МИНОБРНАУКИ РОССИИ

федеральное государственное автономное образовательное учреждение высшего образования «Самарский национальный исследовательский университет имени академика С.П. Королева»

(Самарский университет)

Институт \_информатики, математики и электроники

Факультет \_\_\_\_\_математики\_\_\_\_\_\_\_\_

Кафедра \_\_ безопасности информационных систем\_\_\_\_

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

**ПРОЕКТИРОВАНИЕ И РАЗРАБОТКА ВЕБ-ФИЛЬТРА ДЛЯ ОБЕСПЕЧЕНИЯ КОНТРОЛЯ ДОСТУПА К СЕТЕВЫМ РЕСУРСАМ**

по специальности 10.05.01 Компьютерная безопасность

(уровень специалитета)

специализация №7 «Информационно-аналитическая и техническая экспертиза компьютерных систем»

Студент Дмитрий Алексеевич Бизин

Руководитель ВКР,

доцент кафедры безопасности

информационных систем,

к.ф.-м.н., доцент Мария Ефимовна Федина

Нормоконтролер / /

Самара 2019

# СОДЕРЖАНИЕ

[СОДЕРЖАНИЕ 2](#_Toc533370770)

[ВВЕДЕНИЕ 3](#_Toc533370771)

[1 Анализ предметной области 5](#_Toc533370772)

[1.1 Веб-фильтр 6](#_Toc533370773)

[1.2 Методы контентной фильтрации 8](#_Toc533370774)

[1.3 Категоризация ресурсов сети интернет 12](#_Toc533370775)

[1.4 Методы автоматической классификации текстов 13](#_Toc533370776)

[2 Используемые технологии и алгоритмы 18](#_Toc533370777)

[2.1 Наивный байесовский классификатор 18](#_Toc533370778)

[2.1.1 Многомерная модель 20](#_Toc533370779)

[2.1.2 Мультиномиальная модель 21](#_Toc533370780)

[2.2 HTTP и HTTPS 23](#_Toc533370781)

[2.3 Trusted MITM 26](#_Toc533370782)

[2.4 Java, Jetty, Servlets, TCP-сокеты 29](#_Toc533370783)

[2.5 JavaScript и технология AJAX 32](#_Toc533370784)

[2.6 База данных H2 и JDBC 33](#_Toc533370785)

[3 Проектирование, разработка и тестирование работы веб-фильтра 34](#_Toc533370786)

[3.1 Проектирование 34](#_Toc533370787)

[3.2 Разработка 40](#_Toc533370788)

[3.3 Тестирование 41](#_Toc533370789)

[ЗАКЛЮЧЕНИЕ 42](#_Toc533370790)

[СПИСОК СОКРАЩЕННЫХ ОБОЗНАЧЕНИЙ 45](#_Toc533370791)

# ВВЕДЕНИЕ

Веб-фильтр (контент-фильтр) – это программное или программно-аппаратное устройство, которое ограничивает доступ к веб-сайтам на основании анализа их содержимого. Кроме анализа содержимого сайтов контент-фильтры также могут ограничивать доступ по *IP*-адресу веб-ресурса (черные списки сайтов).

В настоящее время контентную фильтрацию нельзя выделить в отдельную область компьютерной безопасности, настолько она переплелась с другими направлениями. В обеспечении компьютерной безопасности контентная фильтрация очень важна, поскольку позволяет вычленять потенциально опасные вещи и корректно их обрабатывать. Подходы, появившиеся при разработке продуктов для контентной фильтрации, находят применение, например, в продуктах для предотвращения вторжений (*Intrusion Detection System* (*IDS*)).

В последнее время в интернете все больше появляется противозаконного контента, ориентированного на школьников и подростков. Блокирование таких веб-ресурсов по *IP*-адресу не приносит особого результата, т.к. в основном эти сайты создаются сотнями каждый день, и требуется время для их обнаружения и внесения в черный список. В такой ситуации фильтрация на основании содержимого сайта будет давать наилучший результат. В связи с этим тема фильтрации (классификации) веб-контента очень актуальна в образовательных учреждениях. Также с целью соблюдения политики информационной безопасности на предприятиях можно использовать веб-фильтр для ограничения доступа сотрудников к нежелательному контенту (например, социальные сети), что добавляет актуальности теме контент-фильтрации.

Таким образом, целью работы является проектирование и разработка веб-фильтра для обеспечения контроля доступа к веб-ресурсам.

Для достижения поставленной цели были сформулированы следующие задачи:

* анализ предметной области и текущих решений;
* проектирование веб-фильтра (выбор используемых технологий, в том числе выбор алгоритма классификации текста);
* разработка веб-фильтра и тестирование его работы.

# Анализ предметной области

Одна из общих тенденций развития продуктов информационной безопасности — стремление реализовать различные функции в одном устройстве или программном решении. Как правило, разработчики стараются выполнить решения, которые кроме функций контентной фильтрации еще выполняют и функции антивируса, межсетевого экрана и/или системы обнаружения и предотвращения вторжений. С одной стороны, это позволяет снизить затраты компаний на покупку и сопровождение систем безопасности, но с другой — функциональность таких систем часто оказывается ограниченной. Например, во многих продуктах функции фильтрации веб-трафика сведены только к проверке адресов сайтов относительно какой-либо базы данных категорий сайтов.

В существующих уже достаточно давно областях контентной фильтрации — контроле почты и интернет-трафика — также происходят изменения, появляются новые технологии.

В продуктах для контроля почтового обмена стала выходить на первый план функция защиты от фишинга. А в продуктах для контроля интернет-трафика происходит смещение от использования заранее подготовленных баз данных *IP*/*URL* сайтов к категоризации по содержимому «на лету», что является очень актуальным решением, т.к. современные сайты могут часто изменять свой контент.

Кроме двух указанных выше областей, возникают и новые области применения контентной фильтрации — некоторое время назад начали появляться продукты для контроля за передачей мгновенных сообщений (*instant messaging*) и *peer-to-peer* (*p*2*p*) соединений [1]. В настоящее время также активно разрабатываются продукты для контроля за *VoIP*-трафиком [1].

Во многих странах стали активно развивать средства для перехвата и анализа многих видов информации, которая используется для различного вида расследований. Данные мероприятия проводятся на государственном уровне и наиболее часто привязываются к расследованию террористических угроз. Такие системы перехватывают и анализируют не только данные, передаваемые по каналам сети интернет, но также и по другим видам связи — телефонным линиям, радиоканалам и т.п. Наиболее известной системой для перехвата информации является *Echelon* — система, использовавшаяся американской разведкой для сбора информации [2]. В России также существуют различные реализации систем оперативно-розыскных мероприятий (СОРМ), которые используются для захвата и анализа информации в интересах спецслужб [3].

## Веб-фильтр

Как уже говорилось выше, веб-фильтр (контент-фильтр) – это программное или программно-аппаратное устройство, которое ограничивает доступ к веб-сайтам на основании анализа их содержимого. В данной работе будет рассмотрена программная реализация. По своей сути контент-фильтр – это веб прокси-сервер, который перехватывает *HTTP/HTTPS*-запросы и анализирует их.

Упрощенный алгоритм работы веб-фильтра по шагам:

1. Перехват *HTTP/HTTPS*-запроса пользователя к интернет-ресурсу.
2. Анализ запроса (проверка *IP*-адреса интернет-ресурса, анализ самого *HTTP/HTTPS-*запроса).
3. На основании анализа принять решение: позволить осуществить запрос к интернет-ресурсу или отклонить его.
4. Если было принято решение об отклонении запроса, то производится возврат пользователю информации об отклонении (например, веб-страница). Если же было решено пропустить запрос, то веб-фильтр делает запрос к ресурсу.
5. Перехват ответа от интернет-ресурса.
6. Анализ ответа (анализ возвращенного контента, классификация контента).
7. На основании анализа контента принять решение: пропустить ответ от ресурса пользователю или отклонить его.
8. Если было принято решение об отклонении ответа, то производится возврат пользователю информации об отклонении (например, веб-страница). Если же было решено пропустить ответ, то веб-фильтр возвращает ответ от интернет-ресурса пользователю.

На рисунке 1 показана схема работы веб прокси-сервера.



