МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ ДОНЕЦКОЙ НАРОДНОЙ РЕСПУБЛИКИ

ИНФОРМАТИКА И КИБЕРНЕТИКА

3 (25)

Донецк – 2021

УДК 004.3+004.9+004.2+51.7+519.6+519.7

ИНФОРМАТИКА И КИБЕРНЕТИКА, № 3 (25), 2021,
Донецк, ДонНТУ.

Выпуск подготовлен по материалам XII Международной научно-технической конференции «Информатика, управляющие системы, математическое и компьютерное моделирование – 2021» (ИУСМКМ–2021), проведенной 26 – 27 мая 2021 г. в рамках VII Международного Научного форума Донецкой Народной Республики и посвященной 100-летию ДонНТУ. Также представлены другие материалы по вопросам приоритетных направлений научно-технического обеспечения в области информатики, кибернетики, вычислительной техники и инженерного образования.

Материалы предназначены для специалистов народного хозяйства, ученых, преподавателей, аспирантов и студентов высших учебных заведений.

Рекомендовано к печати ученым советом ГОУ ВПО «Донецкий национальный технический университет» Министерства образования и науки ДНР. Протокол № 8 от 08 ноября 2021 г.

Свидетельство о регистрации СМИ: серия ААА № 000145 от 20.06.2017.

Приказ МОН ДНР № 135 от 01.02.2019 о включении в Перечень рецензируемых научных изданий ВАК ДНР.

Контактный адрес редакции
ДНР, 83001, г. Донецк, ул. Артема, 58, ГОУ ВПО «ДонНТУ»,
4-й учебный корпус, к. 36., ул. Кобозева, 17.
Тел.: +38 (062) 301-07-35, +38 (071) 334-89-11
Эл. почта: infcyb.donntu@yandex.ru
Интернет: http://infcyb.donntu.org

СОДЕРЖАНИЕ

Информатика и вычислительная техника

Информатика и вычислительная техника

УДК 004.451.622: 004.724

Организация клиент-серверного взаимодействия для повышения производительности персональных компьютеров, участвующих в образовательном процессе

К. А. Сидоров, С. В. Иваница, А. Я. Аноприенко
Донецкий национальный технический университет
E-mail: ska@donntu.org

Аннотация
Определены требования к выбору методов повышения производительности ПК. Раскрыты подходы к применению клиент-серверной архитектуры и реализации разработанной технологии по принципу технологии «тонкий клиент» для повышения производительности маломощных ПК. Выполнен анализ группы операционных систем для клиентской части с определением оптимальной для имеющейся аппаратуры. Разработана и внедрена технология, повышающая производительность компьютерной техники, использующейся в образовательном процессе.

Введение

В год 100-летнего юбилея Донецкого национального технического университета (ДонНТУ), с учетом технического профиля вуза и его постоянно растущей информационно-компьютерной ориентации крайне критичным остается вопрос модернизации компьютерной инфраструктуры вуза.

Использование компьютерной техники в ДонНТУ началось в очень символичный день: 12 апреля 1961 года, когда было проведено первое занятие с использованием компьютерной техники и компьютерного моделирования [1, 2]. В 1970-е годы на базе ДонНТУ был организован мощный региональный Вычислительный центр и вуз стал одним из всесоюзных лидеров в разработке и применении средств компьютерного моделирования. При этом обеспечивалось постоянное наращивание выпуска компьютерных профессионалов, что позволяло обеспечивать высококвалифицированными кадрами потребности не только Донбасса, но и во многом всесоюзные потребности (выпускники ДонНТУ, например, неоднократно распределялись на космодром Байконур и другие объекты всесоюзного значения).

В украинский период истории ДонНТУ снабжение вуза компьютерным оборудованием резко ухудшилось. Однако за счет участия в международных научных и научно-образовательных проектах, вуз в этот период сохранил необходимый потенциал и стал, частности, единственным вузом в Украине, имевшим возможность использования в учебном процессе и научных исследованиях различных суперкомпьютерных ресурсов, имея, в частности, в своем распоряжении 2 различных суперкомпьютера (с традиционной архитектурой и на графических процессорах с суммарной производительностью свыше 5 TFlops [3]), полученных в начале 2010-х годов в ходе успешного выполнения международных исследовательских проектов.

С началом военных действий в 2014-м году возможности обновления компьютерной техники продолжили сужаться: в период с 2014-го по настоящее время вуз благодаря помощи Министерства образования и науки Донецкой Народной Республики смог обновить в общей сложности всего несколько десятков рабочих мест. Тотальное устаревание компьютерного парка приводит к тому, что в научных исследованиях и учебном процессе не может использоваться современное программное обеспечение, в том числе современные системы компьютерного проектирования и моделирования, что чрезвычайно критично для любого современного технического вуза.

Согласно сложившейся ситуации с оснащенностью современными ПК компьютерных классов, сотрудниками Центра информационных компьютерных технологий ДонНТУ разрабатываются и внедряются программы повышения производительности участвующей в учебном процессе компьютерной техники с использованием имеющихся суперкомпьютерных ресурсов. В вузе на данный момент продолжают функционировать компьютерные классы с устаревшими персональными компьютерами, типичные характеристики которых имеют следующие показатели: Pentium 333 МГц, ОЗУ 32–64 МБ,

HDD до 2 ГБ, годы выпуска 1995–2005.

Целью данной статьи является раскрытие подходов к применению клиент-серверной архитектуры и реализации разработанной технологии по принципу технологии «тонкий клиент» для повышения производительности использующихся в образовательном процессе устаревших персональных компьютеров, а в целом – повышения эффективности проведения практических и лабораторных занятий в компьютерных классах вуза.

Выбор методов повышения производительности ПК

Задача выбора методов увеличения производительности «слабых» персональных компьютеров основывается на подборе клиент-серверной системы таким образом, чтобы удовлетворить следующим требованиям:

1) Возможность работы ПК с операционной системой не ниже Windows XP (непосредственное использование компьютеров с указанными выше параметрами затруднительно даже с ОС Windows 98).

2) Комфортная работа пользователей с прикладным программным обеспечением и стабильная работа с сетевыми приложениями.

3) Минимальная нагрузка на сетевой трафик, поскольку сервер, который выполняет все операции для ПК, передает клиенту изображение рабочего стола (видеопоток) и получает инструкции от пользователя.

В ходе исследования существующих готовых решений, было выявлено, что все они основаны на клиент-серверной архитектуре с использованием сетевой программы-клиента, которая переносит все или большую часть задач по обработке информации на сервер. Под подобными решениями подразумевается достаточно широкий с точки зрения системной архитектуры ряд устройств и программ, которые объединяются общим свойством: возможностью работы в терминальном режиме – в режиме «тонкого клиента».

Физически тонкий клиент представляет собой компактный компьютер с минимальными параметрами (часто без жесткого диска и без вентиляторов), загрузка основной операционной системы которого происходит на сервере. Все пользовательские приложения выполняются на терминальном сервере (сервере приложений), но для пользователя это совершенно прозрачно. Так как вся вычислительная нагрузка ложится на сервер, то тонкий клиент обладает минимальной аппаратной конфигурацией без какого-либо ущерба производительности [4]. В нашем случае в качестве физического тонкого клиента будут выступать использующие в компьютерных классах ПК низкой производительности, а в качестве сервера – двухпроцессорные серверы NEC Express 5800 120RE-1 с процессором Intel Xeon 3,2 GHz, являющиеся вычитательными блоками параллельного кластера MIMD-архитектуры NEC SX8 на 100 узлов [1].

В результате исследования были рассмотрены семь готовых решений для реализации тонкого клиента из имеющихся ПК, которые предоставляют клиенту удаленный рабочий стол ОС Windows, управляемый с сервера, имеющего ОС семейства Linux:

1. *Windows rdesktop* – программное обеспечение с открытым исходным кодом, которое позволяет подключаться к клиенту с помощью RDP – протокола удаленного рабочего стола [5].

2. *Linux Terminal Server Project (LTSP)* – свободно распространяемый дополнительный пакет для Linux с открытым исходным кодом, который позволяет маломощным компьютерам (терминалами) использовать вычислительные мощности более производительного компьютера (сервера).

3. *Terminal Server X2g0* – терминальный сервер, работающий по протоколу NX [6], имеющий собственную серверную и клиентскую части, которые не совместимы с другими вариантами серверов и клиентов NX.

4. *ThinStation* – дистрибутив Linux, разработанный специально для создания тонких клиентов («урезанный» Linux с предустановленными программами, необходимыми для работы сети) [7].

5. *WTware* – операционная система для тонких клиентов, обеспечивающая рабочий стол терминального сервера Windows на экране пользователя.

6. *SPICE* – протокол для независимой вычислительной среды. Позволяет работать с виртуальным рабочим столом, в том числе, через Интернет, причем и на стороне «клиента», и на стороне «сервера» могут выступать различные операционные системы и аппаратные платформы. Конкурирует с протоколами RDP и NX.

7. *Virtual Network Computing (VNC)* – система удаленного доступа к рабочему столу компьютера, использующая протокол RFB (Remote Frame Buffer – удаленный кадровый буфер). Управление осуществляется путем передачи нажатий клавиш на клавиатуре и движений мыши с одного компьютера на другой и ретрансляции содержимого экрана через компьютерную сеть [8].

В результате анализа каждой из вышеописанных технологий, и ни одна не удовлетворяла всех требований, поскольку разнообразные конфигурации ПК требовали создание, с одной стороны, единой (комплексной) системы, а, с другой стороны,

системы, обеспечивающей повышение быстродействия клиентов, в качестве которых выступали различные конфигурации ПК.

Организация клиент-серверного взаимодействия

Собственное решение проблемы производительности ПК составляет комбинирование рассмотренных выше различных систем и протоколов, которые работают одновременно, но в ряде случаев выбирается одна из них в зависимости от ситуации.

Для серверной части выбрана операционная система Linux Ubuntu 16.04, поскольку данная ОС выделяется стабильной работой, простотой администрирования и большой группой (сообществом) поддержки и решения возникающих проблем. Кроме того, это свободная операционная система (что важно при применении в государственной образовательной организации) и применяется как для персональных компьютеров, так и для серверов. Ubuntu является одним из самых популярных дистрибутивов Linux. В случае с клиентской частью также было проведено исследование различных операционных систем, но основной сложностью было найти «современную» систему, способную работать на 64 МБ оперативной памяти. Параметры исследуемых операционных систем приведены в таблице 1.

Таблица 1. Сравнение различных операционных систем для клиентской части

Операционная система	DSL	Slax	Puppy Linux	Slitaz	Lubuntu
Минимальный объем ОЗУ (заявл. / реал.)	32 / 64 МБ	128 / 256 МБ	32 / 256 МБ	35 / 128 МБ	128 / 512 МБ
Минимальный объем HDD	150 МБ	300 МБ	300 МБ	300 МБ	4 ГБ
Процессор	486DX	i686 (32-bit Intel x86 arch)	x86, x86-64, ARM	i486	Pentium 4

Согласно таблице 1, приведенные операционные системы (дистрибутивы) имеют ряд особенностей.

ОС DSL (англ. Damn Small Linux) изначально была разработана в качестве эксперимента для определения максимального размещения полезных настольных приложений на 50-мегабайтном Live CD. Для данного дистрибутива необходимы незначительные (по сравнению с другими рассматриваемыми системами) системные требования и позволяет работать на «слабых» процессорах 486DX всего с 16 МБ оперативной памяти. Полная версия дистрибутива запускается на машине-клиенте, имеющим минимум 128 МБ оперативной памяти [9]. В ходе экспериментов было выявлено, что для стабильной работы системы необходимо не менее 64 МБ ОЗУ. Одним из недостатков данного дистрибутива является отсутствие в репозиториях современных версий прикладного ПО, поддерживающихся данной ОС.

Операционная система Slax базируется на Debian GNU/Linux и является Live-дистибутивом. Live означает, что система может работать прямо со съемного носителя, без установки. Также для полноценной работы данной системы необходимо минимум 128 МБ оперативной памяти. Slax может работать на процессорах 32bit или 64bit архитектуры. Данная система использует репозитории ОС Debian [10]. Преимуществом данной системы является наличие современных версий всего необходимого программного обеспечения. К недостаткам Slax можно отнести значительный объем необходимой оперативной памяти, что не всегда соответствует имеющимся аппаратным ресурсам.

Семейство дистрибутивов Puppy Linux операционной системы GNU/Linux разработано с целью создания нового дистрибутива, внешне похожего на интерфейс ОС Windows, который на момент создания имел необходимые приложения, но при этом не требовал большого дискового пространства (минимум 70 МБ) и незначительные системные требования (достаточно использовать процессор Pentium и ОЗУ – 32 МБ) [11]. Однако современные версии Puppi linux требуют более 512 МБ ОЗУ, что можно отнести к его недостатку.

Slitaz – компактный дистрибутив Linux, подходящий для устаревших (и, соответственно, маломощных) компьютеров. Имеет системную структуру, схожую с ОС DSL, но, в то же время, занимает меньше дискового пространства и основан на более поздних версиях ядра Linux. Новые версии выпускаются еженедельно. Системные требования могут различаться в зависимости от версии дистрибутива. Для загрузки SliTaz версии 4.0 требуются:

– i486-совместимый процессор;

– 192 Мбайт ОЗУ при загрузке «core» версии с LiveCD («slitaz-loram» требуется 80 МБ, а редакции «slitaz-loram-cdrom» — 16 МБ);

– 16 МБ ОЗУ при загрузке установленного на жесткий диск дистрибутива.

В ходе экспериментов было выявлено, что пи установке и запуске всех необходимых компонентов данная система требовала около 100 МБ ОЗУ, что является недостатком Slitaz. Так же сказывается отсутствие современных

версий программного обеспечения в репозиториях.

Lubuntu – легковесный и энергоэффективный дистрибутив Linux с малым потреблением ресурсов, производный от Ubuntu. В качестве графического окружения используется LXDE. Он подходит для нетбуков, портативных устройств и персональных компьютеров слабой производительности [11].

Системные требования для Lubuntu:

– минимум 1 ГБ ОЗУ;

– процессор Pentium IV и выше;

– минимум 6 ГБ свободного дискового пространства.

Эти требования намного выше исследуемых операционных систем, что является недостатком для Lubuntu.

В результате проведенного анализа данных (табл. 1) принято решение использовать два типа подключений:

1) Для компьютеров до 256 МБ ОЗУ – использование операционной системы Damn Small Linux (DSL). Преимуществом данной системы является ее способность работать на компьютерах с минимум 64 МБ оперативной памяти [12]. Простота использования в локальной сети вуза: необходимо только наличие SSH-клиента для подключения к серверу и Xorg-сервера для отображения графики.

2) Для компьютеров свыше 256 МБ ОЗУ – использование свободно распространяемого пакета LTSP. Если для реализации LTSP достаточно было установить программу сервер, а на клиенте иметь возможность PXE загрузки, то в случае применения операционной системы DSL была разработана следующая методика:

– во время загрузки ПК, на стороне клиента выполняется скрипт:

```
ssh -X user@10.64.0.N startvm.sh,
```

который подключается по протоколу SSH к серверу, который запускает программу в графическом режиме и выводит ее на экран клиента;

– для автозапуска конкретной виртуальной машины для определенного клиента на сервере был написан новый скрипт, который и вызывается при подключении по SSH:

```
#!/bin/sh
MACHINE=b22bc639-0bad-43a8-82fa-
2e63469fd90d
# start vm
/usr/lib/virtualbox/VirtualBox --
comment "winxp_user" --startvm
$MACHINE --fullscreen &
# wait until vm starts
sleep 5
# polling for vm poweroff
until $(VBoxManage showvminfo --
machinereadable $MACHINE | grep -q
^VMState=.poweroff.)
do
  sleep 1
done
```

Серверная часть также имеет два типа работы. Подключение к системе Linux и подключение к виртуальной машине с установленной операционной системой Windows XP.

Следующие моменты, которые подлежали исследованию, – пределы пропускной способности локальной сети и максимальное количество клиентов, с которые может обслуживать один сервер. Сравнение характеристик «исходного» ПК (локальная машина) и ПК-клиента приведены на рис. 1.

Рисунок 1 – Сравнение локальной машины и локальной машины, подключенной к серверу

Эмпирическим путем было определено то, что основным ограничением является объем оперативной памяти. На серверах установлено по 4 ГБ оперативной памяти. В случае подключения клиентов к операционной системе Linux комфортная работа обеспечивается для 4–5 клиентов при условии выполнения простых работ (офисные программы, браузер, редактор кода). Исходя из этого, было принято решение на одном сервере размещать максимум четыре виртуальные машины, под которые отводилось по 700 МБ оперативной памяти, что является достаточным для работы базового пакета программного обеспечения.

В случае расчета сетевой нагрузки анализ осложнялся тем, что нагрузка не могла задаваться постоянной величиной. При этом теоретически достаточная скорость не превышает 3 Мбит/с. Данная цифра является актуальной для передачи постоянного видеопотока в разрешении 480p (704x480 пикселей).

Показатели сетевой нагрузки при воспроизведении потокового видео для разных разрешений приведены на рис. 2. Для получения значений был проведен следующий эксперимент. Два ПК были подключены непосредственно друг к другу, на одной из них запускалась трансляция видеофайла, на второй осуществлялся просмотр данного видеопотока с одновременным замером нагрузки на сеть.

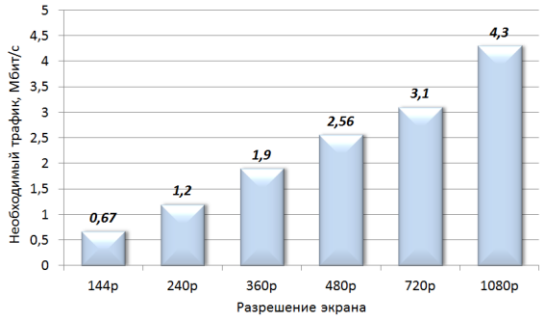


Рисунок 2 – Гистограмма сетевого трафика потокового видео при разном разрешении экрана

При использовании ПК в качестве тонкого клиента, были получены следующие показатели.

На рис. 3 приведен трафик генерируемый одним сервером с 4мя клиентами в которых запущена виртуальная машина, но не ведется больше никакая работа.

Следовательно, трафик в режиме простоя 4х клиентов не превышает 3 Мбит/с. Но в процессе нагрузки и активной работы клиентов наблюдаются резкие пики до 6–7 Мбит/с. При этом подобные пики непродолжительны и не оказывают значимого воздействия на стабильную работу сети и, соответственно тонких клиентов.

На рис. 4 изображен трафик 10 тонких клиентов, с пиком в 6,7 Мбит/с, при этом общий трафик не превышает 20 Мбит/с.

Если принять скорость 20 Мбит/с в качестве усредненного показателя суммарного трафика от 10 клиентов, то рост этого показателя увеличивается пропорционально количеству подключенных клиентов.

Логическая схема сетевого взаимодействия серверов и тонких клиентов для планируемой системы показана на рис.5.

```
10.64.0.7        => 10.176.0.216                          13,4Mb  2,98Mb  1,71Mb
                 <=                                        202Kb  57,8Kb  31,9Kb
10.64.0.7        => 10.176.0.198                          11,3Mb  2,28Mb  4,40Mb
                 <=                                        191Kb  42,5Kb  86,9Kb
10.64.0.7        => 10.176.0.211                          44,2Kb  34,6Kb  90,7Kb
                 <=                                       9,34Kb  6,87Kb  7,78Kb
10.64.0.7        => 10.176.0.212                          12,1Kb  12,9Kb  12,5Kb
                 <=                                       1,06Kb  2,31Kb  1,40Kb

TX:              cum:    211MB   peak:   24,8Mb  rates:   24,8Mb  5,33Mb  6,22Mb
RX:                    8,56MB           2,17Mb           2,17Mb   481Kb   234Kb
```

Рисунок 3 – Трафик с одного сервера с четырьмя клиентами

```
10.176.0.200     => 10.64.0.4                             125Kb   123Kb   110Kb
[redacted]       <=                                      6.73Mb  6.40Mb  5.35Mb
10.176.0.55      => 10.64.0.6                            1.27Kb   135Kb   143Kb
[redacted]       <=                                      12.3Kb  5.97Mb  7.38Mb
10.176.0.216     => 10.64.0.7                             160Kb  96.7Kb  45.5Kb
[redacted]       <=                                      1.56Mb  2.22Mb  1.48Mb
10.176.0.212     => 10.64.0.7                             111Kb  41.2Kb   143Kb
[redacted]       <=                                      6.22Mb  2.26Mb  8.56Mb
10.176.0.198     => 10.64.0.7                             160Kb   151Kb   137Kb
[redacted]       <=                                      1.56Mb  1.47Mb  5.24Mb
10.176.0.193     => 10.64.0.8                            1.39Kb  6.81Kb  3.26Kb
                 <=                                      12.7Kb  36.4Kb  22.0Kb
10.176.0.192     => 10.64.0.8                            1.06Kb  1.06Kb  56.3Kb
                 <=                                      12.1Kb  12.1Kb  3.29Mb
10.176.0.70      => 10.64.0.8                            1.06Kb  1.06Kb  1.05Kb
                 <=                                      12.1Kb  12.1Kb  12.7Kb
10.176.0.211     => 10.64.0.7                            1.06Kb  1.06Kb  1.05Kb
                 <=                                      12.1Kb  12.1Kb  12.7Kb
10.176.0.15      => 10.64.0.6                            1.84Kb  1.49Kb  1.62Kb
                 <=                                      2.84Kb  2.19Kb  2.32Kb

TX:              cumm:  2.48MB   peak:   1.05Mb  rates:    783Kb   774Kb   780Kb
RX:                     106MB           58.1Mb           19.3Mb  20.6Mb  32.5Mb
TOTAL:                  108MB           59.1Mb           20.1Mb  21.3Mb  33.3Mb
```

Рисунок 4 – Трафик 10-ти тонких клиентов

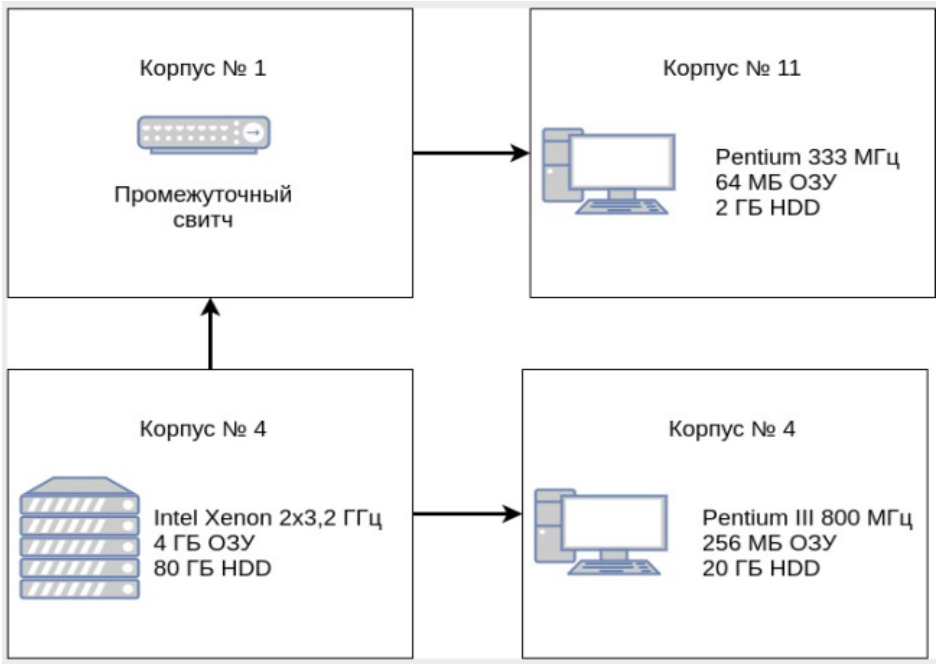


Рисунок 5 – Логическая схема сетевого взаимодействия серверов и тонких клиентов

При этом пропускная способность промежуточных сетевых компонентов ЛВС вуза (например, свитчей, которые расположены на пути от клиентов до серверов) позволяет подключить до 50 клиентов, что является избыточным для планируемого подключения 26 клиентов.

Стоит так же отметить, что серверы и тонкие клиенты расположены в разных учебных корпусах, расстояние между которыми достигает 2,5 км.

Выводы

Выбранная технология полностью отвечает поставленным требованиям, вписывается в граничные показатели сетевых нагрузок и обеспечивает повышение производительности ПК в компьютерных классах.

Так же разработанная и внедренная технология позволила продлить использование оборудования с низкой производительностью, которое в персональном режиме работы использовать не представляется возможным.

Литература

1. Аноприенко, А. Я. Вычислительная техника и информатика в ДонНТУ: люди, события, факты: первые 50 лет / А. Я. Аноприенко, В. А. Святный. – Донецк : УНИТЕХ, 2011. – 264 с.

2. Аноприенко, А. Я. Высокопроизводительные информационно-моделирующие среды для исследования, разработки и сопровождения сложных динамических систем / А. Я. Аноприенко, В. А. Святный // Наукові праці Донецького державного технічного університету. – Донецьк, 2001. – С. 346–367. – (Серія «Проблеми моделювання та автоматизації проектування динамічних систем» ; вип. 29).

3. Аноприенко А. Я. Вызовы времени и постбинарный компьютинг / А. Я. Аноприенко // Информатика и компьютерные технологии / Материалы VI международной научно-технической конференции – 23–25 ноября 2010 г. Т. 1. Донецк, ДонНТУ. – 2010. С. 13–31.

4. Тищенко, Е. Н. Исследование и моделирование систем доверенной загрузки «тонкого клиента» / Е. Н. Тищенко, К. А. Буцик // NBI-technologies. 2017. № 4. URL: https://cyberleninka.ru/article/n/issledovanie-i-modelirovanie-sistem-doverennoy-zagruzki-tonkogo-klienta (дата обращения: 29.04.2021).

5. Рожков, С. А. Терминальная система WTPRO: архитектура, функциональность, схема применения // Вестник ЮУрГУ. Серия: Компьютерные технологии, управление, радиоэлектроника, 2009. № 26 (159). URL: https://cyberleninka.ru/article/n/terminalnaya-sistema-wtpro-arhitektura-funktsionalnost-shema-primeneniya (дата обращения: 29.04.2021).

6. Official X2Go Packages in GNU/Linux Distributions // Retrieved July 3, 2020. URL: http://wiki.x2go.org (дата обращения: 29.04.2021).

7. Двоеглазов, Д. В. Инфраструктура виртуальных рабочих столов на открытых программных продуктах / Д. В. Двоеглазов, И. П. Дешко, К. Г. Кряженков, А. А. Тихонов // Вестник евразийской науки, 2015. № 4 (29). URL: https://cyberleninka.ru/article/n/infrastruktura-virtualnyh-rabochih-stolov-na-otkrytyh-programmnyh-produktah (дата обращения:

29.04.2021).

8. Tristan, R. Virtual Network Computing / Tristan Richardson, Quentin Stafford-Fraser, Kenneth R. Wood & Andy Hopper // IEEE Internet Computing, Vol. 2, No. 1, Jan/Feb 1998. P. 33–38.

9. What is DSL? Damn Small Linux is a very versatile 50MB mini desktop oriented Linux distribution. URL: http://www.damnsmalllinux.org/.

10. Найбук, М. Н. Программно-аппаратный модуль gui-suprem III для проектирования и обучения в глобальной сети Интернет технологии в микроэлектронике / М. Н. Найбук, В. В. Нелаев // Доклады Белорусского государственного университета информатики и радиоэлектроники, № 4 (20), 2007. – С. 136–144.

11. Курицын, А. Рейтинг 30-ти популярных Unix-подобных операционных систем в России / А. Курицын, А. Воробьев, Д. Зайцев. – Иннов: электронный научный журнал, № 3 (20), 2014. – С. 3–8.

12. Andrews, J. Damn Small Linux. Embedded on a USB pen drive – can run on Linux or in Windows / John Andrews. Foster City, CA. URL: http://www.damnsmalllinux.org/usb-qemu.html (дата обращения: 30.04.2021).

Сидоров К. А., Иваница С. В., Аноприенко А. Я. Организация клиент-серверного взаимодействия для повышения производительности персональных компьютеров, участвующих в образовательном процессе. *Определены требования к выбору методов повышения производительности ПК. Раскрыты подходы к применению клиент-серверной архитектуры и реализации разработанной технологии по принципу технологии «тонкий клиент» для повышения производительности маломощных ПК. Разработана и внедрена технология, повышающая производительность компьютерной техники, использующейся в образовательном процессе.*

Ключевые слова: *тонкий клиент, клиент-серверная архитектура, сервер, сетевой трафик, виртуальная машина, операционная система.*

Sidorov K., Ivanitsa S., Anopriyenko A. Organization of client-server interaction to improve the performance of personal computers involved in the educational process. *The requirements for the choice of methods for increasing PC performance have been determined. The approaches to the application of the client-server architecture and the implementation of developed technology on principle of technology "thin client" to improve the performance of low-power PCs. A technology has been developed and implemented that increases the productivity of computer technology used in the educational process.*

Keywords: *thin client, client-server architecture, server, network traffic, virtual machine, operating system.*

Статья поступила в редакцию 20.08.2021
Рекомендована к публикации профессором Мальчевой Р. В.

УДК 004.7

Мультифрактальный анализ задержки TCP-пакетов беспроводной сети

Д. В. Бельков, В. И. Зензеров, Е. Н. Едемская
Донецкий национальный технический университет
E-mail: belkovdv@list.ru

Аннотация
В работе с помощью алгоритма мультифрактального флуктуационного анализа MFDFA выполнен анализ задержки TCP-пакетов беспроводной сети. В первой серии экспериментов отправитель имел GPRS-доступ, получатель–UMTS-доступ, скорость передачи 100 pps, операционная система Windows32, на каждой из сторон. Во второй серии экспериментов – со стороны отправителя – операционная система Windows32, на стороне получателя – операционная система Linux. Изучаемые временные ряды являются мультифрактальными. Ширина мультифрактального спектра изучаемых рядов растет с увеличением объема пакета.

