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 Д.А.Бизин

Руководитель ВКР,

к.ф.-м.н. М.Е.Федина

Нормоконтролер / /

Самара 2019

РЕФЕРАТ

Пояснительная записка: 60 страниц, 15 рисунков, 2 таблицы, 23 источника, 1 приложение.

ВЕБ ПРОКСИ-СЕРВЕР, ПЕРЕХВАТ ТРАФИКА, *TRUSTED MITM*, КЛАССИФИКАЦИЯ ТЕКСТА, КОНТЕНТНАЯ ФИЛЬТРАЦИЯ, НАИВНЫЙ БАЙЕСОВСКИЙ КЛАССИФИКАТОР, *JAVA*, *HTTP*, *HTTPS*, МНОГОПОТОЧНОСТЬ, *JAVASCRIPT*, СЕРТИФИКАТ *X.509*, *JAVA SERVLETS*, *AJAX*.

Объектом разработки является веб прокси-сервер, предназначенный для классификации веб-контента по содержимому и последующей фильтрации данного контента на основании данных классификации.

Цель работы – проектирование и разработка веб-фильтра для обеспечения контроля доступа к веб-ресурсам.

В результате работы был реализован веб-фильтр, работающий по протоколам *HTTP* и *HTTPS* и предназначенный для фильтрации веб-контента.

Разработанный веб-фильтр работает в многопоточном режиме, поддерживает механизм черных списков сайтов и позволяет управлять работой прокси-серверов при помощи веб-консоли.

Веб-фильтр может применяться в образовательных учреждениях и на предприятиях с регламентированной политикой информационной безопасности.

Значимость работы заключается в применении наивного байесовского классификатора для классификации веб-контента.

СОДЕРЖАНИЕ

[Введение 4](#_Toc534634666)

[1 Анализ предметной области 6](#_Toc534634667)

[1.1 Веб-фильтр 7](#_Toc534634668)

[1.2 Методы контентной фильтрации 9](#_Toc534634669)

[1.3 Категоризация ресурсов сети интернет 12](#_Toc534634670)

[1.4 Методы автоматической классификации текстов 14](#_Toc534634671)

[2 Используемые технологии и алгоритмы 19](#_Toc534634672)

[2.1 Наивный байесовский классификатор 19](#_Toc534634673)

[2.1.1 Многомерная модель 21](#_Toc534634674)

[2.1.2 Мультиномиальная модель 22](#_Toc534634675)

[2.2 Протоколы *HTTP* и *HTTPS* 24](#_Toc534634676)

[2.3 Технология *Trusted MITM* 27](#_Toc534634677)

[2.4 Технологии для разработки серверной части 30](#_Toc534634678)

[2.5 Технологии для разработки клиентской части 32](#_Toc534634679)

[2.6 Хранение данных 33](#_Toc534634680)

[3 Реализация веб-фильтра 35](#_Toc534634681)

[3.1 Проектирование 35](#_Toc534634682)

[3.2 Разработка 47](#_Toc534634683)

[3.3 Тестирование производительности 49](#_Toc534634684)

[Заключение 51](#_Toc534634685)

[Определения, обозначения и сокращения 52](#_Toc534634686)

[Список использованных источников 54](#_Toc534634687)

[Приложение А 57](#_Toc534634688)

# 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 или *URL* веб-ресурса (черные списки сайтов).

В настоящее время контентную фильтрацию не выделяют в отдельную область компьютерной безопасности, поскольку она тесно связана с другими дисциплинами. В обеспечении компьютерной безопасности контентная фильтрация играет важную роль, поскольку позволяет находить 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. Согласно данным отчета Лаборатории Касперского за 2016‒2017 гг. [1], каждый третий сайт, посещаемый ребенком в интернете, содержит нелегальную информацию. Блокирование таких веб-ресурсов по *IP*-адресу не приносит особого результата, потому что 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

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

Контроль почты и контроль интернет-трафика – две области, в которых уже достаточно давно применяется контентная фильтрация, поэтому здесь также происходят изменения и 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, поскольку 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 передачи мгновенных сообщений (*instant messaging*) и продукты для контроля *peer-to-peer* (*p2p*) соединений [2]. В настоящее время также активно разрабатываются продукты для контроля *VoIP*-трафика [2].

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 и радиоканалам. Наиболее известной системой, использовавшейся американской разведкой для перехвата информации, является *Echelon* [3]. В России также существуют различные реализации систем оперативно-розыскных мероприятий (СОРМ), которые используются для захвата и анализа информации в интересах спецслужб [4].

## 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.



Рисунок 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, поскольку 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*)[5, 6], суть которой будет описана во втором разделе.

## Методы контентной фильтрации

Существует большое количество методов и алгоритмов контентной фильтрации. На практике обычно применяется совокупность методов 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 метода фильтрации [7]:

* 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 при наличии в ней 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 таблице 1 приведено сравнение наиболее распространенных систем контентной фильтрации по ряду критериев (по принципу наличия или отсутствия функциональности) [7].

Таблица 1 – Сравнение систем контентной фильтрации по критериям

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Критерий | Система контентной фильтрации | | | | |
| 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 |

Продолжение таблицы 1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Критерий | Система контентной фильтрации | | | |
| *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 |

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.

