

Trafficmodel.py – модуль, отвечающий за взаимодействие с таблицей потоков.

Полный исходный код данного модуля можно посмотреть в Приложении В.

5. Тестирование системы

5.1 Описание набора данных

Для тестирования системы анализа сетевого трафика, необходима большая база захваченного трафика.

В ходе тестирования необходимо будет определить, что система работает в соответствии с ожиданиями, а также необходимо определить точность классификации, если она будет высокой, значит, разработанная система работает корректно.

В качестве набора данных использовался дамп сетевого трафика общим объемом приблизительно 10Гб.

Для обучающей выборки из базы было взято 1/3 от случайных сообщений.

5.2 Описание методики тестирования

5.2.1 Тестирование модуля формирования признакового описания

Из-за большого объема файла, полученного дампа сетевого трафика, он был разделен на 6 частей, 5 из которых по 2 Гб и один 730 Мб.

На вход модуля pcaptocsv.py передаются все 6 pcap файлов, для формирования по ним файла признакового описания в формате csv. Название полученного файла указывается после ключа –o. Как видно из рисунка 9, данная процедура повторяется 3 раза, с указанием различных аргументов, следующих после ключа –s, отвечающего за то, по скольким первым N сегментам строить таблицу признаков.

Выполняемые команды:

python pcaptocsv.py –o data-s10.csv –s 10 part0.pcap part1.pcap part2.pcap part3.pcap part4.pcap part5.pcap

python pcaptocsv.py –o data-s100.csv –s 100 part0.pcap part1.pcap part2.pcap part3.pcap part4.pcap part5.pcap

python pcaptocsv.py –o data-s1000.csv –s 1000 part0.pcap part1.pcap part2.pcap part3.pcap part4.pcap part5.pcap

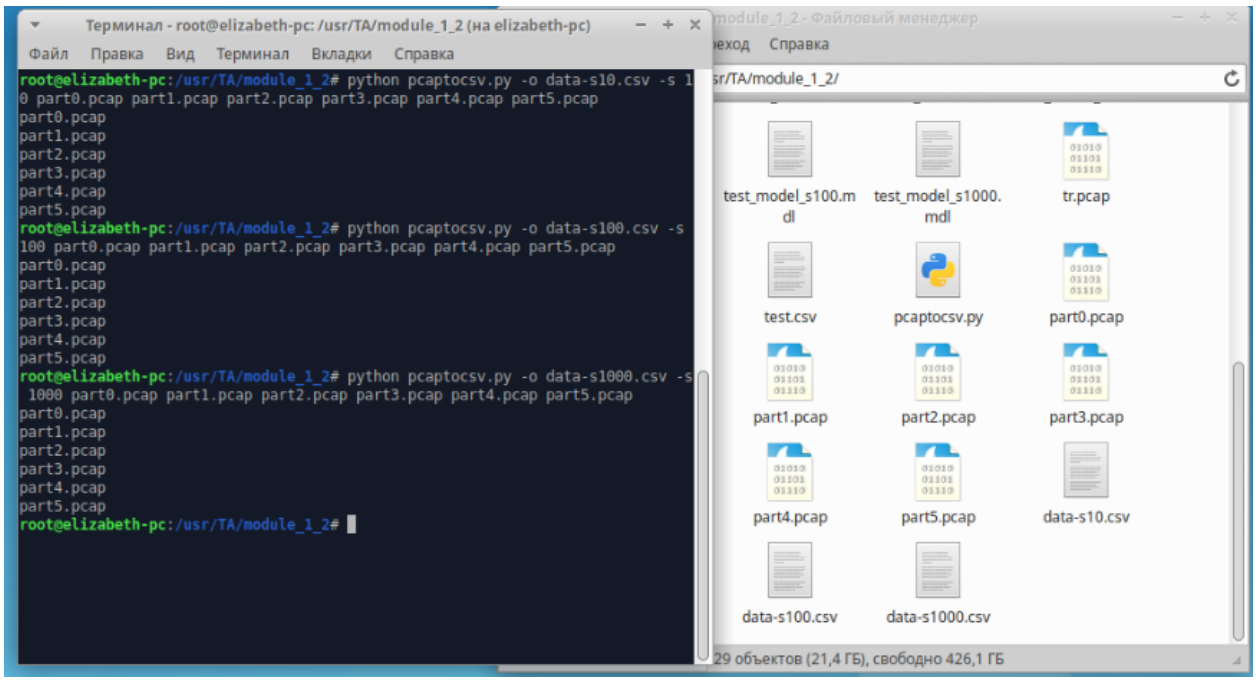


Рис. 9. Результат работы с модулем формирования признакового описания

В результате выполнения данного модуля, появляется файл признаков, формата csv.

5.2.2 Тестирование модуля обучения модели

На вход данному модулю подается таблица признаков в формате csv и производится обучение модели, которая сохраняется в файл формата mdl.

Выполняем данную процедуру 3 раза, для каждого из созданных файлов признаков.

Семя генератора случайных чисел было выбрано случайно.

Выполняемые команды:

