Федеральное государственное бюджетное образовательное учреждение   
высшего образования

|  |  |
| --- | --- |
| Gerb-BMSTU_01 | «Московский государственный технический университет  им. Н.Э. Баумана (национальный исследовательский университет)» (МГТУ им. Н.Э. Баумана) |

ФАКУЛЬТЕТ – Информатика и управления

КАФЕДРА – Информационные системы и телекоммуникации

РАСЧЁТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

По курсу

Разработка программного обеспечения

На тему

Разработка OSGI сервиса перехвата и анализа сетевых пакетов

Студентка группы ИУ3-73 (подпись) 28.12.2017 Коренева Н.Б.

Руководитель курсовой работы (подпись) 28.12.2017 Иванов А.М.

Содержание

[1 Введение 4](#_Toc506218633)

[2 Техническое задание 5](#_Toc506218634)

[3 Теоретическая часть 5](#_Toc506218635)

[3.1 Выявление заинтересованных сторон и их интересов 5](#_Toc506218636)

[3.2 Java Jpcap 6](#_Toc506218637)

[4 Конструкторская часть 6](#_Toc506218638)

[4.1 Выбор технических решений, удовлетворяющих интересам ЗС 6](#_Toc506218639)

[4.2 Структура проекта 7](#_Toc506218640)

[4.2.1 Плагин com.bmstu.jpcap 7](#_Toc506218641)

[4.2.2 Плагин com.bmstu.jpcap.client 8](#_Toc506218642)

[4.2.3 Плагин com.bmstu.jpcap.service 8](#_Toc506218643)

[4.3 Диаграмма компонентов 9](#_Toc506218644)

[4.4 Диаграмма классов 10](#_Toc506218645)

[5 Технологическая часть 11](#_Toc506218646)

[5.1 Запуск разработанного приложения 11](#_Toc506218647)

[5.2 Анализ исходного кода с помощью метрик качества 11](#_Toc506218648)

[5.3 Анализ зависимостей в коде системы 13](#_Toc506218649)

[5.4 Тестирование на корректность работы 13](#_Toc506218650)

[5.5 Оценка покрытия кода. 14](#_Toc506218651)

[6 Выводы 16](#_Toc506218652)

[7 Список источников 17](#_Toc506218653)

# 

# Введение

Задача анализа сетевого трафика приобретает все большую актуальность в

связи с развитием и внедрением новых сетевых технологий (и, как следствие, увеличением объема данных, передаваемых по сети), а также появлением большого количества новых сетевых протоколов прикладного уровня. В качестве наиболее популярных областей практического применения можно

выделить:

* анализ трафика с целью выявления проблем в работе сети (в том

числе, несанкционированной активности);

* восстановление потоков данных («прослушивание»);
* предотвращение различного рода сетевых атак;
* сбор статистики.

Если говорить о комплексном решении задачи анализа сетевого трафика, то в

первую очередь следует разделить ее на три подзадачи (рис. 1): перехват трафика, его хранение и анализ.

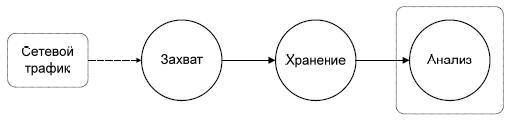


Рисунок 1. Задачи анализа сетевого трафика

Целью данной курсовой работы является разработка такого программного обеспечения, которое поможет произвести настройку правил перехвата и анализа сетевых пакетов.

# Техническое задание

Разработка OSGi сервиса перехвата и анализа сетевых пакетов:

* изучить соответствующую систему;
* спроектировать интерфейс компонента;
* реализовать компонент;
* спроектировать JUnit тесты, провести тестирование;
* описать требования, конструкцию, особенности сборки и запуска в документации.

# Теоретическая часть

## Выявление заинтересованных сторон и их интересов

В таблице ниже представлены результаты выявления и начального анализа заинтересованных сторон (ЗС) и их интересов по отношению к системе.

