ГОСУДАРСТВЕННЫЙ КОМИТЕТ СВЯЗИ, ИНФОРМАТИЗАЦИИ И ТЕЛЕКОММУНИКАЦИОННЫХ ТЕХНОЛОГИЙ РЕСПУБЛИКИ УЗБЕКИСТАН

ТАШКЕНТСКИЙ УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

На правах рукописи
УДК

Болтаев Рустам Жахонгирович

РАЗРАБОТКА МОДЕЛЕЙ ПОВЫШЕНИЯ ПРОИЗВОДИТЕЛЬНОСТИ КОМПЬЮТЕРНЫХ СЕТЕЙ НА ОСНОВЕ СТОХАСТИЧЕСКИХ МЕТОДОВ

Специальность «5А330201 – Компьютерные системы и их программное обеспечение»

Диссертация
на соискание академической степени магистра

Научный
руководитель
Бабамухамедова М.З.

Ташкент 2013

СОДЕРЖАНИЕ

АННОТАЦИЯ МАГИСТЕРСКОЙ ДИССЕРТАЦИИ

В данной магистерской работе рассматривается тема «Разработка алгоритма и программного обеспечения для установления системных связей на основе интервального анализа». При решении многих прикладных задач приходится сталкиваться с неопределенностью в исходных данных. Учет таких неопределенностей в практических задачах и ее внедрение виде автоматизации какого-либо процесса является актуальной проблемой сегодняшних дней.

Объект исследования – системы и процессы установления системных связей. Предмет исследования – условия неполноты и недостоверности исходных данных в показателях процессов и систем установления системных связей.

Диссертация состоит из введения, трех глав, заключения, списка литературы. В ходе выполнения диссертации разработаны интервальные аналоги решения задач линейного программирования симплекс методом, алгоритмы реализации машинной интервальной арифметики для компьютера, разработаны алгоритм и программное обеспечение для установления системных связей на основе интервального анализа.

МАГИСТРЛИК ДИССЕРТАЦИЯСИ АННОТАЦИЯСИ

Мазкур диссертация ишида “интервал тахлил асосида алоқа тизимларини ўрнатиш учун алгоритм ва дастурий таъминот ишлаб чиқиш” мавзусига бағишланган. Кўп амалий масалаларни ечиш жараенида биз бошланғич маълумотларни ноаниқлилиги билан келганига дуч келамиз. Шундай ноаниқлиликларни амалий масаларда амалда ишлатиш ва тадбиқ қилиш хозирги кунга келиб долзарб хисобланади.

Тадқиқот объекти – алоқа тизимларни ўрнатиш тизим ва жараенлари. Тадқиқот предмети эса алоқа тизимларни ўрнатиш жараенларининг бошланғич маълумотларнинг ноаниқлилиги.

Мазкур диссертация кириш, уч бобдан, хулоса ва адабиётлар рўйхатидан иборат. Диссертацияни бажариш давомида чизиқли дастурлаш симплекс усули масалаларни ечиш интервал ўхшаш усул яратилган, компьютерлар учун машина интервал арифметикасини бажариш алгоритмлари, интервал таҳлил асосида алоқа тизимларини ўрнатиш компьютер тизимларини ҳимоялаш учун алгоритм ва дастурий таъминот ишлаб чиқилган.

MASTER'S DISSERTATION ANNOTATION

This master's thesis deals with the topic "Development of algorithms and software system for establishing relations based on interval analysis." When dealing with many applications have to deal with the uncertainty in the source data. Accounting for these uncertainties in practical problems and its implementation as the automation of a process is an urgent problem today.

The object of study - systems and processes of establishing systemic linkages. The subject of the study - the conditions of incompleteness and unreliability of the source data in terms of processes and systems to establish systemic connections.

The thesis consists of an introduction, three chapters, conclusion, bibliography. During the interval of the thesis developed counterparts for solving linear programming simplex method, the algorithms of the machine interval arithmetic for the computer algorithm and software system for establishing relations based on interval analysis.

ВВЕДЕНИЕ

Развитие современных информационных и коммуникационных технологий (ИКТ) имеет целенаправленную тенденцию к интенсификации и диверсификации, охватывая все новые отрасли экономики, в том числе и области управления государством.

Развитие информационно-коммуникационных технологий (ИКТ), являющееся важнейшим фактором поднятия благосостояния и экономического роста, становится одним из основных приоритетов государственной политики Узбекистана.