Рис. 1 – Схема работы веб-фильтра

В приведенном выше алгоритме умалчивается один факт: контент, передаваемый по протоколу *HTTPS*, зашифрован. Поэтому анализировать его без расшифрования не представляется возможным. Это существенное ограничение функциональности контент-фильтра, т.к. на данный момент в интернете сайты, работающие по протоколу *HTTPS*, преобладают над сайтами, работающими по незащищенному протоколу *HTTP*. Поэтому для того, чтобы веб-фильтр работал максимально эффективно, нужно расшифровывать контент с использованием поддельного сертификата и применением технологии *Trusted Man-In-The-Middle* (*Trusted MITM*)[4, 5], суть которой будет описана во втором разделе.

## Методы контентной фильтрации

Существует большое количество методов и алгоритмов контентной фильтрации. На практике обычно применяется совокупность методов, что необходимо для обеспечения лучших результатов работы веб-фильтра, поскольку различные методы позволяют достигать требуемых результатов только на определенном типе фильтруемых данных. Однако можно выделить три наиболее общих класса методов фильтрации [6]:

* фильтрация с использованием справочника ключевых слов;
* фильтрация на основе списков *IP*/*URL*;
* динамическая фильтрация.

Фильтрация по ключевым словам – наиболее простой способ фильтрации, поэтому он часто применяется как отдельно (в простых системах для домашнего использования), так и в комбинации с другими методами фильтрации. Этот метод позволяет включать блокировку страницы либо сайта целиком при наличии в них слов или словосочетаний из справочника. Метод прост в реализации и использовании, но имеет существенный недостаток: он может блокировать веб-ресурсы, в которых фильтруемые слова используются в другом контексте (избыточная фильтрация).

*IP*/*URL*-фильтрация позволяет блокировать веб-ресурсы по справочнику *IP*-адресов или *URL* (возможен смешанный режим). Справочник может наполняться как вручную, так и автоматически, на основе алгоритма предварительного анализа интернет-ресурсов. Также этот метод носит название черных списков.

Динамическая фильтрация – широкий класс методов, в которых содержимое интернет-ресурса анализируется в момент поступления ответа от него. Доступ к ресурсу блокируется, если его содержимое определяется как несоответствующее политике безопасности.

Основное отличие *IP*/*URL*-фильтрации и динамической фильтрации заключается в моменте анализа содержимого веб-ресурса. Согласно рисунку 1 *IP*/*URL*-фильтрация происходит после шага 1, перед отправкой запроса на сервер, где расположен сайт, а динамическая фильтрация происходит после шага 3, после получения ответа от сервера.

В таблицах 1 и 2 приведено сравнение наиболее распространенных систем контентной фильтрации по ряду наиболее важных критериев (по принципу наличия или отсутствия функциональности) [6]. Из обзора существующих решений можно сделать вывод, что имеются системы различного класса и различной функциональности, но нет единого подхода к реализации одних и тех же функциональных возможностей даже среди систем одного класса. Лидирующие системы контентной фильтрации основываются на принципе анализа и категоризации интернет-ресурсов, что признано наиболее эффективным методом фильтрации нежелательного контента. Эти системы используют регулярно обновляемые базы *URL*, гибкие настройки фильтра и развитые системы отчетности.

Все рассмотренные системы фильтрации либо не используют центральные базы данных (БД), либо используют закрытые БД интернет-ресурсов, свободный доступ к которым невозможен. Содержание таких БД обычно является наибольшей ценностью для компаний-разработчиков подобных систем.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Система контентной  фильтрации  Критерий | Дозор-Джет | Dans-  Guardian | Smooth-  Guardian | Cyber  Patrol | Cyber  Snoop |
| Фильтрация по *IP*/*URL* | Да | Да | Да | Да | Да |
| Фильтрация по терминам входящих данных | Да | Да | Да | Да | Да |
| Фильтрация по терминам исходящих данных | Да | Нет | Нет | Нет | Да |
| Фильтрация по портам | Да | Нет | Нет | Нет | Нет |
| Наличие локальной БД | Да | Да | Да | Да | Да |
| Наличие центральной БД | Да | Нет | Нет | Да | Да |
| Установка на рабочую станцию | Нет | Да | Да | Да | Да |
| Установка на шлюз | Да | Да | Да | Нет | Нет |
| Управление временем работы | Да | Нет | Да | Да | Нет |
| Сбор статистики | Да | Нет | Да | Да | Да |
| Встроенная поддержка русского языка | Да | Нет | Да | Нет | Нет |
| Наличие графического интерфейса | Нет | Нет | Нет | Да | Да |

Таблица 1 – Сравнение систем контентной фильтрации по критериям

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Система контентной  фильтрации  Критерий | Net  Nanny | Cyber  Sitter | Wizguard | Cyber  Sentinel |
| Фильтрация по *IP*/*URL* | Да | Да | Да | Да |
| Фильтрация по терминам входящих данных | Нет | Нет | Нет | Да |
| Фильтрация по терминам исходящих данных | Нет | Нет | Да | Да |
| Фильтрация по портам | Нет | Нет | Нет | Нет |
| Наличие локальной БД | Да | Да | Нет | Да |
| Наличие центральной БД | Да | Да | Да | Да |
| Установка на рабочую станцию | Да | Да | Да | Да |
| Установка на шлюз | Нет | Нет | Нет | Нет |
| Управление временем работы | Нет | Нет | Нет | Нет |
| Сбор статистики | Да | Да | Да | Да |
| Встроенная поддержка русского языка | Нет | Нет | Нет | Нет |
| Наличие графического интерфейса | Да | Да | Да | Да |

Таблица 2 – Сравнение систем контентной фильтрации по критериям

Также был проведен обзор отечественных систем контентной фильтрации: подсистемы контентной фильтрации антивируса *Dr. Web* и обзор Интернет Контроль Сервера (ИКС).

В антивирусе *Dr. Web* имеется возможность блокировки доступа к веб-сайтам по различным категориям (категории заданы в системе), блокировки на основании черных и белых списков и блокировки по *MIME*-типу. Также есть возможность написания своих правил разрешения/запрета доступа к веб-ресурсам.

В состав ИКС входит модуль *Kaspersky Anti-Virus*. ИКС полностью соответствует требованиям законодательства: решение имеет сертификат ФСТЭК, содержит встроенные категории трафика *SkyDNS*, списки Минюста, Госнаркоконтроля и специальный профиль для школ. Готовый набор правил для образовательных учреждений включает в себя запрет доступа к заведомо опасным ресурсам, сканирование трафика контент-фильтром, выдачу результатов запроса в поисковиках через поисковый запрос *SkyDNS*. Также имеется возможность задавать свои правила разрешения/запрета доступа к веб-ресурсам. Поддерживается фильтрация *HTTPS*-трафика без подмены сертификата.

## Категоризация ресурсов сети интернет

Все сайты сети интернет могут быть отнесены к одной или нескольким категориями. Категории задаются разработчиком системы контентной фильтрации. Можно выделить несколько способов категоризации ресурсов в сети интернет [6]:

* использование регулярно обновляемых баз данных категоризированных ресурсов (система категоризации работает со списком категорий, категоризирует новые ресурсы и обновляет связи между категориями и существующими ресурсами);
* категоризация контента «на лету» при помощи анализа содержимого страниц;
* использование данных о категории, информацию о принадлежности к которой предоставляет сам сайт.

Категоризация данных и формирование баз данных категорий обычно производится в полуавтоматическом режиме – сначала выполняются анализ содержимого и определение категории с помощью алгоритмов классификации. На втором этапе полученная информация часто проверяется людьми, принимающими решение о том, к какой категории можно отнести тот или иной сайт.

К преимуществам применения предопределенных баз категорий можно отнести то, что предоставление или запрет доступа производится еще до запроса к самому сайту, что может существенно снизить нагрузку на веб-фильтр. Главный же недостаток использования данного подхода — задержки в обновлении баз категорий сайтов, поскольку для анализа потребуется некоторое время. Кроме того, некоторые сайты достаточно часто меняют свое наполнение, из-за чего информация о категории, хранящаяся в базе, становится неактуальной.

Категоризация сайтов на лету также осуществляется самыми разными способами. Особенно часто используются методы, основанные на статистическом подходе к анализу содержания.