Таблица 1. Заинтересованные стороны и их интересы по отношению к системе

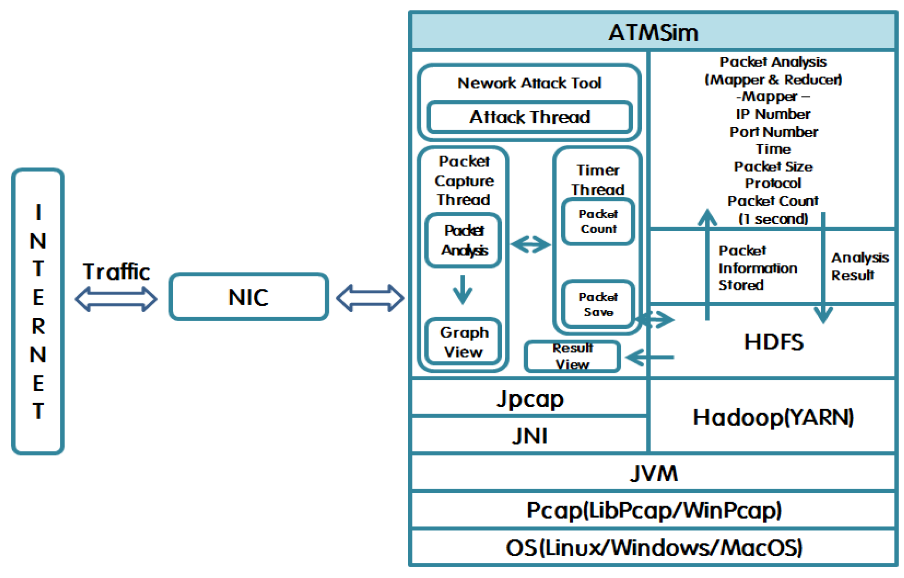
|  |  |
| --- | --- |
| **Заинтересованные стороны** | **Интересы заинтересованных сторон** |
| Системный администратор | Локализовать неисправность сети или ошибку конфигурации сетевых агентов;  Определить злоупотреблением трафика пользователями  (внутри системы так и снаружи нее);  Документирование нормативных требований;  Сбор сетевой статистики;  Фильтрация подозрительного контента, идущего от сетевого трафика. |
| Владелец опенсорсного проекта (project owner) | Быстрая и полная передача исходного кода, настроек, документов.  Возможность в дальнейшем совершенствовать систему, например, добавляя новые модули и возможности.  Для модульного тестирования будет использована система JUnit. |
| Быстрое внесение  изменений и проверка, что они не нарушают работу существующего  функционала. | Исходный код будет структурирован по пакетам.  Повторно-используемые методы будут вынесены в  родительские классы. |

## Java Jpcap

Jpcap - это набор классов Java, которые предоставляют интерфейс и систему для захвата сетевых пакетов. Библиотека протоколов и инструмент для визуализации сетевого трафика. Внутри Jpcap реализует привязки к системной библиотеке libpcap через JNI (Java Native Interface).

Jpcap использует широко развернутую общую библиотеку для захвата пакетов пользовательского уровня libpcap.

Jpcap состоит из небольшой разделяемой библиотеки, которая включает в себя libpcap и набор классов Java. Компонент разделяемой библиотеки обеспечивает событие перехвата, обмен данными между исполняемой виртуальной машиной Java и Libpcap. Пакет 'capture' содержит систему захвата ядра. Пакет «net» содержит абстракции для многих типов сетевых пакетов и протоколов. Пакет 'simulator' содержит сетевой симулятор.



# Конструкторская часть

## Выбор технических решений, удовлетворяющих интересам ЗС

В таблице ниже представлены результаты выбора технических решений, позволяющие удовлетворить интересы заинтересованных сторон по отношению к системе.

Таблица 2. Технические решения, удовлетворяющие интересам ЗС

|  |  |
| --- | --- |
| **Интересы заинтересованных сторон** | **Технические решения** |
| Добавление правил в конфигурационный  файл.  Удаление правил из конфигурационного  файла. | Новое правило заносится одним из классов разработанной системы в конфигурационный файл.  Существующее правило удаляется одним из классов разработанной системы в конфигурационном файле.  Корректность работы будет проверяться в процессе разработки с помощью юнит тестов. |
| Быстрая и полная передача исходного кода, настроек, документов.  Возможность в дальнейшем совершенствовать систему, например, добавляя новые модули и возможности | Код и настройки разрабатываемого менеджера будут находиться в системе GitHub. Контроль версий будет производиться с использованием системы Git.  Для модульного тестирования будет использована система JUnit. |
| Быстро понять принцип работы системы. | Написаны аннотации с помощью Javadoc |