Введение

В связи с тенденциями объединения различных телекоммуникационных приложений на базе универсальной сетевой инфраструктуры актуальной проблемой является разработка методов анализа и синтеза информационных систем. Данная работа посвящена одной из важных задач развития этого направления – исследованию процессов в компьютерных сетях с беспроводным доступом.

Многообразие видов сетей и способов выделения сетевого ресурса для обслуживания трафика требует разработки моделей, которые учитывают реальный характер потоков сообщений и детали обслуживания трафика.

В современных сетях потоки пакетов формируются множеством источников требований на предоставление сетью услуги и сетевых приложений, обеспечивающих услуги передачи информации. Пользователи, создающие потоки пакетов, существенно отличаются между собой значениями удельной интенсивности нагрузки, и поэтому трафик является неоднородным. Он имеет определенные требования к QoS и долгосрочные зависимости в интенсивности, например, из-за повторной передачи неверно принятых пакетов.

Передачу потоков различных служб обеспечивает единая сеть. Поскольку источники каждой службы могут иметь разные скорости передачи информации или изменять ее в процессе сеанса связи, то потокам пакетов свойственна пачечность (burstness). В реализации трафика присутствует значительное количество больших пульсаций при относительно малом среднем уровне трафика. Пакеты поступают на узел не по отдельности, а пачкой. Это явление ухудшает характеристики (увеличивает потери, задержки, джиттер пакетов) при прохождении трафика через узлы сети. Известно [1-4], что пачечный трафик имеет фрактальную структуру.

Фрактальный трафик обладает свойством масштабной инвариантности. В отличие от однородного трафика, он сохраняет внешние признаки при рассмотрении в разном масштабе. Поэтому методы моделирования и расчета сетей, основанные на использовании Пуассоновских потоков, не дают точной картины процессов, происходящих в сети. Учет фрактальности трафика позволит более точно описать трафик. Это обеспечит возможность получения заданных показателей качества обслуживания.

Известно [5,6], что фрактальная структура системы является результатом ее самоорганизации в критическое состояние. Одним из способов, позволяющим исследовать фрактальные свойства временного ряда, является метод нормированного размаха, предложенный Херстом для выделения долговременных тенденций в рядах гидрометеорологических величин и получивший распространение при выявлении фрактальных характеристик временных последовательностей. Однако метод нормированного размаха (R/S-анализ) позволяет получить степень фрактальности лишь монофрактальных рядов. При R/S-анализе не рассматриваются локальные свойства, учет которых достигается при использовании метода мультифрактального флуктуационного анализа.

В данной статье при исследовании сетевого трафика предложено использовать мультифрактальный анализ. Целью работы является исследование трафика сети с беспроводным доступом, направленное на выявление его характерных особенностей.

Изучается один из основных сетевых процессов – процесс RTT-задержки, который служит для получения информации о состоянии сети методом «черного ящика», когда через сеть пропускается последователь последовательность пакетов, и на основании времени их прохождения до удаленного узла и обратно делаются выводы о загрузке сети. Задачей работы является мультифрактальный анализ RTT-задержки TCP-пакетов различного объема при передаче по беспроводной сети.

Статистический анализ задержки TCP-пакетов при передаче по беспроводной сети был выполнен в работе [7]. Мультифрактальный анализ выполнен в среде Octave с помощью алгоритма MFDFA [8,9].

Метод мультифрактального анализа

Метод мультифрактального анализа сводится к следующим шагам [4]:

1. Для исследуемого ряда x(i), i=1, 2…, N следует выделить флуктуационный профиль, где $\bar{x}$ - среднее значение.

2. Полученные значения y(i) разделяются на $N_S = N/s$ непересекающихся сегментов равной длины s. При этом следует учесть, что длина ряда N не всегда кратна шкале s. Поэтому, чтобы не исключать из анализа последний участок, содержащий число элементов, меньшее s, следует повторить процедуру деления, начиная с противоположного конца ряда.

В результате получаем $2N_S$ сегментов $\nu = 1,2,..N_S, N_S+1,..2N_S$ длины s.

3. Используя метод наименьших квадратов, для профиля y(i), отвечающего каждому из этих сегментов, вычислить локальный тренд $y_\nu(i)$, представляющий полином, степень которого обеспечивает заданную точность. Затем для сегментов $\nu = 1,2,..N_S$ по формуле (1) определяется дисперсия:

$$F^2(\nu,s) = \frac{1}{s}\sum_{i=1}^{s}\{y[(\nu-1)s+i] - y_\nu(i)\}^2 . \quad (1)$$

Для $\nu = N_S+1,..2N_S$ используется формула (2):

$$F^2(\nu,s) = \frac{1}{s}\sum_{i=1}^{s}\{y[N-(\nu-N_S)s+i] - y_\nu(i)\}^2 . \quad (2)$$

4. Усредняя значения (1) и (2), деформированные произвольным показателем q, вычисляются моменты:

$$F_q(s) = \{\frac{1}{2N_S}\sum_{\nu=1}^{2N_s}[F^2(\nu,s)]^{q/2}\}^{1/q} . \quad (3)$$

При $q \to 0$ вместо (3) нужно использовать формулу (4)

$$F_0(s) = \exp\{\frac{1}{4N_S}\sum_{\nu=1}^{2N_s}\ln[F^2(\nu,s)]\} \quad (4)$$

Характерно, что при положительных показателях q основной вклад в сумму по ν дают сегменты, отвечающие большим отклонениям $F^2(\nu,s)$, а при отрицательных доминируют вклады малых флуктуаций.

5. Самоподобное поведение, означающее наличие дальнодействующих степенных корреляций, представляется степенной зависимостью моментов (3), (4):

$$F_q(s) \propto s^{h(q)} . \quad (5)$$

При фиксированном значении q эта зависимость в двойных логарифмических координатах представляет собой прямую линию. При больших значениях s зависимость $F_q(s)$ не имеет статистической информативности, поскольку число сегментов N_S, используемых в процедуре усреднения (3), (4), становится малым. При обработке ряда нужно исключить значения s>N/4, а также малые сегменты (s<6), для которых теряет статистическую достоверность усреднение (1), (2) по каждому из сегментов.

Если ряд экспериментальных данных является монофракталом, то обобщенный показатель Херста h(q) в равенстве (5) принимает единственное значение h(q)=H. В случае мультифрактального ряда показатель h зависит от q. Для стационарных рядов h(0) определяет топологическую размерность пространства, содержащего фрактальное множество, h(1) – меру его беспорядка, h(2) – показатель дальних корреляций.

В рамках стандартной фрактальной идеологии используется не только обобщенный показатель Херста h(q), но массовый показатель τ(q) и мультифрактальный спектр f(α). Этот переход достигается преобразованиями Лежандра:

$$\tau(q) = qh(q) - 1 ; \quad (6)$$

$$\alpha = \tau'(q),\ f(\alpha) = q\alpha - \tau(q) . \quad (7)$$

Для монофрактальных объектов функция τ(q) является прямолинейной зависимостью, которая с переходом к мульти фракталам выгибается, сохраняя прямолинейные участки. Наиболее ярко строение самоподобного объекта представляется формой мультифрактального спектра f(α), ширина которого дает набор фрактальных размерностей. Для монофракталов функция f(α) имеет δ-образную форму с фиксированным значением α [9].

Мультифрактальный анализ трафика

В работе выполнен вычислительный эксперимент, результаты которого свидетельствуют о мультифрактальности реального трафика RTT-задержки в беспроводной сети.

Для изучения выбраны три реализации сетевого трафика, полученные в университете города Наполи (Италия). Согласно лицензии, данные свободно доступны для анализа [10]. Изучаемые временные ряды представляют собой измерения задержки TCP-пакетов.

В первой серии экспериментов отправитель имел GPRS-доступ, получатель– UMTS-доступ, скорость передачи 100 pps, операционная система Windows32, на каждой из сторон. Во второй серии экспериментов со стороны отправителя – операционная система Windows32, на стороне получателя – операционная система Linux.

Условные обозначения, принятые в работе, показаны в таблице 1.

Таблица 1 - Условные обозначения

Обозначение	Описание
TCP_d64	Первая серия экспериментов. Ряд измерений задержки TCP-пакетов объемом 64 байт
TCP_d256	Первая серия экспериментов. Ряд измерений задержки TCP-пакетов объемом 256 байт
TCP_d1024	Первая серия экспериментов. Ряд измерений задержки TCP-пакетов объемом 1024 байт
TCP_d_winlin_512	Вторая серия экспериментов. Ряд измерений задержки TCP-пакетов объемом 512 байт
TCP_d_winlin_1024	Вторая серия экспериментов. Ряд измерений задержки TCP-пакетов объемом 1024 байт

Исходные данные первой серии экспериментов показаны на рисунках 1–3.

Исходные данные второй серии экспериментов показаны на рисунках 4,5.

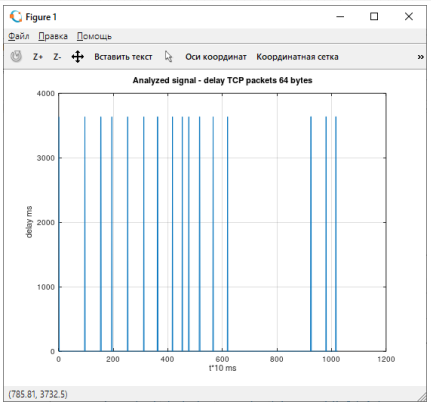


Рисунок 1 – Фрагмент ряда TCP_d64

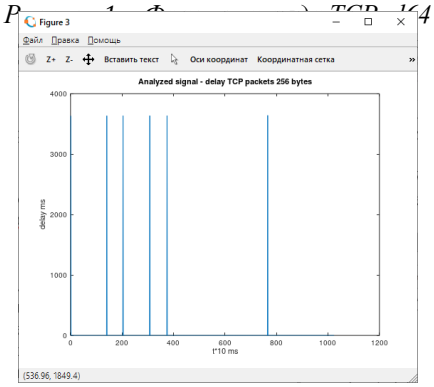


Рисунок 2 – Фрагмент ряда TCP_d256

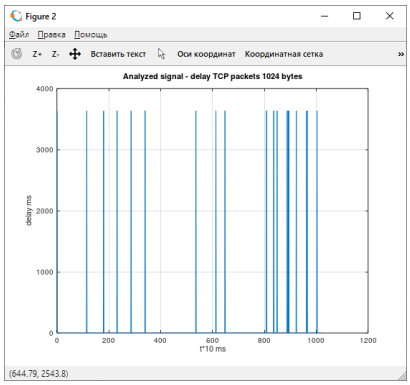


Рисунок 3 – Фрагмент ряда TCP_d1024

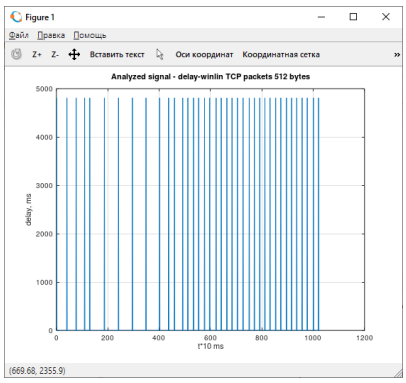


Рисунок 4 – Фрагмент ряда TCP _d_winlin_512

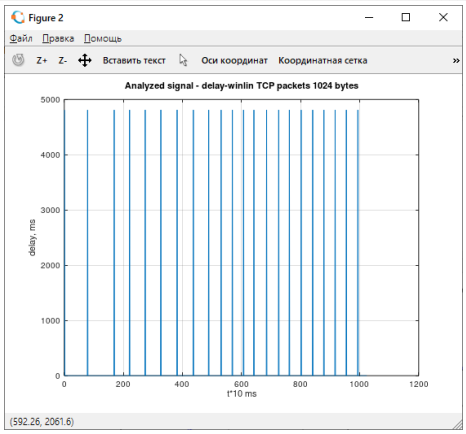


Рисунок 5 – Фрагмент ряда TCP_d_winlin_1024

Результаты мультифрактального анализа ряда TCP_d64 показаны на рисунке 6. Скейлинговая функция F_q и обобщенный показатель Херста H_q зависят от q, следовательно, временной ряд является мультифрактальным.

Массовый показатель $\tau(q)$ не является линейной функцией q, следовательно, временной ряд является мультифрактальным.

Ширина мультифрактального спектра равна 10,663. Поскольку она не является нулевой, то ряд TCP_d64 – мультифрактальный.

Результаты мультифрактального анализа ряда TCP_d256 показаны на рисунке 7. Скейлинговая функция F_q и обобщенный показатель Херста H_q зависят от q, следовательно, временной ряд является мультифрактальным.

Массовый показатель $\tau(q)$ не является линейной функцией q, следовательно, временной ряд является мультифрактальным.

Ширина мультифрактального спектра равна 11,915. Поскольку она не является нулевой, то ряд TCP_d256 – мультифрактальный.

Результаты мультифрактального анализа ряда TCP_d1024 показаны на рисунке 8.

Скейлинговая функция F_q и обобщенный показатель Херста H_q зависят от q, следовательно, временной ряд является мультифрактальным. Массовый показатель $\tau(q)$ не является линейной функцией q, следовательно, временной ряд является мультифрактальным.

Ширина мультифрактального спектра равна 12,108. Поскольку она не является нулевой, то ряд TCP_d1024 – мультифрактальный.

Результаты мультифрактального анализа ряда TCP _d_winlin_512 показаны на рисунке 9.

Скейлинговая функция F_q и обобщенный показатель Херста H_q зависят от q, следовательно, временной ряд является мультифрактальным. Массовый показатель $\tau(q)$ не является линейной функцией q, следовательно, временной ряд является мультифрактальным. Ширина мультифрактального спектра равна 9,335. Поскольку она не является нулевой, то ряд TCP_d_winlin_512 – мультифрактальный.

Результаты мультифрактального анализа ряда TCP _d_winlin_1024 показаны на рисунке 10.

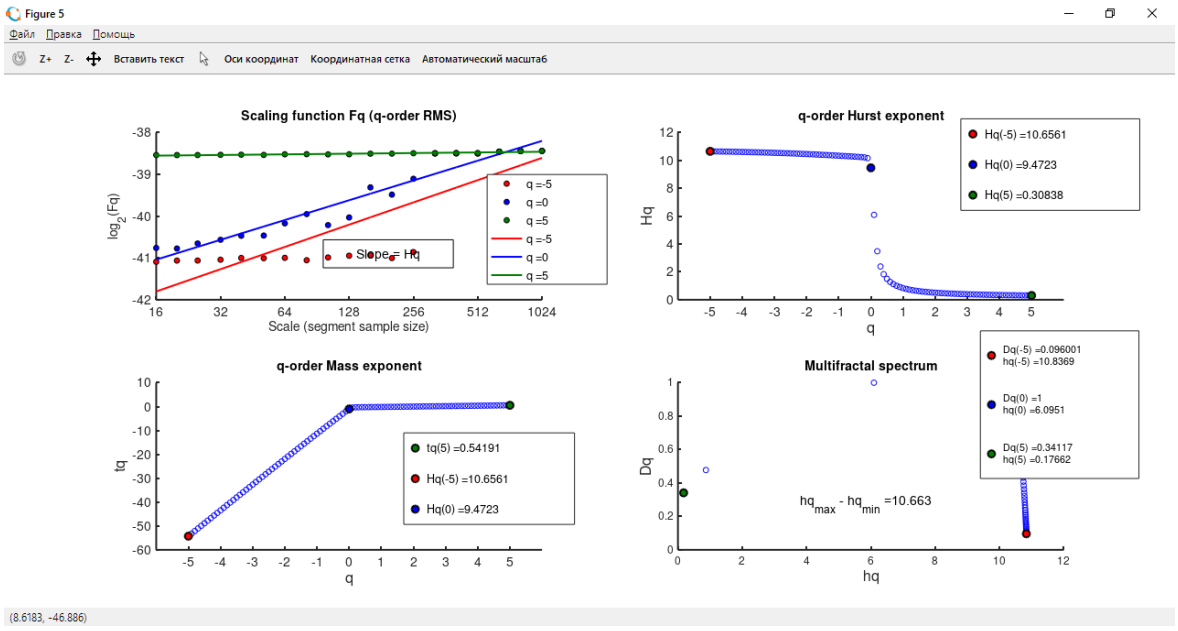


Рисунок 6 – Мультифрактальный анализ ряда TCP_d64

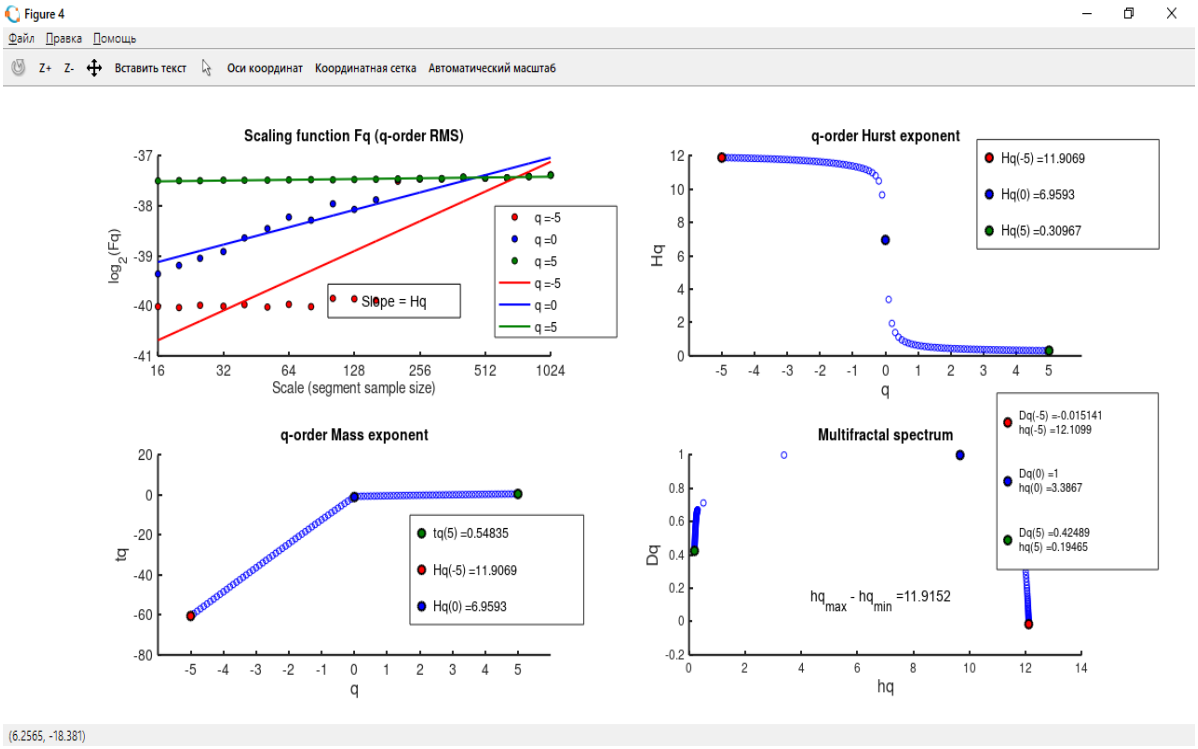


Рисунок 7 – Мультифрактальный анализ ряда TCP_d256

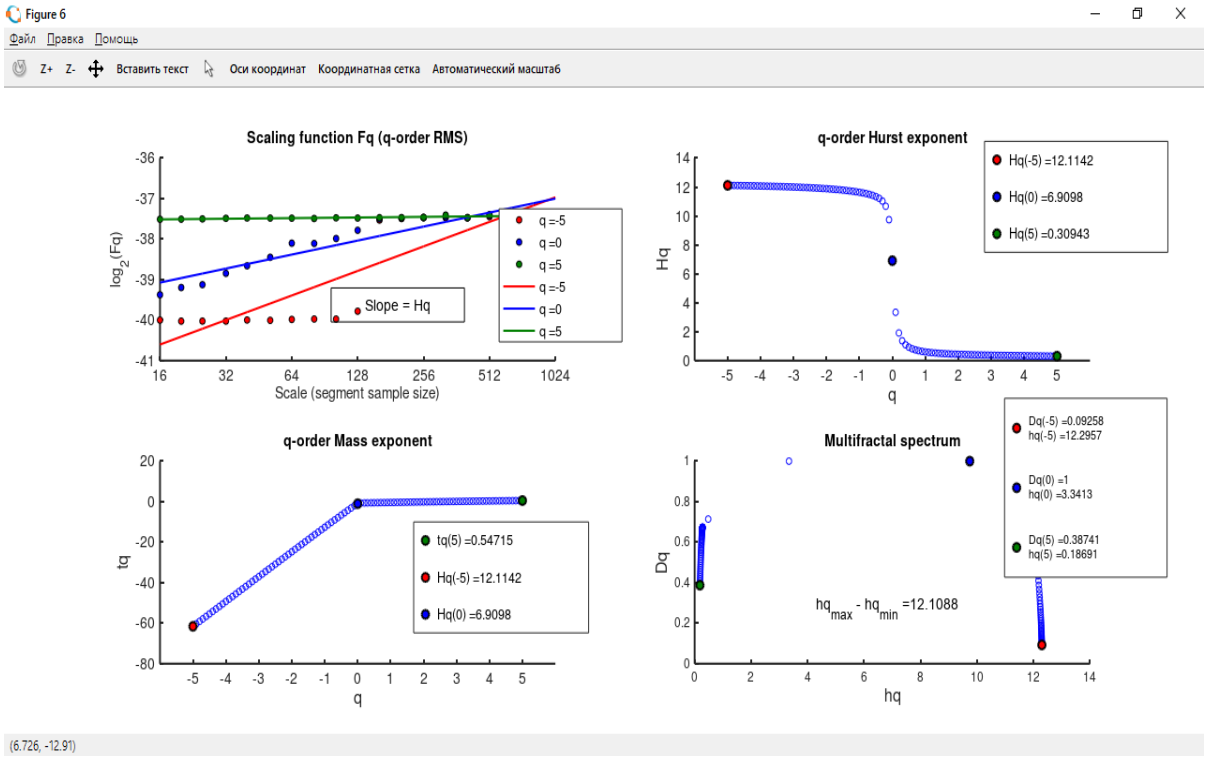


Рисунок 8 – Мультифрактальный анализ ряда TCP_d1024

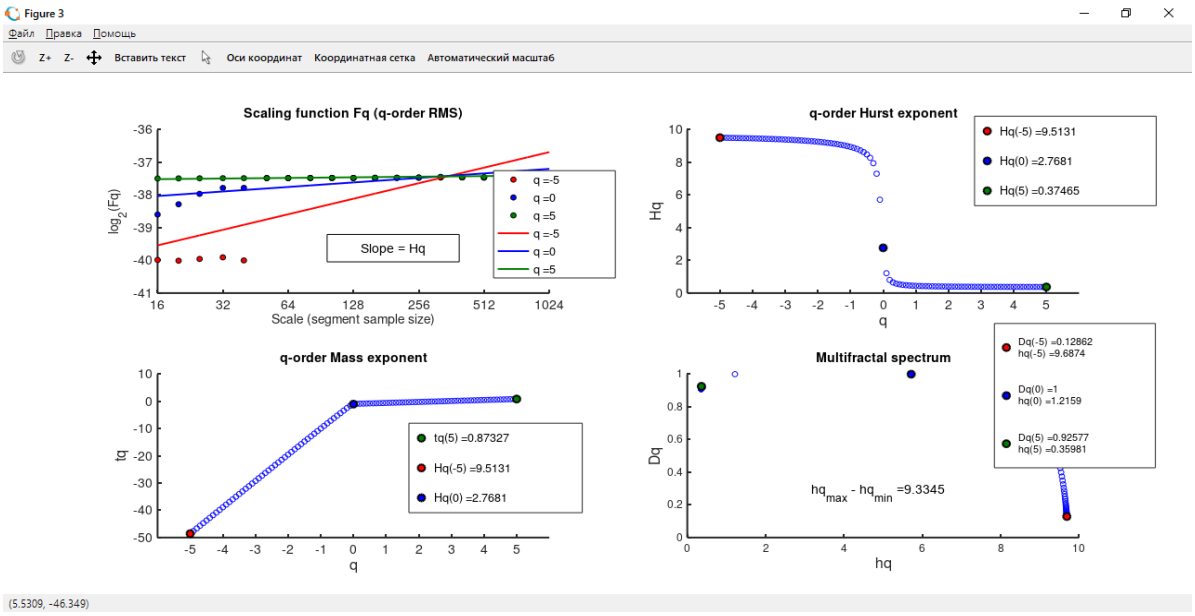


Рисунок 9 – Мультифрактальный анализ ряда TCP_d_winlin_512

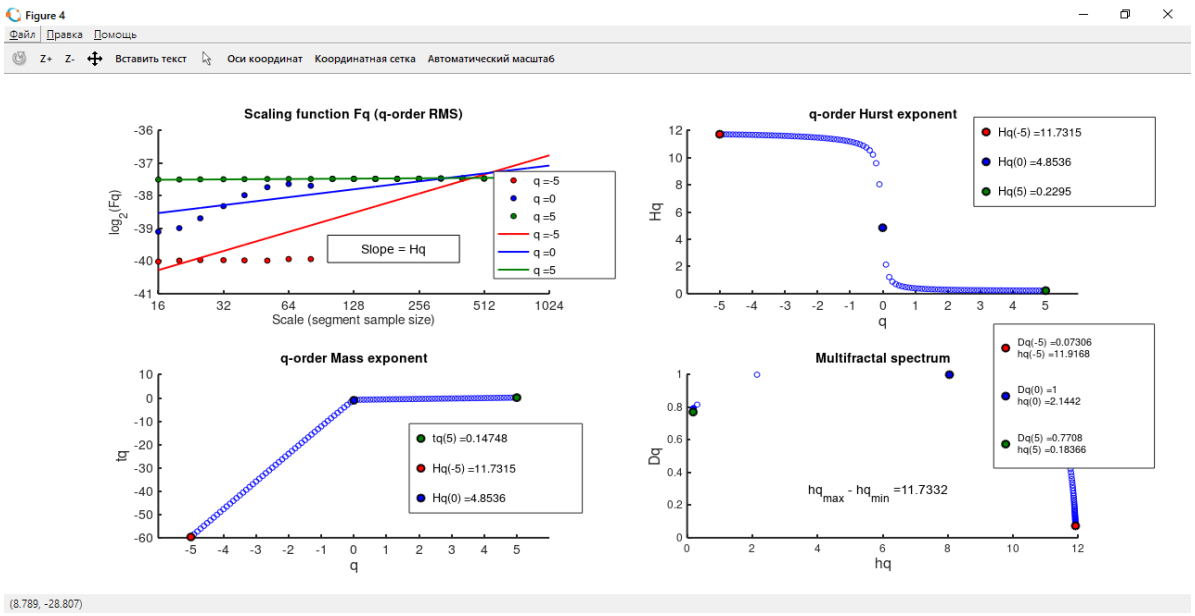


Рисунок 10 – Мультифрактальный анализ ряда TCP_d_winlin_1024

Скейлинговая функция F_q и обобщенный показатель Херста H_q зависят от q, следовательно, временной ряд является мультифрактальным. Массовый показатель $\tau(q)$ не является линейной функцией q, следовательно, временной ряд является мультифрактальным.

Ширина мультифрактального спектра равна 11,733. Поскольку она не является нулевой, то ряд TCP_d_winlin_1024 – мультифрактальный.

Выводы

В работе с помощью алгоритма MFDFA в среде Octave выполнен мультифрактальный анализ задержки TCP-пакетов беспроводной сети. Получены следующие результаты:

1. В первой серии экспериментов отправитель имел GPRS-доступ, получатель–UMTS-доступ, скорость передачи 100 pps, операционная система Windows32, на каждой из сторон. Временные ряды TCP_d64, TCP_d256, TCP_d1024 являются мультифрактальными.
2. Во второй серии экспериментов отправитель имел GPRS-доступ, получатель–

UMTS-доступ, скорость передачи 100 pps, Со стороны отправителя – операционная система Windows32, на стороне получателя – операционная система Linux. Временные ряды TCP_d_winlin_512, TCP_d_winlin_1024 являются мультифрактальными.

3. Ширина мультифрактального спектра изучаемых рядов растет с увеличением объема пакета.

Литература

1. Newton, N. J. Self similar model for bursty traffic – a deterministic approach. [Электронный ресурс], 2013. – Режим доступа: http://www.andonis.eu/documents/MScProject.pdf

2. Ложковський, А. Г. Аналіз і синтез систем розподілу інформації в умовах мультисервісного трафіка. Автореферат дисертації. Одеса. - 2010. – 38 с.

3. Бельков, Д. В. Статистический анализ сетевого трафика / Д. В. Бельков, Е. Н. Едемская, Л. В. Незамова // Зб. Наукових праць ДонНТУ. Серія "Інформатика, кібернетика, обчислювальна техніка". – Донецьк: ДонНТУ, 2011. - Вип. 13 (185). – С. 66 -75.

4. Тарасов, Д. В. Исследование характеристик системы мониторинга сетей связи следующего поколения. Автореферат диссертации. Санкт-Петербург. – 2012. – 22 с.

5. Подлазов, А. В. Теория самоорганизованной критичности – наука о сложности. [Электронный ресурс], 2008. – Режим доступа: http://www.nonlin.ru/articles/podlazov/soc

6. Бак, П. Как работает природа. Теория самоорганизованной критичности. – Москва, 2013. – 276 с.

7. Бельков, Д. В. Статистический анализ трафика сети с беспроводным доступом / Д. В. Бельков, Е. Н. Едемская // Зб. Наукових праць ДонНТУ. Серія "Інформатика, кібернетика, обчислювальна техніка". – Донецьк: ДонНТУ,2011.- Вип. 14 (188). – С. 113-122.

8. Introduction to multifractal detrended fluctuation analysis in Matlab [Электронный ресурс]. – Режим доступа: https://www.frontiersin.org/articles/10.3389/fphys.2012.00141/full

9. Олемской, А. Мультифрактальный анализ рентгеновских дифрактограмм сложных конденсированных сред [Электронный ресурс] / А. И. Олемской, С. Н. Данильченко, В. Н. Борисюк, И. А. Шуда. – Режим доступа: https://www.researchgate.net/publication/279505834_Multifraktalnyj_analiz_rentgenovskih_difraktogramm_sloznyh_kondensirovannyh_sred.