Один из простых вариантов реализации такого решения — использование байесовских алгоритмов, которые себя достаточно хорошо зарекомендовали в борьбе со спамом. Однако у этого варианта есть свои недостатки — необходимо периодически доучивать алгоритм, корректировать словари в соответствии с передаваемыми данными. Поэтому некоторые компании применяют более сложные алгоритмы определения категории сайта по содержимому в дополнение к простым способам.

Категоризация данных на лету позволяет быстро реагировать на появление новых сайтов, поскольку информация о категории сайта не зависит от его адреса, а только от содержания. Но такой подход имеет и недостатки — необходимо проводить анализ всех передаваемых данных, что вызывает некоторое снижение производительности веб-фильтра. Второй недостаток — необходимость поддержания актуальных баз категорий для различных языков. Тем не менее, некоторые продукты применяют этот подход с одновременным использованием баз данных категорий сайтов. Сюда можно отнести использование *Virtual Control Agent* в продуктах компании *SurfControl*, механизмы определения категорий данных в "Дозор-Джет".

## Методы автоматической классификации текстов

Следует отличать классификацию от кластеризации. При классификации документов категории определены заранее, при кластеризации они не заданы и даже информация об их количестве может отсутствовать.

Формально постановку задачи классификации можно определить следующим образом [7]. Имеются множество документов  и множество заранее заданных категорий (классов) . Неизвестная целевая функция  задается формулой (1):

 (1)

Требуется построить классификатор , максимально близкий к .

Если классификатор определяется формулой (1), то он называется точным. Если же классификатор возвращает значение из диапазона [0,1] (вероятность попадания документа  в категорию ), то он называется вероятностным.

В общем случае процесс классификации состоит из следующих шагов:

1. Предобработка.
2. Индексация.
3. Выбор признаков.
4. Построение и обучение классификатора.

На этапе предобработки текста происходит удаление не значащих для классификации слов (союзы, предлоги, частицы) и стемминг. Стемминг – это процесс нахождения основы слова. Таким образом, каждое слово будет заменено на свою основу. В результате проведенных выше операций значительно сокращается размерность пространства признаков (слов), что необходимо для дальнейшей классификации.

На этапе индексации происходит построение числовой модели текста, необходимой для дальнейшей работы. Одна из распространенных моделей индексации текста – *Word2Vec* [8] – представляет каждое слово в виде вектора, который содержит информацию о контекстных словах. Модель *bag-of-words* [9] представляет документ в виде вектора весов слов документа, т.е. координата вектора – это порядковый номер слова в документе, а значение координаты – это вес слова. Также есть модель, которая основана на учете *n*-грамм [9].

На этапе выбора признаков происходит определение весов признаков документа. Наиболее распространенный способ – вычисление функции *TF-IDF* [9, 10, 11]. Суть способа состоит в том, что больший вес получают слова с высокой частотой в пределах конкретного документа, но с низкой частотой употреблений в других документах. Частота термина  (*term frequency*) – оценка важности слова в пределах одного документа – вычисляется по формуле (2):

, (2)

где  – количество слов *t* в документе *d*;

 – общее число слов в документе *d*.

Обратная частота документа  (*inverse document frequency*) вычисляется по формуле (3):

, (3)

где  – общее количество документов;

 – количество документов, в которых встречается термин *t*.

Вес термина  в документе  относительно всех документов вычисляется по формуле (4):

**  (4)

На последнем этапе происходит построение классификатора и его обучение. Выделяют следующие методы классификации:

* вероятностные;
* метрические;
* логические;
* линейные;
* методы на основе искусственных нейронных сетей.

Вероятностные методы классификации текста используют аппарат теории вероятностей и математической статистики и рассматривают термины документа (слова) как события. Например, наивный байесовский классификатор делает «наивное» предположение о независимости любых двух слов документа и использует формулу Байеса для расчета вероятностей попадания документа в ту или иную категорию.

Метрические методы опираются на понятие метрики, введенной на пространстве признаков (слов) документа. Сходство между словами определяется величиной метрики между ними. Например, метод ближайших соседей использует гипотезу о компактности: близкие объекты в метрическом пространстве относятся к одному классу. Таким образом, объект относится к тому классу, к которому принадлежит большинство из  ближайших соседей в обучающей выборке.

Логические методы строят процесс классификации пошагово, т.е. на каждом шаге происходит проверка, обладает ли текущее слово каким-либо признаком. В результате  проверок слово попадает в какую-то категорию. Один из популярных алгоритмов данного метода – дерево принятия решений. В каждом узле дерева происходят те самые проверки. Листы дерева представляют из себя категории.

Линейные методы основываются на построении линейной разделяющей поверхности. Если имеется всего два класса, то линейный классификатор разделит пространство на два полупространства. В задачах со многими классами разделяющая поверхность кусочно-линейная. Рисунок 2 хорошо иллюстрирует линейную классификацию.



Рис. 2 – Пример линейной классификации

На рисунке 2 прямая  не является линейным классификатором, т.к. она неправильно разделяет множества на два класса. Прямые  и  являются линейными классификаторами, причем прямая  классифицирует более точно, т.к. она максимально отстоит от обоих множеств.

Методы на основе искусственных нейронных сетей используют нейронные сети для задач классификации. Существует большое количество разновидностей нейронных сетей (сети прямого распространения, рекуррентные сети, радиально-базисные функции, самоорганизующиеся карты), откуда появляется большое количество алгоритмов классификации.

# Используемые технологии и алгоритмы

В данном разделе описаны технологии, которые будут использованы для разработки веб-фильтра. В качестве алгоритма классификации текста предлагается использовать наивный байесовский классификатор.

## 2.1 Наивный байесовский классификатор

Наивный байесовский классификатор является одним из алгоритмов вероятностной классификации. Относится к категории алгоритмов машинного обучения с учителем.

Пусть дан документ , представленный в виде вектора слов . Также пусть заранее определено множество категорий  Задача классификатора заключается в том, чтобы подобрать такие значения  и , при которых значение  – вероятность того, что документ  принадлежит категории  – будет максимальным, т.е. необходимо посчитать значение по формуле (5):

 (5)

Для вычисления значений  используется формула (6) Байеса:

, (6)

где  – априорная вероятность того, что документ отнесен к категории 

 – вероятность найти документ  в категории ;

 – вероятность того, что документ  можно представить в виде вектора признаков (слов) .

 является отношением количества документов из обучающей выборки, отнесенных к категории , к общему числу документов.

 не зависит от категории , поэтому это значение константное и не влияет на выбор максимального из значений .

Т.к. в большинстве своем документ содержит большое количество слов (признаков), то вычисление значения  затруднительно. Поэтому делается «наивное» предположение о том, что любые два слова из документа  статистически не зависят друг от друга (два независимых события). Тогда для вычисления  можно воспользоваться формулой (7):

 (7)

Преимущества байесовского классификатора перед другими:

* высокая скорость работы;
* простая программная реализация алгоритма;
* легкая интерпретируемость результатов работы алгоритма.

К недостаткам относят низкое качество классификации и неспособность учитывать зависимость между признаками (словами). Но на практике байесовский классификатор показывает высокое качество классификации, и этому есть объяснения [12]:

* слова в документе в большинстве своем зависимы, но эта зависимость одинакова для разных классов и «взаимно сокращается» при оценке вероятностей;
* пусть на самом деле  и , а байесовский классификатор выдаст, что  и , но результат классификации от этого не изменится.

Существует два подхода [12] при реализации наивного байесовского классификатора, которые дают разные результаты: мультиномиальный и многомерный.

### 2.1.1 Многомерная модель

В многомерной модели документ представляется в виде бинарного вектора, показывающего, встретилось ли то или иное слово в документе. Например, пусть имеется обучающая выборка слов . Также пусть все слова из обучающей выборки  уже распределены по категориям . Тогда документ  представляет собой бинарный вектор: , где , если слово  содержится в документе, и  иначе. Формула (7) в данной модели приобретает вид формулы (8):

 (8)

Для обучения такого классификатора необходимо обучить вероятности . Пусть дан набор документов , которые уже распределены по классам . Также имеется обучающая выборка слов . Т.е. множество  разбито на подмножества (возможно, пересекающиеся) – документы  Тогда вероятности  рассчитываются по формуле (9):

, (9)

где  – бит  в документе .