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

В антивирусе *Dr.Web* имеется возможность блокировки доступа к веб-сайтам, попадающим в одну из категорий (категории заданы в системе). Также *Dr.Web* поддерживает механизм черных списков и фильтрацию по *MIME*-типу. Еще одной особенностью является возможность написания своих правил разрешения или запрета доступа к веб-ресурсам.

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 [7]:

* использование регулярно обновляемых баз данных категорированных ресурсов (система категоризации работает со списком категорий, категорирует новые веб-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 [8]. 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* [9] – представляет каждое слово в виде вектора, который содержит информацию о контекстных словах. Модель «мешка слов» (*bag-of-words*) [10] представляет документ в виде вектора весов слов документа, другими словами, координата вектора – это порядковый номер слова в документе, а значение координаты – это вес слова. Также есть модель, которая основана на учете *n*-грамм [10].

На этапе выбора признаков происходит определение весов признаков документа. Наиболее распространенный способ – вычисление функции *TF-IDF* [10, 11, 12]. 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 и 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 – на каждом шаге происходит проверка, обладает ли текущее слово каким-либо признаком. В результате  проверок слово попадает в какую-то категорию. Одним из популярных алгоритмов данного метода является 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 демонстрирует линейную классификацию.



Рисунок 2 – Иллюстрация линейной классификации

На рисунке 2 прямая  не является линейным классификатором, потому что она неправильно разделяет множества на два класса. Прямые  и  являются линейными классификаторами, причем прямая  классифицирует более точно, поскольку 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, другими словами, 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 .

Из-за того, что 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 [13]:

* 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 [13] 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 . 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 [14], 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 формуле (11) 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 . 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 формуле (14) сохраняет монотонность, что не влияет на поиск максимума.

## 2.2 Протоколы HTTP и HTTPS

Веб-фильтр выполняет свою основную задачу (классификация контента) на уровне протокола *HTTP*. Поэтому в данном подразделе будут рассмотрены основные части 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* [15]. 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, стартовая строка *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 не 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* [16].

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 запроса или ответа 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* [17], которые работают на уровне представления сетевой модели *OSI*.

## 2.3 Технология 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.



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

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



Рисунок 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.



Рисунок 5 – Схема работы веб-фильтра с *HTTPS*-трафиком

## 2.4 Технологии для разработки серверной части

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

**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.

**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.

**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*.

**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 надежности 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».

**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 *Eclipse* *Jetty* [18]. *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 Технологии для разработки клиентской части

Для реализации веб-консоли управления веб-фильтром предлагается использовать язык программирования *JavaScript* вместе с языком разметки *HTML* и каскадными таблицами стилей (*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 Хранение данных

Для хранения настроек веб-фильтра и черного списка предлагается использовать реляционную базу данных *H2* [19]. *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 [19].

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.

# Реализация 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 будут описаны проектирование и разработка веб-фильтра с учетом требований, изложенных 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.

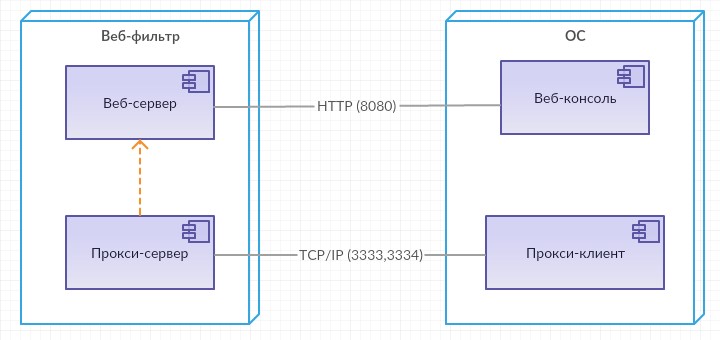


Рисунок 6 – Диаграмма развертывания веб-фильтра

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:

**Класс *BlacklistProxyServlet*.**

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.

**Класс *DownloadRootCertificateServlet.***

Обрабатывает *GET*-запрос на ресурс */proxy/downloadRootCertificate*. *GET*-запрос возвращает файл корневого сертификата, который загружается на клиент.

**Класс *SettingsProxyServlet.***

Обрабатывает *GET/PUT*-запросы на ресурс */proxy/settings*. *GET*-запрос возвращает все настройки веб-фильтра. Через *PUT*-запрос происходит сохранение/обновление настроек.

**Класс *StatusProxyServlet.***

Обрабатывает *GET*-запрос на ресурс */proxy/status.* *GET*-запрос возвращает статусы (запущен или нет) *HTTP* прокси-сервера и *HTTPS* прокси-сервера.

**Класс *StartHttpProxyServlet.***

Обрабатывает *GET*-запрос на ресурс */proxy/startHttp*. Через *GET*-запрос происходит запуск *HTTP* прокси-сервера.

**Класс *StartHttpsProxyServlet.***

Обрабатывает *GET*-запрос на ресурс */proxy/startHttps*. Через *GET*-запрос происходит запуск *HTTPS* прокси-сервера.

**Класс *StopHttpProxyServlet.***

Обрабатывает *GET*-запрос на ресурс */proxy/stopHttp*. Через *GET*-запрос происходит остановка *HTTP* прокси-сервера.