10. Network tools and traffic traces. [Электронный ресурс], 2007. – Режим доступа: http://www.grid.unina.it/Traffic/Traces/ttraces.php

Бельков Д.В., Зензеров В.И., Едемская Е.Н. Мультифрактальный анализ задержки TCP-пакетов беспроводной сети. *В работе с помощью алгоритма мультифрактального флуктуационного анализа MFDFA выполнен анализ задержки TCP-пакетов беспроводной сети. В первой серии экспериментов отправитель имел GPRS-доступ, получатель–UMTS-доступ, скорость передачи 100 pps, операционная система Windows32, на каждой из сторон. Во второй серии экспериментов – со стороны отправителя – операционная система Windows32, на стороне получателя – операционная система Linux. Изучаемые временные ряды являются мультифрактальными. Ширина мультифрактального спектра изучаемых рядов растет с увеличением объема пакета.*

Ключевые слова: *фрактальный трафик, мультифрактальный анализ, беспроводная сеть, TCP-пакеты.*

Belkov D.V., Zenzerov V.I., Edemskaya E.N. Multifractal analysis of delaying TCP-packets of wireless network. *MFDFA analyzes of the delay of wireless network TCP packets using the multifractal fluctuation analysis algorithm. In the first series of experiments, the sender had GPRS access, recipient-UMTS access, 100 pps transmission speed, Windows32 operating system, on each side. In the second series of experiments - on the part of the sender - the operating system Windows32, on the side of the recipient - the operating system Linux. The time series studied are multifractal. The width of the multi-fractal spectrum of the series studied increases with the volume of the packets.*

Keywords: *fractal traffic, multifractal analysis, wireless network, TCP packets.*

Статья поступила в редакцию 20.07.2021
Рекомендована к публикации профессором Павлышом В. Н.

УДК 519.87:004.94

Модель взаимодействия акторов облачной вычислительной среды при решении задач экологического мониторинга

Е. Н. Мащенко, В. И. Шевченко, О. В. Ченгарь
Севастопольский государственный университет
E-mail: elmachenko@mail.ru

Аннотация
Предложена математическая модель для сценариев взаимодействия акторов облачной вычислительной среды при решении задач обработки данных экологического мониторинга. Проведена формализация процессов эффективного назначения функциональных задач потребителей данных на ресурсы вычислительной среды с использованием аппарата линейного программирования, а именно в постановке несбалансированной задачи о назначениях с булевскими переменными.

Введение

Одна из тенденций в области информационных технологий для организации высокопроизводительных систем обработки данных экологического мониторинга связана с миграцией крупномасштабных вычислений в облачные среды. При этом такие проблемы как управление гетерогенными вычислительными ресурсами, поддержка интерактивных сервисов в режиме «реального» времени, обработка и хранение больших данных решаются на стороне провайдеров облачных услуг.

Для систем экологического мониторинга такой переход особенно значим, поскольку позволяет за счет контейнеризации технических компонент функционирования ИТ-сервисов и использования сервиса «самообслуживание по требованию» [1] с минимальными затратами адаптировать вычислительную среду к динамически меняющимся требованиям по предоставляемым ресурсам, объемам обрабатываемых данных (мониторинг в стабильных условиях/в условиях чрезвычайных ситуаций или экологических катастроф).

Согласно эталонной модели облачной вычислительной среды (ОВС), приведенной в работах [2, 3] и ее адаптации для обработки больших данных [4, 5] и распределенных систем экологического мониторинга [6] основными акторами ОВС являются Поставщик данных (датчики системы мониторинга, ДСД), Оркестратор системы (ОС), Провайдер приложений (ПП), Провайдер Сервисов (ПС), Потребитель данных (ПД).

Общая постановка проблемы

В рамках данной статьи будет рассмотрено взаимодействие «Потребитель данных – Провайдер приложений – Провайдер сервисов». При их взаимодействии возникает противоречие: потребители данных заинтересованы в минимизации затрат на обработку своих функциональных задач, Провайдеры сервисов заинтересованы в максимизации доходов от предоставления ИТ-сервисов, а Провайдеры приложений - в максимизации доходов от посреднической деятельности по передаче сырых данных на обработку определенному провайдеру сервисов. Необходимо найти равновесное решение, удовлетворяющее всех взаимодействующих акторов.

Постановка задачи

Целью исследований является разработка методологических основ для построения системы поддержки принятия решений по управлению взаимодействием акторов в гетерогенных облачных средах с целью эффективного управления вычислительными ресурсами и повышения качества ИТ-сервисов при решении крупномасштабных задач экологического мониторинга.

Для достижения цели исследований необходимо решить задачу формализации процессов эффективного назначения функциональных задач пользователей ОВС на ресурсы Провайдеров сервисов с использованием аппарата линейного программирования, а именно в постановке несбалансированной задачи о назначениях с булевскими переменными.

Модель сценария взаимодействия акторов «Поставщик данных –Провайдер сервисов»

В формализованном виде задача о распределении запросов Поставщика данных на обработку данных от систем мониторинга, выполняемых в рамках решения крупномасштабной научной задачи, на вычислительные ресурсы Провайдера сервисов

может быть сформулирована в классе транспортных задач математического программирования, а именно как задача о назначениях.

В обобщенной формулировке задача о назначениях заключается в распределении исполнителей по работам таким образом, чтобы максимизировать (минимизировать) суммарный критерий эффективности (неэффективности) выполнения всех работ [7, 8]. Если число исполнителей и число выполняемых работ совпадают, то задача является сбалансированной, в противном случае - несбалансированной. В случае сбалансированности задачи о назначениях выполняются два условия: каждый исполнитель выполняет только одну работу, каждая работа выполняется только одним исполнителем. Решение задачи назначения проектов Облачных Потребителей исполнителям, претендующим на их выполнение, направлено на повышение экономической эффективности исполнения проектов и сопряжено с большим количеством временных и стоимостных ограничений.

Рассмотрим простейший частный случай (*формулировка сценария S1*), когда каждый из Облачных Потребителей решает одну функциональную задачу (ФЗ) в рамках проекта крупномасштабных вычислений (кейс задач экологического мониторинга), между задачами нет отношений предшествования. Для обработки задач Потребителей в ОВС каждый из Провайдеров использует только один фиксированный тарифный план предоставления ИТ-сервисов. Необходимо назначить все ФЗ из кейса задач Потребителей на обработку в инфраструктурах Провайдеров так чтобы: а) минимизировать расходы Потребителей; б) максимизировать доходы Провайдеров.

Введем следующие обозначения: i – индекс Провайдера, $i = 1,2, \dots I$, где I – множество Провайдеров ОВС; j – индекс Потребителя, в формулировке *S1* соответствует так же индексу ФЗ, $j = 1,2, \dots J$, где J – множество Потребителей ОВС; c_{ij} – элемент матрицы цен C, тариф для j-го Потребителя на обработку ФЗ сервисами i-го Провайдера при фиксированном тарифном плане.

Необходимо определить булеву матрицу назначений $X = |x_{ij}|,\ i = 1,2, \dots I, j = 1,2, \dots J$, где переменные x_{ij} определены следующим образом:

$$x_{ij} = \begin{cases} 1, & \text{если } i-\text{му провайдеру назначена } j-\text{я ФЗ.} \\ 0, & \text{иначе.} \end{cases} \quad (1)$$

Запишем целевую функцию $F(X)$ – суммарную стоимость выполнения кейса функциональных задач:

$$F(X) = \sum_{i=1}^{I}\sum_{j=1}^{J} c_{ij}x_{ij} \to min \quad (2)$$

Сформулируем ограничения рассматриваемой задачи. Выполнение условия соответствия каждой ФЗ только одному Провайдеру означает, что каждый столбец матрицы назначений X содержит только одно значение равное единице, а все остальные равны нулю, т.е.

$$\sum_{i=1}^{I} x_{ij} = 1, \quad \forall j = 1,J. \quad (3)$$

Введем параметры бюджетных ограничений: предположим, что z_{ij} – затраты i-го Провайдера на ИТ-сервис j-му Потребителю; a_i – бюджет i-го Провайдера; b_j – бюджет j-го Потребителя. Тогда бюджетное ограничение на стороне Провайдера примет вид:

$$\sum_{j=1}^{J} z_{ij}x_{ij} \le a_i, \forall i = 1,I. \quad (4)$$

Бюджетное ограничение на стороне Потребителя:

$$\sum_{i=1}^{I} c_{ij}x_{ij} = 1, \quad \forall j = 1,J. \quad (5)$$

Целевая функция (2) и ограничения (1), (3) – (5) составляют математическую постановку задачи эффективного распределения функциональных задач обработки данных экологического мониторинга с точки зрения Облачного Потребителя данных.

Аналогичным образом, на основе ограничений (1), (3) – (5) формулируется задача эффективного распределения ФЗ с точки зрения Провайдера сервисов, при этом целевая функция трактуется как суммарный доход Провайдера от выполнения кейса функциональных задач:

$$E(X) = \sum_{i=1}^{I}\sum_{j=1}^{J} c_{ij}x_{ij} \to max. \quad (6)$$

Функциональные задачи Потребителей данных, входящие в кейс задач обработки данных экологического мониторинга зачастую существенно ограничены по времени выполнения, в этом случае формулировка оптимизационной задачи (1-4) может быть уточнена (*формулировка сценария S2*) путем введения ограничения на время выполнение функциональных задач в кейсе:

$$\sum_{i=1}^{I} t_{ij}x_{ij} \le T_j, \quad \forall j = 1,J. \quad (7)$$

где t_{ij} – время решения функциональной задачи j-го Потребителя по тарифному плану i-го Провайдера; T_j – предельно допустимое время решения функциональной задачи j-го Потребителя.

Задачи (1–4), (7) и (1), (3–5) являются NP-полными задачами и могут быть решены несколькими методами. Вот некоторые из них: метод полного перебора; метод ветвей и границ; эвристические алгоритмы (жадный алгоритм поиска, метод поиска ближайшего соседа, биоинспирированный алгоритм муравьиной колонии); генетические алгоритмы (эволюционные вычисления) и др. Только при относительно небольшой размерности оптимизационные задачи (1) – (5) могут быть решены точными методами. При этом необходимо учитывать, что матрицы назначений (1) будут иметь в общем случае разный состав, эффективный с точки зрения Потребителя данных (2) и Провайдера сервисов (6). Графически взаимодействие этих акторов может быть проиллюстрировано графиками кривых спроса (S) и предложения (R) на рисунке 1.

Кривая спроса Потребителя обобщает набор приемлемых для него комбинаций ИТ-сервисов, которые должны быть оказаны Провайдером за допустимое время [t_{min}, t_{max}], кривая предложений – отображает совокупность тарифных планов $\{P_I\}$ у Провайдеров действующих на рынке облачных услуг.

На сегодняшний день известно [1, 9] три базовые модели предоставления облачных сервисов: программное обеспечение как сервис (Software as a Service, SaaS); платформа как сервис (Platform as a Service, PaaS); инфраструктура как сервис (Infrastructure as a Service, IaaS).

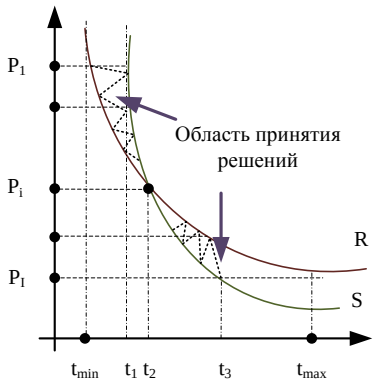


Рисунок 1 – Кривая спроса и предложения по тарифным планам на рынке ОВС

В этих условиях логично предположить, что не все Поставщики облачных услуг обладают технологическими возможностями комплексного предоставления всех типов ИТ-сервисов по обработке данных, а также то, что существует дифференциация требований Пользователей к необходимым наборам ИТ-сервисов.

Уточняя задачу (*формулировка сценария S3*) предположим, что существуют предварительные данные о потенциальной возможности предоставления ИТ-сервиса i-ым Провайдером для j-го Потребителя, фиксируемые матрице достижимости $Y = |y_{ij}|$. Переменные y_{ij} определены следующим образом:

$$y_{ij} = \begin{cases} 1, & \text{если } i-\text{й провайдер имеет сервисы для} \\ & \text{решения } j-\text{й ФЗ}, \\ 0, & \text{иначе.} \end{cases} \quad (8)$$

В этом случае, предполагая, что в вычислительной среде есть Провайдеры, обладающие полным набором технологий, необходимых Потребителю, ограничение (3) примет вид:

$$\sum_{i=1}^{I} x_{ij} y_{ij} = 1, \quad \forall j = 1, J. \quad (9)$$

Поскольку сообщество пользователей-исследователей может обслуживаться одновременно несколькими Провайдерами на основе модели «гибридного облака» задача установления взаимосвязи «Потребитель-Провайдер» усложняется. Модель (1 – 9) может быть расширена для отражения посреднических услуг Провайдера Приложений.

Модель сценария взаимодействия акторов «Поставщик данных –Провайдер приложений – Провайдер сервисов»

В предлагаемой модели (*формулировка S4*) Провайдер приложений должен учитывать динамически меняющиеся ресурсные требования Потребителя. Динамизм ресурсных требований Потребителя обусловлен критичностью ко времени обработки данных экологического мониторинга и уровнем компетенций представителей Потребителя. В этом случае Тарифный план Провайдера Приложений для Потребителя: $c_{ij} + \Delta d_{ij}$, где Δd_{ij} – доход Провайдера приложений. Суммарная стоимость выполнения кейса ФЗ (2) с учетом сервисов Провайдера приложений:

$$F_b(x) = \sum_i \sum_j (c_{ij} + \Delta d_{ij}) x_{ij} \to min, \; F_b > F, \quad (10)$$

при этом $|F_b - F| \le \varepsilon$ зона принятия положительного решения для Потребителя.

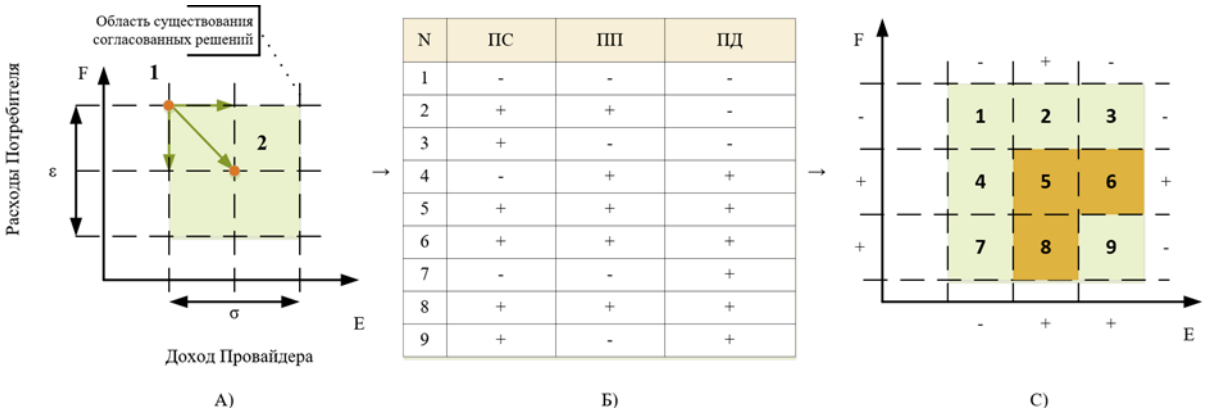
Аналогичным образом определяем тарифный план Провайдера сервисов и Провайдера приложений: $c_{ij} + \Delta k_{ij}$, где Δk_{ij} – доход Провайдера приложений от оказания посреднических услуг на стороне Провайдера. Суммарный доход Провайдера от выполнения кейса ФЗ (6) с учетом сервисов Провайдера приложений:

$$E_b(x) = \sum_{i=1}^{I}\sum_{j=1}^{J}(c_{ij} - \Delta k_{ij})x_{ij} \to max,\ \ E_b < E, \qquad (11)$$

при этом $|E_b - E| \le \sigma$ зона принятия положительного решения для Провайдера.

Поскольку в ОВС используется множество тарифных планов, то областью решения задачи (10,11) будет N-мерный параллелепипед (на рисунке 2 приведен случай для трех уровней тарифных планов модели «ПД-ПС» (А) и трех уровней модели «ПД-ПП-ПС» (Б, С)).

Качество ИТ-сервисов ОВС во многом зависит от того, на сколько эффективно реализовано управление загрузкой имеющихся вычислительных ресурсов и распределением поступающих в облачную систему информационных потоков в условиях возможных деструктивных воздействий на элементы системы и изменения характеристик информационных потоков.



N	ПС	ПП	ПД
1	-	-	-
2	+	+	-
3	+	-	-
4	-	+	+
5	+	+	+
6	+	+	+
7	-	-	+
8	+	+	+
9	+	-	+

Рисунок 2 – Модель взаимодействия акторов ОВС: А) область согласованных решений при взаимодействии «ПС»-«ПД»; Б) матрица принятия решений при взаимодействии «ПД-ПП-ПС»; С) область эффективных решений (5-6-8)

Качество ИТ-сервисов ОВС во многом зависит от того, на сколько эффективно реализовано управление загрузкой имеющихся вычислительных ресурсов и распределением поступающих в облачную систему информационных потоков в условиях возможных деструктивных воздействий на элементы системы и изменения характеристик информационных потоков. Для организации эффективного взаимодействия между Потребителями необходимо учитывать структуру ОВС, в связи с этим далее рассмотрим структурную модификацию задачи (*формулировка S5*).

В распределенной системе мониторинга, несмотря на значительные архитектурные отличия, можно выделить три основных функциональных уровня (рисунок 3): система оперативной обработки данных, поступающих от первичных датчиков (СОО) для решения функциональных задач на стороне клиента (уровень Потребителя); коммуникационная система (КС) передачи и кратковременного хранения данных (уровень приложений); система обработки и долговременного хранения данных (СОД, уровень сервисов).

Переходя к описанию объекта моделирования, сделаем ряд предположений относительно структурной организации ОВС и способов взаимодействия ее элементов. Система оперативной обработки данных состоит из J вычислительных комплексов предварительной обработки информации, генерируемой клиентскими приложениями Потребителей, при выполнении кейса ФЗ $j = 1, J$ типов (каждому Потребителю соответствует свой кейс ФЗ.

Каждый кейс $ФЗ_j$ описывается набором параметров: заданным (предельным) объемом передаваемых данных – v_j; объемом вычислений, необходимых для обработки данных ФЗ – u_j; предельным временем обработки данных в СОД ПС $t_j^{\text{СОД}}$; предельным временем коммуникации с ПП $t_j^{\text{СПД}}$ и интенсивностями затрат на обработку и передачу данных ПП.

Данные от Потребителей передаются для обработки в СОД ПС, предполагаем что число СОД соответствует количеству ПС (I) и любой из кейсов ФЗ может быть назначен на обработку только одной из $i = 1, I$ СХД Провайдера.

Для передачи данных от вычислительных комплексов СОО Потребителей к обрабатывающим узлам СОД Провайдера используется система передачи данных (СПД), состоящая из L коммуникационных комплексов (КК), соединенных каналами связи (КС).

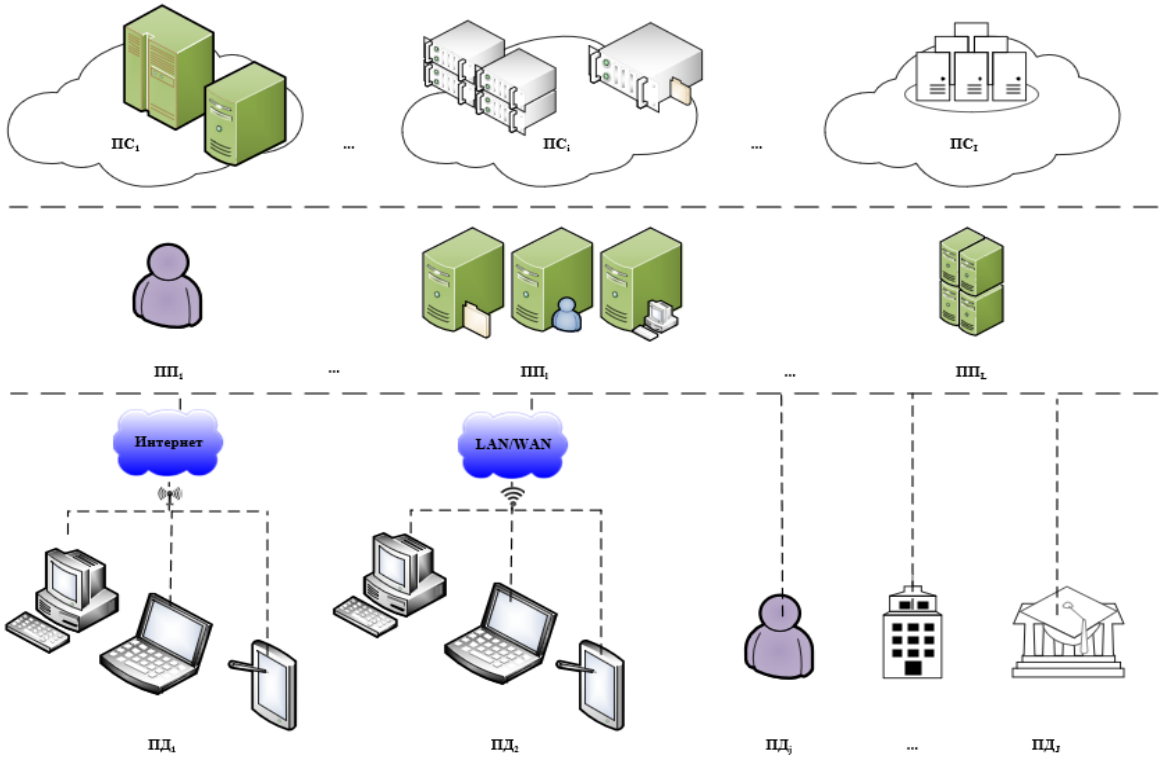


Рисунок 3 – Структурная схема ОВС

Данные от Потребителей передаются для обработки в СОД ПС, предполагаем что число СОД соответствует количеству ПС (I) и любой из кейсов ФЗ может быть назначен на обработку только одной из $i = 1, I$ СХД Провайдера. Для передачи данных от вычислительных комплексов СОО Потребителей к обрабатывающим узлам СОД Провайдера используется система передачи данных (СПД), состоящая из коммуникационных комплексов (КК), соединенных каналами связи (КС).

Коммуникационные комплексы предназначены для буферизации поступающих кейсов ФЗ и передачи для дальнейшей обработки в СОД Провайдера. КС – это каналы взаимодействия «ПД – ПП», «ПП – ПС». Предполагаем, что Пользователи напрямую с Провайдерами не взаимодействуют, это происходит через уровень приложений. На уровне приложений Потребителям предлагается множество тарифных планов $p_l = 1, Pl$, с тарификацией $(c_{ij} + \Delta d_{ij}^{p_l})$ и соответствующей производительностью обслуживания ξ^{p_l}. Для Провайдера это соответствует тарификации $(c_{ij} - \Delta k_{ij}^{p_l})$.

Информационное обеспечение для моделирования ОВС при брокерском взаимодействии описывается кортежем:

$$M = \langle IP1,\ IP2,\ IP3,\ W,\ O,\ F \rangle, \qquad (12)$$

где $IP1$ – входные параметры, связанные с требованиями к уровню качества обработки кейсов ФЗ на стороне ПС; $IP2$ – входные параметры, связанные с ресурсами и тарифными планами ПП; $IP3$ – входные параметры, связанные с затратами на стороне ПД и ПС; W – определяемые булевские переменные, соответствующие назначениям кейсов ФЗ; O – набор системных ограничений; F – целевые функции.

В зависимости от целей управления распределением кейсов ФЗ в распределенных системах экологического мониторинга возможно построение различных математических моделей, и определение на их основе оптимальных режимов межакторного взаимодействия. Одноуровневая стратегия распределения кейсов ФЗ по СОД Провайдеров предполагает одновременное выполнение двух управляющих функций: назначение кейсов ФЗ от клиентских приложений СОО на обработку Провайдерам и выбор тарифных планов приложений с учетом ограничений на время передачи и обработки кейсов ФЗ с минимальными эксплуатационными затратами на обработку (*формулировка S6*).

Входные параметры ($IP1$), связанные с требованиями к уровню качества обработки кейсов ФЗ Провайдера: a_{ij} – интенсивность эксплуатационных затрат на выполнение кейса ФЗ j-го типа у i-го Провайдера; $\xi_i^{p_l}$ – производительность СОД$_i$ (эталонных операций в секунду); u_j – объем вычислений, необходимый

для выполнение кейса ФЗ j-го типа (эталонных операций); v_j – объем данных, передаваемых для обработки в СОД для кейса ФЗ j -го типа; $t_j^{\text{СОД}}$ –предельное время обработки кейса ФЗ j -го типа в СОД Провайдера; $t_j^{\text{СПД}}$ –предельное время передачи кейса ФЗ j -го типа в СПД Брокера; $t_{ij}^{СОД}$ –время, затрачиваемое СОД_i на обработку кейса ФЗ j -го типа, определяется как

$$t_{ij}^{СОД} = \frac{u_j}{\xi_i^{p_l}}. \quad (13)$$

Определяемые булевские переменные (W):

x_{ijl} –признак назначения кейса ФЗ принимает следующие значения:

$$x_{ijl} = \begin{cases} 1, \ \textit{если данные } j-\textit{го ПД обработаны} \\ \quad i-\textit{ым ПС и переданы через } l-\textit{го ПП} \\ 0, \ \text{иначе} \end{cases}$$

ω_{jlp_l} –признак использования тарифного плана p_l-го типа j-ым Потребителем при работе с l-ым Провайдером приложений:

$$\omega_{jlp_l} = \begin{cases} 1, \text{если } j-\text{ый ПД использует } p_l \\ \quad \text{тарифный план } l-\text{го ПП} \\ 0, \text{иначе} \end{cases},$$

v_{ilp_l} – признак использования тарифного плана p_l-го типа i-ым Провайдером сервисов при работе с l-ым Провайдером приложений:

$$v_{jlp_l} = \begin{cases} 1, \text{если } j-\text{ый ПС использует } p_l \\ \quad \text{тарифный план } l-\text{го ПП} \\ 0, \text{иначе} \end{cases},$$

Системные ограничения для одноуровневой модели управления. Данные от j го ПД, обрабатывается только одним ПС при посредничестве единственного ПП:

$$\sum_{i=1}^{I}\sum_{l=1}^{L} x_{ijl} = 1, \quad \forall j = \overline{1,J}. \quad (14)$$

Время обработки данных не должно превышать заданного:

$$\sum_{i=1}^{I}\sum_{l=1}^{L} \frac{u_j}{\xi_i^{p_l}} x_{ijl} \le t_j^{СОД}, \quad \forall j = 1,J. \quad (15)$$

Коммуникационные ограничения для ПП. Между j-ым ПД и l-ым ПП устанавливается соглашение на обслуживание по тарифному плану только одного типа:

$$\sum_{p_l=1}^{P_l} \omega_{jlp_l} = 1, \quad \forall j = \overline{1,J}, \quad \forall l = \overline{1,L}. \quad (16)$$

Коммуникационная связь между j-ым ПД и l-ым ПП формируется в соответствии с назначением j-го кейса ФЗ на обработку в $СОД_i$ Провайдера – x_{ijl}:

$$\sum_{i=1}^{I}\sum_{p_l=1}^{P_l}\sum_{l=1}^{L} \omega_{jlp_l} x_{ijl} = 1, \quad \forall j = \overline{1,J}. \quad (17)$$

Предполагаем, что между l-ым ПП и i-ым ПС реализуются тарифные планы только одного типа:

$$\sum_{p_l=1}^{P_l} v_{ilp_l} = 1, \quad \forall i = \overline{1,I}, \quad \forall l = \overline{1,L}. \quad (18)$$

Ограничения на время передачи данных в ОВС:

$$\sum_{i=1}^{I}\sum_{p_l=1}^{P_l}\sum_{l=1}^{L} \omega_{jlp_l} x_{ijl} \frac{u_j}{\xi_i^{p_l}} +$$

$$+\sum_{i=1}^{I}\sum_{p_l=1}^{P_l}\sum_{l=1}^{L} x_{ijl}\, v_{ilp_l} \frac{u_j}{\xi_i^{p_l}} \le t_j^{СПД}, \forall j = \overline{1,J} \quad (19)$$

Ограничение на время обработки данных экологического мониторинга:

$$\sum_{i=1}^{I}\sum_{j=1}^{J} t_{ij}{}^{СОД} x_{ijl} \le t_j{}^{СОД}. \quad (20)$$

Ограничения на эксплуатационные расходы при передаче данных:

$$\sum_{i=1}^{J}\sum_{p_l=1}^{P_l}\sum_{l=1}^{L} d_{ij}^{p_l} \omega_{jlp_l} x_{ijl} \frac{u_j}{\xi_i^{p_l}} + \sum_{i=1}^{I}\sum_{p_l=1}^{P_l}\sum_{l=1}^{L} k_{ij}^{p_l} x_{ijl}\, v_{ilp_l} \frac{u_j}{\xi_i^{p_l}} \le C_j^{СПД}, \quad \forall j = \overline{1,J} \quad (21)$$

В качестве критерия эффективности распределения информационных ресурсов может быть выбран минимум эксплуатационных расходов на передачу данных в СПД и обработку данных в СОД:

$$F^{СОД} = \sum_{j=1}^{J}\left[\sum_{i=1}^{J}\sum_{p_l=1}^{P_l}\sum_{l=1}^{L} d_{ij}^{p_l}\omega_{jlp_l}x_{ijl}\frac{u_j}{\xi_i^{p_l}} + \sum_{i=1}^{I}\sum_{p_l=1}^{P_l}\sum_{l=1}^{L} k_{ij}^{p_l}x_{ijl}\nu_{ilp_l}\frac{u_j}{\xi_i^{p_l}}\right]\cdot x_{ijl} + \sum_{i=1}^{I}\sum_{j=1}^{J} a_{ij}t_{ij}^{СОД}x_{ijl} \to min. \quad (22)$$

Необходимо определить значения параметров $x_{ijl}, \omega_{jlp_l}, \nu_{jlp_l} \quad \forall i = \overline{1,I}, \quad \forall j = \overline{1,J}, \quad \forall l = \overline{1,L}$ с учетом ограничений (14-21) при которых целевая функция (22) принимает минимальное значение. Данная модель управления распределения кейса ФЗ по обработке данных экологического мониторинга, может быть использована на начальных этапах формирования межакторных коммуникаций, как средство выбора коммуникационных связей, удовлетворяющих заданным критериям качества.