В формуле (9) применено аддитивное сглаживание по Лапласу [13], суть которого заключается в том, чтобы избавиться от проблемы неизвестных слов и сделать вероятность ненулевой. Таким образом, даже если в документе не будет слов из обучающей выборки, этот документ получит ненулевые вероятности попадания в категории.

Априорные вероятности  можно посчитать по формуле (10):

 (10)

Классификация документа  проходит по следующей формуле (11):

(11)

Переход к логарифмам в последнем равенстве сохраняет монотонность, что не влияет на поиск максимума.

### 2.1.2 Мультиномиальная модель

В мультиномиальной модели документ – это последовательность независимых событий. При подсчете вероятности принадлежности документа  классу  происходит перемножение вероятностей того, что текущее слово документа принадлежит обучающей выборке, при условии, что все слова независимы между собой. Таким образом, получается модель, которая учитывает повторы слов в документе, но не учитывает, каких слов из обучающей выборки нет в документе.

Пусть  – обучающая выборка слов. Документ  – это вектор длины , состоящий из слов, каждое из которых с вероятностью  попадает в обучающую выборку. Тогда вероятность принадлежности документа  категории  определяется по формуле (12):

, (12)

где  – количество вхождений слова  в документ .

Для обучения классификатора необходимо обучить вероятности . Пусть как и предыдущем случае дан набор документов , которые уже распределены по классам . Также имеется обучающая выборка слов . Т.е. множество  разбито на подмножества (возможно, пересекающиеся) – документы . Тогда вероятности  рассчитываются по формуле (13):

 (13)

Априорные вероятности  можно посчитать как и раньше по формуле (10).

Тогда классификация документа  проходит по формуле (14):

 (14)

Как и в предыдущей модели переход к логарифмам в последнем равенстве сохраняет монотонность, что не влияет на поиск максимума.

## 2.2 HTTP и HTTPS

Веб-фильтр выполняет свою основную задачу (классификация контента) на уровне протокола *HTTP*. Поэтому далее будут рассмотрены основные моменты протокола, которые использует веб-фильтр.

Каждый пакет, передаваемый по протоколу *HTTP*, можно разделить на три части: стартовая строка, заголовки и тело.

Стартовая строка состоит из трех частей, разделенных пробелом, которые отличаются для запроса и ответа. Начиная с версии 1.0 протокола, стартовая строка для запроса выглядит следующим образом: <метод> <*URI*> *HTTP*/<версия>, где вместо <метод> указывается метод *HTTP*-протокола, вместо <версия> – версия *HTTP*-протокола, а вместо <*URI*> – идентификатор ресурса в сети интернет, которому будет адресован данный запрос. Стартовая строка для ответа выглядит следующим образом: *HTTP*/<версия> <код состояния> <пояснение>, где вместо <версия> снова указывается версия *HTTP*-протокола, вместо <код состояния> указывается трехзначный код ответа сервера (например, всеми известный 404, который указывает на недоступность запрашиваемого ресурса), поле <пояснение> является необязательным и содержит пояснение кода состояния (в случае 404 это, как правило, *Not Found*).

Коды состояния классифицируются по первой цифре: 1 – информационный код (информирует о процессе передачи), 2 – успешно (успешное принятие запроса сервером), 3 – перенаправление (указывает на необходимость сделать запрос к другому *URI*), 4 – ошибка клиента (указывает на ошибку запроса клиентом), 5 – ошибка сервера (указывает на возникновение ошибки на стороне сервера).

Пример стартовой строки для запроса: *GET* /*index.html* *HTTP*/1.1. Пример стартовой строки для ответа: *HTTP*/1.1 200 *OK*.

*HTTP*-протокол предусматривает девять методов: *GET*, *POST*, *PUT*, *DELETE*, *OPTIONS*, *HEAD*, *TRACE*, *PATCH* и *CONNECT*. Метод, согласно протоколу *HTTP*, должен указывать на операцию, которая должна быть произведена над ресурсом. На практике же, из-за того, что некоторые веб-серверы не поддерживают методы *PUT* и *DELETE*, в основном используются методы *GET* (для получения ресурса) и *POST* (для остальных операций, требующих передачу большого количества информации, которую нельзя передать в виде параметров *GET*-запроса). Однако существует практика построения взаимодействия между серверной и клиентской частями с использованием архитектуры *REST* [14]. Один из принципов архитектуры *REST* – раздельное использование *HTTP*-методов. Например, метод *GET* используется для чтения данных, метод *POST* – для создания, метод *PUT* – для изменения, а метод *DELETE* – для удаления. Таким образом получается, что каждый метод используется для одной *CRUD* операции, что улучшает понимание архитектуры веб-приложения.

Клиент передает информацию серверу либо через параметры запроса, либо через тело запроса. Параметры запроса передаются в *URI* после символа ? и разделяются символом &. Например, *GET* /*example*?*p1*=*v1&p2=v2&p3=v3* *HTTP*/1.1. В данном примере передаются три параметра: *p1*, *p2*, *p3* со значениями *v1*, *v2* и *v3* соответственно. Все *HTTP*-методы могут передавать информацию через параметры запроса, но все могут использовать для этого тело запроса. Тело запроса имеют только методы: *POST*, *PUT* и *PATCH*.

Вторая часть после стартовой строки – заголовки – представляет собой множество пар (ключ: значение). Также, как и тело, заголовки отличаются для запроса и ответа. Далее будут перечислены заголовки, используемые веб-фильтром. Информацию по остальным заголовкам можно найти в *Request for Comments* (*RFC*) *HTTP* [15].

Заголовок *Content-Type*. Указывает *MIME*-тип (формат обмена данными), который передается в запросе или ответе и, возможно, кодировку. Например, заголовок *Content-Type: application/json;charset=utf-8* говорит о том, что в теле будет передано сообщение в формате *JSON* в кодировке *UTF*-8.

Заголовок *Host*. Указывает доменное имя и, возможно, порт хоста запрашиваемого ресурса. Для версии протокола 1.1 запрос обязательно должен содержать данный заголовок. Пример: *Host: example.com*.

Заголовок *Content-Length*. Указывает размер содержимого тела в байтах. Пример: *Content-Length: 2353*.

Заголовок *Transfer-Encoding*. Указывает список способов кодирования, которые были применены к содержимому тела для передачи, разделенных запятыми. Возможные значения: *chunked*, *compress*, *deflate*, *gzip*, *identity*. При использовании *chunked* содержимое тела передается частями, с указанием длины каждой части в шестнадцатеричном виде. Если длина части равна 0, то это означает, что достигнут конец передаваемого содержимого. В случае использования *chunked* не обязательно использовать заголовок *Content-Length* для указания длины содержимого. При использовании других способов кодирования заголовок *Content-Length* обязателен. При использовании значения *identity* содержимое передается как есть.

Третья часть – тело – содержит передаваемые данные в формате и кодировке, указанных в заголовке *Content-Type*. Содержимое тела может быть закодировано согласно значению, указанному в заголовке *Transfer-Encoding*.

Веб-фильтр извлекает из *HTTP*-ответа от сервера содержимое тела (используя при этом заголовки для правильного извлечения). Функционал черных списков использует заголовок *Host* для определения запрашиваемого хоста.

Также стоит отметить, что управление веб-фильтром происходит через веб-интерфейс, взаимодействие которого с веб прокси-сервером происходит согласно архитектуре *REST*.

*HTTPS* не является самостоятельным протоколом. Он представляет собой расширение протокола *HTTP* для поддержания шифрования. В качестве криптографических протоколов используются *SSL* и *TLS* [16], которые работают на уровне представления сетевой модели *OSI*.

## 2.3 Trusted MITM

Как уже было сказано в первом разделе, для максимальной эффективности работы веб-фильтра следует расшифровывать и анализировать *HTTPS*-трафик. Для этого предлагается использовать технологию *Trusted MITM*, суть которой заключается в динамической генерации поддельных сертификатов на стороне веб прокси-сервера. Сгенерированные сертификаты подписываются корневым сертификатом веб-фильтра. *Trusted* в данном случае означает, что пользователь знает о применении подхода *MITM* и устанавливает корневой сертификат веб-фильтра в хранилище сертификатов для того, чтобы доверять сгенерированным веб-фильтром сертификатам.