python model.py –o model-s10.csv –r 12345 data-s10.csv

python model.py –o model-s100.csv –r 12345 data-s100.csv

python model.py –o model-s1000.csv –r 12345 data-s1000.csv

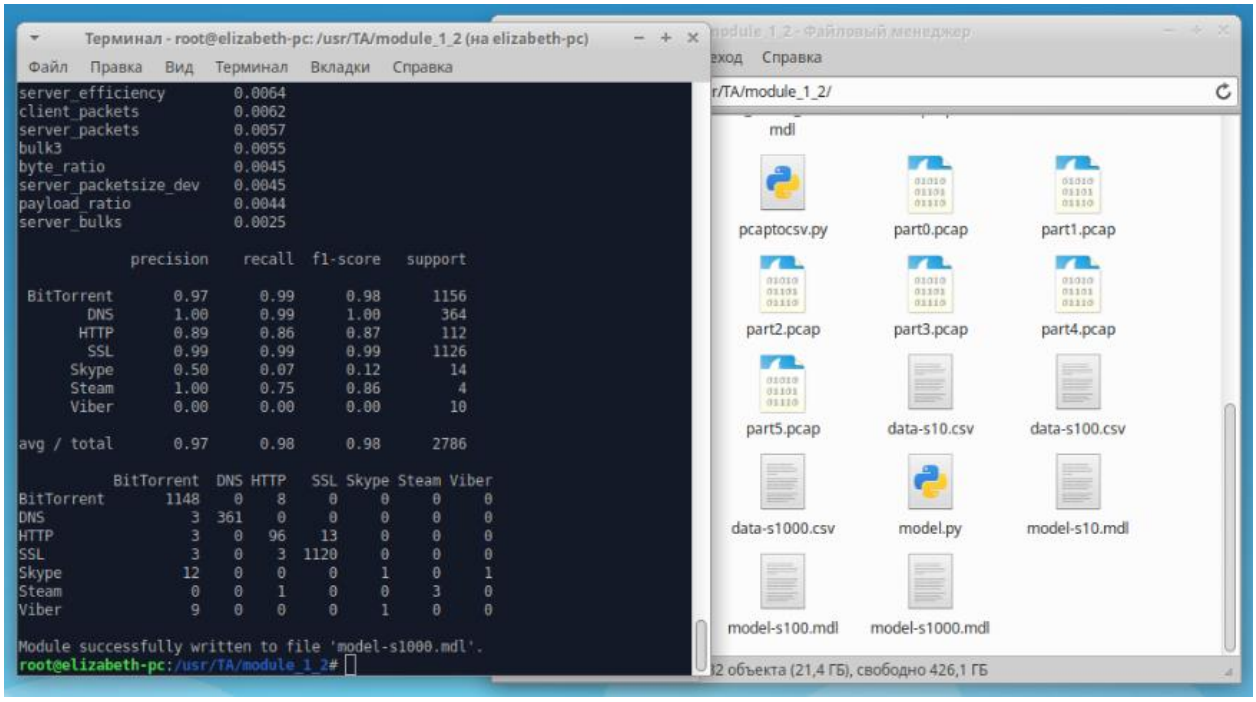


Рис. 10. Результат работы с модулем обучения модели

По полученной информации во время тестирования производительности, выведенной в консоль, можно увидеть, что средняя точность предсказания приблизительно равно 97%, что является довольно неплохим результатом.

В результате появилось три файла обученных моделей, соответствующих каждому из файлов признаков.

5.2.3 Тестирование графического приложения визуализации захвата трафика в реальном времени

Когда все файлы обученных моделей получены, можно указать их в качестве предсказываемых моделей. Делается это в модуле mainwindow.py:

S10_FILE = “model_s10.mdl”

S100_FILE = “model_s100.mdl”

S1000_FILE = “model_s1000.mdl”

После этого, можно запустить приложение и проверить работоспособность.

Начальный интерфейс:

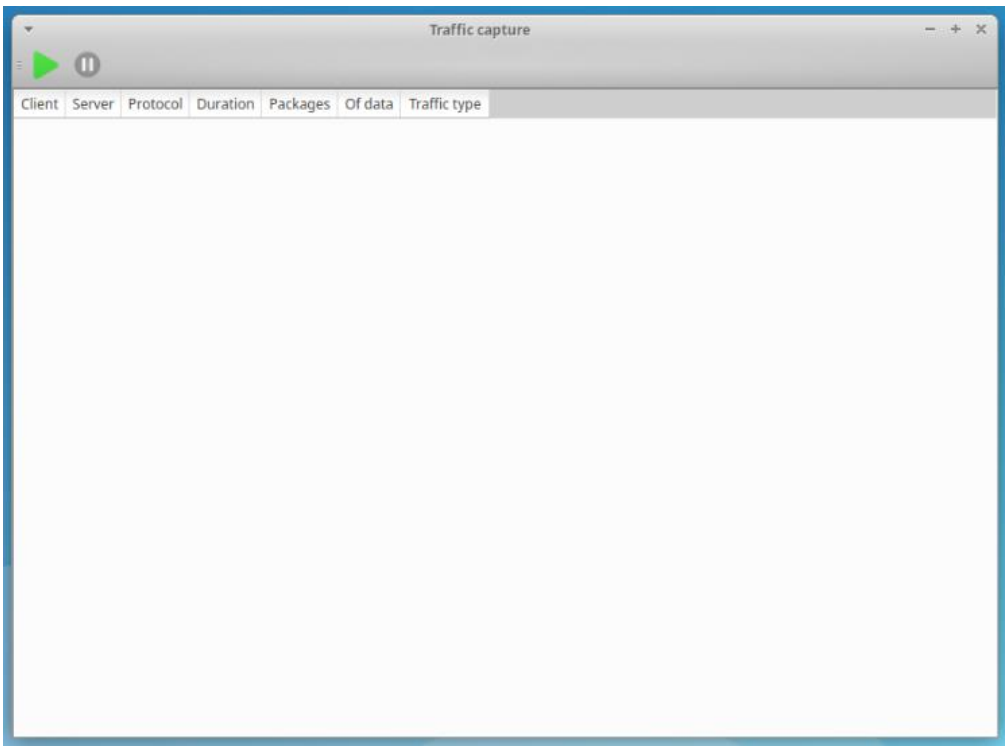


Рис. 11. Начальный интерфейс приложения

После нажатия на кнопку старта (зеленая стрелочка) открывается диалоговое окно с возможность выбора конкретного сетевого интерфейс, с которого будет производиться захват трафика.