Таким образом, предложено шесть моделей описания сценариев взаимодействия акторов ОВС на основе которых может быть сформирован программный комплекс поддержки принятия решений по эффективному управлению ИТ-сервисами или эксплуатационными затратами при решении задач обработки данных экологического мониторинга.

Выводы

От эффективной организации межакторного взаимодействия в облачных вычислительных средах во многом зависит гарантированность уровня поддержки ИТ-сервисов и своевременность выполнения кейсов функциональных задач обработки данных. В данной работе описаны особенности организации взаимодействия основных акторов облачной среды для решения задач экологического мониторинга. Были предложены шесть сценариев взаимодействия акторов ОВС, базирующихся на применении методов целочисленного линейного программирования. Применение такого подхода позволяет эффективно решать задачи малой размерности (в данном случае крупномасштабность задач обработки данных мониторинга скрыта в кейс функциональных задач Потребителя Данных). Консолидация комплекса таких решений на основе представленных сценариев, в свою очередь, позволит сформировать многомерный куб решений для параметрического анализа зависимости эксплуатационных расходов Потребителя и Провайдера от ресурсных ограничений Провайдера, полноты тарифных планов.

В дальнейшем планируются расширение и доработка представленных сценариев и на их основе формирование специализированной системы поддержки принятия решений по управлению распределением кейсов функциональных задач обработки данных с учетом критериев ценности и старения мониторируемой информации.

Также планируется построение многомерных матриц принятия решений в пространстве «Технология» — «Структура» — «KPI» и создание базы знаний по управлению качеством для провайдеров гетерогенных облачных вычислительных сред, предоставляющих информационные сервисы для анализа больших данных экологического мониторинга.

Работа выполнена при поддержке Российского фонда фундаментальных исследований (грант № 18-47-920005\20).

Литература

1. Mell P., Grance T. The NIST Definition of Cloud Computing. Recommendations of the National Institute of Standards and Technology, 2011. Available at: http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf (accessed 12 November 2015).

2. Hogan, M. NIST Cloud Computing Standards Roadmap / M. Hogan, F. Liu, A. Sokol, J. Tong // Computer Security Division Information Technology Laboratory National Institute of Standards and Technology Gaithersburg, 2011.- 76 p.

3. NIST Special Publication 1500-1r1 NIST Big Data Interoperability Framework: Volume 1, Definitions. NIST Big Data Public Working Group Definitions and Taxonomies Sub-group Version 2, June 2018. - URL: https://doi.org/10.6028/NIST.SP.1500-1r1, last accessed 2019/01/17.

4. NIST Special Publication 1500-6r1. NIST Big Data Interoperability Framework: Volume 6, Reference Architecture NIST Big Data. Public Working Group Reference Architecture Subgroup, Version 2, June 2018. –URL: https://doi.org/10.6028/NIST.SP.1500-6r1, Дата доступа – 10.05.2021.

5. Mashchenko, E. Conceptual big data processing model for the tasks of smart cities environmental monitoring / E. Mashchenko, D. Voronin, V. Shevchenko, O. Chengar // Communications in Computer and Information Science. 2019. Т. 1038 CCIS. С. 212-222. / DOI (CrossRef): 10.1007/978-3-030-37858-5_17. – URL: http://link.springer.com/10.1007/978-3-030-37858-5_17, Дата доступа – 10.05.2021.

6. Мащенко, Е. Н. Сбалансированная система показателей качества ИТ-сервисов анализа больших данных экологического мониторинга / Е. Н. Мащенко, В. И. Шевченко, О. В. Ченгарь, Ю. П. Николаева // Экономика и

управление: Теория и практика, 2020. - №4. - С. 90 – 99.

7. Щукина Н. А. Некоторые подходы к решению задачи о назначениях // Проблемы экономики и менеджмента, 2016. - №5 (57). - С.169–174.

8. Пантелеев, А. В. Методы оптимизации в примерах и задачах / А. В. Пантелеев, Т. А. Летова. 3-е изд. – М.: Высш. школа, 2008. – 544 с.

9. Прудникова, А. А. Анализ облачных сервисов с точки зрения информационной безопасности / А. А .Прудникова, Т. М. Садовникова // T-Comm Телекоммуникации и Транспорт, 2012. - №7. - С. 153–156.

Мащенко Е.Н., Шевченко В.И., Ченгарь О.В. Модель взаимодействия акторов облачной вычислительной среды при решении задач экологического мониторинга. *Предложена математическая модель для сценариев взаимодействия акторов облачной вычислительной среды при решении задач обработки данных экологического мониторинга. Проведена формализация процессов эффективного назначения функциональных задач потребителей данных на ресурсы вычислительной среды с использованием аппарата линейного программирования, а именно в постановке несбалансированной задачи о назначениях с булевскими переменными.*

Ключевые слова: *экологический мониторинг, обработка больших данных, акторы процессов, качество ИТ-сервисов, облачные вычислительные среды, системы поддержки принятия решений.*

Mashchenko E.N., Shevchenko V.I., Chengar O.V. Investigation of cloud audit processes for the analysis of environmental monitoring data. *The mathematical model is proposed for scenarios of interaction between actors of a cloud computing environment when solving problems of processing environmental monitoring data. The formalization of the processes of effective assignment of functional tasks of data consumers to the resources of the computing environment is carried out using the apparatus of linear programming, namely, in the formulation of an unbalanced assignment problem with Boolean variables.*

Keywords: *environmental monitoring, big data processing, process actors, quality of IT services, cloud computing environments, decision support systems.*

Статья поступила в редакцию 25.07.2021
Рекомендована к публикации профессором Мальчевой Р. В.

УДК 004.04

Компиляция математических выражений с помощью Linq.Expression

Д. Ю. Луценко
Санкт-Петербургский политехнический университет Петра Великого (СПбПУ)
E-mail: lutsenkodm@mail.ru

Аннотация
Целью данной статьи является реализация программного решения, позволяющего производить компиляцию математических выражений с помощью Linq.Expression. Задача заключается в следующем: скомпилировать выражение в функцию с произвольным количеством аргументов произвольного типа, как числового, так и логического. Предложен протокол для трансляции выражения с целью обеспечения гарантии произвольности типов, а также ряд перегрузок с протоколом по умолчанию для сокращения длины кода. Рассмотренный функционал может быть использован в приложениях, где существует необходимость во время выполнения получить быстро работающую математическую функцию из строки.

Введение

Для начала рассмотрим примеры прототипов выражений, реализацию которых хотелось бы получить в результате данной работы:

```
fun1 = "a + b".Compile<int, double, double>("a", "b");
fun2 = "u > 0 or (u < v) or t".Compile<int, int, bool, bool>("u", "v", "t");
```

Поскольку работа ведется в рамках существующей библиотеки символьной алгебры, можно немедленно приступить к компиляции, уже имея синтаксический анализатор и дерево выражений. Воспользуемся базовым классом Entity. Его иерархия типов выглядит следующим образом:

```
Entity
|
+--Operators
  |
  +--Sumf
  |
  +--Minusf
  |
  ...
+--Trigonometry
  |
  +--Sinf
  |
  +--Cosf
  |
  ...
+--Discrete
  |
  +--Andf
…
```

Дерево выражений – это граф, в котором дочерние элементы узла являются операндами оператора или функции. Деревья выражений представляют код в виде древовидной структуры, где каждый узел является выражением, например, вызовом метода или двоичной операцией. [1]

Протокол компиляции

Определим класс интерфейса, определяющий каким образом подтипы Entity отображаются в операторах, структурах, или функциях Linq.Expression [3]. Поскольку заранее не известны типы, запрашиваемые пользователем в качестве входных и выходных данных, пользователь должен будет предоставить эту информацию.

```
public sealed record CompileProtocol
{
    public Func<Entity, Expression> ConstantTranslator { get; init; }
    public Func<Expression, Expression, Entity, Expression> BinaryNodeTranslator { get; init; }
    public Func<Expression, Entity, Expression> UnaryNodeTranslator { get; init; }
    public Func<IEnumerable<Expression>, Entity, Expression> AnyArgumentConverter { get; init; }
}
```

Наш компилятор будет вызывать все эти лямбды внутри себя при преобразовании унарных узлов, бинарных узлов, константных узлов в Linq.Expression.

То есть теперь известны правила, по которым будут преобразовываться узлы. Напишем сам компилятор, а точнее алгоритм, который будет строить дерево.

Компилятор

Прототип метода, который должен получиться в итоге, выглядит следующим образом:

```
internal        static        TDelegate
Compile<TDelegate>(
            Entity e,
            Type? typeToReturn,
            CompileProtocol cProtocol,
            IEnumerable<(Type        t,
Variable       v)>       typesWithNames
)  where TDelegate : Delegate
```

Значения:
Entity e - компилируемое выражение;
Type? typeToReturn - тип возвращаемого значения из делегата [8], поэтому необходимо передать его как отдельный аргумент;
CompileProtocol - протокол правил, с помощью которого будет производиться преобразование каждого узла выражения;
IEnumerable <(Type t, Variable v)> typesWithNames – комплект кортежей типов и переменных, передаваемых пользователем результирующему делегату [8].

Реализация метода:

```
internal        static        TDelegate
Compile<TDelegate>(Entity    e,    Type?
typeToReturn,           CompileProtocol
cProtocol,    IEnumerable<(Type    type,
Variable    variable)>    typesWithNames
) where TDelegate : Delegate
{
    var       subexpressionCache       =
typesWithNames
.ToDictionary(t  =>  (Entity)t.variable,
t => Expression.Parameter(t.type));
    var          functionArgs          =
subexpressionCache.Select(t          =>
t.Value).ToArray();
    var       locals       =       new
List<ParameterExpression>();
    var     varAssignments     =     new
List<Expression>();
    var    builtTree    =    FormTree(e,
subexpressionCache,     varAssignments,
locals, cProtocol);
    var         treeAndLocals         =
Expression.Block(locals,
varAssignments.Append(builtTree ));
    Expression      entExpresion      =
typeToReturn    is    not    null    ?
Expression.Convert(treeAndLocals,
typeToReturn) : treeAndLocals;
    var          resultLambda          =
Expression.Lambda<TDelegate>(entExpresi
on, functionArguments);
    return resultLambda.Compile();
}
```

Основная цель данного метода - создание хранилищ кеша для поддеревьев, локальных переменных или иных сущностей [2]. Более подробно рассмотрим функцию FormTree. Она строит дерево выражений Linq [3] из Entity. Её прототип выглядит следующим образом:

```
internal static Expression FormTree(
    Entity e,
    Dictionary<Entity,
ParameterExpression>
subexpressionsInCache,
    List<Expression> varAssignments,
    List<ParameterExpression>
newLocalVars,
      CompileProtocol cProtocol)
```

Подробнее об ее аргументах:
Entity e - подвыражение или выражение, из которого нужно построить дерево;
Dictionary <Entity, ParameterExpression> subexpressionsInCache - набор кэшированных поддеревьев (записаных в локальные переменные, которые уже существуют);
List <Expression> varAssignments - список соответствий уникальных поддеревьев локальным переменным;
List <ParameterExpression> newLocalVars - локальные переменные, создаваемые с помощью FormTree. (для хранения результатов поддеревьев);
CompileProtocol cProtocol - правила, по которым узел Entity преобразуется в узел Linq.Expression. Он остается неизменным и передается всем вызовам FormTree.

Далее рассмотрим функцию - FormTree:

```
internal        static        Expression
FormTree(Entity e, ...)
{
    if
(subexpressionsInCache.TryGetValue(e,
out var ready))
        return ready;
    Expression workTree = e switch
    {
        Entity.Number  or  Boolean  =>
cProtocol.ConstantTranslator(e),
IUnaryNode  argOne                  =>
cProtocol.UnaryNodeTranslator(FormTree(
argOne.NodeChild, ...), e),
        IBinaryNode              argTwo
=> cProtocol.BinaryNodeTranslator(

FormTree(argTwo.NodeFirstChild, ...),

FormTree(argTwo.NodeSecondChild,  ...),
e),
        var          further          =>
cProtocol.AnyArgumentConverter(

further.DirectChildren.Select(c      =>
FormTree(c, ...)),
            e)
    };
var            newVariable            =
Expression.Variable(workTree.Type);
varAssignments.Add(Expression.Assign(ne
wVariable, workTree));
```

```
    subexpressionsInCache[e]          =
newVariable;
    newLocalVars.Add(newVariable);
    return newVariable;
}
```

Для каждого поддерева либо немедленно возвращается локальная переменная, соответствующая этому выражению, либо строится новое дерево Linq.Expression [3] и сохраняется в новой локальной переменной, которая будет возвращена.

На этом реализация компилятора завершена, но правила для преобразования выражений в Linq.Expression [4] не были реализованы, потому ожидается, что эти правила будут предоставлены пользователем. Но почему бы не реализовать некоторые правила по умолчанию для встроенных типов?

Протокол по умолчанию

Метод Compile <TDelegate> (Entity, Type ?, CompileProtocol, IEnumerable <(Type, Variable)>) будет предоставлен пользователю, но очевидно, что это длинная и не очень удобная для пользователя конструкция, так как придется написать огромное количество кода, описывающего преобразование каждого узла и константы.

Таким образом, существует возможность предоставить протокол компиляции по умолчанию, который будет работать с некоторыми встроенными примитивами (bool, int, long, float, double, Complex, BigInteger).

ConstantTranslator

Данное правило преобразует константу из Entity в Linq.Constant [5] и выглядит так:

```
public         static         Expression
ConverterConstant(Entity  e)       => e
switch      {      Number      n      =>
Expression.Constant(DownCast(n)),
Entity.Boolean            b           =>
Expression.Constant((bool)b),          _
=>               throw               new
AngouriBugException("Undefined constant
type")    };
```

Entity.Number приводится к числу в зависимости от его типа, логические константы, конечно, преобразуются в bool.

UnaryNodeTranslator

Правило этого протокола - это делегат [8], который преобразует узел с одним аргументом в Linq.Expression.

```
public         static         Expression
OneArgumentEntity (Expression e, Entity
typeHanler => typeHanler switch
    {
     Sinf                              =>
Expression.Call(GetDef("Sin",         1,
e.Type), e),
     Cosecantf                         =>
Expression.Call(GetDef("Csc",
1,e.Type),e),
     Arcsinf                           =>
Expression.Call(GetDef("Asin",
1,e.Type),e),
     Arccosecantf                      =>
Expression.Call(GetDef("Acsc",1,e.Type)
, e),
     Absf                              =>
Expression.Call(GetDef("Abs",         1,
e.Type), e),
     Signumf                           =>
Expression.Call(GetDef("Sgn",         1,
e.Type), e),
     Notf => Expression.Not(e),    _   =>
throw     new     AngouriBugException("A
node seems to be not added")
    };
```

BinaryNodeTranslator

Следующее правило преобразует узел с двумя аргументами в Linq.Expression

```
public static Expression
TwoArgumentEntity(Expression l,
Expression r, Entity typeHanler)
{
    var typeForCast = MaxType(l.Type,
r.Type);
    if (l.Type != typeForCast)
        l = Expression.Convert(l,
typeForCast);
    if (r.Type != typeForCast)
        r = Expression.Convert(r,
typeForCast);
    return typeHanler switch
    {
        Dividef => Expression.Divide(l,
r),
       ...
        Orf => Expression.Or(l, r),
        ...
        Greaterf =>
Expression.GreaterThan(l, r),
        ...
        _ => throw new
AngouriBugException("Node does not
exist")
    };
}
```

Здесь возникает вопрос о приведении типов [7]. Поскольку в качестве входных параметров могут быть два выражения разных типов, то следует найти наиболее примитивный тип, к которому приводятся оба операнда. Для этого каждому типу присвоим ранг: Complex - 10; double – 9; float – 8; long -8; BigInteger – 8; int – 7.

Если типы совпадают, MaxType вернет один из них.

Если уровень типа операнда A выше, чем уровень типа операнда B, то B приводится к A., например, MaxType (long, double) -> double

Если уровни равны, а типы нет, то будет найден ближайший общий тип, то есть любой такой тип, уровень которого выше на 1. Например, MaxType (long, float) -> double.

Операнды при необходимости приводятся к выбранному типу, после чего находится нужная перегрузка или оператор. Например, для Sumf - Expression.Add [10], а Andf превратится в Expression.And [11].

Все необходимые правила для протокола определены. Теперь во время создания существует возможность передать эти правила в требуемые свойства протокола.

Программный интерфейс приложения

Далее представлена низкоуровневая версия того, как может выглядеть окончательный API [9]:

```
public                          TDelegate
Compile<TDelegate>(CompileProtocol
cProtocol,      Type      typeToReturn,
IEnumerable<(Type     type,     Variable
variable)> typesWithNames             )
where TDelegate : Delegate
```

Такая форма неудобна и требует много работы, прежде чем кто-то сможет вызвать данный метод, но существует возможность улучшить данный подход: передать протокол по умолчанию и перегрузить данный метод для делегатов [8] с одним аргументом, двумя, тремя и так далее. Поскольку правила уже назначены свойствам протокола по умолчанию, при передаче протокола создается его экземпляр. Второй вариант немного сложнее - генерирование кода с помощью текстового шаблона T4 Text Template [6].

Пример сгенерированного кода:

```
public Func<TIn1, TIn2, TIn3, TOut>
Compile<TIn1,        TIn2,        TIn3,
TOut>(Variable  var1,  Variable  var2,
Variable                          var3)
=>  IntoLinqCompiler.Compile<Func<TIn1,
TIn2, TIn3, TOut>>(this, typeof(TOut),
new(),                          new[] {
(typeof(TIn1),   var1),   (typeof(TIn2),
var2) , (typeof(TIn3), var3)  });
```

Производительность

Соответствие тестов и их номеров в таблице 1:

1 - Простое лямбда-выражение. Компиляция производилась с помощью стандартными средствами.

2 - Простое лямбда-выражение. Компиляция производилась с помощью предложенного в данной статье решения.

3 - Сложное лямбда-выражение. Компиляция производилась с помощью стандартными средствами

4 - Сложное лямбда-выражение. Компиляция производилась с помощью предложенного в данной статье решения.

Таблица 1 - Результаты сравнения производительности

Тест	Среднее время	Погрешность
1	191.1 ns	3.81 ns
2	187.6 ns	3.93 ns
3	1385.0 ns	26.55 ns
4	295.6 ns	5.76 ns

Из результатов тестов на производительность видно, что при компиляции простых лямбда-выражений выигрыш в производительности минимальный, но при компиляции более сложных выражений затраченное на выполнение время в разы меньше чем у стандартного решения.

Примеры кода

```
      var             fun             =
"cos(x)".Compile<double, double>("x");
Console.WriteLine(fun(Math.PI / 2));
>>> 0
var fun1 = "x > y".Compile<double, int,
bool>("x", "y");
Console.WriteLine(fun1(10.1d, 5));
Console.WriteLine(fun1(1d, 1));
>>> True
>>> False
var cpr = new CompileProtocol()
{
    ConstantTranslator    =    ent    =>
Expression.Constant(ent.ToString()),
    BinaryNodeTranslator = (a, b, t) =>
t switch
    {
        Sumf                            =>
Expression.Call(typeof(string)
            .GetMethod("Concat",  new[]
{ typeof(string), typeof(string) }) ??
throw new Exception(), a, b),
        _ => throw new Exception()
    }
};
var fun2 = "a + b + c + 1234"
    .Compile<Func<string,       string,
string, string>>(
        cpr, typeof(string),
        new[] {
            (typeof(string), Var("a")),
            (typeof(string), Var("b")),
            (typeof(string),  Var("c"))
}
        );
Console.WriteLine(fun2("aa",      "bb",
"cc"));
>>> aabbcc1234
```

Заключение

В данной статье успешно был реализован процесс компиляции математических выражений с помощью Linq.Expression. Для гарантии произвольности типов был придуман протокол для трансляции выражения. Чтобы не заставлять пользователя писать один и тот же код несколько раз, предлагается ряд перегрузок с протоколом по умолчанию, который отлично работает с встроенными типами. Данный протокол будет автоматически преобразовывать типы в бинарных операторах и функциях к ближайшему универсальному типу при необходимости.

Рассмотренный функционал можно использовать в тех местах, где существует необходимость получить быстро работающую математическую функцию из строки во время выполнения.

Литература

1. Адам Фримен, Джозеф С. Раттц-мл. LINQ: язык интегрированных запросов в C# для профессионалов. 2010. — М.: «Вильямс», 2011. — 656 с.

2. Альфред В. Ахо, Моника С. Лам, Рави Сети, Джеффри Д. Ульман. Компиляторы: принципы, технологии и инструментарий. — 2-е изд. — М.: Вильямс, 2008.

3. https://docs.microsoft.com/ru-ru/dotnet/csharp/programming-guide/concepts/expression-trees

4. https://docs.microsoft.com/ru-ru/dotnet/csharp/programming-guide/concepts/linq/converting-data-types

5. https://docs.microsoft.com/ru-ru/dotnet/api/system.linq.expressions.expression.constant

6. https://docs.microsoft.com/ru-ru/visualstudio/modeling/code-generation-and-t4-text-templates

7. Mössenböck, Hanspeter (25 March 2002). Advanced C#: Checked Type Casts. Institut für Systemsoftware, Johannes Kepler Universität Linz, Fachbereich Informatik.

8. https://docs.microsoft.com/ru-ru/dotnet/csharp/programming-guide/delegates

9. Меджуи М., Митра Р. Непрерывное развитие API. Правильные решения в изменчивом технологическом ландшафте. 2020. — П.: «Питер», 2011. — 272 с.

10. https://docs.microsoft.com/ru-ru/dotnet/api/system.linq.expressions.expression.add

11. https://docs.microsoft.com/ru-ru/dotnet/api/system.linq.expressions.expression.and

Луценко Д.Ю. Компиляция математических выражений с помощью Linq.Expression. *Целью данной статьи является реализация программного решения, позволяющего производить компиляцию математических выражений с помощью Linq.Expression. Задача заключается в следующем: скомпилировать выражение в функцию с произвольным количеством аргументов произвольного типа, как числового, так и логического. Предложен протокол для трансляции выражения с целью обеспечения гарантии произвольности типов, а также ряд перегрузок с протоколом по умолчанию для сокращения длины кода. Рассмотренный функционал может быть использован в приложениях, где существует необходимость во время выполнения получить быстро работающую математическую функцию из строки.*

Ключевые слова: *компиляция, Linq.Expression, выражение, математическая функция, произвольный, числовой и логический.*

Lutsenko D.Y. Compiling mathematical expressions with Linq.Expression. *The purpose of this article is to implement a solution that allows you to perform the function of compiling mathematical expressions using Linq.Expression. The task is as follows: compile the expression into a function with arbitrary arguments, arbitrary, not only numeric, but also logical. A protocol for translating an expression in order to guarantee the arbitrariness of types is proposed, as well as a number of overloads with the default protocol to reduce the length of the code. The considered functionality can be used in applications where there is a need to get a fast-running mathematical function from a string at runtime.*

Keywords: *compilation, Linq.Expression, expression, math function, arbitrary, numeric and logical.*

Статья поступила в редакцию 20.05.2021
Рекомендована к публикации профессором Скобцовым Ю. А.

УДК 004.424 + 004.522

Исследование способов синхронизации текстовой и аудио информации для мобильных приложений

В. А. Мишустин, С. В. Иваница
ГОУ ВПО «Донецкий национальный технический университет» (г. Донецк)
E-mail: mishustin.post@yandex.ru

Аннотация
Рассмотрены особенности мобильных приложений для чтения электронных книг. Отмечаются схожесть и различие таких программ, определен факт отсутствия программного обеспечения на рынке мобильных приложений, способных предоставлять пользователю синхронное прочтение/прослушивание электронной книги. Предложены способы синхронизации текстовой и аудио информации для мобильных приложений. Проведено исследование процентной синхронизации, заключающейся в нахождении процентного отношения текущей позиции к тестируемым в текстовом и аудиофайлах.

Введение

В настоящее время более 60% информации, представленной в Вебе, является текстовой. Еще до глобального перехода к информационным технологиям (до 60-х гг. XX в.), книгоиздание являлось единственным источником текстовой информации. В ходе развития компьютерных технологий появились новые электронные форматы, которые стали дополнять печатные издания книг. Так, печатное издание можно дополнить электронной книгой, или аудиокнигой. Благодаря новым форматам, информацию можно воспринимать не только посредством зрения, но еще и посредством слуха.

По оценкам экспертов, ежегодно количество пользователей электронных книг увеличивается на 80% [1]. Наряду с этим большую популярность приобрели аудиокниги. Только за 2018 год, количество проданных экземпляров аудиокниг увеличилось на 27,3% [2].

К преимуществам электронных книг можно отнести их меньшую стоимость относительно печатных изданий, ведь они стоят на порядок дешевле. Современные карманные персональные компьютеры имеют объем памяти, позволяющий хранить целые библиотеки как электронных, так и аудиокниг. Подобные библиотеки всегда находятся на устройстве пользователя. Возможности современных интернет технологий, позволяют пользователям пополнять библиотеку в любой период времени, в любой точке мира [3].

Большое количество компаний разрабатывают приложения для мобильных устройств, способных работать с различными форматами электронных книг. Приложения для смартфонов способны воспроизводить один из самых популярных форматов аудиокниг – mp3.

Целью работы является анализ мобильных приложений, позволяющих использовать различные форматы электронных книг, и разработка способов синхронизации текстовой и аудио информации для таких приложений.

Обзор программного обеспечения для работы с электронными книгами

Существует большое количество готовых решений, предоставляющих возможность просмотра содержимого электронных книг. Некоторые из приложений позволяют также создавать заметки, оставлять закладки, сохранять цитаты и другое.

Самыми популярными приложениями для чтения электронных книг на мобильных приложениях являются программы - Moon+Reader, Google Play Книги, eBook, DjVuViwer, Adobe Acrobat Reader, ReadEra.

Возможности мобильного приложения Moon+Reader позволяют просматривать содержимое самых популярных форматов - ePub, FB2 и PDF. Также, позволяет читать электронные книги, заархивированные в форматы RAR и ZIP. Имеются разные способы отображения каталога книг – списком, таблицей, или «книжная полка». Также есть возможность менять тему оформления и цветовую палитру интерфейса. На главное меню можно вывести статистику: количеств книг в списке читаемых, количество прочитанных книг, количество часов проведенного за чтением, количество пролистанных страниц. Присутствует функция 3D перелистывания страниц. Параметры отображения текста все настраиваемые: большое количество шрифтов, цвет фона и текста, ширина полей, расстояние между абзацами, строками и буквами. Пометки и выделенные фрагменты содержаться в отдельном разделе.

Преимуществами данного приложения является – дизайн, гибкие настройки, работа с

пометками, статистика, имитация перелистывания страниц.

Google Play Книги – предустановленное приложение операционной системы под управлением Android. Поддерживает два формата ePub и PDF. Преимуществом является возможность синхронизации посредством служб компании Google. Список книг делиться на три раздела: не начато, начато и прочитано. Существует дополнительный функционал - встроенный магазин электронных книг. Приложение простое и удобное, в силу ограниченных настроек: можно выбрать несколько шрифтов, указать размер шрифта и интервал между строками. Также имеются функции смены тем, ночной режим, создание пометок и сносок.

Приложение Google Play Книги является простым приложением и в то же время ограниченным в функциональности.

DjVuViwer, DjVu. Существует несколько групп форматов электронных книг – графические и растровые форматы [4]. DjVuViwer – приложение для чтения узкоспециализированного растрового графического формата DjVu. Этот формат отлично подходит для хранения рукописей, текстов с рукописными заметками и т. д. Преимущество формата DjVu над PDF обусловлено более эффективным методом сжатия информации [5]. Список книг можно сортировать по жанрам, дате или алфавиту. Также имеется возможность создания заметок.

Популярность использования данного приложения на современных мобильных телефонах обусловлена популярностью использования формата DjVu.

Ebook Reader. Приложение способное отображать текстовую информацию следующих форматов: FB2, EPUB, MOBI, PDF, CBR, CBZ и TXT. Недостатком является отсутствие возможностей редактирования текстового файла, создания аннотаций к файлу, и копирования текста.

ReadEra читайте файлы в форматах FB2, EPUB, PDF, DOC, TXT, DJVU, из архивов с расширением .rar и .zip. Создание автоматических заголовков к электронным книгам является преимуществом. Настройки чтения позволяют изменять параметры шрифтов, цвета фона и текста, ширину полей, расстояние межсимвольного и межстрочного интервала. На главном меню приложения есть возможность сортировки по времени создания и времени чтения. Каталог имеет разделы: хочу прочитать, прочитанные, избранное. В общем, приложение имеет большое количество функционала поэтому имеет большую популярность, более 700 тысяч официальных скачиваний.

Компания Adobe разработала программу для мобильных приложений – Adobe Acrobat Reader для PDF. Формат PDF является универсальным межплатформенным форматом, разработанным компанией Adobe [6]. Поэтому программа является надежным редактор документов и конвертером PDF. Приложение позволяет просмотр файла формата PDF, добавление комментариев, редактирование файла и реализует возможности общего доступа к хранилищам таких как Microsoft OneDrive, Dropbox и Google Диск.

Рассмотренные выше приложения стараются предоставить рынку пользователей удобный для взаимодействия дизайн (UX-дизайн [7]) и большое количество функционала.

Отличительной чертой большинства подобных приложений – чтение множества форматов электронных книг, от самых популярных до узкоспециализированных форматов.

Отдельная группа приложений предоставляют анимацию взаимодействия со страницами электронных книг, к примеру 3D перелистывание страниц.