## Структура проекта

Разберем работу написанной системы, для этого опишем состав каждого плагина, из которых состоит наша система:

Таблица 3. Разработанные плагины и их описание

|  |  |
| --- | --- |
| **Плагин** | **Содержимое** |
| com.bmstu.jpcap | Содержит интерфейсы и модельные объекты разрабатываемого ПО |
| com.bmstu.jpcap.client | Содержит реализацию клиентов |
| com.bmstu.jpcap.service | Содержит реализацию сервиса |

### Плагин com.bmstu.jpcap

Данный плагин содержит 2 пакета:

* com.bmstu.jpcap
* com.bmstu.jpcap.model

Пакет com.bmstu.jpcap содержит следующие интерфейсы:

* IJpcapClient
* IJpcapService
* IJpcapServiceFactory

которые включают в себя методы:

Таблица 4. Методы интерфейса IJpcapClient и их описание

|  |  |
| --- | --- |
| **Метод** | **Описание** |
| **void** processPacket(PcapPacket packet) | Обрабатывает события службы Jpcap |

Таблица 5. Методы интерфейса IJpcapService и их описание

|  |  |
| --- | --- |
| **Метод** | **Описание** |
| **void** addClient(IJpcapClient client) | Добавляет клиента к службе Jpcap |
| **void** removeClient(IJpcapClient client) | Удаляет клиента из службы Jpcap |
| **void** start() | Запускает службу Jpcap |
| **void** stop(); | Останавливает службу Jpcap |

Таблица 6. Методы интерфейса IJpcapServiceFactory и их описание

|  |  |
| --- | --- |
| **Метод** | **Описание** |
| getService(String deviceName) | Возвращает службу для данного имени устройства. |

Пакет com.bmstu.jpcap.model содержит 1 класс:

Таблица 7. Класс пакета com.bmstu.jpcap.model и его описание

|  |  |
| --- | --- |
| **Класс** | **Состав и описание** |
| PcapPacket | **private** Date date – для pcap javadoc  **private** int caplen – размер данных захвата пакета  **private** int wirelen – значения поля hdr\_wirelen  **public** PcapPacket(long millis, int caplen, int wirelen) – время пакета в миллисекундах, пакет caplen, пакет wirelen  **public** Date getDate() – возвращает дату пакета  **public** int getCaplen() – возвращает пакет caplen  **public** int getWirelen() – возвращает пакет wirelen |

### Плагин com.bmstu.jpcap.client

Данный плагин содержит пакет com.bmstu.jpcap.client, который включает в себя 1 класс:

Таблица 8. Класс пакета com.bmstu.jpcap.client и его описание

|  |  |
| --- | --- |
| **Класс** | **Состав и описание** |
| JpcapClient | Вывод информации о пакете  **public void** processPacket(PcapPacket packet) |

### Плагин com.bmstu.jpcap.service

Содержит пакет com.bmstu.jpcap.service, который включает в себя 3 класса:

* DeviceNamesPrinter
* JpcapFactory
* JpcapService

Таблица 9. Классы пакета com.bmstu.jpcap.service и их описание

|  |  |
| --- | --- |
| **Класс** | **Состав и описание** |
| DeviceNamesPrinter | Вывод имен устройств.  **public static void** main(String[] args) |
| JpcapFactory | Реализация JpcapServiceFactory  **private** Collection<IJpcapClient> clients  **private** Map<String, IJpcapService> deviceNameToServiceMap  **public** JpcapFactory() – конструктор;  **public void** bindClient(IJpcapClient client, Map<String, String> properties)  **private** PcapIf getPcapDevice(String deviceName)  **public** IJpcapService getService(String deviceName)  **public void** unbindClient(IJpcapClient client) |
| JpcapService | Реализация JpcapService  **private class** JPcapLoop implements Runnable  **private class** JPcapPacketHandler implements PcapPacketHandler<String>  **public void** nextPacket(PcapPacket packet, String userString)  **private void** sendEvent(com.bmstu.jpcap.model.PcapPacket packet)  **public** JpcapService(PcapIf device) – конструктор;  **public void** addClient(IJpcapClient client)  **public void** removeClient(IJpcapClient client)  **public void** start()  **private void** startPcap()  **public void** stop() |

## Диаграмма компонентов

На рисунке 2 представлена диаграмма компонентов нашего проекта.