**Класс *StopHttpsProxyServlet.***

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*. Внутри себя класс *Proxy* 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* оформлен в виде полей в сервлетах *StatusProxyServlet*, *StartHttpProxyServlet*, *StartHttpsProxyServlet*, *StopHttpProxyServlet*, *StopHttpsProxyServlet*.

Компонентами взаимодействия на клиентской стороне являются скрипты *JavaScript*:

**Скрипт *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.

**Скрипт *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 сообщения. 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.

**Скрипт *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.

**Скрипт *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 изменении положения 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.

Все описанные выше компоненты (классы-сервлеты, клиентские скрипты и *html*-страница), а также типы запросов между компонентами (*GET*, *POST*, *PUT*, *DELETE*) показаны на *UML* диаграмме компонентов веб-сервера на рисунке 7.

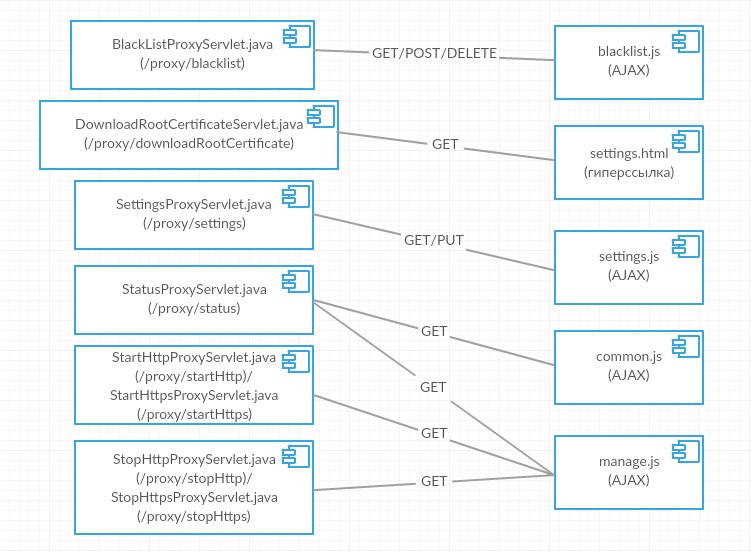


Рисунок 7 – Диаграмма компонентов веб-сервера

*UML* диаграмма компонентов прокси-сервера показана на рисунке 8.