Различие между мобильными приложениями составляет их различие в дополнительном функционале, помимо чтения.

Можно заметить, что способы синхронизации текстовой и аудио информации предоставят пользователям в подобных приложениях дополнительную возможность – переключение между способами восприятия: чтение текстовой информации и прослушивание звуковой информации.

Предлагаемые способы синхронизации

Версии аудио и электронной книги могут считаться одинаковыми, когда озвученный текст в аудио файле полностью и однозначно соответствует тексту электронной книги, тогда получаемая человеком информация, воспринимается одинаково [8]. В этом случае, можно считать, что разница заключается только в способе получении информации: посредством слуха или посредством зрения. Синхронизировать можно только одинаковые версии книг.

Позиция синхронизации в текстовом файле является определенное слово, на котором закончилось чтение. Аналогичная позиция в аудио, это секунда, после которой закончилось воспроизведение аудио. Позиции можно считать синхронизированными, когда информация, следующая после данных позиций, воспринимается человеком одинаково.

Позиция в текстовой информации может измеряться количеством слов от начала текстового файла электронной книги. Позицией в аудио является количество пройденных секунд от начала аудиофайла.

Таким образом, имеют место следующие способы синхронизации:

1. *Способ процентной синхронизации*, который заключается в нахождении процентного отношения текущей позиции ко всему текстовому и аудиофайлу.

2. *Способ поиска уникальных наборов*, который заключается в поиске искомой позиции в электронной книге путем многократной конвертации аудио потока (аудиофайл электронной книги) в текстовый формат.

3. *Способ семплирования*. Анализ аудио потока с целью определения (выделения) элементов текста (глава, абзац, предложение и пр.) с последующей синхронизацией с текстовым файлом одной и той же электронной книги.

Синхронизация текстовой и аудио информации с применением способа процентной синхронизации

При проведении исследования был взят отрывок книги «Дорога в будущее» Билл Гейтса [9] – фрагмент «Послесловие». Данный фрагмент книги, имеющий 450 слов, был озвучен мужским и женским голосом. Длительность полученных аудио файлов составило 3 минуты 45 секунд и 3 минуты 52 секунд соответственно для мужского и женского голоса.

Для анализа предложенного способа было использовано программное обеспечение, использующее компьютерные голоса, поддерживающие SAPI 4, SAPI 5 или Microsoft Speech Platform [10]. Подобранное программное обеспечение дало возможность получить третий файл «озвучки», имеющий длительность 4 минуты 24 секунды.

Выбранный отрывок текста состоит из 6 абзацев. В процессе исследования синхронизировались начала каждого абзаца, начиная со второго. Таким образом, было получено пять позиций для синхронизации.

В ходе проведении исследования было выполнено следующее:

1. Определен объем информации каждого абзаца в словах.

2. Рассчитано отношение объема информации в абзаце по отношению к выбранному отрывку текста. Таким образом стало известно сколько места занимает каждый абзац в данном тексте.

3. По исходной длительности аудиофайлов рассчитана длительность озвучки каждого из абзацев (табл. 1).

Результаты исследований

Далее на рисунках отображены – звуковая дорожка, маркер, указывающий на рассчитанное место синхронизации, линия, показывающая реальное место синхронизации. Маркер и линия сопровождаются временными отметками времени. Исследование проведено в программе Adope Premiere [10].

Синхронизация начала второго абзаца изображена на рисунке 1, где соответственно показывается синхронизация аудио мужской и женской озвучки, а также озвучивание программой.

Таблица 1. Данные синхронизации текстовой и аудио информации

№ абзаца	Кол-во слов	Отношение объема информации в абзаце ко всему тексту, %	Рассчитанные данные		
			Длительность озвучки отрывка мужским голосом, сек	Длительность озвучки отрывка женским голосом, сек	Длительность озвучки отрывка программой, сек
1	79	17,5	39,375	40,60	46,2
2	155	34,4	77,4	79,808	90,816
3	94	20,8	46,8	48,256	54,912
4	64	14,2	31,95	32,944	37,488
5	51	11,3	25,425	26,216	29,832
6	7	1,5	3,375	3,48	3,96

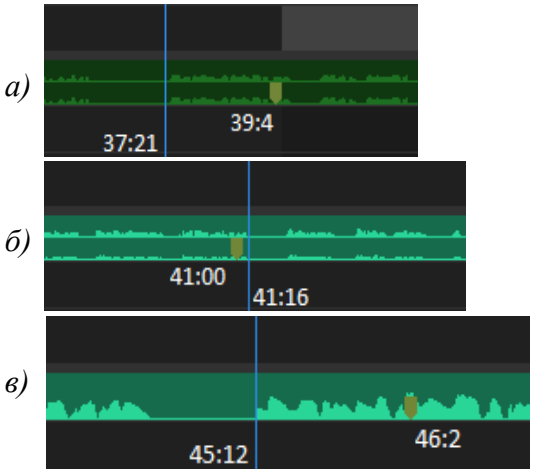


Рисунок 1 – Синхронизация второго абзаца: а) озвучка мужским голосом, б) озвучка женским голосом, в) озвучка программой

Рассчитаем погрешность синхронизации. Для озвучки мужским голосом она составляет – 2,19 секунд, в процентах – 0,97%. Для озвучки женским голосом погрешность равна 1,16 (0,5%). И озвучивание программой имеет погрешность 11,08 секунды (4,2%).

Синхронизация начала третьего абзаца представлена на рисунке 2. Погрешность синхронизации третьего абзаца для озвучки мужским голосом – 1,10 секунд, или (0,48%); озвучка женским голосом 0,11 секунд, или (0,05%); озвучка программой – 2,2 секунды (0,83%).

Результат синхронизации четвертого абзаца

отображен на рисунке 3. Погрешности синхронизации составляют: для озвучки мужским голосом – 0,8 секунд, 0,36%; для озвучки женским голосом – 0,03 секунды (0,013%); озвучка программой – 0.94 секунды (0.36%).

Синхронизация пятого абзаца отображена на рисунке 4. Этот случай имеет наилучший результат синхронизации, мужская озвучка и озвучка программой имеет погрешность в 0 секунд, а женская озвучка – 1,12 секунд (0,48%).

Синхронизации шестого абзаца имеет также не плохие результаты, данные отображены на рисунке 5.

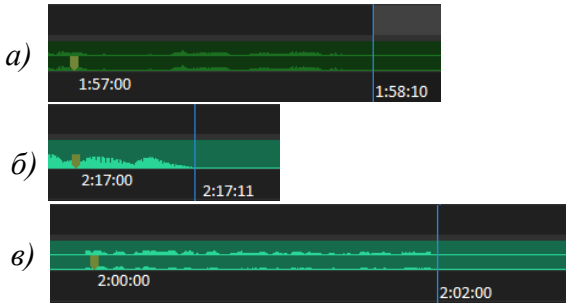


Рисунок 2 – Синхронизация второго абзаца: а) озвучка мужским голосом, б) озвучка женским голосом, в) озвучка программой

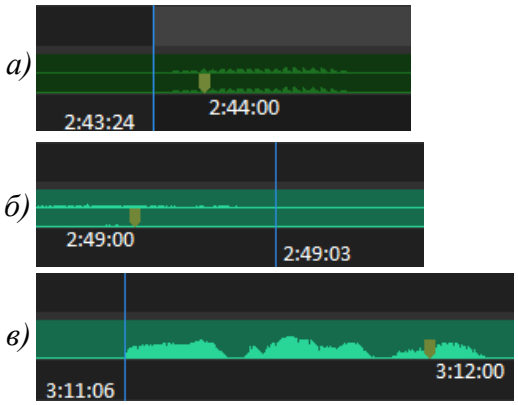


Рисунок 3 – Синхронизация третьего абзаца (а – озвучивание мужским голосом; б – озвучивание женским голосом; в – озвучивание программой)

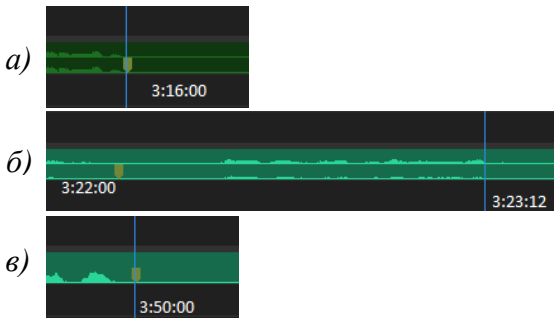


Рисунок 4 – Синхронизация третьего абзаца (*а* – озвучивание мужским голосом; *б* – озвучивание женским голосом; *в* – озвучивание программой)

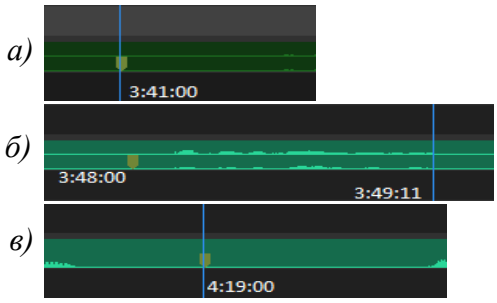


Рисунок 5 – Синхронизация третьего абзаца (а – озвучивание мужским голосом; б – озвучивание женским голосом; в – озвучивание программой)

В этом случае мужская и программная озвучка имеют погрешность 0%, а женская – 1,11 секунд (также 0,48%).

Выводы

В ходе работы были проанализированы мобильные приложения способные работать с разными форматами электронных книг.

Выявлен дополнительный функционал, способный повысить популярность, для подобных приложений.

Предложены три способа синхронизации текстовой и аудио информации. Рассмотрен один из способов синхронизации – *способ процентной синхронизации.*

В результате (по анализу полученных данных) способ процентной синхронизации является достаточно точным, в среднем погрешность имеет показатель меньше одного процента. При уменьшении расчетной позиции для аудио файла на 18 секунд, данный способ можно использовать в пользовательских приложениях для чтения и прослушивания электронных книг с возможностью переключения режимов (чтение, прослушивание) в произвольном месте аудио и текстового файлов.

Литература

1. Евгений Милица: Новая эра развития электронных книг началась. [Электронный ресурс]. – Режим доступа: https://www.cnews.ru/articles/ evgenij_militsa_novaya_era_razvitiya.

2. Электронные книги по продажам обгоняют книги в мягких обложках. 2011-2013 [Электронный ресурс]. – Режим доступа: http://www.segodnya.ua/world/elektronnyeknihi-po-prodazham-obhonjajut-knihi-vmjahkikh-oblozhkakh.html.

3. Самбулов, Д. В. Сравнительный анализ форматов файлов электронных книг [Электронный ресурс]. – Режим доступа: https://cyberleninka.ru/article/n/sravnitelnyy-analiz-formatov-faylov-elektronnyh-knig.

4. Куликова, Е. В. Преобразование форматов графических файлов // Вестник

Сибирского института бизнеса и информационных технологий, 2014. - № 4 (12). - URL:https://cyberleninka.ru/article/n/preobrazovanie-formatov-graficheskih-faylov.

5. Богачева, Е. О. Особенности сжатия формата DjVu для экономии сетевого трафика и исследование методов сегментации изображений / Е. О. Богачева, Д. В. Дмитриев, О. Н. Корелин // Труды Нижегородского государственного технического университета им. Р.Е. Алексеева. - URL: https://cyberleninka.ru/article/n/osobennosti-szhatiya-formata-djvu-dlya-ekonomii-setevogo-trafika-i-issledovanie-metodov-segmentatsii-izobrazheniy.

6. Веретихина, С.В. Оцифровка архивных документов в формате PDF/A [Электронный ресурс] // Инновации в науке, 2016. - № 2(51). – Режим доступа: https://cyberleninka.ru/ article/ n/otsifrovka-arhivnyh-dokumentov-v-formate-pdf-a

7. Яхонтова, И. М. UX-дизайн как инструмент реинжиниринга бизнес-процессов [Электронный ресурс] / И. М. Яхонтова, Л. В. Сорокина // Научный журнал КубГАУ, 2003. - Том 11. – Вып. 4. – Режим доступа: https://cyberleninka.ru/article/n/ux-dizayn-kak-instrument-reinzhiniringa-biznes-protsesso.

9. Демиш, В. О. Синхронизация данных на мобильных платформах [Электронный ресурс] / В. О. Демиш, Б. Н. Пищик // Вестник Новосибирский государственный университет, 2003. - Том 11. – Вып. 4. – Режим доступа: https://cyberleninka.ru/article/n/ sinhronizatsiya-dannyh-na-mobilnyh-platformah.

9. Бил Гейтс: Дорога в будущее. [Электронный ресурс]. – Режим доступа: //www.rulit.me/books/ doroga-v-budushchee-read-301952-1.html.

10. Балаболка [программное обеспечение]. – Режим доступа: http://www.cross-plus-a.ru/balabolka.html.

Мишустин В. А., Иваница С. В. Исследование способов синхронизации текстовой и аудио информации для мобильных приложений. *Рассмотрены особенности мобильных приложений для чтения электронных книг. Отмечаются схожесть и различие таких программ, определен факт отсутствия программного обеспечения на рынке мобильных приложений, способных предоставлять пользователю синхронное прочтение/прослушивание электронной книги. Предложены способы синхронизации текстовой и аудио информации для мобильных приложений. Проведено исследование процентной синхронизации, заключающейся в нахождении процентного отношения текущей позиции к тестируемым в текстовом и аудиофайлах.*

Ключевые слова: *электронная книга, мобильное приложение, аудио поток, спектральный анализ звука, текстовая и аудиоинформация.*

Mishustin V., Ivanitsa S. Research of way to synchronize text and audio information for mobile applications. *The article shown the benefits of audio and electronic books(e-books). It compares the e-book apps to mobile platforms. It is noted similarities and differences between e-book apps. That article suggests possible methods of synchronization text and audio information. A study was made interest the percentage synchronization - search for the percentage of the current position in text and audio files.*

Keywords: *electronic book, mobile app, audio stream, spectral analysis of sound, text and audio information.*

Статья поступила в редакцию 01.11.2021
Рекомендована к публикации профессором Мальчевой Р.В.

УДК 004.771

Программное обеспечение для удалённого управления персональным компьютером

В.А. Мамутова, А.В. Чернышова
Донецкий национальный технический университет
E-mail: vlada.mamutova@yandex.ru, chernyshova.alla@rambler.ru

Мамутова В.А., Чернышова А.В. Программное обеспечение для удалённого управления персональным компьютером. *В статье представлен обзор и сравнительный анализ популярных приложений для удалённого управления компьютером с помощью мобильного устройства. Описан принцип работы подобной программной системы. Разработан собственный протокол обмена информацией между устройствами. Выполнено проектирование клиент-серверного программного обеспечения для дистанционного управления компьютером и разработаны модули, позволяющие на расстоянии выполнять действия компьютерной мыши и отправлять символы на экран.*

Введение

С каждым годом на смартфоны возлагается всё больше задач. Теперь они умеют говорить с нами, думать за нас и помогать нам. Управлять бытовой техникой с мобильного гаджета – уже давно не роскошь, а доступный для каждого способ сделать жизнь проще и комфортнее за счёт своего умного помощника.

Удалённое управление компьютером через смартфон существенно облегчает работу человека, который использует компьютер в качестве основного инструмента. Спрос на приложения для дистанционного управления растёт, и разработчики создают всё новые программы, чтобы удовлетворить возрастающие потребности пользователей.

Актуальность темы исследования обусловлена повышенным спросом на приложения для удалённого управления бытовой техникой и компьютерами, а также практической потребностью реализации данных приложений.

Научная новизна исследования заключается в создании авторского протокола взаимодействия клиента и сервера в программной системе для удаленного управления компьютером с мобильного устройства под ОС Android.

Практическая ценность исследования заключается в том, что на основе разработанной архитектуры было создано приложение, позволяющее управлять мышью и клавиатурой компьютера, и полученные результаты могут стать отправной точкой для реализации новых, ещё более функциональных версий подобных программ.

Анализ существующих программных продуктов

На данный момент существует несколько десятков приложений для удалённого управления компьютером. Но очень немногие из них хорошо зарекомендовали себя. Наиболее популярными приложениями для мобильных устройств под Android, имеющими высокие баллы в Play Market, являются следующие: KDE Connect [1] (4,6 балла), Unified Remote [2] (4,5 балла), Remote Mouse [3] (4,3 балла), ASUS Remote Link (4,2 балла).

Все программы были тщательно протестированы. Каждое из них имеет следующие базовые модули управления компьютером: модуль подключения к ПК, модуль управления событиями мыши и жестами тачпада, модуль клавиатуры, модули управления воспроизведением медиафайлов и демонстрации презентаций, а также модуль, отвечающий за изменение режима активности компьютера.

Помимо данных модулей, приложение Remote Mouse поддерживает основные команды браузера, содержит полноценную клавиатуру и позволяет быстро переключаться между запущенными программами на компьютере. Приложение Unified Remote позволяет просматривать файлы от корня дисков и выполнять все основные действия над элементами в проводнике. Программа KDE Connect является самой многофункциональной. Она позволяет полностью синхронизировать компьютер и смартфон с помощью общего буфера обмена, отправки уведомлений, файлов в обе стороны и многого другого.

Описание основных модулей и сравнение их функциональности представлено в таблице 1.

Таблица 1 – Описание и сравнение модулей приложений

Модуль	Описание	Сравнение
Модуль подключения к компьютеру	Приложение позволяет искать в беспроводной локальной сети компьютеры, подключаться к ним по Wi-Fi, а также быстро соединяться с устройством с помощью истории подключений.	Приложения Remote Link и Unified Remote поддерживают также соединение по Bluetooth. Remote Mouse позволяет подключаться с помощью QR-кода. Подключение и поиск устройств в приложениях происходит быстро, однако в Remote Link после подключения присутствует 7-10-секундная задержка перед началом работы.
Виртуальная мышь и тачпад	Приложение содержит большое поле, имитирующее сенсорную панель в ноутбуках, поддерживает мультисенсорные жесты.	Масштабирование на Windows реализовано только в Remote Link и Unified Remote. В приложениях Remote Link и Remote Mouse для действий мыши предусмотрены отдельные кнопки, что является ощутимым преимуществом. В программах Unified Remote и KDE Connect имеются некоторые проблемы при работе с сенсорной панелью.
Виртуальная клавиатура	Приложение содержит функцию удалённого ввода текста с помощью привычной клавиатуры Android.	Все приложения, кроме Remote Link, отправляют на компьютер по одному символу по мере ввода текста, что является более удобным, чем нажатие кнопки отправки после ввода всего текста, как это реализовано в Remote Link
Пульт управления медиафайлами	Приложение позволяет в проигрывателях Windows Media Player и VLC Media Player воспроизводить песни и видео, ставить их на паузу, регулировать громкость, отключать и включать звук на компьютере.	Программы Unified Remote и KDE Connect, помимо управления медиафайлами в плеере, позволяют управлять ими в браузере, что является значительным преимуществом. Однако приложение Remote Link предоставляет возможность включать режимы циклического повтора песен и случайного порядка, а также позволяет перематывать песни и просматривать библиотеку медиафайлов, но только в Windows Media Player. В приложении Remote Mouse данная функция платная и протестирована не была.
Пульт управления презентациями	Приложение реализует функции открытия презентации в полноэкранном режиме, переключение слайдов вперёд и назад, завершение демонстрации.	В приложениях Remote Link и KDE Connect имеются гироскопические указки для презентаций, а Remote Link также предоставляет удобную функцию просмотра миниатюр слайдов и перемещения между ними. В приложении Remote Mouse данная функция платная и протестирована не была.
Модуль управления режимами активности	Приложение позволяет перезагружать компьютер, переводить его в спящий режим, выключать, выходить из текущей учётной записи.	Приложение Unified Remote, помимо перечисленных функций, поддерживает блокировку, гибернацию, технологию Wake-on-LAN. KDE Connect не умеет данного модуля.

На основе проведенного анализа существующих приложений для дистанционного управления компьютером, следует отметить, что ни одно из приложений не является идеальным для работы, но каждое из них имеет собственные преимущества, отличающие его от других.

Например, программа Remote Link прекрасно справляется с управлением компьютером с помощью жестов тачпада, имеет очень удобный модуль показа презентаций, однако управлять воспроизведением песен она позволяет только в плеере. При этом значительная часть пользователей предпочитает слушать музыку в браузере, управлять которой в данном случае с помощью Remote Link нельзя. Функция управления треками в браузере реализована в приложениях KDE Connect и Unified Remote, однако и в первом, и во втором имеются небольшие проблемы в главном модуле программы – в модуле управления виртуальной мышью и тачпадом. Приложение Remote Mouse, несмотря на удобство его использования, имеет много платных функций, что превращает бесплатную версию приложения в обычную беспроводную мышь и клавиатуру.

Проведённое тестирование и анализ приложений позволяет сделать определённые выводы насчёт разработки многих функций системы и не допустить ошибок, которые были найдены в системах-аналогах.

Постановка задачи

Целью данной работы является анализ существующих программных систем, проектирование авторского протокола и архитектуры программной системы для удалённого управления компьютером под ОС Windows с помощью мобильного устройства под ОС Android, а также программная реализация основных модулей такой системы.

Чтобы организовать дистанционное взаимодействие в беспроводной локальной сети между компьютером и смартфоном, необходимо иметь модули Wi-Fi и специальное программное обеспечение. Общие принципы организации взаимодействия в сети определяет архитектура «клиент-сервер», на основе которой и должна быть построена данная система.

Клиент-серверное приложение разбито на два составляющих процесса:

– клиент, который просит у сервера оказания определённой услуги и отправляет ему сообщение с запросом;

– сервер, который получает и обрабатывает запрос, выполняя при этом соответствующие действия [4].

Для достижения коммуникационной цели между клиентом и сервером используются межсетевые протоколы. Протокол TCP обеспечивает прикладные программы безошибочным полнодуплексным каналом связи, сохраняя целостность данных [5]. Поверх TCP и будем создавать собственный протокол обмена информацией, который будет описывать правила взаимодействия между клиентом и сервером.

Таким образом, чтобы удалённо управлять компьютером с помощью смартфона, необходимо создать клиент-серверное программное обеспечение и разработать протокол обмена данными. Необходимо так спроектировать архитектуру, чтобы программа могла управлять действиями компьютерной мыши и выполнять отправку символов с клавиатуры, воспроизводить медиафайлы и регулировать громкость, управлять показом презентаций и выполнять основные команды в браузере, переключать запущенные приложения и управлять режимами активности компьютера.

Разработка протокола обмена информацией

Для обмена сообщениями между клиентом и сервером был выбран формат JSON. Он реализует текстовое представление структурированных данных, основанное на принципе пар ключ-значение [6].

Рассмотрим структуру пакета, передаваемого между клиентом и сервером. Условно пакет делится на 2 части: заголовок и тело. В заголовке указывается тип пакета, а тело содержит данные в формате JSON. Структура пакета представлена в таблице 2.

Таблица 2 – Пример структуры пакета

Тип	Поле	Размерность	Описание
PacketTypes	type	1 байт	Тип пакета
JsonObject	body	не фиксирована	Тело пакета

В качестве признака окончания данных пакета выступает символ перевода строки, занимающий 1 байт и добавляющийся непосредственно при передаче данных сокетом.

Поле «type» (тип) является перечислением PacketTypes (тип пакета) и может иметь следующие значения, представленные в таблице 3.

Таблица 3 – Константы в перечислении PacketTypes

Значение	Имя	Значение	Имя
0	PING	6	KEYBOARD
1	HELLO	7	MEDIA
2	OK	8	PRESENTATION
3	ERROR	10	DISPATCHER
4	BYE	11	POWER
5	MOUSE		

Первые пять команд необходимы для организации соединения между клиентом и сервером. Команды «PING» и «HELLO» инициируются исключительно клиентом, а «OK» и «ERROR» - сервером. Команда «BYE» может быть отправлена как клиентом, если он завершает удалённое управление компьютером, так и сервером, если его работа была прекращена пользователем. Остальные команды отвечают за удалённое управление и отправляются от клиента на сервер.

Пакет типа «PING» с пустым телом посылается клиентом во время обнаружения сервера в сети на все доступные IP-адреса подсети. При получении данного пакета от нового клиента сервер отправляет пакет типа «OK» с сетевым именем компьютера и закрывает соединение.

Пакет типа «HELLO» с именем мобильного устройства (в качестве него используется модель телефона) посылается клиентом при подключении к серверу для начала взаимодействия с ним. При получении данного пакета от нового клиента сервер отправляет пакет типа «OK» с сетевым именем компьютера и начинает работу с клиентом. Если подключение нового клиента невозможно (например, при превышении лимита клиентов), то сервер отправляет пакета типа «ERROR» с сообщением об ошибке.

Пакет типа «BYE», как и пакет «PING», имеет пустое тело.

Пакет типа «MOUSE» клиент отправляет, когда работает с модулем управления мышью и жестами тачпада. В нём содержится название действия (перемещение курсора мыши, правый или левый клик, двойной клик, перетаскивание и выделение элементов, скроллинг, масштабирование) и дополнительные параметры при необходимости (например, относительные координаты при перемещении курсора или направление прокрутки при скроллинге).

Пакет типа «KEYBOARD» отправляется при вводе символов с клавиатуры или нажатии кнопок для ввода специальных клавиш клавиатуры. В нём содержится описание нажатых клавиш (отдельный символ либо комбинация) и конкретно клавиши (символ или, например, специальная клавиша и символ).

Следующие пакеты описаны кратко, так как среди поставленных задач не было задачи программной реализации данных модулей. Тем не менее, примеры содержания данных пакетов будут представлены ниже.

Пакет типа «MEDIA» отправляется клиентом на сервер для управления воспроизведением медиафайлов.

Пакет типа «PRESENTATION» передаёт серверу команды при демонстрации презентаций.

Пакет типа «BROWSER» отправляется клиентом для выполнения основных команд браузера.

Пакет типа «DISPATCHER» отправляется клиентом с целью получения списка запущенных приложений либо активации какого-либо из них.

Пакет типа «POWER» передаёт серверу команды управления питанием компьютера.

Примеры содержания всех пакетов представлены в таблице 4.

Таблица 4 – Примеры пакетов

Тип	Содержание
PING	0{}\n
HELLO	1{"name": "Redmi 4X"}\n
OK	2{"name": "AsusX550VX"}\n
ERROR	3{"message": "Превышен лимит подключений"}\n
BYE	4{}\n
MOUSE	5{"action": "move", "dx": "1", "dy": "-1"}\n
KEYBOARD	6{"type": "shortcuts", "key1": "Ctrl", "key2": "A"}\n
MEDIA	7{"volume": "up"}\n
PRESENTATION	8{"command": "next"}\n
BROWSER	9{"command": "open new tab"}\n
DISPATCHER	10{"command": "get application list"}\n
POWER	11{"command": "power off"}\n

Проектирование архитектуры программной системы

Как было определено ранее, программная система для удалённого управления компьютером имеет клиент-серверную архитектуру. На мобильном устройстве под операционной системой Android устанавливается клиентское приложение Remote Stick. Посредством TCP-соединения клиент взаимодействует с серверным приложением Remote Stick Server, работающим на компьютере под Windows. Исполнимый файл «RemoteStickServer.exe» при запуске подключает динамическую библиотеку «win32.dll». Чтобы реализовать такие функции операционной системы Windows, как передвижение мыши, осуществление кликов, отправку кодов клавиш клавиатуры, и многие другие, необходимо использовать нативные API-функции. Для этого должна быть разработана и подключена в проект данная библиотека, в которой и будут созданы обёртки для всех необходимых вызовов системных функций. Диаграмма развёртывания программы представлена на рисунке 1.

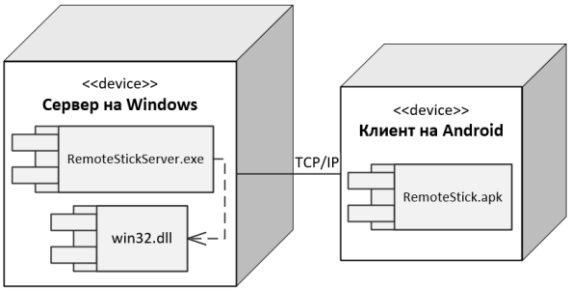


Рисунок 1 – Диаграмма развёртывания

Внутренняя структура сервера будет частично схожа со структурой клиента. В каждом из них находится управляющий класс, отвечающий за поддержку соединения между сторонами и осуществляющий координацию действий между модулями приложения. Объект данного класса содержит список модулей, или плагинов, которые на клиенте будут формировать и отправлять пакеты, а на сервере – получать и обрабатывать их, совершая те или иные действия по управлению компьютером. Все плагины наследуются от абстрактного класса Plugin, который содержит формируемый или обрабатываемый им тип пакета и собственно метод формирования или его обработки. Данный метод является абстрактным и переопределяется в дочерних классах, конкретизирующих плагин.

Управляющим классом на сервере является RemoteStickServer. Он содержит методы запуска сервера, подключения и обработки клиентов, а также список плагинов, которые выполняют команды, полученные от клиента.

Диаграмма классов серверного приложения изображена на рисунке 2.