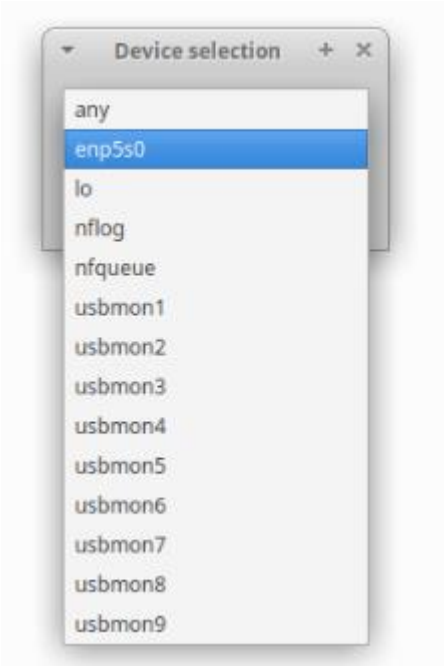
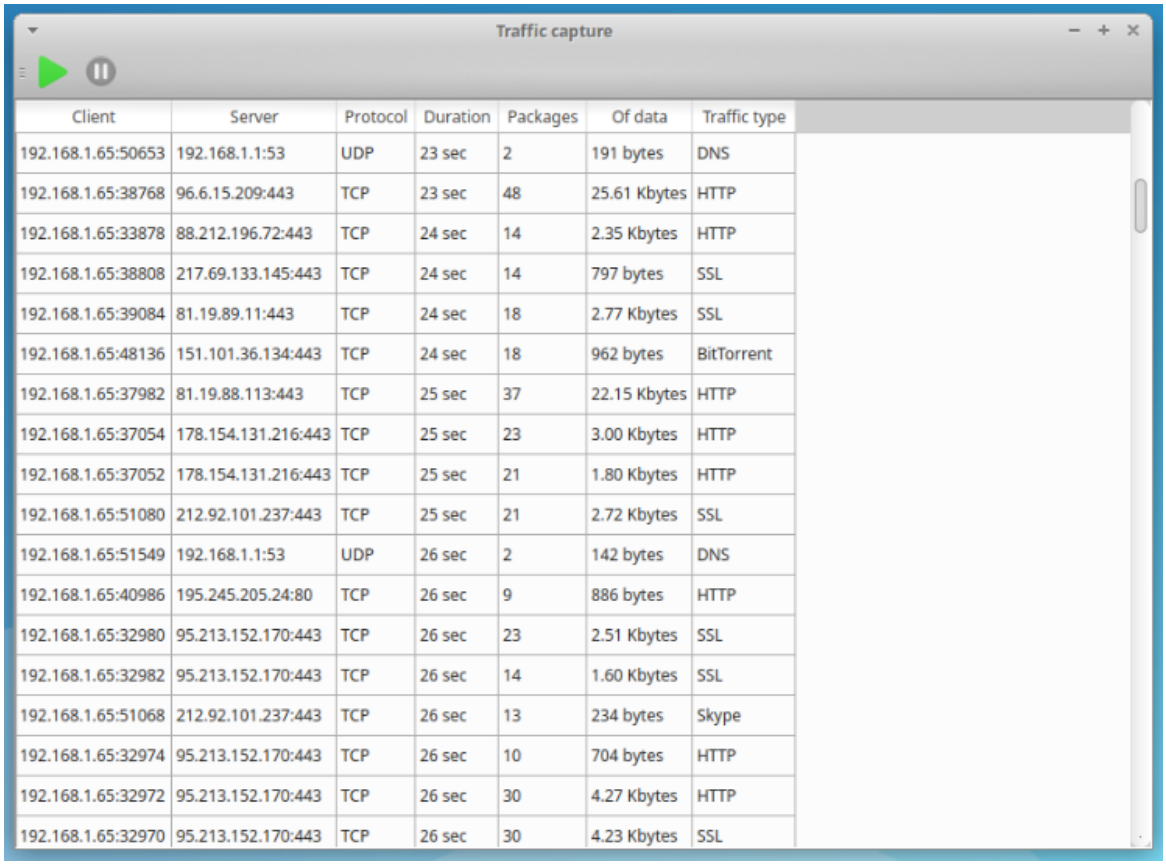


Рис. 12. Диалоговое окно выбора сетевого интерфейса

Далее начнется захват трафика в реальном времени и предсказывание его протокола прикладного уровня (Traffic type).



Client	Server	Protocol	Duration	Packages	Of data	Traffic type
192.168.1.65:50653	192.168.1.1:53	UDP	23 sec	2	191 bytes	DNS
192.168.1.65:38768	96.6.15.209:443	TCP	23 sec	48	25.61 Kbytes	HTTP
192.168.1.65:33878	88.212.196.72:443	TCP	24 sec	14	2.35 Kbytes	HTTP
192.168.1.65:38808	217.69.133.145:443	TCP	24 sec	14	797 bytes	SSL
192.168.1.65:39084	81.19.89.11:443	TCP	24 sec	18	2.77 Kbytes	SSL
192.168.1.65:48136	151.101.36.134:443	TCP	24 sec	18	962 bytes	BitTorrent
192.168.1.65:37982	81.19.88.113:443	TCP	25 sec	37	22.15 Kbytes	HTTP
192.168.1.65:37054	178.154.131.216:443	TCP	25 sec	23	3.00 Kbytes	HTTP
192.168.1.65:37052	178.154.131.216:443	TCP	25 sec	21	1.80 Kbytes	HTTP
192.168.1.65:51080	212.92.101.237:443	TCP	25 sec	21	2.72 Kbytes	SSL
192.168.1.65:51549	192.168.1.1:53	UDP	26 sec	2	142 bytes	DNS
192.168.1.65:40986	195.245.205.24:80	TCP	26 sec	9	886 bytes	HTTP
192.168.1.65:32980	95.213.152.170:443	TCP	26 sec	23	2.51 Kbytes	SSL
192.168.1.65:32982	95.213.152.170:443	TCP	26 sec	14	1.60 Kbytes	SSL
192.168.1.65:51068	212.92.101.237:443	TCP	26 sec	13	234 bytes	Skype
192.168.1.65:32974	95.213.152.170:443	TCP	26 sec	10	704 bytes	HTTP
192.168.1.65:32972	95.213.152.170:443	TCP	26 sec	30	4.27 Kbytes	HTTP
192.168.1.65:32970	95.213.152.170:443	TCP	26 sec	30	4.23 Kbytes	SSL

Рис. 13. Результат работы приложения

Заключение

В данной работе наглядно продемонстрирован метод анализа, который позволяет качественно классифицировать трафик с целью определения протокола прикладного уровня без средств DPI, на основе алгоритма машинного обучения. Данный метод можно применять для идентификации используемых протоколов прикладного уровня в тех условиях, когда полезная нагрузка зашифрована или для профилактического сбора статистики для анализа.

Тестирование системы подтвердило её работоспособность и возможность использования для решения поставленной задачи. Для дальнейшего развития данного метода можно увеличить количество исследуемых протоколов прикладного уровня, а также проводить тестирование в системах с большим потоком сетевого трафика.