Инициатива Президента послужила сигналом к крупным стратегическим изменениям. Правительство сейчас четко осознает важность ИКТ для достижения своих целей развития. Поэтому, в последние годы руководство республики принимает энергичные меры по развитию и широкому внедрению ИКТ в различные сферы общественного и государственного строительства.

В связи с мировым финансовым кризисом совсем недавно принят и сегодня реализуется Указ Президента Республики Узбекистан Ислама Каримова по оказанию дополнительной помощи банковским и финансовым структурам, поддержке деловой активности предприятий и компаний реального сектора экономики, повышению рентабельности производства и экспортных возможностей, выделению им в этих целях дополнительных налоговых льгот и преференций и реализации наряду с этим других крупномасштабных мер и проектов.

Актуальность темы. Диссертационная работа посвящена актуальной проблеме разработки методологических вопросов оптимизации вычислительных сетей, а именно разработки математических методов и алгоритмов повышения производительности основных составляющих компьютерных сетей.

В настоящее время в связи с интеграцией корпоративных сетей передачи данных все более остро встает проблема управления распределенными гетерогенными сетями, состоящими из множества локальных сетей, функционирующих на основе различных стандартов и протоколов.

Объект исследования. Объектом исследования диссертационной работы является компьютерные сети, маршрутизаторыи серверы.

Методом исследования являются методы и способы повышение производительности компьтерных сетей на основе стохастических методов.

Цели и задачи работы. Основная на сегодняшний день цель — создание системы интегрированного сетевого управления - требует решения целого ряда задач.

В их число входят:

- традиционные задачи сетевого управления (управление конфигурацией, управление производительностью, управление сбоями, управление безопасностью, учет использования ресурсов);
- управление распределенными приложениями в гетерогенных сетях;
- мониторинг текущего состояния системно-технического обеспечения организации (ведение визуализированной базы данных, содержащей полную информацию, как о технических, так и об учетных параметрах всего технического и программного обеспечения, имеющегося в той или иной организации);
- поддержка принятия решений по модернизации технического и программного обеспечения с учетом текущего состояния технического прогресса, информации о производителях и поставщиках технических и программных средств и о сравнительных характеристиках этих продуктов;
- управление модернизацией (контроль и управление установкой нового технического и программного обеспечения, включая оптимизацию этого процесса);

- моделирование работы существующих сетей (включая анализ нагрузок на отдельные их участки и поддержку принятия решений по перепланированию).

Краткий анализ литературы по теме диссертации. Основные вопросы в области компьютерных систем и сетей освещаются в работах ученых М. Федотова, Н.А. Олифера, В.Кумара, Н.Лагари, А.Мунгале, Т.Паркера. В результате предлагается множество методов и алгоритмов повышения производительности компьютерных сетей, предложены множество решений проблем связанные с производительностью коммуникационных сетей.

Научная новизна работы. Научная новизна работы заключается в применении стохастических методов для повышения эффективности компьютерных сетей.

Разработонное программное обеспечение выделяет основные показатели повышения производительности компьютерных сетей, такие как:

- Интенсивность поступления заявок на обработку;
- Интенсивность работы серверов (циклических, параллельных и т.д.);
- Размер буферного пространства серверного оборудования;
- Среднее время ожидания заявки в буфере;
- Среднее время обработки заявки…

и алгоритмы реализации;

Практическая значимость. Разработана структура программного комплекса повышения производительности компьютерных сетей на основе стохастических моделей, подробно описана его архитектура, а также приведены доказательства, на основании которых был сделан выбор программных средств для реализации данного программного комплекса.

Следует заметить, что ни один из имеющихся на сегодняшний день на рынке программного обеспечения продуктов не решает целиком ни одной из перечисленных задач. Поэтому наиболее целесообразным решением в данном случае является либо разработка такой интегрированной системы

самостоятельно, либо заказ на ее разработку фирме — системному интегратору.

В настоящее время под собственно сетевым управлением обычно подразумевают совокупность пяти взаимосвязанных задач, в число которых входят:

1. Управление конфигурацией (Configuration Management)
2. Управление безопасностью (Security Management)
3. Управление сбоями (Fault & Problem Management)
4. Учет использования ресурсов (Accounting Management)
5. Управление производительностью (Performance Management)

Подробнее рассматривая данный пункт, необходимо отметить, что он подразумевает под собой оценку состояния ресурсов и эффективности их использования и включает в себя такие важные этапы, как:

- сбор и анализ статистических данных о функционировании сети;
- анализ трафика;
- планирование и оценку эффективности использования ресурсов сети;
- выявление узких мест сети;
- анализ сетевых протоколов;
- планирование развития сети.