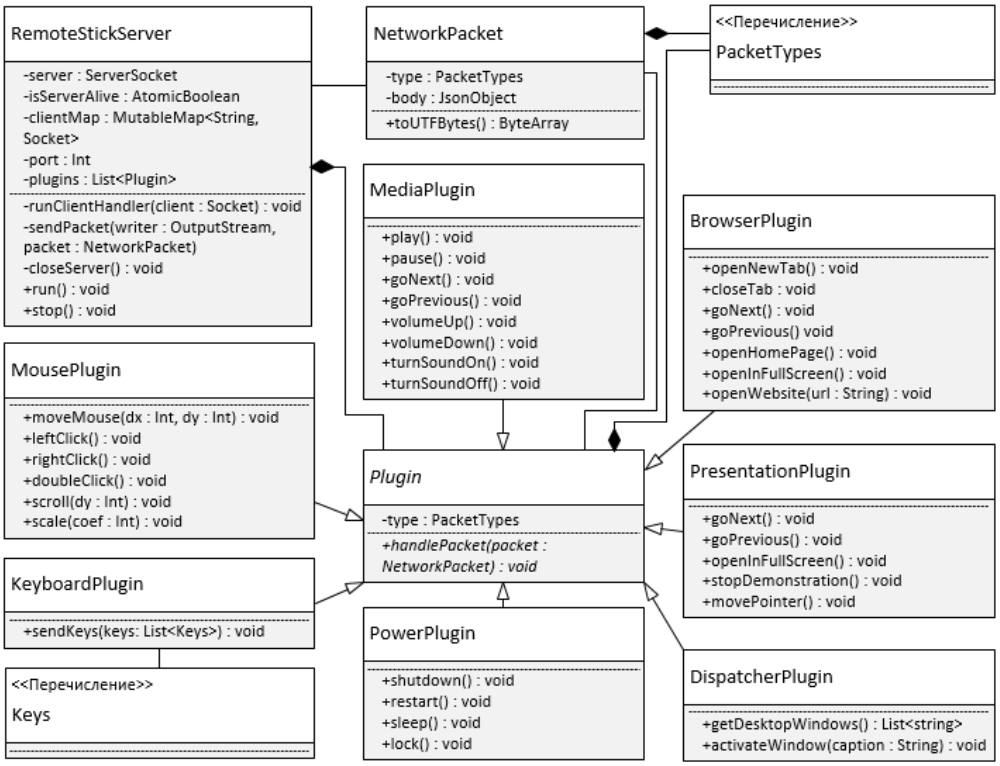


Рисунок 2 - Диаграмма классов серверного приложения

Диаграмма классов клиента, на которой для лаконичности классы плагинов были свернуты, представлена на рисунке 3.

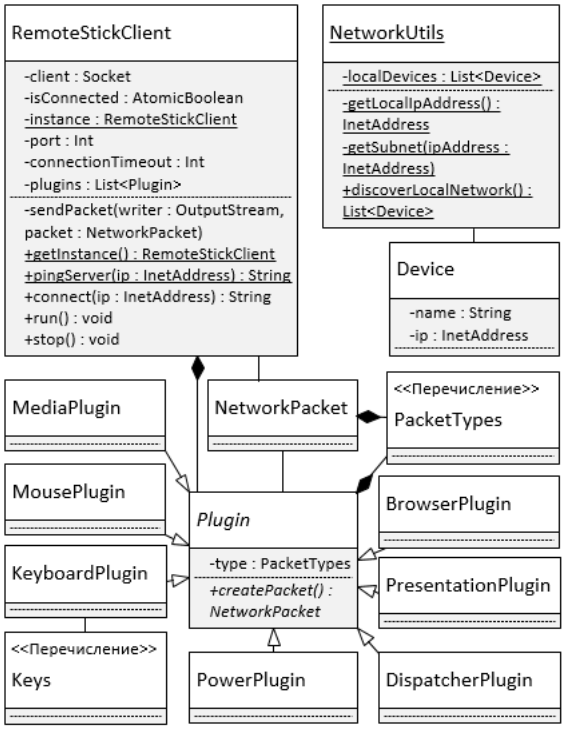


Рисунок 3 - Диаграмма классов клиента

Основным классом, осуществляющим управляющую работу в клиентском приложении, является RemoteStickClient. Класс реализуется в виде синглтона, для того чтобы гарантировать во всём приложении единую, согласованную точку доступа к единственному объекту клиента [7]. Он содержит статический метод, отправляющий пинг на сервер, метод подключения к серверу, запуска и прекращения работы клиента. Аналогично устройству класса RemoteStickServer, класс RemoteStickClient содержит список плагинов, которые создают и отправляют пакеты на сервер. Помимо этого, для обнаружения устройств в сети используется статический класс NetworkUtils. Метод обнаружения устройств возвращает список найденных серверов с их именами и IP-адресами, которые инкапсулируются классом Device.

Программная реализация

Для реализации программной системы была выбрана программная платформа Java SE и высокоуровневый объектно-ориентированный язык программирования Kotlin [8], провозглашённый компанией Google на конференции Google I/O 2017 одним из официальных языков для разработки под Android (наряду с Java и C++) [9].

Для создания графического приложения на платформе Java используется инструментарий JavaFX, позволяющий создавать приложения с богатой насыщенной графикой благодаря использованию аппаратного ускорения графики и возможностей GPU [10]. Для того, чтобы из созданного приложения можно было создать исполнимый файл для Windows 7/8/10 x32 и x64, для разработки используется 32-битный JDK

версии 8u251 и встроенный в него модуль JavaFX. В качестве интегрированной среды разработки (IDE) используется IntelliJ IDEA версии 2020.1.1.

Первоначальной задачей при разработке программной системы является модуль обнаружения доступных устройств в сети и подключение клиента к серверу. На рисунке 4 изображена диаграмма последовательности при обнаружении сервера в сети и взаимодействие объектов системы при этом.

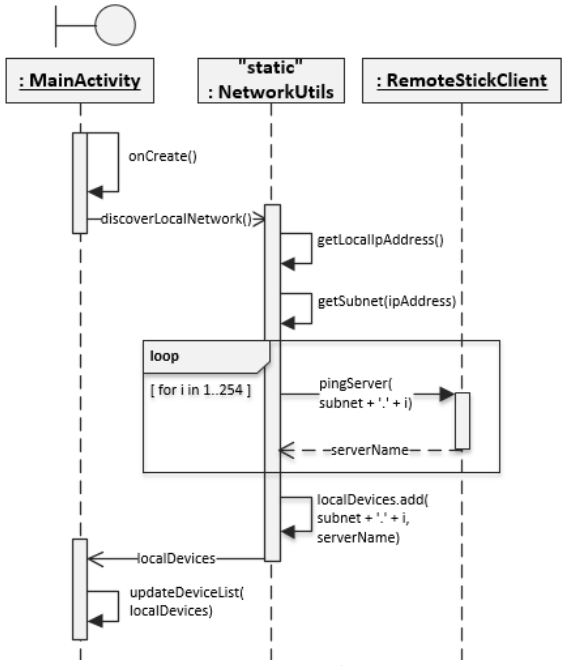


Рисунок 4 - Диаграмма последовательности при обнаружении устройств в сети

После запуска основной активности клиентского приложения вызывается статический метод класса NetworkUtils для обнаружения устройств в локальной сети. По диаграмме видно, что он запускается асинхронно. В Android не рекомендуется создавать "долгоиграющие" процессы, каким и является работа с сетью, в основном потоке, это может вызвать зависание и прекращение работы приложения [**Ошибка! Источник ссылки не найден.**].

Чтобы интерфейс был отзывчивым, метод обнаружения устройств запускается в дополнительном потоке. Данный метод сначала получает IP-адрес мобильного устройства, затем определяет маску подсети и в цикле перебирает возможные IP-адреса, проверяя их на доступность и посылая на них пинг с помощью статического метода pingServer(InetAddress) класса RemoteStickClient. При удачном завершении вызова устройство с полученными IP-адресом и именем сервера добавляется в список локальных устройств. После окончания работы цикла список устройств передаётся в интерфейс, который обновляет соответствующий компонент.

Следующей и основной задачей реализации программной системы является обеспечение функционирования алгоритма перемещения курсора мыши. Когда пользователь в приложении начинает перемещать палец по специальному полю, имитирующему тачпад в ноутбуках, формируются пакеты типа «MOUSE» и отправляются на сервер. Данные пакеты содержат команду перемещения и относительные координаты курсора, которые рассчитывается по алгоритму, представленному на рисунке 5.

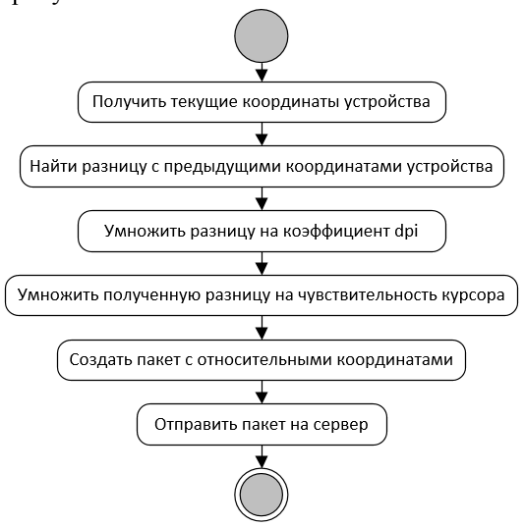


Рисунок 5 - Диаграмма деятельности для алгоритма отправки координат курсора на сервер

После получения пакета сервером и передачи его модулю управления курсором мыши начинает работать вторая часть алгоритма, которая обрабатывает пакет и перемещает курсор. Этот алгоритм представлен на рисунке 6.

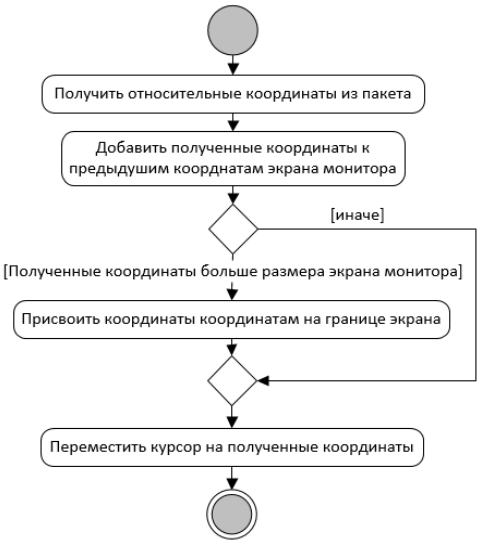


Рисунок 6 - Диаграмма деятельности для алгоритма перемещения курсора по экрану

Выводы

В ходе данной работы был проведён обзор и сравнительный анализ популярных приложений для удалённого управления компьютером с помощью смартфона. Был сделан вывод, что ни одно из них не позволяет в полной мере и с удобством выполнять основные функции при работе с компьютером.

Для решения данной проблемы было предложено спроектировать и разработать собственное приложение для удалённого управления компьютером. На начальном этапе был описан принцип работы программы. Затем был разработан протокол обмена информацией между устройствами и спроектировано клиент-серверное ПО для удалённого управления ПК. На основе разработанной архитектуры были созданы модули обнаружения устройств в сети и подключения к компьютеру, а также модули, позволяющие выполнять действия мыши компьютера и отправлять символы на экран.

В дальнейшем планируется реализовать все спроектированные модули и создать удобный многофункциональный продукт, который будет конкурентоспособным на рынке приложений.

Литература

1. KDE Connect // KDE Connect. [Электронный ресурс]. – Режим доступа: https://kdeconnect.kde.org/. - Загл. с экрана.

2. Unified Remote – Remote Control App for your computer // Unified Remote. [Электронный ресурс]. – Режим доступа: https://www.unifiedremote.com/. – Загл. с экрана.

3. Turn iPhone, iPad and Android into wireless mobile mouse / trackpad / keyboard with Remote Mouse // Remote Mouse. [Электронный ресурс]. – Режим доступа: https://remotemouse.net/. – Загл. с экрана.

4. Клиент-серверная архитектура // Oracle Patches. [Электронный ресурс]. – Режим доступа: https://oracle-patches.com/is/3875-клиент-серверная-архитектура. – Загл. с экрана.

5. Барфилд, Эд. Программирование "клиент-сервер" в локальных вычислительных сетях: [пер. с англ.] / Эд Барфилд, Брайен Уолтерс. – М.: Филинъ. 1997. – 422. – ISBN 5-89568-015-1.

6. Что такое JSON? // Losst. [Электронный ресурс]. – Режим доступа: https://losst.ru/chto-takoe-json. – Загл. с экрана.

7. Carlo Chung. Pro Objective-C Design Patterns for iOS Pro Objective-C Design Patterns for iOS / Carlo Chung. – N.Y.: Apress, 2011. – P. 450. – ISBN 978-1-4302-3331-2.

8. Kotlin Programming Language // Kotlin. [Электронный ресурс]. – Режим доступа: https://kotlinlang.org/.– Загл. с экрана.

9. Что такое Kotlin. Первая программа // Metanit.com. [Электронный ресурс]. – Режим доступа: https://metanit.com/kotlin/tutorial/1.1.php. – Загл. с экрана.

10. Введение в Java FX // Metanit.com. [Электронный ресурс]. – Режим доступа: https://metanit.com/java/javafx/1.1.php. – Загл. с экрана.

Мамутова В.А., Чернышова А.В. Программное обеспечение для удалённого управления персональным компьютером. *В статье представлен обзор и сравнительный анализ популярных приложений для удалённого управления компьютером с помощью мобильного устройства. Описан принцип работы подобной программной системы. Разработан собственный протокол обмена информацией между устройствами. Выполнено проектирование клиент-серверного программного обеспечения для дистанционного управления компьютером и разработаны модули, позволяющие на расстоянии выполнять действия компьютерной мыши и отправлять символы на экран.*

Ключевые слова: *удалённое управление компьютером, клиент-серверное приложение, протокол обмена информацией, дистанционное управление, Android.*

Mamutova V. A., Chernyshova A.V. Software for remote control of a personal computer. *This article provides an overview and comparative analysis of popular applications for remote computer management using a mobile device. The principle of operation of such a software system is described. A proprietary protocol for exchanging information between devices has been developed. Client-server software was developed for remote computer management and modules were developed that allow performing computer mouse actions and sending symbols to the screen at a distance.*

Keywords: *remote computer management, client-server application, information exchange protocol, remote control, Android.*

Статья поступила в редакцию 20.09.2021
Рекомендована к публикации профессором Мальчевой Р. В.

УДК 004.03 ИВТ

Анализ влияния развития компонентов компьютерных систем на учебный процесс.

Н. С. Максименко, Е.И. Приходченко, Л.И. Дорожко
Донецкий национальный технический университет, г. Донецк
E-mail: nata.demesh@gmail.com

Аннотация

В статье проведен анализ развития компонентов компьютерных систем, приведена хронология развития центральных процессоров и развитие языков программирования. Также в статье приведена история развития факультета «Компьютерные науки и технологии», проанализированы темпы и динамика развития компьютерной техники в ДонНТУ, выявлено влияние развития компонентов компьютерных систем на учебный процесс. Проанализировано развитие курса «Программирование».

Введение

На сегодняшний день развитие вычислительных систем позволяет внедрять перспективные технологии, требующие как больших вычислительных мощностей, так и серьёзного уменьшения габаритов устройств. Однако столь быстрый прогресс требует тщательного анализа и оценки дальнейших перспектив развития вычислительной техники. Происходит активная интеграция современных достижений во многие сферы человеческой деятельности, что требует дополнительного изучения и анализа.

Цель и задачи исследования

Целью данной работы является анализ влияния развития таких компонентов компьютерных систем как процессоры на учебный процесс, что способствует внедрению новых технологий в учебные дисциплины вузов.

Необходимо провести анализ динамики развития аппаратных и программных средств, а также рассмотреть эволюцию компьютерного обеспечения учебного процесса и его влияние на подготовку специалистов на примере ДонНТУ.

Закономерности развития компьютерных систем

Гордон Мур (один из основателей фирмы Intel) в 1965 году, подводя итоги первого десятилетия развития полупроводниковой элементной базы и начального периода развития интегральных микросхем, формулирует закон, который звучит следующим образом: «Количество элементов на кристаллах электронных микросхем будет и далее удваиваться каждый год».

Однако, спустя 10 лет в 1975 году Мур вносит в свой закон коррективы, согласно которым удвоение числа транзисторов будет происходить каждые два года (24 месяца) (рис.1). Теперь закон приобрел следующий вид: «Число транзисторов на кристалле удваивается каждые два года».

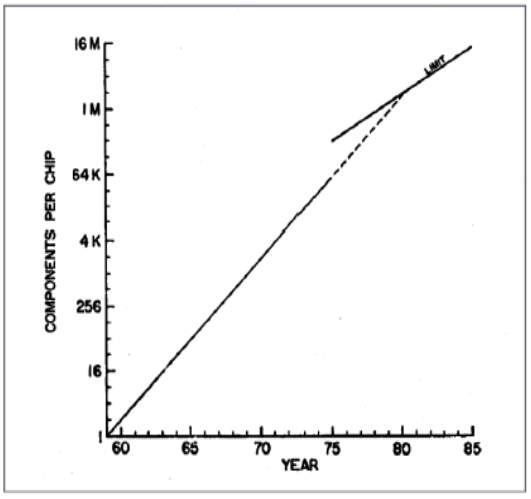


Рисунок 1 - Этим рисунком завершается доклад Гордона Мура 1975 года

На данный момент существует два варианта закона Мура, которые можно обозначить в соответствии с годом их появления как «закон Мура 1965» и «закон Мура 1975» [1-5].

Закономерности развития компонентов компьютерных систем изучались и продолжают исследоваться в Донецком национальном техническом университете под руководством А.Я. Аноприенко.

В работе [1] приводится пример более точной оценки роста производительности, определяющей десятикратный рост каждые 4 года, что предполагает более точную оценку ежегодного коэффициента роста (EKP): 1,77828. Это примерно соответствует значению EKP $\sqrt{\pi} = 1{,}7725$ [6].

Закономерности развития компонентов компьютерных систем

Проследим за тем, как развивались центральные процессоры — наиболее важная составляющая любого компьютера.

Первые процессоры появились в 40-х годах XX века. На момент выпуска они работали на электромеханических реле и вакуумных лампах, а роль запоминающих устройств выполняли ферритовые сердечники. С 1950-х годов в конструкции процессоров стали применяться транзисторы.

Технология изготовления интегральных схем получила свое развитие в 1960-х годах, что позволило создавать микрочипы с расположенными на них транзисторами.

Однако выпуск IBM System/360 в 1964 году, немного приблизил дизайн компьютеров и процессорных систем к современному виду. До появления этого компьютера все системы, в том числе и процессорные системы, работали только с тем программным кодом, который был написан специально для них. Компания IBM в своих компьютерных системах начала использовать новую политику: вся линейка разных по производительности процессорных систем начала поддерживать один и тот же набор инструкций, что позволило писать программное обеспечение, которое работало бы под управлением любой модификации System/360 [6,7].

Процессоры с 4- и 8-битной архитектурой

В 1971 году вошел в историю как год появления первых микропроцессоров: компания Intel, выпускает на рынок первый в мире коммерчески доступный однокристальный процессор – модель 4004. В 1974 году выходит 4-битный Intel 4040.

В начале 70-х годов в компания Intel начинает выпускать и 8-битную модель — 8008.

Компания Motorola в этот период времени становится главным конкурентом Intel. В 1974 году компанией Motorola был выпущен процессор 6800. Кристалл выпускался по 6-мкм технологии, а его тактовая частота составляла 2 МГц. Набор инструкций процессора содержал 78 команд. Процессор 6800 имел 16-битную адресную шину с прямой адресацией 64 Кбайт памяти [7,8].

Процессоры с 16-битной архитектурой

В 1970-х годах центральные процессоры получили очень быстрое развитие. Вслед за 4-битными и 8-битными кристаллами, производители взялись за разработку новых 16-разрядных процессоров.

8 июня 1978 года был официально анонсирован процессор Intel 8086. Параллельно с 80186, в Intel велась разработка процессора с индексом 80286 [6,8].

Процессоры с 32-битной архитектурой

В 1981 году компания Intel выпускает новый кристалл с индексом 80386, ставший первым 32-разрядным процессором. Процессор i386 сохранил обратную совместимость со своими предшественниками 8086 и 80286.

13 октября 1985 года был выпущен процессор i386 с тактовой частотой 16 МГц. С июня 1988 года после выхода процессора 386SX модификация процессора i386 получила приставку DX (Double-word eXternal, что подчеркивало поддержку процессором 32-битной внешней шины данных), соответственно модель получила название 386DX сразу.

В 1987 году частота была повышена до 20 МГц, в 1988 году — до 25 МГц. А в 1990 году в продажу поступила модификация с частотой 33 МГц [12].

В 1993 году выходит следующее поколение процессоров – всем известные Pentium на базе архитектуры P5. Архитектура P5 стала суперскалярной, шина данных – 64-битной, а кэш-память данных и инструкций была разделена на два отдельных блока объемом 8 Кбайт каждый. В 1995 году архитектура P54C еще раз получила обновление – переведена на 350-нм техпроцесс.

В 1996 году P5 получила последнее обновление — P55C.

В 1995 году была представлена архитектура P6 — CISC-платформа с RISC-ядром.

В 1995 году были выпущены процессоры следующего поколения Pentium Pro, кристаллы которых работали на частоте 150-200 МГц, а также имели 16 Кбайт кэш-памяти первого уровня и до 1 Мбайт кэша второго уровня. В 1997 году компанией Intel были представлены Pentium II.

В 1998 году в продаже появились процессоры с улучшенным ядром P6 (Deschutes).

В 1999 году были представлены компанией Intel первые процессоры Pentium III, которые базировались на новой генерации ядра P6–Katmai (модифицированная версия Deschutes). В конце года ядро Coppermin сменило Katmai. Новые процессоры имели интегрированную кэш-память второго уровня в отличие от Katmai, выпускались они по 180-нм технологическому процессу. Их частота достигала 1,13 ГГц [6, 9].

В 2008 году выпуск процессоров с архитектурой NetBurst был остановлен. На

смену NetBurst приходит более совершенная микроархитектура Core [6, 10].

Процессоры с 32-битной архитектурой

После постигнувшей Intel неудачи в архитектуре NetBurst, компания начинает обдумать новую стратегию на ближайшее будущее. Процессоры Pentium 4 показали, что NetBurst не может достойно конкурировать с AMD K8. С течением времени преимущество решений конкурента лишь возрастало. В начале 2006 года представлена новая микроархитектура следующего поколения, которая получила название Core. В ней было принято решение вернуться к корням и позаимствовать лучшие черты архитектуры P6.

Выпуск на рынок микроархитектуры Core ознаменовал собой процессоры со следующими названиями:

- Merom – предназначался для мобильных систем;
- Conroe, Allendale – были предназначены для настольных компьютеров. Ядро Allendale было упрощенной версией Conroe, в нем была уменьшена частота системной шины с 1066 МГц до 800 МГц, а также уменьшен объем кэш-памяти 2-го уровня с 4 Мбайт до 2 Мбайт. К этому добавилось отсутствие поддержки аппаратной виртуализации;
- Woodcrest – предназначался для серверных систем.

После выхода микроархитектуры Core компания Intel успела выпустить на рынок 32-нм процессоры следующего поколения — Sandy Bridge, их 22-нм модификацию Ivy Bridge и кристаллы на базе 22-нм архитектуры Haswell.[6,11]

Развитие языков программирования

Программирование — это процесс создания программ (программного обеспечения). Для этого программисты пишут исходный код на одном из языков программирования.

Рассмотрим их историю развития.

В 1843 году был изобретен первый язык программирования. Ада Лавлейс изобрела первый в истории машинный алгоритм для одной из первых вычислительных машин

В 1944-1945 годах Конрад Цузе разработал «настоящий» язык программирования под названием Plankalkül (Расчет плана).

В 1949 году был создан Ассемблер, который использовался в автоматическом калькуляторе с электронным запоминанием задержки (EDSAC). Ассемблер был разновидностью низкоуровневого языка программирования, который упростил язык машинного кода. В этом же году был изобретен Shortcode. Он стал первым языком высокого уровня (HLL).

Автокод был общим термином, используемым для семейства языков программирования. Autocode, впервые разработанный Аликом Глении в 1952 году для компьютера Mark 1 в Университете Манчестера, был первым в истории скомпилированным языком, который был реализован, что означает, что он может быть переведен непосредственно в машинный код с помощью программы, называемой компилятором.

FORTRAN был создан Джоном Бэкусом в 1957 году и считается старейшим языком программирования, используемым сегодня. Язык программирования был создан для научных, математических и статистических вычислений высокого уровня. FORTRAN до сих пор используется в некоторых из самых передовых суперкомпьютеров в мире.

В 1958 году был создан ALGOL (Алгоритмический язык) совместным комитетом американских и европейских компьютерных ученых. Алгол послужил отправной точкой для разработки некоторых из наиболее важных языков программирования, включая Pascal, C, C ++ и Java.

Процессор списков или LISP был изобретен в 1958 году Джоном Маккарти в Массачусетском технологическом институте (MIT). Первоначально предназначенный для искусственного интеллекта, LISP является одним из старейших языков программирования, которые все еще востребованны сегодня, и его можно использовать вместо Ruby или Python.

В 1959 году был изобретен КОБОЛ (Общий бизнес-ориентированный язык) Общий бизнес-ориентированный язык (COBOL) - это язык программирования, лежащий в основе многих процессоров кредитных карт, банкоматов, телефонных и сотовых вызовов, сигналов больниц и систем сигналов светофора (и это лишь некоторые из них).

Универсальный код символических инструкций для начинающих или BASIC был разработан группой студентов Дартмутского колледжа в 1964 году. Он был написан для студентов, которые плохо разбирались в математике или компьютерах. Этот язык был разработан основателями Microsoft Биллом Гейтсом и Полом Алленом и стал первым товарным продуктом компании.

В 1970 году был изобретен ПАСКАЛЬ. Названный в честь французского математика Блеза Паскаля, Никлаус Вирт разработал язык программирования в его честь. Он был разработан как средство обучения компьютерному программированию, что означало, что его легко освоить. Apple

предпочитала его на заре своей деятельности из-за простоты использования и мощности.

Smalltalk, разработанный в 1972 году в исследовательском центре Xerox в Пало-Альто Аланом Кей, Аделем Голдбергом и Дэном Ингаллсом, позволял программистам изменять код на лету. Он представил множество аспектов языка программирования, которые сегодня являются видимыми языками, такими как Python, Java и Ruby. Такие компании, как Leafly, Logitech и CrowdStrike, заявляют, что используют Smalltalk в своих технических стеках.

В 1972 году был разработан C (Си) Деннисом Ричи из Bell Telephone Laboratories для использования с операционной системой Unix. Он был назван C, потому что был основан на более раннем языке под названием «B». Многие из ведущих в настоящее время языков являются производными от C, включая C #, Java, JavaScript, Perl, PHP и Python. Он также использовался / до сих пор используется такими крупными компаниями, как Google, Facebook и Apple.

В 1972: SQL (в то время SEQUEL) был впервые разработан исследователями IBM Рэймондом Бойсом и Дональдом Чемберленом. SEQUEL (как его тогда называли) используется для просмотра и изменения информации, хранящейся в базах данных. В настоящее время язык является аббревиатурой – SQL (от **Structured Query Language**), что означает язык структурированных запросов. Существует множество компаний, использующих SQL. Некоторые из них включают Microsoft и Accenture.

Изначально Ada была разработана в 1980-1981 годах командой во главе с Джин Ичбиа из CUU Honeywell Bull по контракту с Министерством обороны США. Названный в честь математика середины 19-го века Ады Лавлейс.

В 1983 году Бьярн Страуструп модифицировал язык C в Bell Labs, C ++ - это расширение C с такими улучшениями, как классы, виртуальные функции и шаблоны. Он был включен в 10 лучших языков программирования с 1986 года и получил статус Зала славы в 2003 году. C ++ используется в MS Office, Adobe Photoshop, игровых движках и другом высокопроизводительном программном обеспечении.

В 1983 году Objective-C, разработанный Брэдом Коксом и Томом Лавом, является основным языком программирования, используемым для написания программного обеспечения для операционных систем Apple macOS и iOS.

Perl был создан в 1987 году Ларри Уоллом и представляет собой универсальный язык программирования высокого уровня.

В 1990 году разработан Haskell. Это язык программирования общего назначения, названный в честь американского логика и математика Хаскелла Брукса Карри. Это чисто функциональный язык программирования, то есть в первую очередь математический. Он используется во многих отраслях, особенно в тех, которые имеют дело со сложными вычислениями, записями и обработкой чисел. Как и многие другие языки программирования той эпохи, не так уж часто можно увидеть, что Haskell используется для хорошо известных приложений. С учетом сказанного, язык программирования был использован для написания ряда игр, одна из которых - Nikki and the Robots.

Названный в честь британской комедийной труппы «Монти Пайтон», Python был разработан в 1991 году Гвидо Ван Россумом. Это универсальный язык программирования высокого уровня, созданный для поддержки различных стилей программирования и приятный в использовании (ряд руководств, примеров и инструкций часто содержат ссылки на Monty Python). Python по сей день является одним из самых популярных языков программирования в мире, который используют такие компании, как Google, Yahoo и Spotify.

Visual Basic, разработанный Microsoft в 1991 году, позволяет программистам использовать стиль перетаскивания для выбора и изменения предварительно выбранных фрагментов кода через графический интерфейс пользователя (GUI). В наши дни этот язык не используется слишком часто, однако Microsoft частично использовала Visual Basic для ряда своих приложений, таких, как Word, Excel и Access.

В 1993 году Ruby, созданный Юкихиро Мацумото, представляет собой интерпретируемый язык программирования высокого уровня. Язык обучения, на который повлияли Perl, Ada, Lisp и Smalltalk - среди прочих. В основном Ruby используется для разработки веб-приложений и Ruby on Rails. Twitter, Hulu и Groupon - известные примеры компаний, использующих Ruby.

Java - это универсальный язык высокого уровня, созданный Джеймсом Гослингом в 1995 году для проекта интерактивного телевидения. Он обладает кросс-платформенной функциональностью и неизменно входит в число самых популярных языков программирования в мире. Java можно найти везде, от компьютеров до смартфонов и парковочных счетчиков.

Ранее известный как «Персональная домашняя страница», что теперь означает «Препроцессор гипертекста», PHP был разработан Расмусом Лердорфом в 1995 году. Его основное применение включает создание и поддержку динамических веб-страниц, а также разработку на стороне сервера. Некоторые из крупнейших компаний по всему миру используют PHP, включая Facebook, Wikipedia, Digg, WordPress и Joomla. В этом же году был создан Бренданом Эйхом JavaScript, этот язык в основном используется для динамической веб-разработки, документов PDF, веб-браузеров и виджетов рабочего стола. Почти каждый крупный веб-сайт использует JavaScript. Gmail, Adobe Photoshop и Mozilla Firefox включают несколько хорошо известных примеров.

В 2000 году Microsoft разрабатывает C # с надеждой на объединение вычислительных возможностей C ++ с простотой Visual Basic, C # основан на C ++ и имеет много общего с Java. Этот язык используется почти во всех продуктах Microsoft и используется в основном при разработке настольных приложений.