Список литературы

1. А. Гетельман, Е.Евстропов, Ю. Маркин Анализ сетевого трафика в режиме реального времени: обзор прикладных задач, подходов и решений-2015
2. А. Гетельман Ю. Маркин, Д. Обыденков, Е Евстропов Обзор задач и методов их решения в области классификации сетевого трафика-2017
3. Chris Riley, Ben Scott Deep – Packet Inspection: The End of Internet as We Know It? // Free Press, March 2009.
4. Ю. Маркин, А. Санаров Обзор современных инструментов анализа сетевого трафика - 2016.
5. С. П. Чистяков – Случайные леса: Обзор // Труды Карельского научного центра РАН №1 2013, с.117-

Приложение А

pcaptocvs.py

```
#!/usr/bin/python2.7
# -*- coding: utf-8 -*-

from __future__ import division

import argparse, re, dpkt, sys
from subprocess import Popen, PIPE, call
import pandas as ps
import numpy as np

FEATURES = [
    "proto",
    "subproto",
    "bulk0",
    "bulk1",
    "bulk2",
    "bulk3",
    "client_packet0",
    "client_packet1",
    "server_packet0",
    "server_packet1",
    "client_bulksize_avg",
    "client_bulksize_dev
    "server_bulksize_avg",
    "server_bulksize_dev",
    "client_packetsize_avg",
    "client_packetsize_dev",
    "server_packetsize_avg",
    "server_packetsize_dev",
    "client_packets_per_bulk",
    "server_packets_per_bulk",
```

```
    "client_effeciency",
    "server_efficiency",
    "byte_ratio",
    "payload_ratio",
    "packet_ratio",
    "client_bytes",
    "client_payload",
    "client_packets",
    "client_bulks",
    "server_bytes",
    "server_payload",
    "server_packets",
    "server_bulks",
    "is_tcp"
]

def ip_from_string(ips):
    return "".join(chr(int(n)) for n in ips.split("."))

def parse_flows(pcapfile):
    pipe = Popen(["./ndpiReader", "-i", pcapfile, "-v2"], stdout=PIPE)
    raw = pipe.communicate()[0].decode("utf-8")
    reg = re.compile(r'(UDP|TCP) (\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}):(\d{1,5}) <->
(\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}):(\d{1,5}) \[proto: [\d+\.]*\d+\/(\w+\.?\w+)*\]')
    flows = {}
    apps = {}
    for captures in re.findall(reg, raw):
        transp_proto, ip1, port1, ip2, port2, app_proto = captures
        ip1 = ip_from_string(ip1)
        ip2 = ip_from_string(ip2)
        port1 = int(port1)
        port2 = int(port2)
        key = (transp_proto.lower(),
            frozenset(((ip1, port1), (ip2, port2))))
```

```
            flows[key] = []
            apps[key] = app_proto.split(".")
            if len(apps[key]) == 1:
                apps[key].append(None)

        for ts, raw in dpkt.pcap.Reader(open(pcapfile, "rb")):
            eth = dpkt.ethernet.Ethernet(raw)
            ip = eth.data
            if not isinstance(ip, dpkt.ip.IP):
                continue
            seg = ip.data
            if isinstance(seg, dpkt.tcp.TCP):
                transp_proto = "tcp"
            elif isinstance(seg, dpkt.udp.UDP):
                transp_proto = "udp"
            else:
                continue
            key = (transp_proto, frozenset(((ip.src, seg.sport),
                (ip.dst, seg.dport))))
            try:
                assert key in flows
            except AssertionError:
                print repr(ip.src)
                raise
            flows[key].append(eth)
        for key, flow in flows.items():
            yield apps[key][0], apps[key][1], flow

    def forge_flow_stats(flow, strip = 0):
        ip = flow[0].data
        seg = ip.data
        # Если пакет TCP
        if isinstance(seg, dpkt.tcp.TCP):
            try:
```

```
            seg2 = flow[1].data.data
        except IndexError:
            return None
        if not (seg.flags & dpkt.tcp.TH_SYN and seg2.flags & dpkt.tcp.TH_SYN):
            return None
        proto = "tcp"
        flow = flow[3:]
    elif isinstance(seg, dpkt.udp.UDP):
        proto = "udp"
    else:
        raise ValueError("Unknown transport protocol: `{}`".format(
            seg.__class__.__name__))

    if strip > 0:
        flow = flow[:strip]

    client = (ip.src, seg.sport)
    server = (ip.dst, seg.dport)

    client_bulks = []
    server_bulks = []
    client_packets = []
    server_packets = []
    cur_bulk_size = 0
    cur_bulk_owner = "client"
    client_fin = False
    server_fin = False
    for eth in flow:
        ip = eth.data
        seg = ip.data
        if (ip.src, seg.sport) == client:
            if client_fin: continue
            if proto == "tcp":
                client_fin = bool(seg.flags & dpkt.tcp.TH_FIN)
```

```
            client_packets.append(len(seg))
            if cur_bulk_owner == "client":
                cur_bulk_size += len(seg.data)
            elif len(seg.data) > 0:
                server_bulks.append(cur_bulk_size)
                cur_bulk_owner = "client"
                cur_bulk_size = len(seg.data)
        elif (ip.src, seg.sport) == server:
            if server_fin: continue
            if proto == "tcp":
                server_fin = bool(seg.flags & dpkt.tcp.TH_FIN)
            server_packets.append(len(seg))
            if cur_bulk_owner == "server":
                cur_bulk_size += len(seg.data)
            elif len(seg.data) > 0:
                client_bulks.append(cur_bulk_size)
                cur_bulk_owner = "server"
                cur_bulk_size = len(seg.data)
        else:
            raise ValueError("There is more than one flow here!")

    if cur_bulk_owner == "client":
        client_bulks.append(cur_bulk_size)
    else:
        server_bulks.append(cur_bulk_size)
    stats = {
        "bulk0": client_bulks[0] if len(client_bulks) > 0 else 0,
        "bulk1": server_bulks[0] if len(server_bulks) > 0 else 0,
        "bulk2": client_bulks[1] if len(client_bulks) > 1 else 0,
        "bulk3": server_bulks[1] if len(server_bulks) > 1 else 0,
        "client_packet0": client_packets[0] if len(client_packets) > 0 else 0,
        "client_packet1": client_packets[1] if len(client_packets) > 1 else 0,
        "server_packet0": server_packets[0] if len(server_packets) > 0 else 0,
        "server_packet1": server_packets[1] if len(server_packets) > 1 else 0,
```

```
}

if client_bulks and client_bulks[0] == 0:
    client_bulks = client_bulks[1:]

if not client_bulks or not server_bulks:
    return None

stats.update({
    "client_bulksize_avg": np.mean(client_bulks),
    "client_bulksize_dev": np.std(client_bulks),
    "server_bulksize_avg": np.mean(server_bulks),
    "server_bulksize_dev": np.std(server_bulks),
    "client_packetsize_avg": np.mean(client_packets),
    "client_packetsize_dev": np.std(client_packets),
    "server_packetsize_avg": np.mean(server_packets),
    "server_packetsize_dev": np.std(server_packets),
    "client_packets_per_bulk": len(client_packets)/len(client_bulks),
    "server_packets_per_bulk": len(server_packets)/len(server_bulks),
    "client_effeciency": sum(client_bulks)/sum(client_packets),
    "server_efficiency": sum(server_bulks)/sum(server_packets),
    "byte_ratio": sum(client_packets)/sum(server_packets),
    "payload_ratio": sum(client_bulks)/sum(server_bulks),
    "packet_ratio": len(client_packets)/len(server_packets),
    "client_bytes": sum(client_packets),
    "client_payload": sum(client_bulks),
    "client_packets": len(client_packets),
    "client_bulks": len(client_bulks),
    "server_bytes": sum(server_packets),
    "server_payload": sum(server_bulks),
    "server_packets": len(server_packets),
    "server_bulks": len(server_bulks),
    "is_tcp": int(proto == "tcp")
})
```

```
    return stats


def main():
    parser = argparse.ArgumentParser()
    parser.add_argument("file", nargs="+", help="pcap file")
    parser.add_argument("-o", "--output", help="output csv file", default="flows.csv")
    parser.add_argument("-s", "--strip", help="leave only first N datagramms", metavar = "N",
default=0, type=int)
    args = parser.parse_args()
    flows = {feature: [] for feature in FEATURES}
    for pcapfile in args.file:
        if len(args.file) > 1:
            print pcapfile
        for proto, subproto, flow in parse_flows(pcapfile):
            stats = forge_flow_stats(flow, args.strip)
            if stats:
                stats.update({"proto": proto, "subproto": subproto})
                for feature in FEATURES:
                    flows[feature].append(stats[feature])
    data = ps.DataFrame(flows)
    data.to_csv(args.output, index=False)
if __name__ == "__main__":
    main()
```