Была разработана фабрика (IJpcapServiceFactory), которая ищет клиентов и по их параметрам даёт им сервисы (IJpcapService). Параметры клиентов записаны в аннотации компонента клиентов. В параметрах указано только название устройства, которое они хотят слушать. IJpcapService запускает поток, которой ловит сетевые пакеты при помощи Pcap. Когда IJpcapService ловит пакет, то всем клиентам, которые были переданы в этот сервис, посылается событие. Клиенты в этом событии обрабатывают пришедший пакет.

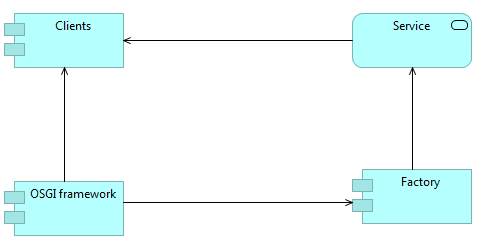


Рисунок 2. Диаграмма компонентов

## Диаграмма классов

Для наглядности, с помошью редактора UML Papirus была создана диаграмма классов. Функционал всех пакетов и их компонентов был описан выше.

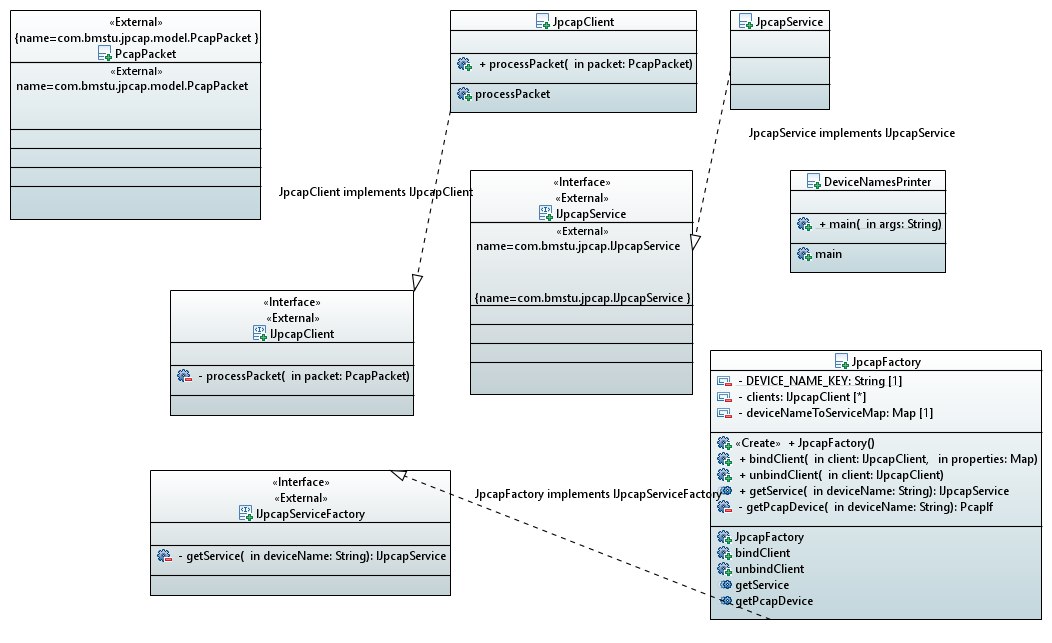


Рисунок 3. Диаграмма классов

# Технологическая часть

## Запуск разработанного приложения

Исходный код проекта доступен в репозитории GitHub.

Запуск приложения осуществляется с помощью конфигурационного файла, который содержит настройки, необходимые для успешного запуска приложения.

## Анализ исходного кода с помощью метрик качества

На рисунке 4 показано соотношение пакетов проекта по их размеру. Видно, что самый большой размер имеет пакет service.