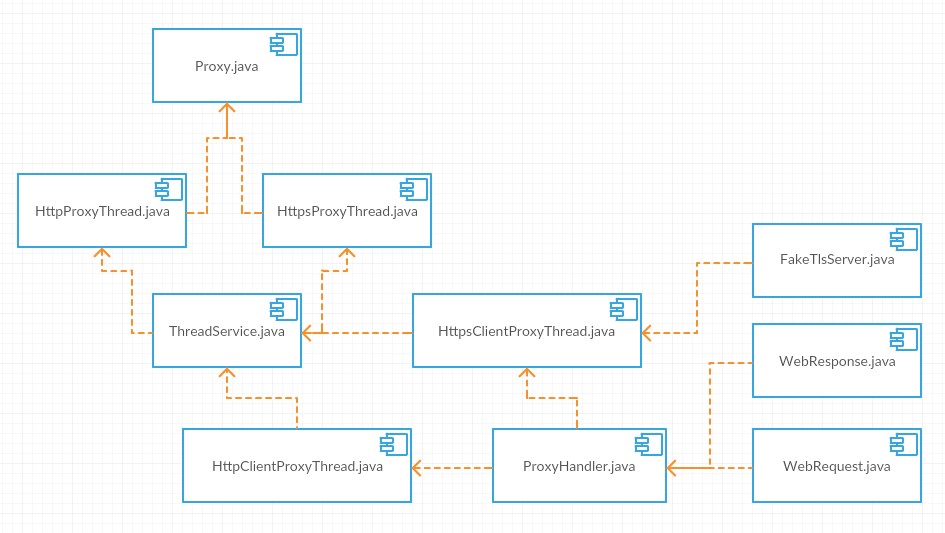


Рисунок 8 – Диаграмма компонентов прокси-сервера

На рисунке 8 показаны компоненты прокси-сервера. В данном случае компоненты – это единицы компиляции программы (*Java* классы). Класс *Proxy*, про который было написано выше, содержит в себе два объекта серверных *TCP*-сокетов – для обработки *HTTP* и *HTTPS* запросов. При запуске *HTTP* прокси-сервера создается поток (объект класса *HttpProxyThread*), в котором будут приниматься *HTTP*-запросы. Аналогично при запуске *HTTPS* прокси-сервера создается поток (объект класса *HttpsProxyThread*), в котором будут приниматься *HTTPS*-запросы. Внутри каждого из потоков используется *ThreadService* – это сервис, обеспечивающий общий пул потоков для обоих прокси-серверов. Количество потоков, находящихся в его пуле, задается в настройках и по умолчанию равно пяти. При помощи *ThreadService* начинают свое выполнение потоки *HttpClientProxyThread* и *HttpsClientProxyThread* для обработки запросов по *HTTP* и по *HTTPS* протоколам соответственно. Внутри данных потоков используется *ProxyHandler* – обработчик запросов к сайту и ответов от сайта. Внутри *ProxyHandler* проверяется принадлежность запрашиваемого сайта черному списку при запросе к сайту, и также происходит классификация контента при получении ответа от сайта. Также дополнительно в *HttpsClientProxyThread* используется объект класса *FakeTlsServer*, который содержит методы обработки для каждого шага установления *TLS*-соединения между прокси-сервером и браузером. На шаге отправления сертификата браузеру происходит генерация поддельного сертификата, подписанного корневым сертификатом веб-фильтра, которому должен доверять браузер. Также в классе *FakeTlsServer* содержится кэш сертификатов – ассоциативный массив, ключом которого является название хоста, а значением – сгенерированный сертификат для данного хоста. Поэтому генерация сертификата не будет происходить, если кэш содержит уже сгенерированный ранее сертификат для данного хоста, что уменьшает время установления соединения. Класс *ProxyHandler* использует объекты классов *WebRequest* и *WebResponse* для хранения информации о запросах и ответах как от браузера, так и от сайта.

На рисунке 9 показана диаграмма последовательности веб-фильтра в нотации *UML*.

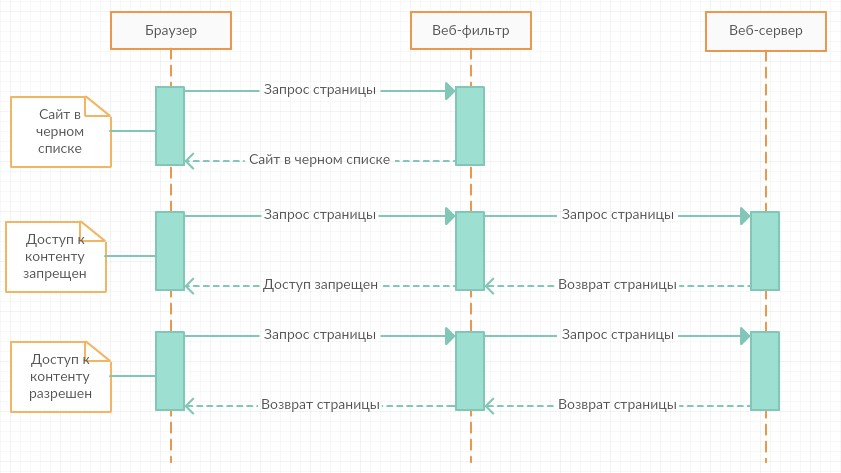


Рисунок 9 – Диаграмма последовательности веб-фильтра

Согласно рисунку 9 возможны три варианта при запросе страницы в браузере:

1. запрашиваемая страница расположена на хосте, который внесен в черный список;
2. запрашиваемая страница содержит запрещенный контент (на основании данных классификации);
3. доступ к запрашиваемой странице разрешен.

В первом случае пользователю будет возвращена информационная страница, интерфейс которой показан на рисунке 10.

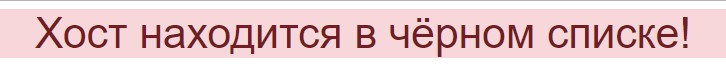


Рисунок 10 – Страница с информацией о нахождении хоста в черном списке

Во втором случае пользователю будет возвращена страница с информацией о вероятностях принадлежности запрашиваемой страницы каждому из классов. Информация на странице оформлена в виде таблицы, где наиболее вероятный класс выделен розовым цветом. Интерфейс страницы показан на рисунке 11.



Рисунок 11 – Страница с информацией о вероятностях попадания запрашиваемой страницы в каждый из классов

В третьем случае пользователю будет возвращена запрашиваемая страница.

Для выполнения последнего требования к разрабатываемому веб-фильтру – возможности управления веб-фильтром через пользовательский интерфейс (веб-консоль) – был спроектирован пользовательский интерфейс веб-консоли управления. Интерфейс включает в себя навигационную панель, которая расположена сверху. При помощи навигационной панели можно перейти на четыре страницы:

1. главная страница;
2. страница управления;
3. страница настроек;
4. страница с черным списком.

Интерфейс главной страницы показан на рисунке 12.

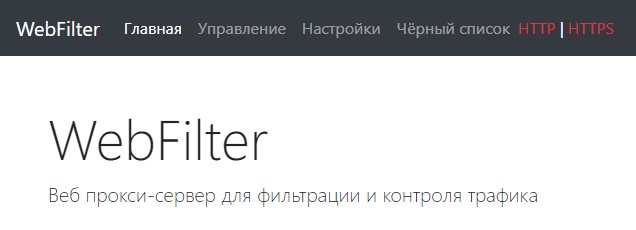


Рисунок 12 – Пользовательский интерфейс главной страницы

На главной странице располагается название и небольшое описание программы. В правом верхнем углу рисунка 12 написаны названия протоколов *HTTP* и *HTTPS*. Эти надписи отражают статус соответствующих прокси-серверов веб-фильтра. Если название протокола написано красным цветом, то прокси-сервер для данного протокола выключен, если же название написано зеленым, то соответствующий прокси-сервер включен. Данные статусы прокси-серверов присутствуют на всех четырех страницах. За управление отображения статусов отвечает скрипт *common.js*, про который было написано выше. Разметка главной страницы располагается в файле *index.html*.