Приложение Б

model.py

```
import argparse, pickle

import pandas as ps
import numpy as np
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.metrics import classification_report

def preprocess(data, seed=None):
    drop_protos = ["Unknown", "Unencryped_Jabber", "NTP", "Apple"]
    replace_protos = [("SSL_No_Cert", "SSL")]
    data = data[~data["proto"].isin(drop_protos)]
    for old_proto, new_proto in replace_protos:
        data = data.replace(old_proto, new_proto)

    if seed:
        np.random.seed(seed)
        data = data.iloc[np.random.permutation(len(data))]

    scaler = StandardScaler()
    labeler = LabelEncoder()
    X = scaler.fit_transform(data.drop(["proto", "subproto"], axis=1))
    y = labeler.fit_transform(data["proto"])

    cols = [col for col in data.columns if col not in ("proto", "subproto")]
    return ps.concat([ps.DataFrame(X, columns=cols),
            ps.DataFrame({"proto": y})], axis=1), scaler, labeler

def split_data(data):
    proto_clusters = [data[data["proto"] == proto] for proto in data["proto"].unique()]
    clusters = [[], [], []]
    for cluster in proto_clusters:
```

```
        split_index = len(cluster)//3
        for i in range(3):
            clusters[i].append(
                cluster.iloc[i*split_index : (i+1)*split_index])
    return [ps.concat(clus) for clus in clusters]

def train_model(data_train, seed=None):
    X_train = data_train.drop(["proto"], axis=1)
    y_train = data_train["proto"]
    model = RandomForestClassifier(32, "entropy", 8, random_state=seed)
    model.fit(X_train, y_train)
    return model

def score_model(model, data_test, labeler):
    X_test = data_test.drop(["proto"], axis=1)
    y_test = data_test["proto"]
    y_predicted = model.predict(X_test)

    true_labels = labeler.inverse_transform(y_test)
    predicted_labels = labeler.inverse_transform(y_predicted)

    print feature_importances_report(model, X_test.columns)
    print "\n", classification_report(true_labels, predicted_labels)
    print cross_class_report(true_labels, predicted_labels)

def cross_class_report(y, p):
    classes = np.unique(y)
    res = ps.DataFrame({"y": y, "p": p}, index=None)
    table = ps.DataFrame(index=classes, columns=classes)
    for true_cls in classes:
        tmp = res[res["y"] == true_cls]
        for pred_cls in classes:
            table[pred_cls][true_cls] = len(tmp[tmp["p"] == pred_cls])
    return table
```

```
def feature_importances_report(model, columns):
    imp = {col: imp for imp, col
        in zip(model.feature_importances_, columns)}
    assert len(imp) == len(columns)
    return "\n".join("{}{:.4f}".format(str(col).ljust(25), imp)
        for col, imp in sorted(imp.items(), key=(lambda x: -x[1])))

def main():
    parser = argparse.ArgumentParser()
    parser.add_argument("file", help="csv file")
    parser.add_argument("-o", "--output", help="output model into file", metavar="FILE")
    parser.add_argument("-r", "--random", help="random seed value", metavar="SEED",
type=int)
    args = parser.parse_args()

    data = ps.read_csv(args.file)
    data, scaler, labeler = preprocess(data, args.random)

    clusters = split_data(data)
    for i, data_train in enumerate(clusters):
        print("\n\n*** FOLD: {} ***\n".format(i+1))
        data_test = ps.concat([c for c in clusters if c is not data_train])
        model = train_model(data_train, args.random)
        score_model(model, data_test, labeler)

    if args.output:
        pickle.dump((model, scaler, labeler), open(args.output, "wb"))
        print "\nModule successfully written to file '{}'.".format(args.output)

if __name__ == "__main__":
    main()
```