Параллельно с этим необходимо учитывать особенности построения самой сети, наличия в структуре различных топологий и их взаимного соединения.

Всякий раз, когда должна быть построена новая система, или реконструирована уже имеющаяся, анализ таких сетей может использоваться для предсказывания влияния архитектурных или других изменений на характер выполнения работ и протекания процессов в коммуникационных сетях.

В зависимости от модели, можно непосредственно вычислять важные параметры производительности выполнения работ коммуникационных сетей, используя так называемые неявные формулы. Эти аналитические модели конечно очень удобны, но большинство реальных систем не могут быть смоделированы таким способом.

Для несколько более широкого класса моделей используются способы получения систем уравнений, решение которых может быть получено в цифровой форме на основе рекурсивных вычислений. Хотя эти числовые модели не дают явные формулы, все еще можно получить точные результаты, конечно в пределах допустимой ошибки компьютера, который используется для числовых вычислений.

Для самого широкого класса систем для получения образцовых решений числовые и аналитические методы не существуют. В этих случаях обращаются к имитационному моделированию. В имитационных моделях, можно однозначно представить возможное поведение системы. При этом используется моделирование важных параметров системы во временном разрезе, чтобы вычислить средние параметры поведения моделирования коммуникационных сетей.

Структура работы. Диссертация состоит из введения 3 глав, заключения, списка использованной литературы и приложения.

ГЛАВА 1. АНАЛИЗ ПРОГРАММНЫХ СРЕДСТВ ПОВЫШЕНИЯ ПРОИЗВОДИТЕЛЬНОСТИ КОМПЬЮТЕРНЫХ СЕТЕЙ

1. Анализ методов и программных средств повышения производительности коммуникационных сетей

Каждый раз, когда выполняется построение новых систем или проводится реконструкция уже имеющихся, предварительный анализ таких сетей может использоваться для предсказывания влияния архитектурных или других изменений на характер выполнения работ и протекания процессов в коммуникационных сетях.

Основой анализа систем является количественная оценка параметров коммуникационных сетей. Не менее важным также, является наблюдение предпосылок выполнения работ и пригодность (готовность) фактической системы для этой работы.

Одним из средств оценки параметров коммуникационных сетей являются анализаторы. Анализаторы незаменимы для исследования реальных сетей, но они не позволяют получать количественные оценки характеристик для еще не существующих сетей, находящихся в стадии проектирования. В этих случаях проектировщики могут использовать средства моделирования, с помощью которых разрабатываются модели, воссоздающие информационные процессы, протекающие в сетях.

Методы аналитического, имитационного и натурного моделирования. Моделирование представляет собой мощный метод научного познания, при использовании которого исследуемый объект заменяется более простым объектом, называемым моделью. Основными разновидностями процесса моделирования можно считать два его вида - математическое и физическое моделирование. При физическом (натурном) моделировании исследуемая система заменяется соответствующей ей другой материальной системой, которая воспроизводит свойства изучаемой системы с сохранением их физической природы. Примером этого вида моделирования может служить пилотная сеть, с помощью которой изучается принципиальная возможность построения сети на основе тех или иных компьютеров, коммуникационных устройств, операционных систем и приложений.

Возможности физического моделирования довольно ограничены. Оно позволяет решать отдельные задачи при задании небольшого количества сочетаний исследуемых параметров системы. Действительно, при натурном моделировании вычислительной сети практически невозможно проверить ее работу для вариантов с использованием различных типов коммуникационных устройств - маршрутизаторов, коммутаторов и т.п. Проверка на практике около десятка разных типов маршрутизаторов связана не только с большими усилиями и временными затратами, но и с немалыми материальными затратами.

Но даже и в тех случаях, когда при оптимизации сети изменяются не типы устройств и операционных систем, а только их параметры, проведение экспериментов в реальном масштабе времени для огромного количества всевозможных сочетаний этих параметров практически невозможно за обозримое время. Даже простое изменение максимального размера пакета в каком-либо протоколе требует переконфигурирования операционной системы в сотнях компьютеров сети, что требует от администратора сети проведения очень большой работы.