Scala, разработанная Мартином Одерски в 2003 году, объединяет математическое функциональное программирование и организованное объектно-ориентированное программирование. Совместимость Scala с Java делает его полезным при разработке под Android. Linkedin, Twitter, Foursquare и Netflix - это всего лишь несколько примеров многих компаний, которые используют Scala в своих технических стеках.

Go был разработан Google в 2009 году для решения проблем, возникающих из-за больших программных систем. Благодаря своей простой и современной структуре, Go завоевал популярность среди некоторых крупнейших технологических компаний по всему миру, таких, как Google, Uber, Twitch и Dropbox.

В 2014 году разработанный Swift компанией Apple в качестве замены C, C ++ и Objective-C, Swift был разработан с целью быть проще, чем вышеупомянутые языки, и оставлять меньше места для ошибок. Универсальность Swift означает, что его можно использовать для настольных, мобильных и облачных приложений. Ведущее языковое приложение Duolingo запустило новое приложение, написанное на Swift [12].

Развитие и влияние компьютерной техники на читаемые дисциплины в ДонНТУ

Донецкий национальный технический университет (ДонНТУ) – первое высшее учебное заведение в Донбассе, основанное 30 мая 1921 года. За прошедшие годы он несколько раз переименовывался: от Донецкого Горного техникума до нынешнего названия. ДонНТУ был всегда ведущим и передовым ВУЗом Донбасса. И в то время, когда началась компьютеризация, был в числе первых.

В 1959 году в Донецком индустриальном институте создан электротехнический факультет, первым деканом которого стал Матвей Борисович Шумяцкий. По его инициативе и при его активном участии многое было сделано для создания будущего факультета КНТ. Уже в 1960 году для студентов электротехнических специальностей доцент Сергей Ражденович Буачидзе начал читать курс «Основы вычислительной техники». Важной вехой в истории факультета и института явился 1961 год, когда для студентов всех специальностей института начал читаться курс «Математические машины и программирование», подготовку которого осуществили доцент С.Г.Буачидзе и ассистенты В.И. Назаренко и В.А.Святный. Приобретение в этом же году вычислительной машины «Минск-12» положило начало вычислительному центру ДПИ. В 1962 г. выполнена первая научно-исследовательская работа в области информатики по созданию систем управления с применением управляющих ЭВМ (доц. Дударев Л.С., асс. Святный В.А.).

В 1963 г. была создана кафедра вычислительной техники (ВТ).

В 1965 г. на этой кафедре была открыта специальность «Математические счетно-решающие приборы и устройства», а в 1971 г. – «Прикладная математика».

Все эти годы факультет стремительно развивался. На данный момент он насчитывает восемь направлений подготовки:

1. ИВТ − Информатика и вычислительная техника
2. ИСТ − Информационные системы и технологии
3. ПИ − Программная инженерия
4. САУ − Системный анализ и управление
5. ИНФ− Прикладная информатика
6. ПМК – Прикладная математика
7. МКН – Математика и компьютерные науки
8. БИ - Бизнес-информатика

На базе данного факультета рассматривается развитие компьютерной техники в рамках ВУЗа, а также влияние развития компьютерной техники на читаемые дисциплины.

В 2004 году был проведен ряд исследований, который показал уровень развития компьютерной техники на факультете (рисунки 2-4).

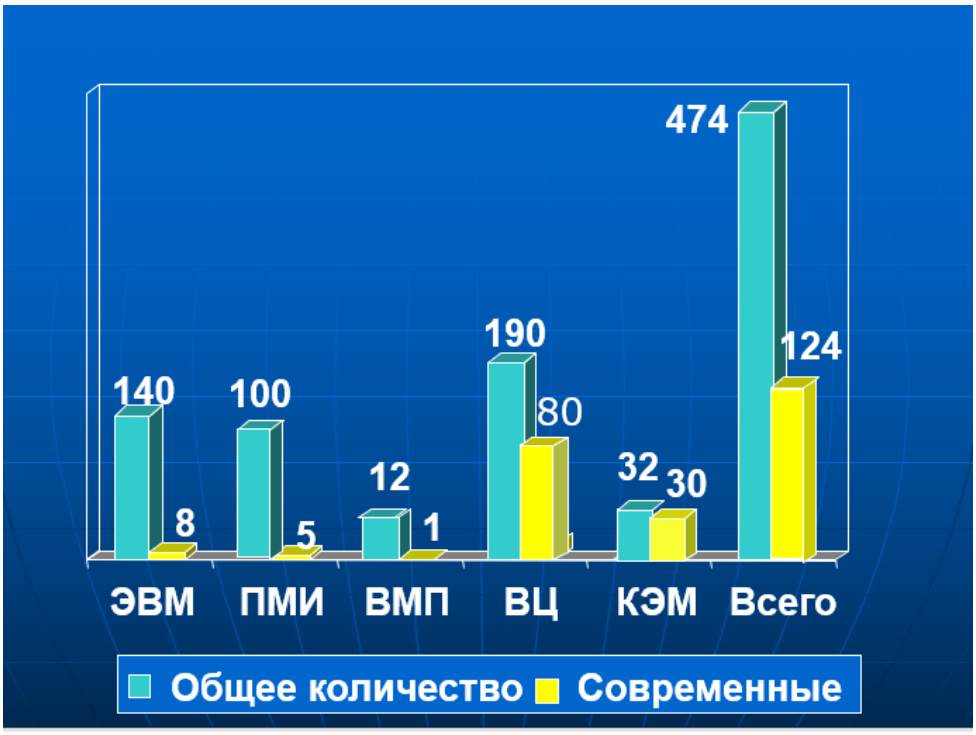


Рисунок 2 - Обеспеченность кафедр ФВТИ компьютерами по состоянию на 2004 год

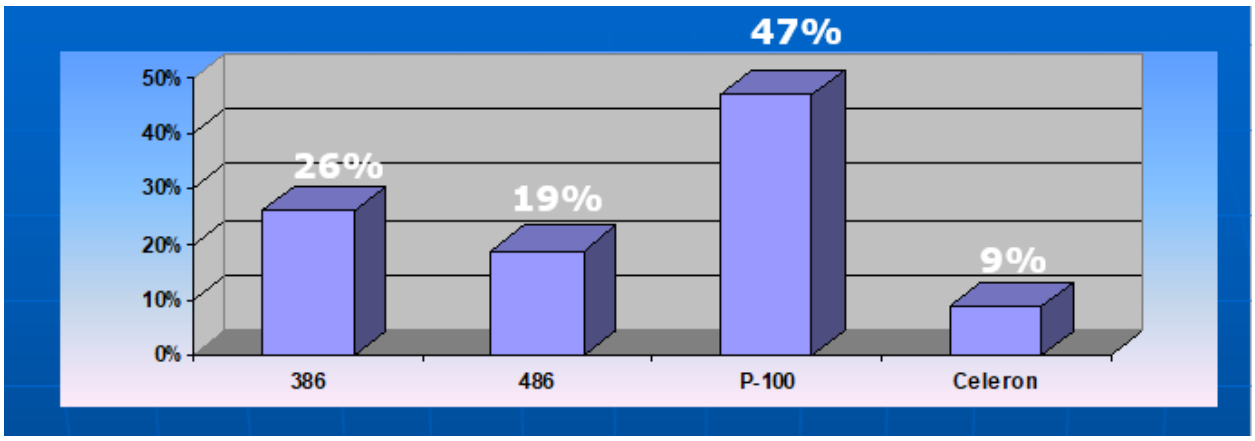


Рисунок 3 - Типы используемых процессоров в компьютерных системах ФВТИ

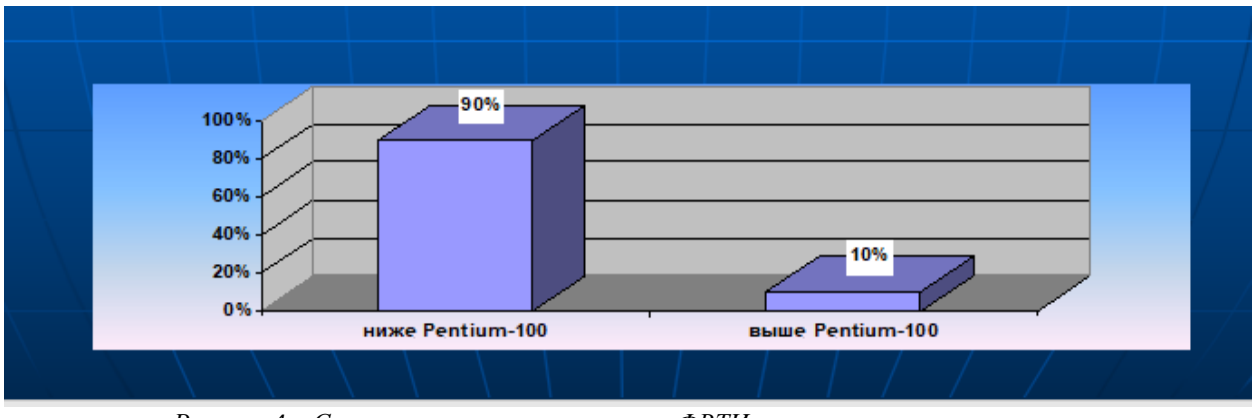


Рисунок 4 - Соотношение компьютеров на ФВТИ по мощности процессора

Позднее, в 2011 году, более детальный анализ компьютеров в учебных аудиториях показал следующие статистические данные (табл. 1). Пример оснащенности некоторых аудиторий на данный момент приведен в табл.2.

Таблица 1 – Характеристики компьютеров по состоянию на 2011г.

Компьютерный класс	**Тип процессора**
4.10	Celeron 366 MHz
4,28	Pentium 1000 MHz
4,26	Celeron 433 MHz-2gHz
4.23, 24, 31	Celeron 133 - 300 MHz
4.36	Celeron 1.2gHz
4.37	Celeron 330 mHz
4.19, 20	PC 550-850 mHz
4.16	Celeron 750 mHz
4.40	Celeron 300 mHz - 1gHz

Таблица 2 – Характеристики компьютеров по состоянию на 2021г.

Компьютерный класс	**Тип процессора**
4.23, 24	PC 550-850 mHz
4.37	Intel(R) Celeron(R) CPU G540 2,50 gHz
4.16	AMD A4-6300 APU with Radeon HD Graphics 3.70 gHz
4.3a	AMD A4-6300 APU with Radeon HD Graphics 3.70 gHz.
4.19	Intel(R) Atom(TM) CPU D410 1,66 gHz
4.20	PC 550-850 mHz
4.40	Intel(R) Celeron(R) CPU G540 2,50 gHz

Как видно из таблиц 1 и 2 компьютерная техника развивается недостаточными темпами. Далее проведен анализ влияния развития компьютерной техники на развитие дисциплины «Программирование».

Долгое время дисциплина «Программирование» велась в аудитории 4.23 и 4.24. Параметры процессоров в этих аудиториях представлены в табл. 1. Начиная с 2000 годов, студенты специальностей «Компьютерные системы и сети» и «Системное программирование» изучали в рамках данного курса язык программирования Pascal (среда программирования Turbo Pascal). Данный язык был выбран на тот момент по причине того, что:

- он был признан лучшим учебным языком программирования;
- он не требователен к аппаратуре.

В 2011 году после обновления некоторых компьютерных классов. Предмет «Программирование» читается в аудиториях с компьютерной техникой имеющей следующие пераметры - Celeron 750 mHz. На этот момент язык программирования остается тот же, а вот среда разработки меняется, на более дружелюбную для пользователей ABC Pascal.

И в 2014 году происходит качественный переворот. В рамках данного предмета, который закладывает базу для большинства курсов цикла программирования, начинает изучаться программирование на языке С. Среда разработки Visual Studio 10. Компьютерная техника, предназначенная для этого предмета, имеет следующие параметры – AMD A4-6300 APU with Radeon HD Graphics 3.70 GHz.

Вывод

Интенсивное развитие компьютерных технологий с начала 90-х годов привело к качественному скачку в развитии человечества. То же можно сказать и об истории развития языков программирования, которые прошли долгий путь от ранних машинных кодов до сложного, удобочитаемого исходного кода, используемого в современных компьютерных системах.

Несомненно, развитие компьютерных средств позволит и в дальнейшем совершенствовать преподавание учебных дисциплин, связанных с программированием, моделированием и проектированием средств вычислительной техники.

Литература

1. Аноприенко, А. Я. Закономерности развития компьютерных систем // Научная дискуссия: инновации в современном мире: сб. ст. по материалам XVIII междунар. заоч. науч.-практ. конф./ Междунар. центр науки и образования. – М., 2013. – № 10 (18). - С. 19-29.

2. Аноприенко, А. Я. Основные закономерности эволюции компьютерных систем и сетей // Науч. тр. Донец. нац. техн. ун-та. – Донецк: ДонНТУ, 2013. – Вып.1(12)-2(13). – С. 10-32 (серия: Проблемы моделирования и автоматизации проектирования).

3. Аноприенко, А. Я. Модели эволюции компьютерных систем и средств компьютерного моделирования // Моделирование и компьютерная графика: материалы пятой междунар. науч. - техн. конф., 24-27 сент. 2013 года / Донец. нац. техн. ун - т. – Донецк, 2013. - С. 403-423.

4. Аноприенко, А. Я. Обобщения закона Мура // Информатика и кибернетика, 2017. - № 3 (9). - С. 14-23.

5. Аноприенко, А. Я. Будущее компьютерных технологий в контексте технической и кодо-логической эволюции. // Вестник Инженерной Академии Украины. Теоретический и научно-практический журнал.

6. Максименко, Н. С. Сравнительный анализ и закономерности развития процессоров AMD и Intel // Информатика и кибернетика, 2020. – №1(19). – С. 63-71.

7. Коленченко, О. Эволюция процессоров. Часть 1: 8-битная эпоха [Электронный ресурс]. - Режим доступа: https://www.ferra.ru/review/computers/processor-evolution-part-1.htm (Дата обращения 22.09.2019).

8. Коленченко, О. Эволюция процессоров. Часть 2: 16-битная эпоха [Электронный ресурс]. - Режим доступа: https://www.ferra.ru/review/computers/processor-evolution-part-2.htm (Дата обращения 22.09.2019).

9. Коленченко, О. Эволюция процессоров. Часть 3: 32-битные процессоры во второй половине 1980-х годов [Электронный ресурс]. - Режим доступа: https://www.ferra.ru/review/computers/processor-evolution-part-3.htm (Дата обращения 22.09.2019).

10. Коленченко, О. Эволюция процессоров. Часть 4: архитектура RISC и развитие индустрии в 1990-е годы [Электронный ресурс]. - Режим доступа: https://www.ferra.ru/review/computers/processor-evolution-part-4.htm (Дата обращения 22.09.2019).

11. Коленченко, О. Эволюция процессоров. Часть 5: современность [Электронный ресурс]. - Режим доступа: https://www.ferra.ru/review/computers/processor-evolution-part-5.htm (дата обращения 22.09.2019).

12. История языков программирования [Электронный ресурс]. – Режим доступа: https://itanddigital.ru/historycoding (дата обращения 27.04.2021).

Максименко Н.С., Приходченко Е.И., Дорожко Л.И. Анализ влияния развития компонентов компьютерных систем на учебный процесс*. В статье проведен анализ развития компонентов компьютерных систем, приведена хронология развития центральных процессоров и развитие языков программирования. Также в статье приведена история развития факультета «Компьютерные науки и технологии», проанализированы темпы и динамика развития компьютерной техники в ДонНТУ, выявлено влияние развития компонентов компьютерных систем на учебный процесс. Проанализировано развитие курса «Программирование».*

Ключевые слова: *компьютерные системы, развитие компьютерных систем, развитие процессоров, учебный процесс, языки программирования*

Maksimenko N.S., Prikhodchenko E.I., Dorozhko L.I. Analysis of the impact of the development of components of computer systems on the educational process. *The article analyzes the development of components of computer systems, provides a chronology of the development of central processors and the development of programming languages. Also in the article the history of the development of the faculty of "Computer Science and Technology" is given, the rates and dynamics of the development of computer technology in DonNTU are analyzed, the influence of the development of the components of computer systems on the educational process is revealed. The development of the course "Programming" is analyzed.*

Keywords: *computer systems, development of computer systems, development of processors, educational process, programming languages*

Статья поступила в редакцию 20.10.2021
Рекомендована к публикации профессором Аноприенко А. Я.

УДК 004.92

Анализ алгоритмов и протоколов маршрутизации данных в беспроводных локальных сетях

Я. О. Кравченко, Р. В. Мальчева
Донецкий национальный технический университет, г. Донецк
e-mail: fftggf4g@gmail.com

Аннотация
В данной статье проанализированы различные алгоритмы и протоколы маршрутизации сбора данных в беспроводных локальных сетях, их преимущества и недостатки. Также описаны ключевые тенденции и выделили десятку беспроводных технологий, которые займут доминирующее положение на рынке в качестве корпоративных архитектур нового поколения. Целью работы является обоснование и выбор наиболее эффективной организации передачи данных при проектировании конкретной беспроводной сети, параметры которой должны соответствовать требованиям технического задания.

Введение

За последние два десятилетия беспроводные технологии успели вытеснить из обихода множество традиционных средств связи и передачи информации. В ближайшие годы ожидается появление новых типов беспроводных коммуникаций, которые станут основой развития перспективных технологий, например робототехники, автономного наземного и авиатранспорта, медицинских гаджетов [1].

В промышленной автоматизации наибольшее распространение получили три типа беспроводных сетей: Bluetooth на основе стандарта IEEE 802.15.1, ZigBee на основе IEEE 802.15.4 и Wi-Fi на основе IEEE 802.11. Физические уровни модели OSI для этих сетей основаны на соответствующих стандартах IEEE, а протоколы верхних уровней разработаны и поддерживаются организациями Bluetooth, ZigBee и Wi-Fi, соответственно [2].

Быстрое развитие технологий привело к тому, что многие из тех, кто специализировался на беспроводной технологии, увеличили производительность и получили инвестиции и прибыль. Технология быстро развивается и становится более совершенной. В то же время сектор динамично меняется, оставаясь одним из самых важных для европейской экономики с годовым оборотом в 290 млрд. Евро и составляет 4% рабочих мест в Европейском Союзе [3].

Следующее поколение стандартов мобильной связи - 5G или система, также известная как WWWW (Wireless World Wide Web), поддерживает всю беспроводную связь неограниченно. Беспроводные веб-приложения 5G включают в себя полную мультимедийную возможность за пределами скорости 4G. Некоторые из преимуществ этой технологии заключаются в том, что она обеспечивает высокую скорость и быструю передачу данных по сравнению с предыдущими поколениями, поддерживает интерактивные медиа, потоковое видео с потоком голоса, бесконечную передачу данных в рамках последней мобильной операционной системы и т.д. В целом, текущая тенденция 5G технология имеет светлое будущее, потому что она обрабатывает лучшие технологии на доступных мобильных телефонах своим клиентам. Новаторские решения сделать мир более приятным [3].

На сегодняшний день существует множество проблем в области создания беспроводных самоорганизующихся сетей с переменной топологией. Одной из главных является проблема маршрутизации. Каждый тип протоколов маршрутизации потенциально имеет свои преимущества и недостатки при различных условиях (плотности узлов и скорости перемещения) использования.

Целью данной статьи является анализ существующих алгоритмов и протоколов маршрутизации для выбора наиболее эффективной организации передачи данных при проектировании конкретной беспроводной сети.

Анализ алгоритмов маршрутизации

Алгоритмы маршрутизации применяются для определения наилучшего пути пакетов от источника к приёмнику и являются основой любого протокола маршрутизации. Для формулирования алгоритмов маршрутизации сеть рассматривается как граф. При этом маршрутизаторы являются узлами, а физические линии между маршрутизаторами — рёбрами

соответствующего графа. Каждой грани графа присваивается определённое число — стоимость, зависящая от физической длины линии, скорости передачи данных по линии или стоимости линии.

Большинство алгоритмов маршрутизации можно свести к трем основным алгоритмам [4].

Маршрутизация по вектору расстояния, в соответствии с которым определяются направление (вектор) и расстояние до каждого канала в сети.

Маршрутизация на основе оценки состояния канала (также называемый выбором наикратчайшего пути), при котором воссоздается точная топология всей сети (или по крайней мере той части, где размещается маршрутизатор).

Гибридная маршрутизация, объединяющая аспекты алгоритмов с определением вектора расстояния и оценки состояния канала.

Алгоритмы маршрутизации на основе вектора расстояния (известные под названием алгоритмы Беллмана—Форда) предусматривают периодическую передачу копий таблицы маршрутизации от одного маршрутизатора другому. Алгоритм аккумулирует сетевые расстояния и поэтому способен поддерживать базу данных информации о топологии сети, но не позволяет маршрутизатору знать точную топологию всего сетевого комплекса.

Каждый маршрутизатор, использующий алгоритм маршрутизации по вектору расстояния, начинает с исследования своих соседей и открывает наилучший путь до сети пункта назначения на основе информации от каждого соседа. При изменении топологии сети таблицы маршрутизации должны быть обновлены шаг за шагом от одного маршрутизатора к другому.

Таким образом этот алгоритм аккумулирует сетевые расстояния и поэтому способен поддерживать базу данных информации о топологии сети. Но алгоритмы на основе вектора расстояния не позволяют маршрутизатору знать точную топологию всего сетевого комплекса.

Алгоритмы маршрутизации с учетом состояния канала связи (известные под названием алгоритмов выбора первого кратчайшего пути) поддерживают сложную базу данных топологической информации и собирают полные данные о дальних маршрутизаторах и о том, как они соединены друг с другом [5].

В этом типе маршрутизации, каждый маршрутизатор рассылает сообщения, когда он впервые становится активным, список маршрутизаторов к которым он непосредственно подсоединен, а также информацию об активности соединений с каждым маршрутизатором. Другие маршрутизаторы использует эту информацию для построения карты топологии сети, а затем использует карту для выбора лучшего пути к точке назначения.

Протоколы маршрутизации по состоянию канала быстро отслеживают изменения в сети, рассылая триггер обновления только при изменениях в сети, и рассылая периодические обновления о состоянии канала через большие промежутки времени.

Когда соединение изменяет состояние, устройство, которое обнаружит изменения, создает объявления о состоянии канала (LSA), и эти LSA распространяются ко всем маршрутизаторам, использующим алгоритм первоочередного открытия кратчайших маршрутов(OSPF). Каждый маршрутизатор OSPF получает копию LSA, обновляя свою базу данных о состоянии каналов, и пересылая LSA всем соседним OSPF маршрутизаторам. Все OSPF маршрутизаторы обновят свои базы данных, до создания обновленной таблицы маршрутизации, которая отражает новую топологию.

База данных о состоянии каналов, используется для вычисления лучшего пути через сеть, применяя алгоритм выбора кратчайшего маршрута (SPF) также называемый алгоритмом Дейкстра, для построения дерева кратчайших маршрутов из базы данных о состоянии каналов. Лучший путь потом выбирается из дерева кратчайших маршрутов и размещается в таблицу маршрутизации.

Алгоритмы маршрутизации различаются по нескольким ключевым характеристикам:

- на работу протокола маршрутизации влияют цели, которые ставились разработчиком алгоритма;
- различные виды алгоритмов по-разному используют ресурсы сети и маршрутизаторов;
- алгоритмы маршрутизации применяют различные метрики, влияющие на выбор оптимальных маршрутов.

При разработке алгоритмов маршрутизации обычно ставится одна или несколько из следующих целей:

- оптимальность, простота и минимальный объем передаваемой служебной информации;
- надежность и устойчивость алгоритма, быстрая сходимость, гибкость.

Под оптимальностью алгоритма маршрутизации понимается его способность выбрать лучший маршрут, что зависит от используемой при вычислениях метрики и удельного веса отдельных параметров.

Кроме того, алгоритмы маршрутизации стараются сделать как можно более простыми (должны эффективно выполнять свои функции с минимальными затратами на передачу служебной информации — как программными, так и аппаратными).

Эффективность алгоритма особенно важна в том случае, когда реализующее его программное обеспечение работает на

компьютере с ограниченными физическими ресурсами.

Алгоритмы должны быть надежными, т.е. должны безошибочно работать в непредвиденных условиях, таких как аппаратные сбои, высокая нагрузка и неправильная установка.

Алгоритмы маршрутизации должны быстро сходиться. Под сходимостью понимается процесс согласования оптимальных маршрутов всеми маршрутизаторами. Если вследствие роста, реконфигурирования или отказа изменяется топология сети, то база знаний о сети должна изменяться тоже. При выходе из строя маршрутизатора или возобновлении его работы, другие маршрутизаторы распространяют по всем сетям сообщения об обновлении маршрутов, вследствие чего происходит повторное вычисление оптимальных маршрутов и согласование их между всеми маршрутизаторами.

Принято считать, что сетевой комплекс сошелся, когда все имеющиеся в нем маршрутизаторы работают с одной и той же информацией. Процесс и время, требующиеся для возобновления сходимости маршрутизаторов, меняются в зависимости от протокола маршрутизации. Если алгоритм маршрутизации медленно сходится, то это может привести к появлению петель маршрутизации или к недоступности части сети.

Алгоритм маршрутизации должен по возможности быть гибким, то есть адаптироваться к изменениям полосы пропускания, длины очереди на маршрутизаторе и сетевым задержкам, а также к другим параметрам.

Все алгоритмы маршрутизации для беспроводных сетей малой дальности можно разделить на две группы: алгоритмы общего назначения и специализированные алгоритмы.

Алгоритмы общего назначения разрабатываются без учета специфики конкретной беспроводной сети датчиков. К этим алгоритмам можно отнести AODV и DSR. Они более или менее работоспособны в любой среде, поэтому они используются в стандартных стеках ZigBee разных производителей. Недостатком алгоритмов общего назначения можно считать неоптимальное использование ресурсов сети.

Специализированные алгоритмы маршрутизации разрабатываются для более оптимального решения задач, стоящих перед сетью датчиков. При разработке этих алгоритмов в расчет берутся такие параметры, как топология сети, гомогенность или гетерогенность аппаратных средств сети, плотность размещения устройств, особенности архитектуры узлов датчиков, характер данных, которые собирает сеть и т.д.

Следует отметить, что приемы и идеи, лежащие в основе специализированных алгоритмов маршрутизации, вызванные трудностями и проблемами, которые возникают во время практической разработки и внедрения сетей датчиков. Специализированные алгоритмы должны работать там, где использование алгоритмов общего назначения неэффективно. Но это одновременно является и недостатком специализированных алгоритмов, поскольку сфера их применения ограничена классом задач, который решает специализированная сенсорная сеть. Поэтому задача сбора данных в беспроводных сетях остается актуальной.

Анализ протоколов маршрутизации в беспроводных сетях

Маршрутизация в беспроводных сетях имеет свои особенности: мобильные устройства должны функционировать в автономном режиме, самостоятельно проводя установление связи с другими узлами сети, тем самым выполняя некоторые функции маршрутизатора, узлы-пользователи могут когда угодно изменять свое местоположение, тем самым постоянно изменяя топологию и общаясь между собой без создания каких-либо определенных стационарных путей передачи данных. Такие сети носят название MANET (mobile ad hoc networks).

Протоколы маршрутизации MANET делятся на следующие группы:

- проактивные;
- реактивные;
-гибридные протоколы;
- протоколы, использующие данные о географическом положении узлов.

Особенность протоколов проактивной группы в том, что узлы сети постоянно собирают и обновляют информацию о ее состоянии, обмениваясь ею с соседями. Проактивные протоколы требуют от узла ведения таблиц маршрутизации, где указаны маршруты, которые позволяют достичь любого абонента сети. Специальные алгоритмы используются для поддержки актуальности этой информации. В связи с этим все изменения в топологии сети распространяются в ней. К проактивным протоколам относятся TBRPF, OLSR, DSDV.

Один из наиболее применяемых проактивных протоколов OLSR основан на сборе и распространении служебной информации о состоянии сети. В результате обработки этой информации каждый узел может построить модель текущего состояния сети в виде формального описания графа, вершины которого ставятся в соответствие узлам сети, рёбра – линиям связи. Имея такой граф, любой узел может вычислить «длины» кратчайших путей до всех адресатов в сети и выбрать «оптимальный»

маршрут, ведущий к любому конкретному узлу сети.

Данный алгоритм хорошо реагирует на множество непредвиденных событий: спонтанные отказы/восстановления узлов и линий, повреждения и ремонт узлов сети, агрессивные воздействия «внешней среды» с блокировкой отдельных элементов системы, подключения и отключения узлов и линий при оперативной передислокации абонентов. Применение ресурса пропускной способности для служебного трафика протокола OLSR наиболее эффективно в сетях с высокой плотностью узлов.

Протокол DSDV основан на идее классического алгоритма маршрутизации Беллмана-Форда с некоторыми улучшениями. DSDV проактивный, дистанционно векторный алгоритм. Каждый узел поддерживает таблицу маршрутизации, в которой перечислены все доступные направления, количество маршрутизаторов («прыжков») до конечного пункта и номер версии. Узлы периодически передают свои таблицы маршрутизации ближайшим соседям. Узел также передает свою таблицу маршрутизации, если в ней произошло изменение с момента последнего отправленного обновления.

Основная задача алгоритма в том, чтобы исключить возможность создания циклических маршрутов. Для минимизации объема трафика, протокол предусматривает обмен полными таблицами маршрутизации только при серьезных изменениях в топологии сети.

Главным недостатком протоколов на базе DSDV является необходимость регулярной передачи служебной информации между узлами для обновления своих таблиц маршрутизации, что в условиях беспроводной сети ведет к увеличению расхода энергии батареи мобильного устройства и занимает часть полосы пропускания радиоканала, даже когда сеть не используется. Кроме этого, каждый раз, когда изменяется топология сети, создается новый порядковый номер для версии маршрутной информации.

При очень динамичных сетях, возможно переполнение данного параметра, т.е. DSDV не подходит для сетей с быстро изменяющейся топологией.