Приложение В

Приложение с графическим интерфейсом

main.py

```
import sys
from PyQt4 import QtGui

from dialogs.mainwindow import MainWindow

def main():
    app = QtGui.QApplication(sys.argv)
    window = MainWindow()
    window.show()
    sys.exit(app.exec_())

if __name__ == "__main__":
    main()
```

mainwindow.py

```
import pickle, os, subprocess

import sklearn, dpkt, pcapy

from PyQt4 import QtGui
from ui.ui_mainwindow import Ui_MainWindow

from models.trafficmodel import TrafficModel

from capthread import CapThread

S10_FILE = "model_s10.mdl"
S100_FILE = "model_s100.mdl"
```

```
S1000_FILE = "model_s1000.mdl"

class MainWindow(QtGui.QMainWindow):
    def __init__(self, parent = None):
        super(MainWindow, self).__init__(parent)
        self.ui = Ui_MainWindow()
        self.ui.setupUi(self)

        self.prediction_models = []
        for num, file in zip((10, 100, 1000), (S10_FILE, S100_FILE, S1000_FILE)):
            model, scaler, labeler = pickle.load(open(file, "rb"))
            self.prediction_models.append({"num": num, "model": model, "scaler": scaler, "labeler":
labeler})

        self.ui.action_start.triggered.connect(self.startCapture)
        self.ui.action_stop.triggered.connect(self.stopCapture)
        self.ui.tableView.doubleClicked.connect(self.viewDoubleClicked)
        self.thread = None

        self.model = TrafficModel(self.prediction_models, self)
        self.ui.tableView.setModel(self.model)
        self.ui.tableView.resizeColumnsToContents()

        ## Иконки:
        self.setWindowIcon(QtGui.QIcon("res/sherlock.png"))
        self.ui.action_start.setIcon(QtGui.QIcon("res/start.png"))
        self.ui.action_stop.setIcon(QtGui.QIcon("res/pause.png"))

    def startCapture(self):
        all_devs = sorted(pcapy.findalldevs())
        dev, ok = QtGui.QInputDialog.getItem(self, u"Device selection", u"Select device:",
all_devs, 0, False)
        if not ok: return
```

```
        self.model = TrafficModel(self.prediction_models, self)
        self.ui.tableView.setModel(self.model)
        self.ui.tableView.resizeColumnsToContents()

        self.ui.action_start.setEnabled(False)
        self.ui.action_stop.setEnabled(True)
        self.thread = CapThread(str(dev))
        self.thread.framesReceived.connect(self.addFrames)
        self.thread.start()

    def stopCapture(self):
        self.ui.action_start.setEnabled(True)
        self.ui.action_stop.setEnabled(False)
        self.thread.terminate()
        self.thread = None

    def addFrames(self, raws):
        flows = {}
        for raw in raws:
            eth = dpkt.ethernet.Ethernet(raw)
            if eth.type != dpkt.ethernet.ETH_TYPE_IP:
                continue
            ip = eth.data
            if ip.p == dpkt.ip.IP_PROTO_TCP:
                proto = "TCP"
            elif ip.p == dpkt.ip.IP_PROTO_UDP:
                proto = "UDP"
            else:
                continue
            seg = ip.data
            flow_key = (frozenset([(ip.src, seg.sport), (ip.dst, seg.dport)]), proto)
            if flow_key not in flows:
                flows[flow_key] = []
            flows[flow_key].append(eth)
```

```
        self.model.addFlows(flows)
        self.ui.tableView.resizeColumnsToContents()

    def viewDoubleClicked(self, index):
        flow = self.model.getFlow(index)
        outfile = dpkt.pcap.Writer(open("/tmp/dump.pcap", "wb"))
        for i, eth in enumerate(flow["dump"]):
            outfile.writepkt(eth, i)
        outfile.close()
        subprocess.call("echo 'wireshark /tmp/dump.pcap' | runuser mikhail &", shell=True)
```