Поэтому, при оптимизации сетей во многих случаях предпочтительным оказывается использование математического моделирования. Математическая модель представляет собой совокупность соотношений (формул, уравнений, неравенств, логических условий), определяющих процесс изменения состояния системы в зависимости от ее параметров, входных сигналов, начальных условий и времени.

Особым классом математических моделей являются имитационные модели. Такие модели представляют собой компьютерную программу, которая шаг за шагом воспроизводит события, происходящие в реальной системе. Применительно к вычислительным сетям их имитационные модели воспроизводят процессы генерации сообщений приложениями, разбиение сообщений на пакеты и кадры определенных протоколов, задержки, связанные с обработкой сообщений, пакетов и кадров внутри операционной системы, процесс получения доступа компьютером к разделяемой сетевой среде, процесс обработки поступающих пакетов маршрутизатором и т.д. При имитационном моделировании сети не требуется приобретать дорогостоящее оборудование - его работы имитируется программами, достаточно точно

воспроизводящими все основные особенности и параметры такого оборудования.

Преимуществом имитационных моделей является возможность подмены процесса смены событий в исследуемой системе в реальном масштабе времени на ускоренный процесс смены событий в темпе работы программы. В результате за несколько минут можно воспроизвести работу сети в течение нескольких дней, что дает возможность оценить работу сети в широком диапазоне варьируемых параметров.

Результатом работы имитационной модели являются собранные в ходе наблюдения за протекающими событиями статистические данные о наиболее важных характеристиках сети: временах реакции, коэффициентах использования каналов и узлов, вероятности потерь пакетов и т.п.

Существуют специальные языки имитационного моделирования, которые облегчают процесс создания программной модели по сравнению с использованием универсальных языков программирования. Примерами языков имитационного моделирования могут служить такие языки, как SIMULA, GPSS, SIMDIS.

Существуют также системы имитационного моделирования, которые ориентируются на узкий класс изучаемых систем и позволяют строить модели без программирования.

Модели теории массового обслуживания. Используемые в настоящее время в локальных сетях протоколы канального уровня используют методы доступа к среде, основанные на ее совместном использовании несколькими узлами за счет разделения во времени. В этом случае, как и во всех случаях разделения ресурсов со случайным потоком запросов, могут возникать очереди. Для описания этого процесса обычно используются модели теории массового обслуживания.

Механизм разделения среды протокола Ethernet упрощенно описывается простейшей моделью типа M|M|1 - одноканальной моделью с пуассоновским потоком заявок и показательным законом распределения времени обслуживания. Она хорошо описывает процесс обработки случайно поступающих заявок на обслуживание системами с одним обслуживающим прибором со случайным временем обслуживания и буфером для хранения поступающих заявок на время, пока обслуживающий прибор занят

выполнением другой заявки. Передающая среда Ethernet представлена в этой модели обслуживающим прибором, а пакеты соответствуют заявкам.

Введем обозначения: l - интенсивность поступления заявок, в данном случае это среднее число пакетов, претендующих на передачу в среде в единицу времени, b - среднее время обслуживания заявки (без учета времени ожидания обслуживания), то есть среднее время передачи пакета в среде с учетом паузы между пакетами в 9.6 мкс, r - коэффициент загрузки обслуживающего прибора, в данном случае это коэффициент использования среды, r = lb.

В теории массового обслуживания для данной модели получены следующие результаты: среднее время ожидания заявки в очереди (время ожидания пакетом доступа к среде) W равно:

$$W = \rho b/(1-\rho).$$

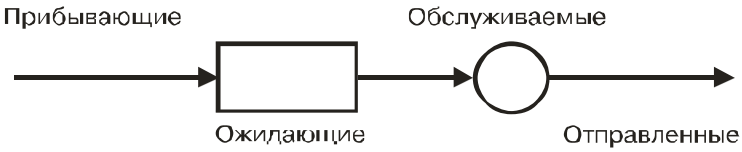

Рис. 1. Модель теории массового обслуживания M/M/1

Специализированные системы имитационного моделирования вычислительных сетей. Существуют специальные, ориентированные на моделирование вычислительных сетей программные системы, в которых процесс создания модели упрощен. Такие программные системы сами генерируют модель сети на основе исходных данных о ее топологии и используемых протоколах, об интенсивностях потоков запросов между компьютерами сети, протяженности линий связи, о типах используемого оборудования и приложений. Программные системы моделирования могут быть узко специализированными и достаточно универсальными, позволяющие имитировать сети самых различных типов. Качество результатов моделирования в значительной степени зависит от точности исходных данных о сети, переданных в систему имитационного моделирования.