На странице управления располагаются два переключателя: для управления *HTTP* прокси-сервером и для управления *HTTPS* прокси-сервером. При включении или выключении любого из прокси-серверов меняется отображаемый статус и появляется всплывающее сообщение сверху, которое уведомляет об изменении статуса прокси-сервера. Разметка страницы управления располагается в файле *manage.html*. Интерфейс страницы управления показан на рисунке 13.

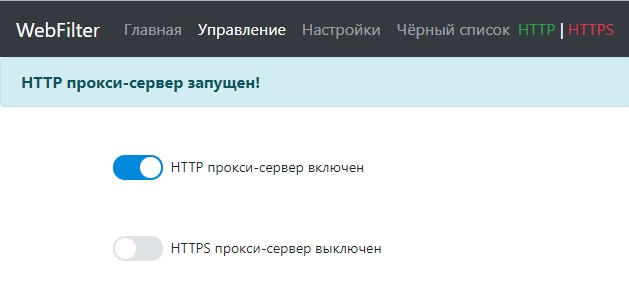


Рисунок 13 – Пользовательский интерфейс страницы управления

Интерфейс страницы настроек показан на рисунке 14.

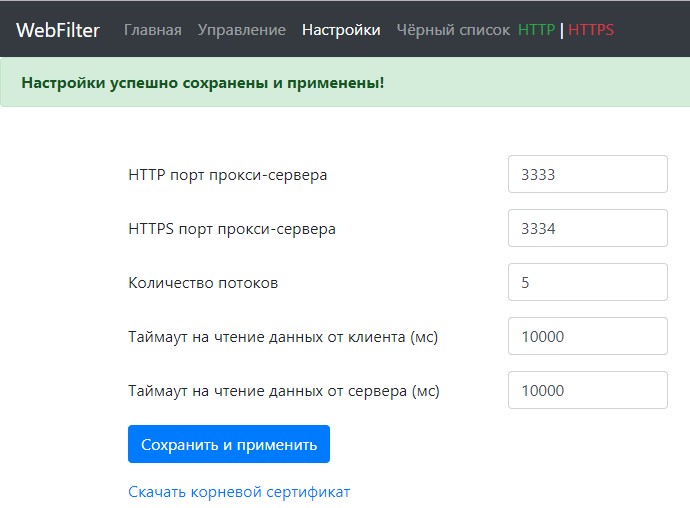


Рисунок 14 – Пользовательский интерфейс страницы настроек

На странице настроек располагаются настройки веб-фильтра и гиперссылка для скачивания корневого сертификата. При сохранении настроек происходит их проверка. Если настройки корректны, то посылается запрос на веб-сервер для их сохранения. В случае успешного сохранения настроек, происходит перезапуск прокси-серверов с новыми параметрами (настройками). В случае ошибок или успеха происходит уведомление пользователя при помощи всплывающего сообщения сверху, как показано на рисунке 14. Разметка страницы настроек располагается в файле *settings.html*.

Интерфейс страницы с черным списком показан на рисунке 15.

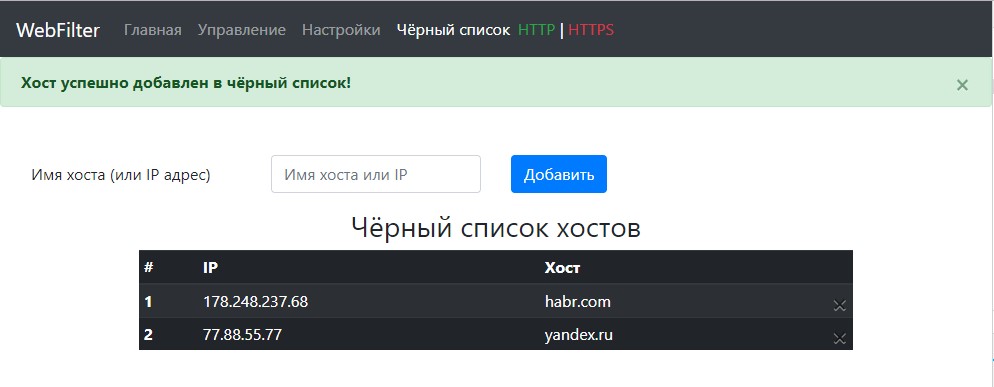


Рисунок 15 – Пользовательский интерфейс страницы с черным списком

На данной странице производится управление черным списком. При добавлении хоста можно указать как имя хоста, так и его *IP*-адрес. Для удаления записи из черного списка достаточно нажать на иконку крестика, расположенную в конце каждой записи. Как и на предыдущих страницах в случае успеха или ошибок будет показано всплывающее сообщение. Разметка страницы с черным списком располагается в файле *blacklist.html*.

## Разработка

Для разработки серверной части веб-фильтра использовался язык программирования *Java* версии 1.8. Для разработки клиентской части веб-фильтра (веб-консоли управления) использовался язык программирования *JavaScript*, реализующий стандарт *ECMAScript* версии 6, каскадные таблицы стилей (*CSS*) версии 3 и язык разметки *HTML* версии 5.