В протоколах реактивной группы узел ищет путь к пункту назначения только при возникновении необходимости. Для этого существуют две основных операции: поиск маршрута и поддержка маршрута. Когда узел намерен установить связь, и начинает устанавливать маршрут, информацию о доступных каналах он получает по запросам. Для поддержки информации о маршрутах узлы должны реагировать на изменения в топологии сети. Узел, у которого есть информация о каком-то канале, должен стремиться детектировать его отказ, если это происходит. Основные реактивные протоколы: DSR, AODV.

Протокол маршрутизации DSR – протокол динамической маршрутизации от источника. Основные механизмы DSR включают определение маршрута и его обслуживание. Эти два механизма работают совместно, чтобы определять и/или поддерживать маршруты в любую точку сети. При первоначальном определении маршрута пакеты отправляются по всем возможным направлениям и в заголовок добавляется информация о пройденном узле. В итоге по достижению цели, заголовок пакета содержит полностью сформированный маршрут между заданными узлами. В случае возникновения петель, т.е. повторного приема первого пакета, узел уничтожает данный пакет.

Одно из основных преимуществ этого протокола – это избавление от необходимости постоянной рассылки сообщений обновления таблицы маршрутизации. установка маршрута происходит по требованию и строится он только к узлу, который нам требуется, поэтому отсутствует необходимость в поиске путей ко всем узлам, что снижает загруженность сети. Для уменьшения времени передачи промежуточными узлами используется информация из кэша.

Из недостатков можно выделить то, что механизм поддержки маршрута не позволяет восстанавливать разорванные соединения. В кэше может находиться старая информация о маршруте, что будет вести за собой ошибки передачи. Также этот протокол имеет большую задержку при установке соединения, в отличие от табличных протоколов. Производительность теряется прямо пропорционально возрастанию подвижности узлов.

Дистанционно-векторный протокол AODV использует другой механизм для актуализации маршрутной информации [6, 10]. Протокол AODV строит таблицы маршрутизации на каждом узле сети для минимизации времени передачи информации между узлами и находит пути маршрутизации независимо от использования маршрутов.

Первым шагом является построение таблиц маршрутизации на каждом узле. В таблице содержится длина кратчайшего пути к каждому узлу в сети через каждый соседний узел. На каждом следующем шаге каждый узел обменивается с соседними узлами информацией о каждом известном ему кратчайшем пути к каждому узлу сети. После некоторого количества шагов, зависящего от количества узлов в сети, таблицы маршрутизации на узлах перестают изменяться, после чего начинается передача данных по кратчайшему найденному пути [6].

Протокол AODV, как и протокол DSR, создает маршруты по необходимости. Тем не менее, AODV принимает традиционные таблицы маршрутизации. Однако используется одна запись на узел назначения, в отличие от DSR, в котором поддерживается несколько записей маршрута для каждого узла назначения. Как и DSDV, AODV предоставляет информацию о нарушении или изменении в сети и предоставляет альтернативные маршруты, но в отличие от DSDV, не требует глобальных периодических объявлений маршрутизации.

Помимо уменьшения количества трансляций в результате разрыва линии связи, AODV также имеет и другие существенные особенности. Каждый раз, когда маршрут от источника к получателю доступен, дополнительные поля заголовка к пакетам не добавляются. Процесс обнаружения маршрута начинается, когда маршруты не используются и/или истекло время жизни. Еще одна отличительная черта AODV заключается в способности обеспечивать однонаправленную, групповую и широковещательную передачу данных.

Гибридные протоколы совмещают в себе механизмы проактивных и реактивных протоколов. Как правило, они разбивают сеть на множество подсетей, внутри которых функционирует проактивный протокол, тогда как взаимодействие между ними осуществляется реактивными методами. В крупных сетях это способствует сокращению размеров таблиц маршрутизации, которые ведут узлы сети, так как им необходимо знать точные маршруты лишь для узлов подсети, к которой они принадлежат.

Также сокращается и объем пересылаемой по сети информации служебного характера, поскольку её основная часть распространяется лишь в пределах подсетей. Один из самых известных гибридных протоколов носит название HwMP [7].

В отдельную группу протоколов маршрутизации MANET выделяются протоколы, использующие данные о местоположении абонентов сети (протоколы геомаршрутизации). Их основные преимущества заключаются в отсутствии необходимости в хранении маршрутной информации на транзитных узлах, возможностях оптимизировать маршруты, исходя из имеющейся информации о местоположении узлов.

Что касается протоколов маршрутизации MANET, то в данном случае, они должны, по возможности, свести к минимуму время, затраченное на построение маршрута, и время задержки доставки пакетов. А также максимизировать коэффициент доставки пакетов, рассылая как можно меньше служебной информации. При этом они должны успешно справляться с увеличением нагрузки при добавлении узлов.

Обеспечение данных условий требует от протоколов геомаршрутизации использования различных стратегий поиска маршрутов. Например, протокол геомаршрутизации GAF формирует виртуальную сетку покрытой области, в которой каждый узел соотносит себя с ближайшим пунктом на виртуальной сетке [8]. Узлы, связанные с конкретным пунктом на сетке, считаются равнозначными с точки зрения стоимости маршрутизации. Данный подход позволяет увеличивать время жизни сети при увеличении числа узлов. При этом узлы могут изменять свое состояние и переходить от спящего к активному, чтобы балансировать нагрузки.

Существуют три состояния, в которых могут находиться узлы:

- «состояние обнаружения», когда существует возможность определения присутствия соседей в сетке,
- «активное состояние», при котором предполагается участие в маршрутизации,
- «спящий режим», продолжительность которого имеет прямую зависимость от приложения.

Каждый узел в сетке оценивает свое время выхода из неё и посылает данную информацию своим соседям. Спящие соседние узлы корректируют своё время сна, чтобы сохранить актуальность маршрутной информации. Прежде, чем наступит время, установленное для выхода узла из активного режима, один из соседних спящих узлов «просыпается» и переходит в активное состояние. Таким образом, протокол GAF всегда сохраняет сеть связанной, поддерживая один из узлов в активном состоянии для каждой области на виртуальной сетке.

Геопротокол GPSR — использует информацию о расположении узла для определения маршрута при пересылке пакетов. Пересылка осуществляется на основе «жадной» стратегии. Процесс ретрансляции пакетов промежуточными узлами продолжается до достижения пункта назначения.

В некоторых случаях данная стратегия может привести к ошибкам. Чтобы их исключить применяется «правило правой руки»: текущий узел, в случае отсутствия такового соседнего, более близкого к узлу-приемнику, передает пакеты первому узлу, передвигаясь против часовой стрелки. При увеличении подвижности узлов сети интервал пересылаемых служебных пакетов с геоинформацией, позволяющий держать таблицы маршрутизации в актуальном состоянии, должен быть уменьшен. Однако в действительности подобное приводит к большим

накладным расходам, и чтобы их уменьшить, информация о местоположении узла должна отправляться вместе с пакетами данных.

Один из наиболее используемых протоколов географической маршрутизации — LAR [9]. Данный протокол использует информацию о местоположении узла-источника для ограничения области (зоны запроса), где производится поиск маршрута. В итоге количество сообщений о запросе искомого маршрута сокращается.

Также были предложены другие протоколы для пакетных радиосетей, в которых были сделаны попытки соединить преимущества и избавиться от недостатков каждой из групп [10]. Примером является протокол BVR, который использует технологии «жадного продвижения пакетов» (greedy forwarding) и построения системы логических координат, унаследованные от предыдущих протоколов. Его особенностью является создание ряда «маяков» (beacons), случайно выбранных узлов, которые играют роль синхронизаторов в сети.

На их основе строится «дерево» сетевой структуры, определяются показатели маршрутов и осуществляется построение путей к пунктам назначения: поиск ближайшего соседа, назначение его как следующего элемента маршрута и переход до его ближайшего соседа (реализация алгоритму «жадного продвижения»). Отличием BVR является применение при этом не географических, а логических координат. Главное назначение протокола - поддержка соединениий «точка-точка» (point-to-point). Позднее на его основе был разработан протокол LCR.

Выводы

Выполненный анализ алгоритмов и протоколов маршрутизации данных показал не только их значительное количество и разнообразие, но и постоянное развитие. Существующие беспроводные решения совершенствуются и подстраиваются под нужды разработок нового поколения.

В некоторых случаях, напротив, появление принципиально новых перспективных технологических направлений диктует необходимость в разработке необычных коммуникаций со специфическими требованиями к мощности, экономии энергии, программному управлению, большой автономии и надежности [11]. Специалисты исследовательской компании Gartner проанализировали ключевые тенденции и выделили десятку беспроводных технологий, которые займут доминирующее положение на рынке в качестве корпоративных архитектур нового поколения.

Литература

1. Мальчева, Р. В. Анализ проблем сбора данных в беспроводной локальной сети / Р. В. Мальчева, Я. О. Кравченко // Информационное пространство Донбасса: проблемы и перспективы : материалы IV Респ. с междунар. участием науч.-практ. конф., 28 окт. 2019 г. – Донецк : ГО ВПО «ДонНУЭТ», 2021. - С. 154-156.

2. Коптев, Д. С. Сравнительный анализ наиболее перспективных стандартов беспроводных сетей связи [Электронный ресурс]/ Д. С. Коптев, А. Н. Щитов, А. Н. Шевцов // Информатика, 2016. - Режим доступа: https://cyberleninka.ru/article/n/sravnitelnyy-analiz-naibolee-perspektivnyh-standartov-besprovodnyh-setey-svyazi

3. Димитров, Г. Л. Тенденции развития беспроводных средств коммуникаций [Электронный ресурс] / Г. Л. Димитров. – Режим доступа: https://cyberleninka.ru/article/ n/tendentsii-razvitiya-besprovodnyh-sredstv-kommunikatsiy

4. Винокуров, В. М. Маршрутизация в беспроводных мобильных Ad hoc-сетях [Электронный ресурс] / В. М. Винокуров, А. В. Пуговкин, А. А. Пшенников, Д. Н. Ушарова, А. С. Филатов // Доклады ТУСУРа, 2010. - № 2 (22). - Часть 1. - С. 288-292. - Режим доступа: https://cyberleninka.ru/article/n/marshrutizatsiya-v-besprovodnyh-mobilnyh-ad-hoc-setyah/viewer

5. Золотарев, С. П. Алгоритмы маршрутизации состояния связей [Электронный ресурс] / С. П. Золотарев. - Режим доступа: https://cyberleninka.ru/article/n/algoritmy-marshrutizatsii-sostoyaniya-svyazey

6. Карманов, М. Л. Протокол маршрутизации для ad-hoc сетей [Электронный ресурс] / М. Л. Карманов // Вестник ЮурГУ, 2009. - № 26. – С. 47-51. - Режим доступа: https://dspace.susu.ru/bitstream/handle/0001.74/768/10.pdf

7. Альбекова, З. М. Анализ эволюции технологии беспроводных сетей и прогнозы развития инфокоммуникационных сетей в России [Электронный ресурс] / З. М. Альбекова, В. О. Квашурин, Н. А. Тутик // Инженерный вестник Дона, 2016. - Режим доступа: https://cyberleninka.ru/article/n/analiz-evolyutsii-tehnologii-besprovodnyh-setey-i-prognozy-razvitiya-infokommunikatsionnyh-setey-v-rossii

8. Бершадский, А. М. Обзор методов маршрутизации в беспроводных сенсорных сетях [Электронный ресурс] / А. М. Бершадский, Л. С. Курилов, А. Г. Финогеев // Известия высших учебных заведений. Поволжский регион. Технические науки, 2012. – № 1 (21). – С. 47–57. - Режим доступа: https://cyberleninka.ru/article/n/obzor-metodov-marshrutizatsii-v-besprovodnyh-sensornyh-setyah

9. Метелёв, А. П. Протоколы маршрутизации в беспроводных самоорганизующихся сетях [Электронный ресурс] / А. П. Метелёв, А. В. Чистяков, А. Н. Жолобов // Вестник ННГУ, 2013. - № 3 (1). - С. 75-78. - Режим доступа: https:// cyberleninka.ru/article/n/protokoly-marshrutizatsii-v-besprovodnyh-samoorganizuyuschihsya-setyah

10. Шипицин, С. П. Алгоритм адаптивной настройки параметров протокола маршрутизации AODV [Электронный ресурс] / С. П. Шипицин // Инженерный вестник Дона, 2018. - №2. - Режим доступа: https://cyberleninka.ru/article/n/algoritm-adaptivnoy-nastroyki-parametrov-protokola-marshrutizatsii-aodv/viewer

11. Malcheva, R. Development of the data transferring system using SoC / R. Malcheva, H. Naaem // European Scientific Journal, 2014. - Т. 10. - № 7. - С. 191-195.

Мальчева Р. В., Кравченко Я. О. Анализ алгоритмов и протоколов маршрутизации данных в беспроводных локальных сетях. *В данной статье проанализированы различные алгоритмы и протоколы маршрутизации сбора данных в беспроводных локальных сетях, их преимущества и недостатки. Также описаны ключевые тенденции и выделили десятку беспроводных технологий, которые займут доминирующее положение на рынке в качестве корпоративных архитектур нового поколения. Целью работы является обоснование и выбор наиболее эффективной организации передачи данных при проектировании конкретной беспроводной сети, параметры которой должны соответствовать требованиям технического задания.*

Ключевые слова: *беспроводные сенсорные сети, маршрутизация, алгоритм маршрутизации, протокол маршрутизации, MANET, управление передачей данных.*

Malcheva R.V., Kravchenko Y.O. Analysis of data routing algorithms and protocols in wireless local area networks. *This article analyzes various algorithms and routing protocols for data collection in wireless local area networks, their advantages and disadvantages. The key trends are also described and the top ten wireless technologies that will occupy a dominant position in the market as a new generation of enterprise architectures are highlighted. The purpose of the work is to substantiate and select the most effective organization of data transmission when designing a specific wireless network, the parameters of which must meet the requirements of the terms of reference.*

Key words: *wireless sensor networks, routing, routing algorithm, routing protocol, MANET, data transmission control.*

Статья поступила в редакцию 28.10.2021
Рекомендована к публикации профессором Аноприенко А.Я.

Об авторах

Аноприенко Александр Яковлевич – кандидат технических наук, доцент, профессор кафедры и заведующий кафедрой компьютерной инженерии факультета интеллектуальных систем и программирования, директор института компьютерных наук и технологий, ректор Донецкого национального технического университета.

Бельков Дмитрий Валерьевич – кандидат технических наук, доцент, доцент кафедры прикладной математики факультета интеллектуальных систем и программирования Донецкого национального технического университета.

Дорожко Леонид Иванович – кандидат технических наук, доцент кафедры компьютерной инженерии факультета интеллектуальных систем и программирования Донецкого национального технического университета.

Зензеров Владимир Иванович - кандидат технических наук, доцент, доцент кафедры прикладной математики факультета интеллектуальных систем и программирования Донецкого национального технического университета.

Едемская Евгения Николаевна – старший преподаватель кафедры искусственного интеллекта и системного анализа факультета интеллектуальных систем и программирования Донецкого национального технического университета.

Иваница Сергей Васильевич - кандидат технических наук, доцент кафедры компьютерной инженерии факультета интеллектуальных систем и программирования Донецкого национального технического университета.

Кравченко Ярослав Олегович - магистрант кафедры компьютерной инженерии факультета интеллектуальных систем и программирования Донецкого национального технического университета.

Луценко Дмитрий Юрьевич - Санкт-Петербургский политехнический университет Петра Великого (СПбПУ)

Максименко Наталья Сергеевна - ассистент кафедры компьютерной инженерии факультета интеллектуальных систем и программирования Донецкого национального технического университета.

Мальчева Раиса Викторовна - кандидат технических наук, доцент, профессор кафедры компьютерной инженерии факультета интеллектуальных систем и программирования Донецкого национального технического университета.

Мамутова Влада – выпускницы бакалавриата кафедры программной инженерии факультета интеллектуальных систем и программирования Донецкого национального технического университета; магистрант кафедры «Программное обеспечение ЭВМ и ИТ» МГТУ им. Н. Э. Баумана по специальности «Программная инженерия».

Мащенко Елена Николаевна - кандидат технических наук, доцент, доцент кафедры «Информационные технологии и компьютерные системы» Севастопольского государственного университета (г. Севастополь, РФ).

Мишустин Виктор Андреевич - магистрант кафедры компьютерной инженерии факультета интеллектуальных систем и программирования Донецкого национального технического университета.

Приходченко Екатерина Ильинична – доктор педагогических наук, профессор, профессор кафедры социологии и политологии социально-гуманитарного института Донецкого национального технического университета, заслуженный учитель Украины, академик Международной академии наук педагогического образования.

Сидоров Константин Андреевич – ассистент кафедры компьютерной инженерии факультета интеллектуальных систем и программирования Донецкого национального технического университета.

Ченгарь Ольга Васильевна – кандидат технических наук, доцент, доцент базовой кафедры «Корпоративные информационные системы» Севастопольского государственного университета (г. Севастополь, РФ).

Чернышова Алла Викторовна – старший преподаватель кафедры программной инженерии факультета интеллектуальных систем и программирования Донецкого национального технического университета.

Шевченко Виктория Игоревна - кандидат технических наук, доцент, заведующая базовой кафедрой «Корпоративные информационные системы» Севастопольского государственного университета (г. Севастополь, РФ).

**Требования к статьям,
направляемым в редакцию научного журнала
«Информатика и кибернетика»**

Редколлегией принимаются к рассмотрению статьи, в которых рассматриваются важные вопросы в области информатики и кибернетики. Научный журнал издаётся с 2015 года, периодичность издания – 4 раза в год.

В журнале предусмотрены следующие рубрики:

- информатика и вычислительная техника;
- компьютерные и информационные науки;
- инженерное образование.

В соответствии с номенклатурой специальностей научных работников МОН ДНР первые две рубрики соответствуют следующим укрупненным группам специальностей научных работников:

05.01 – «Инженерная геометрия и компьютерная графика»,

05.13 – «Информатика, вычислительная техника и управление».

С 01.02.2019 Научный журнал включён в Перечень рецензируемых научных изданий, в которых должны быть опубликованы основные научные результаты диссертаций на соискание учёной степени кандидата наук, на соискание учёной степени доктора наук (приказ МОН ДНР № 135) по группам специальностей 05.01.00 и 05.13.00.

Рубрика «Инженерное образование» предназначена опубликования сотрудниками научно-методических статей.

Журнал также включён в базу данных РИНЦ (Российский индекс научного цитирования) (лицензионный договор № 425-07/2016 от 14.07.2016).

Статьи, представляемые в данный сборник, должны отвечать следующим требованиям. ***Содержание статьи*** должно быть посвящено актуальным научным проблемам и включать следующие необходимые элементы:

- постановку проблемы в общем виде, её связь с важными научными и практическими задачами;
- анализ последних исследований и публикаций, в которых решается данная задача и на которые опирается автор, выделение нерешенных ранее частей общей проблемы, которым посвящается статья;
- формулировка цели статьи и постановка задач, решаемых в ней;
- изложение основного материала с полным обоснованием полученных научных результатов;
- выводы и перспективы последующих исследований в данном направлении.

Каждый элемент должен быть выделен соответствующим названием раздела, например, «введение», «постановка задачи», «цель и задачи работы», «цель статьи», «цель исследования», «цель разработки», «анализ … », «сравнительная оценка … », «разработка … », «проектирование … », «программная реализация», «тестирование … », «полученные результаты», «выводы», «литература». Разделы «введение», «выводы», «литература» являются обязательными. Включать в названия разделов нумерацию не разрешается.

В основном тексте статьи формулируются и обосновываются полученные авторами утверждения и результаты. Выводы должны полностью соответствовать содержанию основного текста. Языки публикаций: русский, английский.

Объём статьи, формат страницы

Для оформления статьи следует использовать листы формата А4 (210х297 мм) с полями по 2,5 см со всех сторон. Нумерацию страниц выполнять не нужно.

Рекомендуемый объём статьи – 6-12 страниц. Рукописи меньшего объёма могут быть рекомендованы к публикации в качестве коротких сообщений.

Последняя страница текста статьи должна быть заполнена не менее чем на две трети, но содержать не менее трёх пустых строк в конце.

Форматирование текста

Подготовка статьи осуществляется в текстовом редакторе Microsoft Office Word.

Весь текст статьи оформляется шрифтом Times New Roman 10 пт с одинарным междустрочным интервалом, если ниже в требованиях не сказано иного. Абзацный интервал «перед» – 0 пт, «после» – 0 пт.

На первой строке с выравниванием по левому краю располагается УДК.

Заголовок (название) статьи оформляется шрифтом Times New Roman 14 пт, полужирное начертание, с выравниванием по центру (без абзацных отступов). Заголовок статьи следует печатать с прописной буквы без точки в конце, переносы слов не допускаются. Абзацный интервал «перед» – 12 пт, «после» – 12 пт.

После названия статьи следует информация об авторах, которая выравнивается по центру (без абзацных отступов). На одной строке указываются инициалы и фамилии всех авторов через запятую. Между двумя инициалами ставится пробел. С новой строки указывается название вуза (организации) и город (для каждого автора, если не совпадают). На следующей строке указываются адреса электронной почты (один адрес либо каждого автора – по желанию). Адрес электронной почты оформляется в виде гиперссылки.

К тексту аннотации применяется курсивное начертание, с выравниванием по ширине, отступы слева и справа по 1 см. Заголовок «Аннотация» выделяется полужирным начертанием. Объём аннотации – 450-550 символов (без пробелов). Абзацный интервал «перед» – 12 пт, «после» – 12 пт.

Основной текст статьи разбивается на две колонки шириной по 7,5 см (промежуток между столбцами – 0,99 см), выравнивается по ширине. Абзацный отступ первой строки – 1 см. Автоматический перенос слов не применяется.

Заголовки разделов выполняются шрифтом Arial 10 пт, полужирное курсивное начертание. Абзацный отступ отсутствует, интервал перед абзацем – 12 пт, после абзаца – 6 пт. Для заголовка «Введение» установить интервал «перед» – 0 пт, «после» – 6 пт.

Таблицы в тексте статьи

Название следует помещать над таблицей с абзацного отступа (1 см) в формате: слово «Таблица», пробел, номер таблицы, пробел, тире, пробел, название таблицы. Название таблицы записывают с прописной буквы без точки в конце строки и выравнивают по ширине. В ячейках таблицы устанавливается выравнивание текста по центру по вертикали. По горизонтали текст выравнивается по центру либо по левому краю. Границы ячеек таблицы должны быть только чёрного цвета, толщина линии – 1 пт. На все таблицы должны быть приведены ссылки в тексте статьи, при ссылке следует писать слово «табл.» с указанием её номера, например, « … данные приведены в табл. 5». Таблицы нумеруются в пределах статьи. Таблица располагается сразу после ссылки на неё, если это возможно (например, после окончания абзаца). Если же таблица не помещается на текущей странице, то она должна быть расположена в начале следующей страницы (или колонки). При необходимости допускается включение в статью таблицы, ширина которой превышает ширину колонки. В этом случае таблица и её название размещаются по центру страницы. Таблица не должна выступать за границы полей страницы. Таблица и её название отделяются от основного текста статьи одной пустой строкой до и после.

Рисунки в статье

Ссылки на иллюстрации по тексту статьи обязательны и оформляются в виде « ... на рис. 2» и т. п. Рисунок и его подпись выравниваются по центру колонки (без абзацных отступов), положение рисунка – «в тексте». Размещается рисунок после его первого упоминания в тексте, если это возможно (например, после окончания абзаца). Если же иллюстрация не помещается на текущей странице, то она должна быть расположена в начале следующей страницы (или колонки). При необходимости допускается включение в статью рисунка, ширина которого превышает ширину колонки. В этом случае рисунок и его подпись выравниваются по центру страницы. Иллюстрация не должна выступать за границы полей страницы. Подпись рисунка оформляется в формате: слово «Рисунок», пробел, номер иллюстрации, пробел, тире, пробел, название рисунка. Название рисунка записывают с прописной буквы без точки в конце строки. Для подписи иллюстрации применяют курсивное

начертание. Иллюстрация и её подпись отделяются от основного текста статьи одной пустой строкой до и после. Не допускается выполнять рисунки с помощью встроенного графического редактора Microsoft Office Word. Если на иллюстрации имеется текст, размер шрифта должен быть не менее чем аналогичный текст, набранный шрифтом Times New Roman 10-го размера. Иллюстрация не должна содержать много незаполненного пространства.

Формулы

Формулы и уравнения рекомендуется набирать с использованием MathType (предпочтительно) или MS Equation. Формулы и математические символы не должны существенно отличаться по размеру от основного текста. Обязательной является нумерация формул, на которые имеется ссылка в тексте статьи. Ссылки в тексте на порядковые номера формул дают в скобках, например, « ... согласно формуле (2)». Формулы размещаются по центру колонки, а их номера – по правому краю. Как для строки с формулой, так и для первой строки пояснений (при наличии), абзацный отступ убирается. Первая строка пояснения начинается со слова «где», после которого следует поставить табуляцию на 1 см, затем само пояснение в формате: символ, подлежащий объяснению, пробел, тире, пробел, поясняющий текст, запятая, обозначение единицы измерения физической величины. Пояснения перечисляются через точку с запятой, выравниваются по ширине. Вторая и последующие строки пояснений начинаются с абзацного отступа (1 см). Весь блок текста, связанный с формулой (только формула, несколько формул подряд или формула с пояснениями), отделяется от основного текста одной пустой строкой до и после. Переносить формулы на следующую строку допускается только на знаках выполняемых операций, причем знак в начале следующей строки повторяют. При переносе формулы на знаке умножения применяют знак «×». Формулы и математические уравнений могут быть записаны в тексте документа, если их высота не превышает высоту строки. При этом следует учитывать, что знаки математических операций отделяются от чисел или символов пробелами с обеих сторон. Например, «Если учесть, что $y < 0$ и $2x + y = 1$, то из формулы (3) можно выразить x…». К символам, которые приведены в формуле, при дальнейшем их употреблении (в том числе в пояснениях к формуле) должно применяться курсивное начертание. При этом к любым числам (верхние и нижние индексы, содержащие цифры и т. п.), а также к математическим знакам курсивное начертание не применяется. Не допускается вставлять формулы, выполненные в виде рисунков.

Перечисления: оформление списков

Основной текст статьи может содержать перечисления, оформленные в виде маркированного списка. В качестве маркера элемента списка разрешается использовать только короткое тире «–». Каждый элемент перечисления записывается с новой строки с абзацного отступа, равного 1 см. После символа короткого тире текст располагается с отступом в 1,5 см от левой границы строки, выравнивается по ширине, при переносе на новые строки располагается без отступов. Нумерованные и многоуровневые списки включать в статью не разрешается.

Литература

В тексте статьи обязательны ссылки на все литературные источники, номер источника указывается в квадратных скобках. Ссылки на неопубликованные работы не допускаются. Рекомендуемое количество источников, на которые ссылается автор, не менее 10. Перечень источников приводится в порядке их упоминания в статье. Библиографическое описание каждого литературного источника оформляется в соответствии с ГОСТ Р 7.0.100–2018. Перечень литературных источников оформляется в виде нумерованного списка. В качестве маркеров элементов списка используют порядковые арабские цифры с точкой. Каждый источник представляет собой отдельный элемент перечисления, записывается с новой строки с абзацного отступа, равного 1 см. После порядкового номера с точкой текст располагается с отступом в 1,5 см от левой границы строки, выравнивается по ширине, при переносе на новые строки располагается без отступов.

В конце статьи обязательно приводятся аннотации на русском и английском языках, каждая заканчивается перечнем 5-6 ключевых слов.

К тексту аннотации применяется курсивное начертание, с выравниванием по ширине, отступы слева и справа по 1 см. Слово «Аннотация» опускается. Текст аннотации начинается с ФИО авторов и названия статьи, выделяемых полужирным начертанием. Аннотация на русском языке совпадает с аннотацией, приведенной в начале статьи. В тексте аннотации на английском языке после фамилии автора указывается только первая буква имени с точкой. Абзацный интервал «перед» – 12 пт, «после» – 12 пт. Ключевые слова оформляются с новой строки аналогично тексту аннотации. Заголовок «Ключевые слова:» (англ. «Keywords:») выделяется полужирным начертанием. Ключевые слова перечисляются через запятую.

Порядок представления статьи и сопроводительные документы

В редакцию необходимо представить:

- файл с текстом статьи;
- файл, содержащий фамилию, имя и отчество авторов полностью; ученую степень, ученое звание; место работы с полным указанием должности, подразделения и наименования организации, города (страны); номера телефонов и e-mail для связи;
- экспертное заключение о возможности публикации статьи, подписанное руководителем и заверенное печатью организации, в которой работает автор статьи;
- выписка из заседания кафедры или письмо организации с просьбой об опубликовании и указанием, что изложенные в статье результаты ранее не публиковались.

Статьи и сопроводительные документы следует высылать на электронный адрес infcyb.donntu@yandex.ru.

К сведению авторов

Если статья оформлена с нарушением указанных выше требований и правил, редакция после предварительного рассмотрения может отклонить статью.

На рецензирование статьи направляются членам редакционной коллегии журнала. Все статьи публикуются при наличии положительной рецензии.

В статью могут быть внесены изменения редакционного характера без согласования с автором. Ответственность за содержание статьи и качество перевода аннотаций несут авторы.

Публикация статей в научном журнале «Информатика и кибернетика» осуществляется на некоммерческой основе.

Все номера Научного журнала размещаются на сайте http://infcyb.donntu.org/.

.

CONTENT

Informatics and computer engineering

Электронное периодическое издание

Научный журнал

ИНФОРМАТИКА И КИБЕРНЕТИКА

(на русском, английском языках)

№ 3 (25) - 2021

Ответственный за выпуск Р. В. Мальчева

Технический редактор Р. В. Мальчева

Компьютерная верстка Р. В. Мальчева

Подписано к выпуску 05.11.2021. Усл. печ. лист. 7,6. Уч.-изд. лист.5,0.
Адрес редакции: ДНР, 83001, г. Донецк, ул. Артема, 58, ГОУ ВПО «ДонНТУ»,
4-й учебный корпус, к. 36., ул. Кобозева, 17.
Тел.: +38 (062) 301-07-35, +38 (071) 334-89-11
E-mail: infcyb.donntu@yandex.ru, URL: http://infcyb.donntu.org