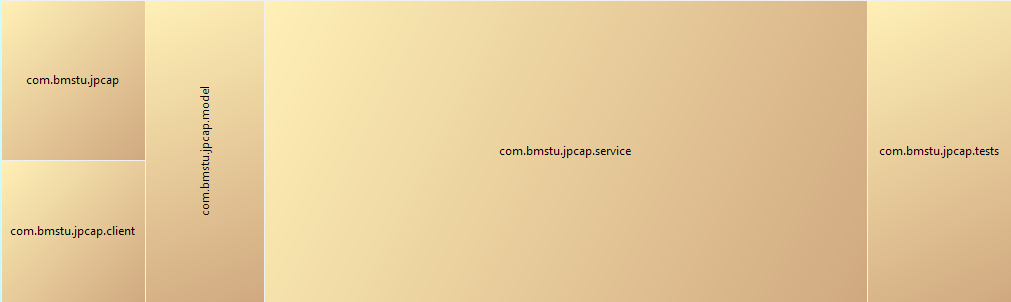


Рисунок 4. Соотношение пакетов по размеру

Далее на рисунке 5 отображен список всех метрик по разделам. Всего имеется четыре раздела:

* метрики количества (Count);
* метрики сложности (Complexity);
* метрики Роберта Мартина (Robert C. Martin);
* метрики Чидамбера-Кемерера (Chidamber & Kermerer).

Первый раздел с метриками количества (Count) содержит следующие метрики:

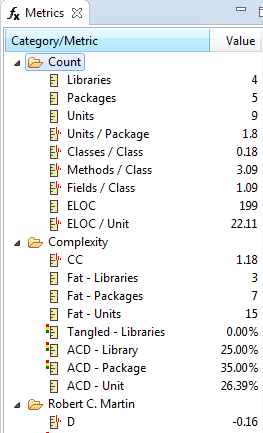
* количество классов верхнего уровня (Unit);
* среднее число внутренних классов на класс (Classes / Class);
* среднее число методов в классе (Methods / Class);
* среднее число полей в классе (Fields / Class);
* число строчек кода (ELOC);
* число строчек кода на модуль (ELOC / Unit).

Второй раздел с метриками сложности (Complexity) содержит всего три различных метрики:

* средняя циклическая сложность (CC);
* метрика Fat (Fat);
* средняя зависимость компонентов между модулями (ACD - Unit).

Третий раздел с метриками Роберта Мартина содержит следующие метрики:

* нормализованное расстояние от основной последовательности (D);
* абстрактность (A);
* нестабильность (I);
* число афферентных соединений (Ca);
* число эфферентных соединений (Ce).



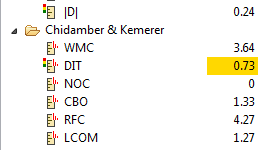


Рисунок 5. Значения метрик

Последний раздел с метриками Чидамбера-Кемерера содержит следующие метрики:

* средняя длина метода на класс (WMC);
* средняя глубина наследования (DIT);
* среднее количество классов-наследников (NOC);
* среднее число соединений класса (CBO);
* среднее число методов, которые потенциально могут быть выполнены в ответ на сообщение, полученное объектом этого класса (RFC);
* отсутствие единства методов (LCOM).

## Анализ зависимостей в коде системы

Рисунок 6 описывает зависимости между пакетами разработанного программного обеспечения:

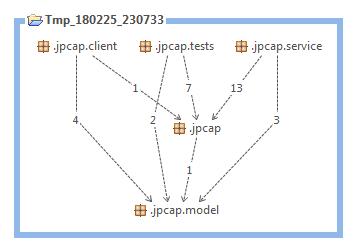


Рисунок 6. Зависимости между пакетами

## Тестирование на корректность работы

Для последующего тестирования разработанного программного обеспечения была поставлена задача написания Junit тестов.

В ходе выполнения курсовой работы было написано 2 теста:

* serviceTest
* clientTest

Связь Junit с OSGI происходит с помощью ServiceTracker. ServiceTracker позволяет получить сервисы из OSGI. Для его использования тест необходимо запускать как plugin test, иначе система не будет подгружать необходимые плагины.