Разработку веб-фильтра можно поделить на несколько этапов:

1. на первом этапе проводилась разработка общего каркаса веб-фильтра в виде *HTTP* прокси-сервера с использованием *TCP*-сокетов в многопоточном режиме. На данном этапе было реализовано простое пропускание *HTTP*-трафика между веб-сервером и браузером через веб-фильтр с использованием многопоточности – каждый запрос браузера выполнялся в отдельном потоке;
2. на втором этапе проводилась разработка серверной части веб-фильтра с использованием технологии *Java Servlets* и библиотеки для создания веб-сервера *Jetty* версии *9.4.10.v20180503*. Данный этап решал задачу управления *HTTP* прокси-сервером при помощи веб-консоли на серверной стороне. Были реализованы такие функциональности, как запуск и остановка *HTTP* прокси-сервера, сохранение и применение настроек, отслеживание статуса *HTTP* прокси-сервера, добавление хоста в черный список, удаление хоста из черного списка;
3. на третьем этапе проводилась разработка компонентов, отвечающих за работу с базой данных *H2* версии *1.4.197*. На данном этапе была определена структура базы данных: таблица настроек и таблица для черного списка, а также написан код для выполнения *CRUD* операций с данными таблицами;
4. на четвертом этапе проводилась разработка клиентской части веб-фильтра – веб-интерфейса для управления контент-фильтром – с использованием языка программирования *JavaScript*, библиотек *jQuery* [20] версии *3.3.1*, *Bootstrap* [21] версии *4* и применением технологии *AJAX*. На данном этапе были сделаны страницы *HTML* с управляющими *JavaScript* скриптами. Также на данном этапе были сделаны информационные страницы: страница с информацией о нахождении хоста в черном списке и страница с информацией о классификации контента запрашиваемой страницы;
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 ассоциативный массив, ключами которого являются категории, а значениями – вероятности попадания данного текста в категории. Также были реализованы компоненты, которые предварительно обрабатывают слова с использованием библиотек *Apache Lucene Morphology* последней версии и *Jsoup* [22] версии *1.11.2*. Исходный код класса, содержащего метод классификации и метод обучения классификатора, приведен в приложении А. Данные для обучения берутся из файлов, которые должны располагаться в директории *learn*. Название файла интерпретируется веб-фильтром как название категории. Внутри файла должны располагаться ключевые слова для данной категории по одному слову на строку;
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* [23] версии *1.49*. Также на данном этапе был написан *HTTPS* прокси-сервер с методами запуска, остановки и проверки статуса. Элементы управления *HTTPS* прокси-сервером также были добавлены в веб-консоль.

## Тестирование производительности

В качестве тестирования производительности разработанного веб-фильтра проводились замеры времени загрузки страницы с одного и того же сайта по разным протоколам. Тестирование проводилось в три этапа:

1. замер времени загрузки веб-страницы, с использованием веб-фильтра, для *HTTP* и *HTTPS* протоколов и замер времени загрузки веб-страницы без фильтрации для тех же протоколов;
2. замер времени генерации сертификата;
3. оценка сложности классификации веб-страницы и оценка сложности обучения классификатора.

Для замеров первого этапа были написаны юнит-тесты, которые делали подряд в одном потоке 100 *GET*-запросов к одному и тому же сайту. Время ответа на каждый запрос замерялось и после всех запросов усреднялось. Запросы делались как по протоколу *HTTP*, так и по протоколу *HTTPS*. В одном случае запросы делались без использования веб-фильтра, а в другом случае использовался веб-фильтр, причем в двух режимах: с кэшированием сертификатов и без кэширования. Результаты замеров времени работы для первого этапа представлены в таблице 2.

Таблица 2 – Замеры времени загрузки страницы

|  |  |  |  |
| --- | --- | --- | --- |
| Протокол | Тип замера | | |
| Без веб-фильтра | Веб-фильтр с генерацией сертификата | Веб-фильтр без генерации сертификата |
| *HTTP* | 1120 мс | 1285 мс | 1285 мс |
| *HTTPS* | 1419 мс | 12626 мс | 11253 мс |

Из результатов тестирования видно, что при использовании веб-фильтра время загрузки страницы по протоколу *HTTPS* в среднем в 8 раз больше. Время загрузки веб-страницы по протоколу *HTTP* практически не меняется при использовании веб-фильтра. Стоит отметить, что время загрузки зависит от используемых аппаратных средств компьютера, на котором проходит тестирование.

Для замеров второго этапа также был написан юнит тест, который вызывал подряд в одном потоке метод генерации сертификатов. Время генерации каждого сертификата замерялось и после всех запросов усреднялось. В результате тестирования среднее время генерации сертификатов оказалось равным 1362 мс.

На третьем этапе проводилась оценка сложности классификации и оценка сложности обучения классификатора. Исходный код класса, содержащего метод классификации и метод обучения байесовского классификатора, приведен в приложении А. Пусть – количество слов в обучающей выборке, – количество категорий, – количество слов в веб-странице. Тогда оценка сложности обучения классификатора равна , а оценка сложности классификации веб-страницы равна . Таким образом, время обучения классификатора квадратично зависит от количества слов в обучающей выборке, а время классификации страницы линейно зависит от количества слов на данной странице.