Программные системы моделирования сетей - инструмент, который может пригодиться любому администратору корпоративной сети, особенно при проектировании новой сети или внесении кардинальных изменений в

уже существующую. Продукты данной категории позволяют проверить последствия внедрения тех или иных решений еще до оплаты приобретаемого оборудования. Конечно, большинство из этих программных пакетов стоят достаточно дорого, но и возможная экономия может быть тоже весьма ощутимой.

Программы имитационного моделирования сети используют в своей работе информацию о пространственном расположении сети, числе узлов, конфигурации связей, скоростях передачи данных, используемых протоколах и типе оборудования, а также о выполняемых в сети приложениях.

Обычно имитационная модель строится не с нуля. Существуют готовые имитационные модели основных элементов сетей: наиболее распространенных типов маршрутизаторов, каналов связи, методов доступа, протоколов и т.п. Эти модели отдельных элементов сети создаются на основании различных данных: результатов тестовых испытаний реальных устройств, анализа принципов их работы, аналитических соотношений. В результате создается библиотека типовых элементов сети, которые можно настраивать с помощью заранее предусмотренных в моделях параметров.

Системы имитационного моделирования обычно включают также набор средств для подготовки исходных данных об исследуемой сети - предварительной обработки данных о топологии сети и измеренном трафике. Эти средства могут быть полезны, если моделируемая сеть представляет собой вариант существующей сети и имеется возможность провести в ней измерения трафика и других параметров, нужных для моделирования. Кроме того, система снабжается средствами для статистической обработки полученных результатов моделирования.

В следующей таблице приведены характеристики нескольких популярных систем имитационного моделирования различного класса - от простых программ, предназначенных для установки на персональном компьютере, до мощных систем, включающих библиотеки большинства имеющихся на рынке коммуникационных устройств и позволяющих в значительной степени автоматизировать исследование изучаемой сети.

Компания и продукт	Стоимость (долл)	Тип сети	Требуемые ресурсы	Примечания
American	1800	ЛС	16МбОП, 6	Оценивание

HYTech, Prophesy			Мбдиск, DOS, Windows, OS/2	производительности при работе с текстовыми и графическими данными по отдельным сегментам и сети в целом
CACI Product, COMNET III	44500-49500	ЛС, ГС	64 МбОП, 100 Мбдиск, Windows, Windows NT, OS/2, Unix	Моделируетсети X.25, ATM, Frame Relay, связи LAN-WAN, SNA, DECnet, протоколы OSPF, RIP. Доступ CSMA/CD и токенный доступ, FDDI и др. Встроенная библиотека марщрутизаторов 3COM, Cisco, DEC, HP, Wellfleat, ...
Make System, NetMaker XA	7995-16995	ЛС, ГС	256 МбОП, 2000 Мбдиск, AIX, Sun OS, Sun Solaris	Проверка данных о топологии сети; импорт информации о трафике, получаемой в реальном времени
NetMagic System, StressMagik	3995	ЛС	4 МбОП, 8 МБдиск, Windows	Поддержка стандартных тестов измерения производительности; имитация пиковой нагрузки на файл-сервер
Network Analysis Center, MIND	10400-80000	ГС	16 МбОП, 65 Мбдиск, DOS, Windows	Средство проектирования, оптимизации сети, содержит данные о стоимости типичных конфигураций с возможностью точного оценивания производительности
Network Design and Analysis Group, AutoNet/ Designer	28000	ГС	16 МбОП, 40 Мбдиск, Windows, OS/2	Определение оптимального расположения концентратора в ГС, возможность оценки экономии средств за счет снижения тарифной платы, смены поставщика услуг и обновления оборудования; сравнение вариантов связи через ближайшую и оптимальную точку доступа, а также через