В общем случае взаимодействие пользователя и веб-сервера с использованием протокола *HTTPS* выглядит следующим образом:

1. Клиент (браузер) запрашивает у веб-сервера ресурс (веб-страницу) по протоколу *HTTPS*.
2. Веб-сервер отправляет клиенту свой открытый ключ и сертификат.
3. Браузер выполняет проверку сертификата сервера: по сроку действия, не отозван ли сертификат, совпадение поля *Common Name* в сертификате и запрашиваемого доменного имени. Если все проверки пройдены успешно, браузер генерирует симметричный ключ, шифрует его открытым ключом сервера и отправляет его обратно.
4. Веб-сервер расшифровывает симметричный ключ, используя свой закрытый ключ. Затем шифрует контент симметричным ключом и отправляет его клиенту.
5. Браузер расшифровывает контент симметричным ключом и отображает его.

Взаимодействие браузера с веб-сервером по протоколу *HTTPS* по шагам показано на рисунке 3.



Рис. 3 – Взаимодействие браузера и веб-сервера по протоколу *HTTPS*

Схема использования технологии *Trusted MITM* веб-фильтром упрощенно показана на рисунке 4.



Рис. 4 – Упрощенная схема перехвата и обработки *HTTPS* трафика

Более подробная схема описана ниже:

1. Перехват веб-фильтром запроса браузера к веб-серверу по протоколу *HTTPS*.
2. Прокси-сервер генерирует ключевую пару (открытый и закрытый ключи) и сертификат для запрашиваемого браузером доменного имени (в поля *Common Name* и *Subject Alternative Name* сертификата проставляется доменное имя). Сгенерированный сертификат подписывается доверенным корневым сертификатом (которому доверяет клиент). Веб-фильтр отправляет сгенерированный сертификат клиенту.
3. Браузер проверяет сгенерированный сертификат веб-фильтра. В случае прохождения проверок, браузер генерирует симметричный ключ, шифрует его открытым ключом из сертификата прокси-сервера и отправляет обратно.
4. Веб-фильтр расшифровывает симметричный ключ, используя закрытую часть ключа, и сохраняет его.
5. Веб прокси-сервер делает запрос из шага 1 к веб-серверу по протоколу *HTTPS*.
6. Веб-сервер отправляет прокси-серверу свой сертификат.
7. Веб-фильтр проверяет сертификат и, в случае прохождения проверок, генерирует симметричный ключ, зашифровывает его открытым ключом из сертификата сервера и отправляет обратно веб-серверу.
8. Веб-сервер расшифровывает симметричный ключ, используя закрытую часть, и отправляет прокси-серверу контент, зашифрованный симметричным ключом.
9. Веб-фильтр расшифровывает контент при помощи симметричного ключа и затем шифрует сохраненный контент симметричным ключом, сохраненным на шаге 4, и отправляет его клиенту.
10. Браузер расшифровывает контент при помощи симметричного ключа и отображает его.

Описанная выше схема по шагам показана на рисунке 5.



Рис. 5 – Схема работы веб-фильтра с *HTTPS*-трафиком

## 2.4 Java, Jetty, Servlets, TCP-сокеты

В качестве языка программирования (ЯП) для написания веб-фильтра предлагается использовать *Java*. Есть несколько причин выбора этого языка для реализации веб прокси-сервера:

1. Удобный *Application Programming Interface* (*API*) для работы с сетью. Язык программирования *Java* изначально разрабатывался с уклоном на сетевые технологии и разработку распределенных приложений. Примером может служить набор спецификаций *Jakarta EE* (в прошлом *Java EE*), которые описывают архитектуру серверной части языка. По этой причине *Java* используется для разработки высоконагруженных и распределенных систем.
2. Независимость от платформы. Как известно, программы, написанные на ЯП *Java*, при компиляции преобразуются в байт-код, который может исполняться виртуальной машиной *Java* (*Java Virtual Machine* (*JVM*)). Таким образом, один и тот же байт-код может быть исполнен на *JVM*, установленных на разных платформах (операционных системах (ОС)). Поэтому программы, написанные на *Java*, являются кроссплатформенными.
3. Безопасность среды исполнения. *JVM* является в каком-то смысле песочницей (безопасной средой исполнения) для исполняемых ею программ. Можно ограничивать часть используемых ресурсов (например, размер используемой оперативной памяти (ОП)) путем задания опций запуска *JVM*.
4. Достаточная простота разработки и надежность программ. *Java* является языком программирования со строгой типизацией (типы каждых переменных должны быть известны на стадии компиляции), поэтому предотвращает многие ошибки программирования на моменте компиляции программы. Развитый подход к обработке исключительных ситуаций добавляет надёжности написанным программам. Также, создавая программу на *Java*, программист может не заботиться о выделении и высвобождении памяти под данные – за этим автоматически следит *JVM*. Поэтому в *Java* практически не встречается понятие «утечка памяти».
5. Поддержка сообществом. *Java* занимает высокие позиции в ежегодных рейтингах ЯП и имеет большую поддержку сообществом программистов: есть множество тематических форумов, проектов с открытыми исходными кодами и большое разнообразие библиотек и фреймворков.

Для реализации веб-сервера предлагается использовать библиотеку *Jetty* [17]. *Jetty* представляет реализацию спецификации сервлетов (*Java Servlets*) и поддержку их жизненного цикла при помощи контейнера сервлетов. Сервлет представляет собой класс с реализацией методов обработки *HTTP*-запросов (основные методы *doGet*, *doPost*, *doPut*, *doDelete*). Каждому сервлету сопоставляется ресурс (*URL*), запросы на который он будет обрабатывать. Методы обработки *HTTP*-запросов сервлета принимают в качестве параметров объекты с информацией о запросе и ответе. Таким образом, каждый метод сервлета обрабатывает *HTTP*-запрос с соответствующим методом запроса (например, *GET*, *POST*, *PUT*, *DELETE*). При помощи объекта запроса методы сервлета получают информацию о клиенте (например, *User-Agent*, *IP*-адрес) и запросе (параметры, заголовки, тело запроса). Через объект ответа методы сервлета отправляют ответ на запрос, устанавливают заголовки и тело ответа.

Контейнер сервлетов играет роль контроллера, управляющего жизненным циклом объектов сервлетов. Каждый запрос клиента обрабатывается в отдельном потоке. Принимая запрос, контейнер сервлетов выбирает подходящий обработчик запроса (объект сервлета) и передает ему управление с объектами запроса и ответа. Если же подходящего сервлета не находится, то клиенту будет возвращен ответ с кодом ошибки 404 (либо 405).

Сервлеты будут обеспечивать взаимодействие с веб-фильтром через веб-консоль. Например, такие операции, как получение и сохранение настроек, получение статуса веб-фильтра, запуск и остановка веб прокси-сервера, управление черным списком будут.

Перехват *HTTP*/*HTTPS*-запросов будет происходить на уровне протоколов *TCP*/*IP* сетевой модели *OSI* (ОС будет перенаправлять запросы на указанный порт). В *Java*, как и в любом другом языке программирования, есть программные *TCP*-сокеты, которые позволяют взаимодействовать с другими программами, используя *IP*-сеть. Веб прокси-сервер будет работать в многопоточном режиме, обрабатывая каждый запрос от клиента в отдельном потоке.

## 2.5 JavaScript и технология AJAX

Для реализации веб-консоли управления веб-фильтром будет использован язык программирования *JavaScript* в связке с *HTML* и *CSS*. *JavaScript* вместе с *CSS* придают веб-страницам динамичность и интерактивность, приближая их интерфейс к интерфейсу настольных программ.

*JavaScript*, также как и *Java*, имеет *C*-подобный синтаксис и большую поддержку сообществом: количество библиотек и фреймворков очень велико. Однако есть существенное отличие между этими двумя языками: *JavaScript* является интерпретируемым языком с динамической типизацией. Последнее означает, что в различных участках программы одна и та же переменная может принимать значения разных типов (в отличие от *Java*, где тип переменной задается при ее объявлении).