# ЗАКЛЮЧЕНИЕ

В результате проделанной работы был спроектирован и разработан веб-фильтр для обеспечения контроля доступа к сетевым ресурсам. Разработанный веб-фильтр работает по протоколам *HTTP* и *HTTPS* в многопоточном режиме. Для управления веб-фильтром была реализована веб-консоль, позволяющая задавать настройки веб-фильтра, запускать и останавливать прокси-серверы, отслеживать статусы прокси-серверов, а также управлять черным списком сайтов. В качестве алгоритма классификации текстов используется наивный байесовский классификатор. Для перехвата и анализа трафика, передаваемого по защищенному протоколу *HTTPS*, используется технология *Trusted MITM*. Суть данной технологии заключается в генерации поддельных сертификатов для запрашиваемых хостов. Поддельные сертификаты подписываются корневым сертификатом веб-фильтра. Данный корневой сертификат следует импортировать на компьютеры, на которых будет использоваться веб-фильтр (пользователь доверяет веб-фильтру, позволяя ему перехватывать и анализировать трафик). Веб-фильтр реализован на языке программирования *Java*, одним из преимуществ которого является кроссплатформенность.

В ходе работы также было проведено тестирование производительности веб-фильтра. Тестирование показало, что время загрузки страницы через веб-фильтр при использовании протокола *HTTP* практически не отличается от обычного времени загрузки страницы. При загрузке страницы по протоколу *HTTPS* время увеличивается примерно в 8 раз по сравнению с обычным временем загрузки. В качестве уменьшения времени загрузки была сделана оптимизация, заключающаяся в кэшировании сгенерированных сертификатов.

Разработанный веб-фильтр можно устанавливать как на шлюз сети, так и на конечные рабочие станции.

В качестве работ по дальнейшему улучшению функциональности веб-фильтра можно отметить добавление реализаций других алгоритмов классификации, упомянутых в данной работе.

# ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

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

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

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

*CSS* (*Cascading Style Sheets*) – язык описания внешнего вида документа.

*HTML* (*HyperText Markup Language*) – язык разметки документов, используемых в сети интернет.

*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.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

БД – база данных.

ИКС (Интернет Контроль Сервер) – программный комплекс для контроля за информационными потоками между локальной сетью и сетью интернет.

ОП – оперативная память.

ОС – операционная система.

СОРМ – система оперативно-розыскных мероприятий.

СУБД – система управления базами данных.

ФСTЭК – Федеральная служба по техническому и экспортному контролю.

ЯП – язык программирования.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 Ларкина, А. Чем интересуются дети в Сети [Электронный ресурс]. – 2017. – URL: https://securelist.ru/what-are-children-doing-online/30779/ (дата обращения: 06.12.2018).

2 Современные тенденции в области контентной фильтрации [Электронный ресурс]. – 2013. – URL: http://alexott.net/ru/writings/cf/ (дата обращения: 08.10.2018).

3 Антонов, И. Под колпаком Эшелона [Электронный ресурс]. – 2014. – URL: http://iantonov.me/page/pod-kolpakom-eshelona (дата обращения: 15.11.2018).

4 Хазов, В. СОРМ-1, СОРМ-2, СОРМ-3: особенности и отличия [Электронный ресурс]. – 2016. – URL: https://vasexperts.ru/blog/osobennosti-i-otlichiya-sorm/ (дата обращения: 15.11.2018).

5 Пискунов, И. Перехват и расшифровка HTTPS трафика [Электронный ресурс]. – 2016. – URL: https://ipiskunov.blogspot.com/2016/06/https.html (дата обращения: 15.10.2018).

6 Decrypt SSL (trusted man-in-the-middle technique) [Электронный ресурс]. – 2011. – URL: http://webos-internals.org/wiki/Decrypt\_SSL\_%28trusted\_man-in-the-middle\_technique%29 (дата обращения: 15.10.2018).

7 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.

8 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.

9 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.

10 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. – URL: https://arxiv.org/abs/1509.01626 (дата обращения: 17.10.2018).

11 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.

12 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.

13 Николенко, С. Байесовские классификаторы [Электронный ресурс]. – 2011. – URL: https://logic.pdmi.ras.ru/~sergey/teaching/mlaptu11/03-classifiers.pdf (дата обращения: 20.10.2018).

14 Баженов, Д. Наивный байесовский классификатор [Электронный ресурс]. – 2012. – URL: http://bazhenov.me/blog/2012/06/11/naive-bayes (дата обращения: 20.10.2018).

15 Архитектура REST [Электронный ресурс]. – 2008. – URL: https://habr.com/post/38730/ (дата обращения: 25.10.2018).

16 Hypertext Transfer Protocol -- HTTP/1.1 [Электронный ресурс]. – 1999. – URL: https://tools.ietf.org/html/rfc2616#page-31 (дата обращения: 05.11.2018).

17 HTTP Over TLS [Электронный ресурс]. – 2000. – URL: https://tools.ietf.org/html/rfc2818 (дата обращения: 08.11.2018).

18 Библиотека Jetty [Электронный ресурс]. – URL: http://www.eclipse.org/jetty/ (дата обращения: 10.11.2018).