				мост и местную телефонную сеть
Network Design and Analysis Group, AutoNet/ MeshNET	35000	ГС	16 МбОП, 40 Мбдиск, Windows, OS/2	Моделирование полосы пропускания и оптимизация расходов на организацию ГС путем имитации поврежденных линий, поддержка тарифной сетки компаний AT & T, Sprint, WiTel, Bell
Network Design and Analysis Group, AutoNet/ Performance-1	6000	ГС	16 МбОП, 1 Мбдиск, Windows, OS/2	Моделирование производительности иерархических сетей путем анализа чувствительности к длительности задержки, времени ответа, а также узких мест в структуре сети
Network Design and Analysis Group, AutoNet/ Performance-3	8000	ГС	16 МбОП, 3 Мбдиск, Windows, OS/2	Моделирование производительности многопротокольных объединений локальных и глобальных сетей; оценивание задержек в очередях, прогнозирование времени ответа, а также узких мест в структуре сети; учет реальных данных о трафике, поступающих от сетевых анализаторов
System& Networks, BONES	25000-45000	ЛС, ГС	64 МбОП, 80 Мбдиск, Sun OS, Sun Solaris, HP-UX	Анализ воздействия приложений клиент-сервер и новых технологий на работу сети

2. Основные проблемы задачи разработки методов и программных средств повышения производительности коммуникационных сетей

Чтобы сеть работала самым эффективным образом, приходится решать для себя следующие задачи:

1. Сформулировать *критерии эффективности* работы сети. Чаще всего такими критериями служат производительность и надежность, для которых в свою очередь требуется выбрать конкретные показатели оценки, например, время реакции и коэффициент готовности, соответственно.

2. Определить множество *варьируемых параметров* сети, прямо или косвенно влияющих на критерии эффективности. Эти параметры действительно должны быть варьируемыми, то есть нужно убедиться в том, что их можно изменять в некоторых пределах по вашему желанию. Так, если размер пакета какого-либо протокола в конкретной операционной системе устанавливается автоматически и не может быть изменен путем настройки, то этот параметр в данном случае не является варьируемым, хотя в другой операционной системе он может относиться к изменяемым по желанию администратора, а значит и варьируемым. Другим примером может служить пропускная способность внутренней шины маршрутизатора - она может рассматриваться как параметр оптимизации только в том случае, если вы допускаете возможность замены маршрутизаторов в сети.

Все варьируемые параметры могут быть сгруппированы различным образом. Например, параметры отдельных конкретных протоколов (максимальный размер кадра протокола Ethernet или размер окна неподтвержденных пакетов протокола TCP) или параметры устройств (размер адресной таблицы или скорость фильтрации моста, пропускная способность внутренней шины маршрутизатора). Параметрами настройки могут быть и устройства, и протоколы в целом. Так, например, улучшить работу сети с медленными и зашумленными глобальными каналами связи можно, перейдя со стека протоколов IPX/SPX на протоколы TCP/IP. Также можно добиться значительных улучшений с помощью замены сетевых адаптеров неизвестного производителя на адаптеры BrandName.

3. Определить *порог чувствительности* для значений критерия эффективности. Так, производительность сети можно оценивать логическими

значениями "Работает" / "Не работает", и тогда оптимизация сводится к диагностике неисправностей и приведению сети в любое работоспособное состояние. Другим крайним случаем является тонкая настройка сети, при которой параметры работающей сети (например, размер кадра или величина окна неподтвержденных пакетов) могут варьироваться с целью повышения производительности (например, среднего значения времени реакции) хотя бы на несколько процентов. Как правило, под оптимизацией сети понимают некоторый промежуточный вариант, при котором требуется выбрать такие значения параметров сети, чтобы показатели ее эффективности существенно улучшились.

Таким образом, можно предложить три различных трактовки задачи оптимизации:

1. Приведение сети в любое работоспособное состояние. Обычно эта задача решается первой, и включает:

- поиск неисправных элементов сети - кабелей, разъемов, адаптеров, компьютеров;
- проверку совместимости оборудования и программного обеспечения;
- выбор корректных значений ключевых параметров программ и устройств, обеспечивающих прохождение сообщений между всеми узлами сети - адресов сетей и узлов, используемых протоколов, типов кадров Ethernet и т.п.

2. Грубая настройка - выбор параметров, резко влияющих на характеристики (надежность, производительность) сети. Если сеть работоспособна, но обмен данными происходит очень медленно (время ожидания составляет десятки секунд или минуты) или же сеанс связи часто разрывается без видимых причин, то работоспособной такую сеть можно назвать только условно, и она, безусловно, нуждается в грубой настройке. На этом этапе необходимо найти ключевые причины существенных задержек прохождения пакетов в сети. Обычно причина серьезного замедления или неустойчивой работы сети кроется в одном неверно работающем элементе или некорректно установленном параметре, но из-за большого количества возможных виновников поиск может потребовать длительного наблюдения за работой сети и громоздкого перебора вариантов. Грубая настройка во

многом похожа на приведение сети в работоспособное состояние. Здесь также обычно задается некоторое пороговое значение показателя эффективности и требуется найти такой вариант сети, у которого это значение было бы не хуже порогового. Например, нужно настроить сеть так, чтобы время реакции сервера на запрос пользователя не превышало 5 секунд.