С целью повышения интерактивности в общении между клиентом и сервером будет использоваться технология асинхронных запросов *Asynchronous Javascript and XML* (*AJAX*). Ее суть заключается в фоновых запросах к веб-серверу, не приводящих к полной перезагрузке страницы. Полученные данные могут дополнить или поменять содержимое страницы. Хотя в названии и указан формат обмена данными *XML*, но *AJAX* поддерживает не только его. При реализации будет использован формат *JSON*. Асинхронные запросы будут использоваться при получении статуса веб-фильтра по таймеру, при сохранении и получении настроек, при включении и выключении веб-фильтра, при работе с черным списком, при скачивании корневого сертификата. *AJAX*-запросы будут обрабатываться на сервере сервлетами. Взаимодействие между клиентом и сервером будет построено по архитектуре *REST*.

## 2.6 База данных H2 и JDBC

Для хранения настроек веб-фильтра и черного списка предлагается использовать реляционную базу данных *H2* [18]. *H2* является кроссплатформенной системой управления базами данных (СУБД) с открытым исходным кодом, написанной полностью на *Java*. Благодаря этому она легко интегрируется с *Java*-приложениями. Также база данных может работать в двух режимах: клиент-серверном и встроенном. Для первого режима работы требуется установка СУБД, которая является сервером в этой архитектуре. Во втором режиме работы база данных представляет собой бинарный файл, работа с которым происходит через библиотеку *H2* (реализация спецификации *Java Database Connectivity* (*JDBC*)). База данных поддерживает транзакции, шифрование данных, хранимые процедуры, планы исполнения запросов и многое другое [18].

Встроенный режим работы баз данных редко используется в промышленной разработке, так как при частых запросах будет заметно снижение производительности. Однако если работа с базой данных происходит не так интенсивно, то такой режим работы оказывается подходящим вариантом. Например, если настроек у программы не так много, то часто они хранятся в файле. Также встроенный режим работы используется в процессе разработки и тестирования приложений, чтобы не тратить время на настройку СУБД. Благодаря тому, что *JDBC* является интерфейсом, замена его реализации на клиент-серверный режим работы не повлечет за собой изменение кода. Таким образом, можно легко перейти на клиент-серверный режим работы (даже другой СУБД) без изменения существующего кода. Также плюсом встроенного режима работы является независимость от сервера БД – приложения не требуют дополнительной установки СУБД. Из минусов стоит отметить снижение производительности при интенсивной работе с базой данных.

# Проектирование, разработка и тестирование работы веб-фильтра

В процессе написания предыдущих глав работы сформировались требования к разрабатываемому веб-фильтру:

1. Многопоточный режим работы.
2. Фильтрация контента «на лету» при помощи байесовского классификатора.
3. Поддержка механизма черных списков *IP*/*URL*.
4. Работа с протоколами *HTTP* и *HTTPS*.
5. Возможность управления веб-фильтром через пользовательский интерфейс (веб-консоль).

В данной главе будет описано проектирование и разработка веб-фильтра с учетом требований выше. Также будет проведено тестирование работы разработанного веб-фильтра.

## Проектирование

На уровне компонентов веб-фильтр состоит из двух частей: веб-сервер и прокси-сервер. Веб-сервер обрабатывает запросы от веб-консоли и управляет работой прокси-сервера. Веб-сервер работает на порту 8080. Прокси-сервер в свою очередь обрабатывает запросы, которые ему перенаправляет операционная система. Так как одно из требований к веб-фильтру – это работа с протоколами *HTTP* и *HTTPS*, то прокси-сервер принимает запросы на два порта (для *HTTP* и для *HTTPS*). Данные порты и хост указываются в настройках прокси-сервера операционной системы. По умолчанию порт для обработки *HTTP*-запросов 3333, а для обработки *HTTPS*-запросов – 3334. Прокси-сервер работает на уровне протоколов *TCP/IP* (*TCP*-сокеты), а веб-сервер на прикладном уровне непосредственно с *HTTP*-трафиком. На рисунке 6 показана *UML* диаграмма развертывания веб-фильтра.

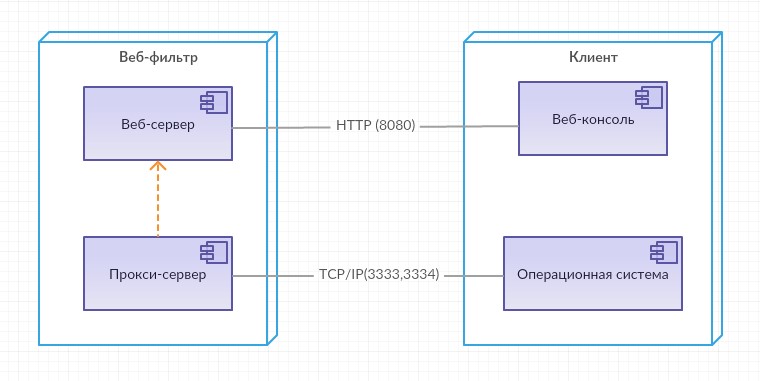


Рис. 6 – Диаграмма развертывания веб-фильтра

На рисунке 6 показана иллюстрация двух основных компонентов веб-фильтра – веб-сервера и прокси-сервера. Прокси-сервер управляется при помощи веб-сервера, который в свою очередь обрабатывает запросы, поступающие от веб-консоли управления.

Взаимодействие веб-сервера и веб-консоли происходит при помощи технологий *Java Servlets* (на серверной стороне) и *AJAX* (на клиентской стороне). Формат обмена данными *JSON*. Компонентами взаимодействия на сервере являются классы-сервлеты:

1. *BlacklistProxyServlet.java*. Обрабатывает *GET/POST/DELETE*-запросы на ресурс */proxy/blacklist*. *GET*-запрос возвращает весь черный список. Через *POST*-запрос происходит добавление *IP/URL* хоста в черный список. Через *DELETE*-запрос происходит удаление записи из черного списка.
2. *DownloadRootCertificateServlet.java.* Обрабатывает *GET*-запрос на ресурс */proxy/downloadRootCertificate*. *GET*-запрос возвращает файл корневого сертификата, который загружается на клиент.
3. *SettingsProxyServlet.java.* Обрабатывает *GET/PUT*-запросы на ресурс */proxy/settings*. *GET*-запрос возвращает все настройки веб-фильтра. Через *PUT*-запрос происходит сохранение/обновление настроек.
4. *StatusProxyServlet.java.* Обрабатывает *GET*-запрос на ресурс */proxy/status.* *GET*-запрос возвращает статусы (запущен или нет) *HTTP* прокси-сервера и *HTTPS* прокси-сервера.
5. *StartHttpProxyServlet.java.* Обрабатывает *GET*-запрос на ресурс */proxy/startHttp*. Через *GET*-запрос происходит запуск *HTTP* прокси-сервера.
6. *StartHttpsProxyServlet.java.* Обрабатывает *GET*-запрос на ресурс */proxy/startHttps*. Через *GET*-запрос происходит запуск *HTTPS* прокси-сервера.
7. *StopHttpProxyServlet.java.* Обрабатывает *GET*-запрос на ресурс */proxy/stopHttp*. Через *GET*-запрос происходит остановка *HTTP* прокси-сервера.
8. *StopHttpsProxyServlet.java.* Обрабатывает *GET*-запрос на ресурс */proxy/stopHttps*. Через *GET*-запрос происходит остановка *HTTPS* прокси-сервера.

Архитектурно прокси-сервер оформлен в виде паттерна одиночка (*singleton*), который гарантирует единственный экземпляр класса *Proxy.java*. Внутри себя класс *Proxy.java* содержит два серверных сокета: для обработки запросов по *HTTP* и по *HTTPS* и методы для управления этими сокетами (методы запуска/остановки прокси-серверов, методы получения статусов прокси-серверов). Экземпляр класса *Proxy.java* оформлен в виде поля в сервлетах 4-8 из списка выше.