19 База данных H2 [Электронный ресурс]. – URL: http://www.h2database.com/html/main.html (дата обращения: 10.11.2018).

20 Библиотека jQuery [Электронный ресурс]. – URL: https://jquery.com/ (дата обращения: 12.11.2018).

21 Библиотека Bootstrap 4 [Электронный ресурс]. – URL: https://getbootstrap.com/docs/4.0/getting-started/introduction/ (дата обращения: 12.11.2018).

22 Библиотека Jsoup [Электронный ресурс]. – URL: https://jsoup.org/ (дата обращения: 12.11.2018).

23 Библиотека Bouncy Castle [Электронный ресурс]. – URL: http://www.bouncycastle.org/ (дата обращения: 20.11.2018).

# ПРИЛОЖЕНИЕ А

Исходный код класса, содержащего методы классификации и обучения байесовского классификатора

package classifiers.bayes;

import org.apache.log4j.Logger;

import org.tartarus.snowball.ext.RussianStemmer;

import util.FileUtil;

import java.util.\*;

public abstract class BayesClassifier {

private static final Logger LOGGER = Logger.getLogger(BayesClassifier.class);

/\*\*

\* Ассоциативный массив категория -> (ассоциативный массив ключевое слово -> принадлежность ключевого слова этой категории).

\* Во втором ассоциативном массиве присутствуют ВСЕ ключевые слова из всех категорий.

\*/

private static final Map<String, Map<String, Boolean>> categoryToContainsKeywordsMap = new HashMap<>();

/\*\*

\* Ассоциативный массив категория -> вероятность этой категории (Dc/D)

\*/

private static final Map<String, Double> categoryProbabilityMap = new HashMap<>();

/\*\*

\* Ассоциативный массив категория -> количество ключевых слов в этой категории

\*/

private static final Map<String, Integer> categoryCountKeywordsMap = new HashMap<>();

/\*\*

\* Метод для обучения (тренировки) классификатора на основе заготовленных наборов слов (документов)

\*/

public static void learn() {

LOGGER.info("Начало обучения классификатора");

Map<String, List<String>> categoryToKeywordsMap = FileUtil.getLearnKeywordsFromFiles();

Set<String> allKeywords = new HashSet<>();/\* уникальные ключевые слова из ВСЕХ категорий \*/

categoryToKeywordsMap.values()

.forEach(allKeywords::addAll);

for (Map.Entry<String, List<String>> entry : categoryToKeywordsMap.entrySet()) {

String category = entry.getKey();

List<String> keywordsForCategory = entry.getValue();

categoryCountKeywordsMap.put(category, keywordsForCategory.size());

categoryProbabilityMap.put(category,

(double) keywordsForCategory.size() / allKeywords.size());

Map<String, Boolean> keywordsContainsMap = new HashMap<>();

allKeywords

.forEach(s -> keywordsContainsMap.put(s, keywordsForCategory.contains(s)));

categoryToContainsKeywordsMap.put(category, keywordsContainsMap);

}

LOGGER.info("Конец обучения классификатора");

}

/\*\*

\* Метод для классификации набора слов (документа)

\*

\* @param words набор слов (документ)

\* @return ассоциативный массив категория -> вероятность попадания набора слов (документа) в эту категорию

\*/

public static Map<String, Double> classify(List<String> words){

RussianStemmer russianStemmer = new RussianStemmer();

words.removeIf(s -> s.length() < 4);

for (int i = 0; i < words.size(); i++) {

russianStemmer.setCurrent(words.get(i)

.toLowerCase());

russianStemmer.stem();

words.set(i, russianStemmer.getCurrent());

}

Map<String, Double> categoryProbabilityMap = new HashMap<>();

for (Map.Entry<String, Map<String, Boolean>> entry : categoryToContainsKeywordsMap.entrySet()) {

Map<String, Boolean> keywordsMap = entry.getValue();/\* V, все ключевые слова \*/

String category = entry.getKey();

double d = Math.log(BayesClassifier

.categoryProbabilityMap.get(category));/\* log(Dc/D) \*/

double sum = 0d;/\* сумма логарифмов \*/

for (String word : words) {

long denominator = categoryToContainsKeywordsMap

.size() + categoryCountKeywordsMap.size();

if (keywordsMap.containsKey(word)) {

sum += Math.log((1d +

(keywordsMap.get(word) ? 1 : 0)) / denominator);

} else {

sum += Math.log(1d / denominator);

}

}

double resultForCategory = d + sum;/\* значение логарифма для категории \*/

categoryProbabilityMap.put(category, resultForCategory);

}

Map<String, Double> copy = new HashMap<>(categoryProbabilityMap);

for (Map.Entry<String, Double> entry : categoryProbabilityMap.entrySet()) {

double s = categoryToContainsKeywordsMap.keySet()

.stream().filter(category -> !entry.getKey().equals(category))

.mapToDouble(category -> Math.exp(copy.get(category) - entry.getValue())).sum();

entry.setValue(1d / (1d + s));/\* делаем из логарифма вероятность (нормируем) \*/

}

return categoryProbabilityMap;

}

}