3. Тонкая настройка параметров сети (собственно оптимизация). Если сеть работает удовлетворительно, то дальнейшее повышение ее производительности или надежности вряд ли можно достичь изменением только какого-либо одного параметра, как это было в случае полностью неработоспособной сети или же в случае ее грубой настройки. В случае нормально работающей сети дальнейшее повышение ее качества обычно требует нахождения некоторого удачного сочетания значений большого количества параметров, поэтому этот процесс и получил название "тонкой настройки".

Даже при тонкой настройке сети оптимальное сочетание ее параметров (в строгом математическом понимании термина "оптимальность") получить невозможно, да и не нужно. Нет смысла затрачивать колоссальные усилия по нахождению строгого оптимума, отличающегося от близких к нему режимов работы на величины такого же порядка, что и точность измерений трафика в сети. Достаточно найти любое из близких к оптимальному решений, чтобы считать задачу оптимизации сети решенной. Такие близкие к оптимальному решения обычно называют рациональными вариантами, и именно их поиск интересует на практике администратора сети или сетевого интегратора.

Поиск неисправностей в сети - это сочетание анализа (измерения, диагностика и локализация ошибок) и синтеза (принятие решения о том, какие изменения надо внести в работу сети, чтобы исправить ее работу).

- *Анализ* - определение значения критерия эффективности (или, что одно и то же, критерия оптимизации) системы для данного сочетания параметров сети. Иногда из этого этапа выделяют подэтап *мониторинга*, на котором выполняется более простая процедура - процедура сбора первичных данных о работе сети: статистики о количестве циркулирующих в сети кадров и пакетов различных протоколов, состоянии портов концентраторов, коммутаторов и маршрутизаторов и т.п. Далее выполняется этап собственно анализа, под которым в этом случае понимается более сложный и интеллектуальный процесс осмысления собранной на этапе мониторинга

информации, сопоставления ее с данными, полученными ранее, и выработки предположений о возможных причинах замедленной или ненадежной работы сети. Задача мониторинга решается программными и аппаратными измерителями, тестерами, сетевыми анализаторами и встроенными средствами мониторинга систем управления сетями и системами. Задача анализа требует более активного участия человека, а также использования таких сложных средств как экспертные системы, аккумулирующие практический опыт многих сетевых специалистов.

- *Синтез* - выбор значений варьируемых параметров, при которых показатель эффективности имеет наилучшее значение. Если задано пороговое значение показателя эффективности, то результатом синтеза должен быть один из вариантов сети, превосходящий заданный порог. Приведение сети в работоспособное состояние - это также синтез, при котором находится любой вариант сети, для которого значение показателя эффективности отличается от состояния "не работает". Синтез рационального варианта сети - процедура чаще всего неформальная, так как она связана с выбором слишком большого и очень разнородного множества параметров сети - типов применяемого коммуникационного оборудования, моделей этого оборудования, числа серверов, типов компьютеров, используемых в качестве серверов, типов операционных систем, параметров этих операционных систем, стеков коммуникационных протоколов, их параметров и т.д. и т.п. Очень часто мотивы, влияющие на выбор "в целом", то есть выбор типа или модели обрудования, стека протоколов или операционной системы, не носят технического характера, а принимаются из других соображений - коммерческих, "политических" и т.п. Поэтому формализовать постановку задачи оптимизации в таких случаях просто невозможно. В данной книге основное внимание уделяется этапам мониторинга и анализа сети, как более формальным и автоматизируемым процедурам. В тех случаях, когда это возможно, в книге даются рекомендации по выполнению некоторых последовательностей действий по нахождению рационального варианта сети или приводятся соображения, облегчающие его поиск.

3. Критерии эффективности работы сети

Все множество наиболее часто используемых критериев эффективности работы сети может быть разделено на две группы. Одна группа характеризует производительность работы сети, вторая - надежность.