Компонентами взаимодействия на клиентской стороне являются скрипты *JavaScript*:

1. *blacklist.js.* Данный скрипт находится на *HTML*-странице *blacklist.html*. При запуске скрипта при помощи *AJAX* делается *GET*-запрос на ресурс */proxy/blacklist* для получения всего черного списка. При удалении элемента черного списка при помощи *AJAX* делается *DELETE*-запрос на ресурс */proxy/blacklist*. В теле запроса передается *IP*-адрес удаляемого элемента. Сервер может вернуть три кода: 200, 400 и 500. Код 200 возвращается при успешном удалении элемента из черного списка. Код 400 возвращается если был передан некорректный *IP*-адрес хоста. Код 500 возвращается в случае серверной ошибки. В каждом из трех случаев происходит уведомление клиента при помощи всплывающего сообщения. При добавлении элемента в черный список при помощи *AJAX* делается *POST*-запрос на ресурс */proxy/blacklist*. В теле запроса передается *IP/URL* введенного хоста. Как и при удалении сервер может вернуть три кода: 200, 400 и 500. Их семантика и обработка точно такие же, как описано выше.
2. *settings.js*. Данный скрипт находится на *HTML*-странице *settings.html*. Также на этой странице находится гиперссылка (при клике на гиперссылку делается *GET*-запрос на указанный ресурс) на ресурс */proxy/downloadRootCertificate*, *GET*-запрос на который возвращает корневой сертификат. При запуске скрипта при помощи *AJAX* делается *GET*-запрос на ресурс */proxy/settings* для получения всех настроек веб-фильтра. При обновлении настроек сначала происходит их проверка (порты должны находиться в диапазоне [1024, 65535], количество потоков должно находится в диапазоне [1, 10], таймауты в миллисекундах должны находиться в диапазоне [1000, 60000]). Если настройки не проходят проверку, то происходит уведомление клиента об этом при помощи всплывающего уведомления. Если же проверка проходит, то при помощи *AJAX* делается *PUT*-запрос на ресурс */proxy/settings*. В теле запроса передаются все настройки. Сервер может вернуть два кода: 200 и 500. Код 200 возвращается при успешном обновлении настроек. Код 500 возвращается в случае серверной ошибки. В каждом случае происходит уведомление клиента при помощи всплывающего сообщения.
3. *common.js*. Данный скрипт находится на всех *HTML*-страницах и выполняет при помощи *AJAX* *GET-*запросы на ресурс */proxy/status* для получения статусов *HTTP* и *HTTPS* прокси-серверов. Запросы выполняются по таймеру с периодом в одну секунду. Сервер может вернуть два кода: 200 и 500. Код 200 возвращается, если веб-фильтр запущен. Код 500 возвращается, если веб-фильтр остановлен. В случае возвращения кода 200 происходит вывод информации о статусах прокси-серверов. Таким образом, если какой-то из прокси-серверов завершит свою работу, клиент узнает об этом максимум через одну секунду. В случае возвращения кода 500 происходит уведомление клиента при помощи всплывающего сообщения.
4. *manage.js.* Данный скрипт находится на *HTML*-странице *manage.html*. На данной странице располагаются два переключателя: для *HTTP* прокси-сервера и для *HTTPS* прокси-сервера. При переключении каждого из переключателей при помощи *AJAX* делается *GET*-запрос на один из четырех ресурсов (зависит от вида переключателя и его положения): */proxy/startHttp*, */proxy/stopHttp*, */proxy/startHttps*, */proxy/stopHttps*. Сервер может вернуть два кода: 200 и 500. Код 200 возвращается в случае успешного запуска (остановки) прокси-сервера. Код 500 возвращается в случае серверной ошибки. В обоих случаях происходит уведомление клиента при помощи всплывающего сообщения. Также для актуального отображения состояния переключателей при помощи *AJAX* делаются *GET*-запросы на ресурс */proxy/status* по таймеру с периодом в одну секунду.

Диаграмма компонентов веб-сервера показана на рисунке 7.

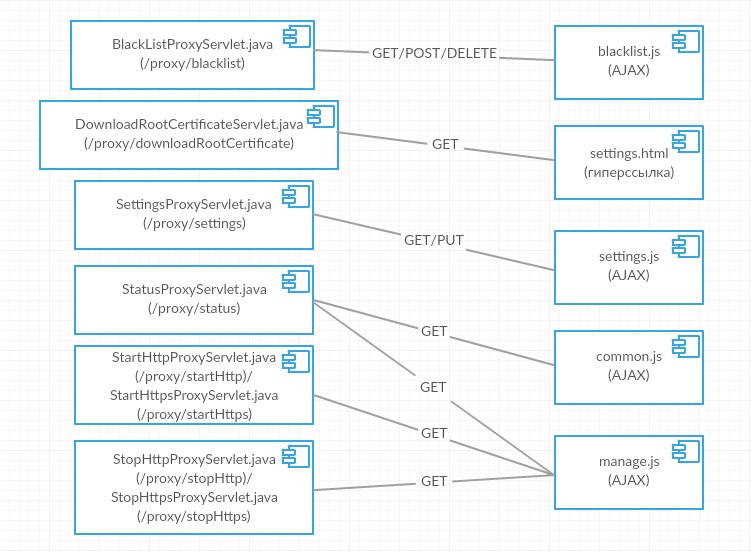


Рис. 7 – Диаграмма компонентов веб-сервера

Диаграмма компонентов прокси-сервера показана на рисунке 8.

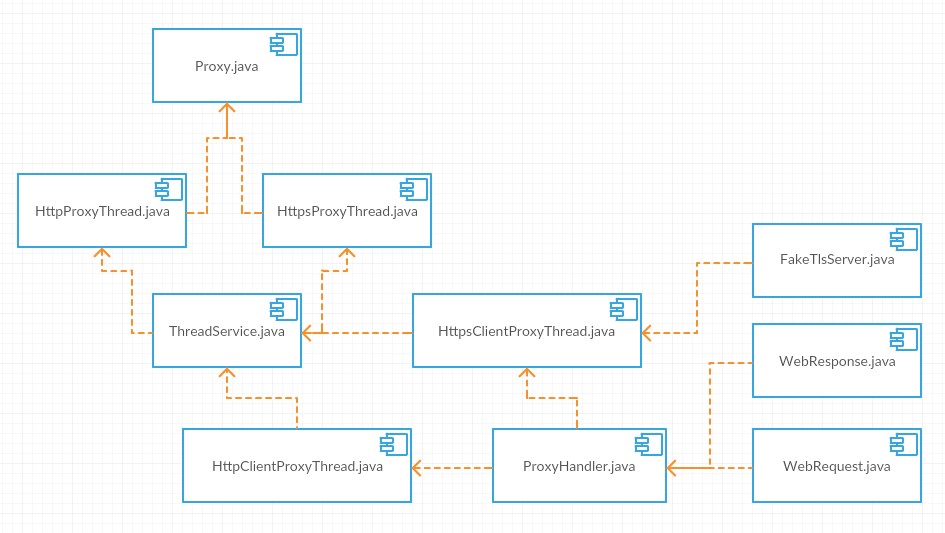


Рис. 8 – Диаграмма компонентов прокси-сервера

## Разработка

Реализацию можно разбить на несколько этапов:

1. На первом этапе проводилась разработка общего каркаса веб-фильтра в виде прокси-сервера на *TCP*-сокетах в многопоточном режиме. На данном этапе было реализовано простое пропускание *HTTP*-трафика между веб-сервером и браузером через веб-фильтр с использованием многопоточности, т.е. каждый запрос браузера выполнялся в отдельном потоке.
2. На втором этапе проводилась разработка серверной части веб-фильтра с применением технологии *Java Servlets* и библиотеки для создания веб-сервера *Jetty*. Данный этап решал задачу управления веб-фильтром через веб-консоль со стороны сервера. Были реализованы такие функциональности, как запуск и остановка веб-фильтра, сохранение и применение настроек, отслеживание статуса веб-фильтра, сохранение *IP*-адреса или *URL* хоста в черный список.
3. На третьем этапе проводилась разработка компонентов, отвечающих за работу с базой данных *H2*. База данных работает во встроенном режиме и представляет собой один файл. На данном этапе была определена структура базы данных: таблица настроек и таблица черного списка, а также реализованы *Java* классы для выполнения *CRUD* операций с данными таблицами. В коде было оформлено в виде паттерна проектирования *DAO*.
4. На четвертом этапе проводилась разработка клиентской части веб-фильтра – веб-интерфейса для управления контент-фильтром – с использованием *JavaScript*, библиотек *jQuery*, *Bootstrap* и применением технологии *AJAX*. На данном этапе были сделаны страницы *HTML* с управляющими *JavaScript* скриптами. Технология *AJAX* применялась для получения состояния веб-фильтра по таймеру.
5. На пятом этапе проводилась разработка компонентов, отвечающих за обучение классификатора, разбор, обработку (стемминг) и классификацию текста. На данном этапе был реализован байесовский классификатор, на вход которому подается список обработанных слов (текст страницы) и который возвращает карту категория -> вероятность попадания данного текста в эту категорию. Также были реализованы компоненты, которые предварительно обрабатывают слова при помощи библиотек *Apache Lucene Morphology* и *Jsoup*.
6. На шестом этапе проводилась разработка компонентов, предназначенных для генерации поддельных сертификатов формата *X.509* и обработки процедуры хэндшейков в протоколе *TLS*. Для разработки использовалась криптографическая библиотека *BouncyCastle*.