Для проверки используется метод assertNotNull. Если сервис не доступен, то выводится соответствующее сообщение.

Результаты тестирования Junit приведены на рисунке 7.

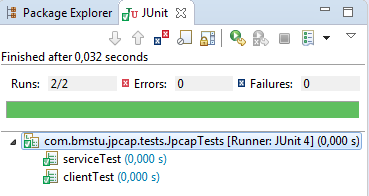


Рисунок 7. Результат работы тестов для проверки на корректность работы

## Оценка покрытия кода.

Покрытие кода – это метрика тестирования программного обеспечения, которая показывает, какие строки исходного кода были протестированы, а какие – нет. Чем больше покрытие, тем выше шанс, что программное обеспечение отработает без ошибок, однако стоит понимать, что высокий процент покрытия требует большего количества усилий и, как следствие, больших затрат ресурсов. Покрытие кода предоставляет важную информацию, позволяющую выявить важные участки кода, на которых стоит сосредоточить свои усилия, так как в дальнейшем эти участки могут быть модифицированы, что, вероятно, приведет возникновению новых ошибок.

На рисунке 8 приведен результат оценки покрытия текущего проекта.

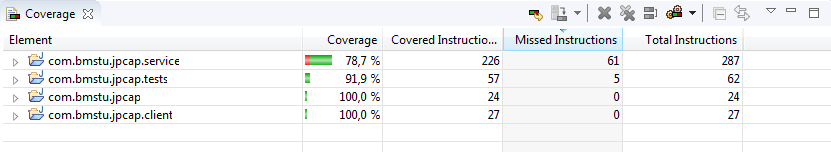


Рисунок 8. Оценка покрытия кода

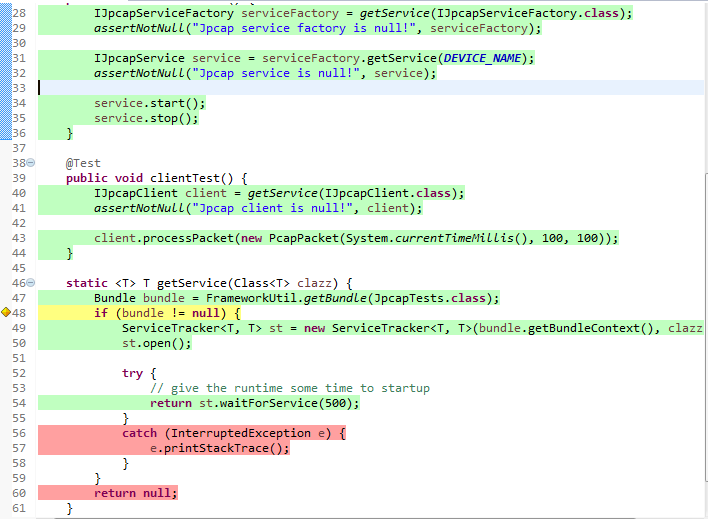


Рисунок 9. Графическое отображение пройденных участков кода

Зеленые участки – были пройдены при тестировании;

Желтые участки – были частично затронуты при тестировании. Такими участками обычно являются условные операторы;

Красные участки – код не был пройден при тестировании.

# Выводы

В ходе выполнения курсовой работы был спроектирован и реализован OSGI компонент перехвата и анализа сетевых пакетов. Работа была выполнена с помощью технологий OSGI и Junit в среде разработки Eclipse.

Была изучена библиотека протоколов и инструмент для визуализации сетевого трафика Jpcap, а также разработана архитектура приложения, структура классов, проведены необходимые тесты работоспособности программного продукта.

Разработанная программа устойчиво выполняет все свои функции.

# Список источников

[1] GitHub – Jpcap [Электронный ресурс]. Режим доступа: https://github.com/jpcap/jpcap

[2] OSGI modulation – Tutorial : [Электронный ресурс]. Режим доступа: http://www.vogella.com/tutorials/OSGi/article.html

[3] OSGI and Equinox : J. McAffer, 2010. – 328 с.

[4] OSGI services – Tutorial : [Электронный ресурс]. Режим доступа: http://www.vogella.com/tutorials/OSGiServices/article.html