trafficmodel.py

```
from PyQt4 import QtCore

import time

import dpkt
import pandas as ps

from util import ip_address, forge_flow

class TrafficModel(QtCore.QAbstractTableModel):
    def __init__(self, models, parent = None):
        super(TrafficModel, self).__init__(parent)
        self.models = models
        self.flows = []
        self.flows_index = {}
        self.idlers = []
        self.timer = QtCore.QTimer()
        self.timer.timeout.connect(self.timerTick)
        self.timer.start(1000)

    def timerTick(self):
```

```
        self.dataChanged.emit(self.index(0, 3), self.index(self.rowCount()-1, 3))
        new_idlers = []
        now = time.time()
        for flow in self.idlers:
            if flow["type"] == u"Not determined" or flow["type"] == u"Unknown":
                if len(flow["dump"]) > 1 and now - flow["ts_started"] > 1:
                    flow["type"] = self.flowType(flow, self.models[0])
                else:
                    new_idlers.append(flow)
        self.idlers = new_idlers

    def addFlows(self, flows):
        for key, frames in flows.iteritems():
            if key not in self.flows_index:
                eth = frames[0]
                ip = eth.data
                seg = ip.data
                proto = key[1]
                flow = {
                    "client_ip": ip.src,
                    "client_port": seg.sport,
                    "server_ip": ip.dst,
                    "server_port": seg.dport,
                    "proto": proto,
                    "ts_started": time.time(),
                    "dump": [],
                    "packets": 0,
                    "payload": 0,
                    "type": u"Not determined",
                    "last_payload": "client",
                    "index": len(flows)
                }
                if isinstance(seg, dpkt.tcp.TCP) and not (seg.flags & dpkt.tcp.TH_SYN):
                    continue
```

```
            self.beginInsertRows(QtCore.QModelIndex(), 0, 0)
            self.flows.append(flow)
            self.flows_index[key] = flow
            self.idlers.append(flow)
            self.endInsertRows()

        flow = self.flows_index[key]
        flow["packets"] += len(frames)
        flow["dump"] += frames
        for eth in frames:
            ip = eth.data
            seg = ip.data
            flow["payload"] += len(seg.data)
        self.checkThreshold(flow, flow["packets"] - len(frames))

        self.dataChanged.emit(self.index(self.rowCount()-flow["index"]-1, 0),
            self.index(flow["index"], self.columnCount()-1))

    def getFlow(self, index):
        return self.flows[self.rowCount()-index.row()-1]

    def checkThreshold(self, flow, old_len):
        new_len = flow["packets"]
        for mdl in self.models:
            if old_len < mdl["num"] <= new_len:
                flow["type"] = self.flowType(flow, mdl)
                break

    def flowType(self, flow, model):
        stats = forge_flow(flow["dump"], model["num"])
        if not stats: return u"Unknown"
        data = ps.DataFrame({key: [val] for key, val in stats.iteritems()})
        X = model["scaler"].transform(data)
        y = model["model"].predict(X)
```

```
        labels = model["labeler"].inverse_transform(y)
        return labels[0]

    def get_size_string(self, bts):
        if bts < 1024:
            return u"{} bytes".format(bts)
        if bts < 2**20:
            return u"{:.02f} Kbytes".format(bts/1024.0)
        else:
            return u"{:.02f} Mbytes".format(float(bts)/2**20)

    def columnCount(self, index = None):
      return 7

    def rowCount(self, index = QtCore.QModelIndex()):
      return len(self.flows)

    def data(self, index, role = QtCore.Qt.DisplayRole):
        if role == QtCore.Qt.DisplayRole:
            row, col = index.row(), index.column()
            flow = self.flows[-row-1]
            return [
                "{}:{}".format(ip_address(flow["client_ip"]), flow["client_port"]),
                "{}:{}".format(ip_address(flow["server_ip"]), flow["server_port"]),
                flow["proto"].upper(), u"{} sec".format(int(time.time() - flow["ts_started"])),
                flow["packets"], self.get_size_string(flow["payload"]),
                flow["type"]
            ][col]

    def headerData(self, col, orientation, role = QtCore.Qt.DisplayRole):
        if orientation == QtCore.Qt.Horizontal and role == QtCore.Qt.DisplayRole:
            return [u"Client", u"Server", u"Protocol", u"Duration", u"Packages", u"Of data",
u"Traffic type"][col]
```

```
    def flags(self, index):
        if index.isValid():
            flags = QtCore.Qt.ItemIsEnabled | QtCore.Qt.ItemIsSelectable
            return flags
```

ui_mainwindow.py

```
from PyQt4 import QtCore, QtGui

try:
    _fromUtf8 = QtCore.QString.fromUtf8
except AttributeError:
    def _fromUtf8(s):
        return s

try:
    _encoding = QtGui.QApplication.UnicodeUTF8
    def _translate(context, text, disambig):
        return QtGui.QApplication.translate(context, text, disambig, _encoding)
except AttributeError:
    def _translate(context, text, disambig):
        return QtGui.QApplication.translate(context, text, disambig)

class Ui_MainWindow(object):
    def setupUi(self, MainWindow):
        MainWindow.setObjectName(_fromUtf8("MainWindow"))
        MainWindow.resize(850, 600)
        self.centralwidget = QtGui.QWidget(MainWindow)
        self.centralwidget.setObjectName(_fromUtf8("centralwidget"))
        self.verticalLayout = QtGui.QVBoxLayout(self.centralwidget)
        self.verticalLayout.setMargin(0)
        self.verticalLayout.setSpacing(0)
        self.verticalLayout.setObjectName(_fromUtf8("verticalLayout"))
        self.tableView = QtGui.QTableView(self.centralwidget)
```

```
        self.tableView.setEditTriggers(QtGui.QAbstractItemView.NoEditTriggers)
        self.tableView.setSelectionMode(QtGui.QAbstractItemView.SingleSelection)
        self.tableView.setSortingEnabled(False)
        self.tableView.setObjectName(_fromUtf8("tableView"))
        self.tableView.verticalHeader().setVisible(False)
        self.verticalLayout.addWidget(self.tableView)
        MainWindow.setCentralWidget(self.centralwidget)
        self.toolBar = QtGui.QToolBar(MainWindow)
        self.toolBar.setObjectName(_fromUtf8("toolBar"))
        MainWindow.addToolBar(QtCore.Qt.TopToolBarArea, self.toolBar)
        self.action_start = QtGui.QAction(MainWindow)
        self.action_start.setObjectName(_fromUtf8("action_start"))
        self.action_stop = QtGui.QAction(MainWindow)
        self.action_stop.setEnabled(False)
        self.action_stop.setObjectName(_fromUtf8("action_stop"))
        self.toolBar.addAction(self.action_start)
        self.toolBar.addAction(self.action_stop)

        self.retranslateUi(MainWindow)
        QtCore.QMetaObject.connectSlotsByName(MainWindow)

    def retranslateUi(self, MainWindow):
        MainWindow.setWindowTitle(_translate("MainWindow", "Traffic capture", None))
        self.toolBar.setWindowTitle(_translate("MainWindow", "toolBar", None))
        self.action_start.setText(_translate("MainWindow", "Start", None))
        self.action_stop.setText(_translate("MainWindow", "Stop", None))
```

capthread.py

```
from PyQt4.QtCore import QThread, pyqtSignal

import pcapy, time

class CapThread(QThread):
```

```
    framesReceived = pyqtSignal(object)

    def __init__(self, dev, parent=None):
        super(CapThread, self).__init__(parent)
        self.dev = dev
        self.exiting = False

    def __del__(self):
        self.exiting = True
        self.wait()

    def run(self):
        pcap = pcapy.open_live(self.dev, 65536, False, 100)
        pcap.setnonblock(1)
        while not self.exiting:
            frames = [1]
            while frames[-1]:
                try:
                    frames.append(pcap.next()[1])
                except Exception, e:
                    break
            if len(frames) > 2:
                self.framesReceived.emit(frames[1:-1])
            time.sleep(0.05)
```