## Тестирование

# ЗАКЛЮЧЕНИЕ

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 Современные тенденции в области контентной фильтрации [Электронный ресурс]. – 2013. – Режим доступа: http://alexott.net/ru/writings/cf/, свободный. – Яз. рус.

2 Антонов, И. Под колпаком Эшелона [Электронный ресурс]. – 2014. – Режим доступа: http://iantonov.me/page/pod-kolpakom-eshelona, свободный. – Яз. рус.

3 Хазов, В. СОРМ-1, СОРМ-2, СОРМ-3: особенности и отличия [Электронный ресурс]. – 2016. – Режим доступа: https://vasexperts.ru/blog/osobennosti-i-otlichiya-sorm/, свободный. – Яз. рус.

4 Пискунов, И. Перехват и расшифровка HTTPS трафика [Электронный ресурс]. – 2016. – Режим доступа: https://ipiskunov.blogspot.com/2016/06/https.html, свободный. – Яз. рус.

5 Decrypt SSL (trusted man-in-the-middle technique) [Электронный ресурс]. – 2011. – Режим доступа: http://webos-internals.org/wiki/Decrypt\_SSL\_%28trusted\_man-in-the-middle\_technique%29, свободный. – Яз. англ.

6 Масюк, А.А, Сараджишвили, С.Э. Тематическая категоризация ресурсов в системах контентной фильтрации [Текст] / А.А. Масюк, С.Э. Сараджишвили // Научно-технические ведомости СПбГПУ. Сер. Информатика. Телекоммуникации. Управление. –2013. –№ 1.

7 Батура Т.В. Методы автоматической классификации текстов // Программные

продукты и системы. 2017. Т. 30. № 1. С. 85–99; DOI: 10.15827/0236-235X.030.1.085-099.

8 Ju R. An Efficient Method for Document Categorization Based on Word2vec and Latent Semantic Analysis. 2015 IEEE Int. Conf. on Computer and Information Technology; Ubiquitous Computing and Communications; Dependable, Autonomic and Secure Computing; Pervasive Intelligence and Computing. Liverpool, UK, 2015, pp. 2276–2283.

9 Zhang X., Zhao J., LeCun Y. Character-level Convolutional Networks for Text Classification. Proc. of the Neural Information Processing Systems Conf. (NIPS 2015). Montreal, Canada, 2015. Available at: https://arxiv.org/abs/1509.01626 (accessed July 18, 2016).

10 Moraes R., Valiati J.F., Gavião Neto W.P. Document-level sentiment classification: An empirical comparison between SVM and ANN. Expert Systems with Applications. 2013, no. 40, pp. 621–633.

11 Pontiki M., Galanis D., Pavlopoulos J., Papageorgiou H., Androutsopoulos I., Manandhar S. SemEval-2014 Task 4: Aspect based sentiment analysis. Proc. 8th Int. Workshop on Semantic Evaluation (SemEval 2014). Dublin, Ireland, 2014,pp. 27–35.

12 Николенко, С. Байесовские классификаторы [Электронный ресурс]. – 2011. – Режим доступа: https://logic.pdmi.ras.ru/~sergey/teaching/mlaptu11/03-classifiers.pdf, свободный. – Яз. рус.

13 Баженов, Д. Наивный байесовский классификатор [Электронный ресурс]. – 2012. – Режим доступа: http://bazhenov.me/blog/2012/06/11/naive-bayes, свободный. – Яз. рус.

14 Архитектура REST [Электронный ресурс]. – 2008. – Режим доступа: https://habr.com/post/38730/, свободный. – Яз. рус.

15 Hypertext Transfer Protocol -- HTTP/1.1 [Электронный ресурс]. – 1999. – Режим доступа: https://tools.ietf.org/html/rfc2616#page-31, свободный. – Яз. англ.

16 HTTP Over TLS [Электронный ресурс]. – 2000. – Режим доступа: https://tools.ietf.org/html/rfc2818, свободный. – Яз. англ.

17 Библиотека Jetty [Электронный ресурс]. – Режим доступа: http://www.eclipse.org/jetty/, свободный. – Яз. англ.

18 База данных H2 [Электронный ресурс]. – Режим доступа: http://www.h2database.com/html/main.html, свободный. – Яз. англ.

# СПИСОК СОКРАЩЕННЫХ ОБОЗНАЧЕНИЙ

*IP*-адрес (*Internet Protocol*) – уникальный сетевой адрес узла в компьютерной сети.

*IDS* (*Intrusion Detection System*) – система обнаружения вторжений.

*VoIP* (*Voice over IP*) – технология передачи звукового сигнала в *IP*-сетях.

*HTTP* (*HyperText Transfer Protocol*) – протокол передачи гипертекста. Широко используется в сети интернет.

*HTTPS* (*HyperText Transfer Protocol Secure*) – расширение протокола *HTTP* с поддержкой шифрования передаваемой информации.

*Trusted MITM* (*Man-In-The-Middle*) – технология перехвата *HTTPS*-трафика, путем подмены сертификата доверенным «лицом» (программой).

*URL* (*Uniform Resource Locator*) – единообразный определитель ресурса в сети интернет.

*URI* (*Uniform Resource Identifier*) – единообразный идентификатор ресурса в сети интернет.

*REST* (*Representational State Transfer*) – архитектурный стиль взаимодействия компонентов распределённого веб-приложения в сети интернет.

*CRUD* (*Create*, *Read*, *Update*, *Delete*) – акроним, обозначающий четыре базовые операции (создание, чтение, обновление и удаление), используемые при работе с БД.

*RFC* (*Request for Comments*) – документ, содержащий техническую спецификацию.

*MIME* (*Multipurpose Internet Mail Extensions*) – стандарт, описывающий передачу типов данных по различным протоколам (например, *HTTP*, интернет-почта).

*JSON* (*JavaScript Object Notation*) – текстовый формат обмена данными, основанный на структуре объектов в ЯП *JavaScript*. По сути, является объектом *JavaScript.*

*API* (*Application Programming Interface*) – в широком смысле интерфейс программного компонента, реализации которого могут быть использованы в других программах.

*JVM* (*Java Virtual Machine*) – виртуальная машина *Java.*

*TCP* (Transmission Control Protocol) – один из основных протоколов передачи данных интернета, предназначенный для управления передачей данных.

*SSL* (*Secure Sockets Layer*) – криптографический протокол.

*TLS* (*Transport Layer Security*) – криптографический протокол, который используется как замена уязвимому протоколу *SSL*.

*OSI* (*Open Systems Interconnection Basic Reference Model*) – сетевая модель стека сетевых протоколов.

*AJAX* (*Asynchronous Javascript and XML*) – технология асинхронного взаимодействия между клиентом и сервером.

*JDBC* (*Java Database Connectivity*) – спецификация взаимодействия *Java*-приложений и различных систем управления базами данных.

*UML* (*Unified Modeling Language*) – язык графического описания для моделирования и проектирования в области разработки программного обеспечения.

БД – база данных.

СОРМ – система оперативно-розыскных мероприятий.

ЯП – язык программирования.

ОС – операционная система.

ОП – оперативная память.

СУБД – система управления базами данных.