util.py

```
from __future__ import division

import dpkt
import numpy as np

def ip_address(bytes):
    return ".".join([str(ord(b)) for b in bytes])
```

```
def forge_flow(frames, strip = None):
    ip = frames[0].data
    seg = ip.data
    if isinstance(seg, dpkt.tcp.TCP):
        proto = "tcp"
        frames = frames[3:]
    elif isinstance(seg, dpkt.udp.UDP):
        proto = "udp"
    else:
        raise ValueErrur("Unknown transport protocol: `{}`".format(
            seg.__class__.__name__))

    if strip:
        frames = frames[:strip]

    client = (ip.src, seg.sport)
    server = (ip.dst, seg.dport)

    client_bulks = []
    server_bulks = []
    client_packets = []
    server_packets = []

    cur_bulk_size = 0
    cur_bulk_owner = "client"
    client_fin = False
    server_fin = False
    for eth in frames:
        ip = eth.data
        seg = ip.data
        if (ip.src, seg.sport) == client:
            if client_fin: continue
            if proto == "tcp":
                client_fin = bool(seg.flags & dpkt.tcp.TH_FIN)
```

```
            client_packets.append(len(seg))
            if cur_bulk_owner == "client":
                cur_bulk_size += len(seg.data)
            elif len(seg.data) > 0:
                server_bulks.append(cur_bulk_size)
                cur_bulk_owner = "client"
                cur_bulk_size = len(seg.data)
        elif (ip.src, seg.sport) == server:
            if server_fin: continue
            if proto == "tcp":
                server_fin = bool(seg.flags & dpkt.tcp.TH_FIN)
            server_packets.append(len(seg))
            if cur_bulk_owner == "server":
                cur_bulk_size += len(seg.data)
            elif len(seg.data) > 0:
                client_bulks.append(cur_bulk_size)
                cur_bulk_owner = "server"
                cur_bulk_size = len(seg.data)
        else:
            raise ValueError("There is more than one flow!")

    if cur_bulk_owner == "client":
        client_bulks.append(cur_bulk_size)
    else:
        server_bulks.append(cur_bulk_size)

    flow = {
        "bulk0": client_bulks[0] if len(client_bulks) > 0 else 0,
        "bulk1": server_bulks[0] if len(server_bulks) > 0 else 0,
        "bulk2": client_bulks[1] if len(client_bulks) > 1 else 0,
        "bulk3": server_bulks[1] if len(server_bulks) > 1 else 0,
        "client_packet0": client_packets[0] if len(client_packets) > 0 else 0,
        "client_packet1": client_packets[1] if len(client_packets) > 1 else 0,
        "server_packet0": server_packets[0] if len(server_packets) > 0 else 0,
```

```
    "server_packet1": server_packets[1] if len(server_packets) > 1 else 0,
}

if client_bulks and client_bulks[0] == 0:
    client_bulks = client_bulks[1:]

if not client_bulks or not server_bulks:
    return None

flow.update({
    "client_bulksize_avg": np.mean(client_bulks),
    "client_bulksize_dev": np.std(client_bulks),
    "server_bulksize_avg": np.mean(server_bulks),
    "server_bulksize_dev": np.std(server_bulks),
    "client_packetsize_avg": np.mean(client_packets),
    "client_packetsize_dev": np.std(client_packets),
    "server_packetsize_avg": np.mean(server_packets),
    "server_packetsize_dev": np.std(server_packets),
    "client_packets_per_bulk": len(client_packets)/len(client_bulks),
    "server_packets_per_bulk": len(server_packets)/len(server_bulks),
    "client_effeciency": sum(client_bulks)/sum(client_packets),
    "server_efficiency": sum(server_bulks)/sum(server_packets),
    "byte_ratio": sum(client_packets)/sum(server_packets),
    "payload_ratio": sum(client_bulks)/sum(server_bulks),
    "packet_ratio": len(client_packets)/len(server_packets),
    "client_bytes": sum(client_packets),
    "client_payload": sum(client_bulks),
    "client_packets": len(client_packets),
    "client_bulks": len(client_bulks),
    "server_bytes": sum(server_packets),
    "server_payload": sum(server_bulks),
    "server_packets": len(server_packets),
    "server_bulks": len(server_bulks),
    #"proto": int(proto == "tcp")
```

```
        "is_tcp": int(proto == "tcp")
    })

    return                                                                    flow
```

mainwindow.ui

```
<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
 <class>MainWindow</class>
 <widget class="QMainWindow" name="MainWindow">
  <property name="geometry">
   <rect>
    <x>0</x>
    <y>0</y>
    <width>850</width>
    <height>600</height>
   </rect>
  </property>
  <property name="windowTitle">
   <string>Traffic capture</string>
  </property>
  <widget class="QWidget" name="centralwidget">
   <layout class="QVBoxLayout" name="verticalLayout">
    <property name="spacing">
     <number>0</number>
    </property>
    <property name="leftMargin">
     <number>0</number>
    </property>
    <property name="topMargin">
     <number>0</number>
    </property>
    <property name="rightMargin">
```

```
  <number>0</number>
 </property>
 <property name="bottomMargin">
  <number>0</number>
 </property>
 <item>
  <widget class="QTableView" name="tableView">
   <property name="editTriggers">
    <set>QAbstractItemView::NoEditTriggers</set>
   </property>
   <property name="selectionMode">
    <enum>QAbstractItemView::SingleSelection</enum>
   </property>
   <property name="sortingEnabled">
    <bool>false</bool>
   </property>
   <attribute name="verticalHeaderVisible">
    <bool>false</bool>
   </attribute>
  </widget>
 </item>
</layout>
</widget>
<widget class="QToolBar" name="toolBar">
 <property name="windowTitle">
  <string>toolBar</string>
 </property>
 <attribute name="toolBarArea">
  <enum>TopToolBarArea</enum>
 </attribute>
 <attribute name="toolBarBreak">
  <bool>false</bool>
 </attribute>
 <addaction name="action_start"/>
```

```
  <addaction name="action_stop"/>
 </widget>
 <action name="action_start">
  <property name="text">
   <string>Start</string>
  </property>
 </action>
 <action name="action_stop">
  <property name="enabled">
   <bool>false</bool>
  </property>
  <property name="text">
   <string>Stop</string>
  </property>
 </action>
</widget>
<resources/>
<connections/>
</ui>
```