Производительность сети измеряется с помощью показателей двух типов - временных, оценивающих задержку, вносимую сетью при выполнении обмена данными, и показателей пропускной способности, отражающих количество информации, переданной сетью в единицу времени. Эти два типа показателей являются взаимно обратными, и, зная один из них, можно вычислить другой. [17]

4. Время реакции

Обычно в качестве временной характеристики производительности сети используется такой показатель как *время реакции*. Термин "время реакции" может использоваться в очень широком смысле, поэтому в каждом конкретном случае необходимо уточнить, что понимается под этим термином.

В общем случае, время реакции определяется как интервал времени между возникновением запроса пользователя к какому-либо сетевому сервису и получением ответа на этот запрос. Очевидно, что смысл и значение этого показателя зависят от типа сервиса, к которому обращается пользователь, от того, какой пользователь и к какому серверу обращается, а также от текущего состояния других элементов сети - загруженности сегментов, через которые проходит запрос, загруженности сервера и т.п.

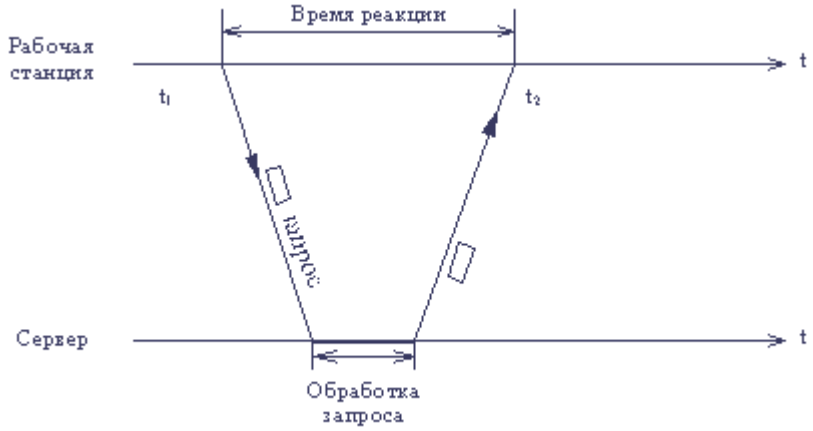

Рис. 2. Время реакции - интервал между запросом и ответом.

5. Основные варианты определения показателя "время реакции".

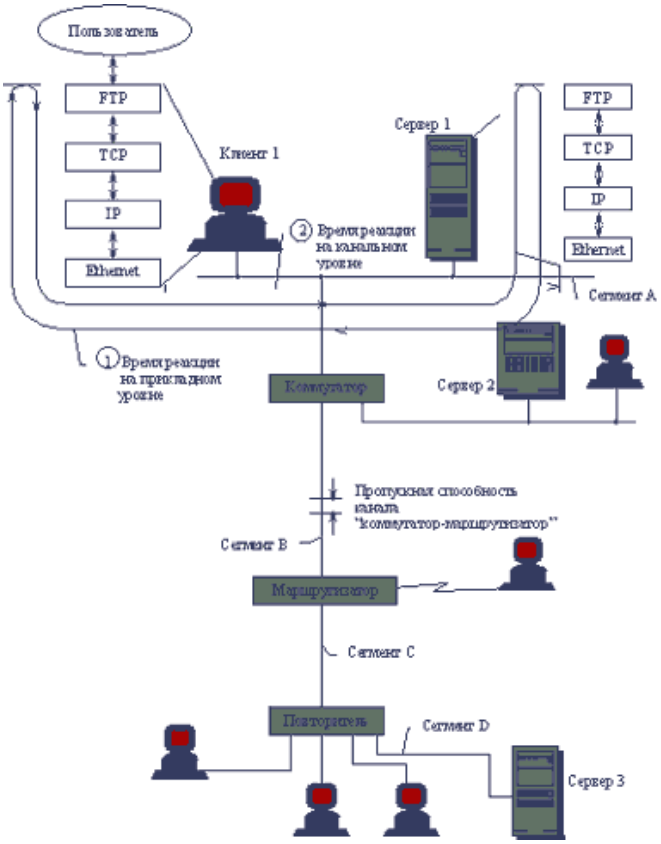

Рис. 3. Показатели производительности сети

В первом случае под временем реакции понимается время, которое проходит с момента обращения пользователя к сервису FTP для передачи файла с сервера 1 на клиентский компьютер 1 до момента завершения этой передачи. Очевидно, что это время имеет несколько составляющих. Наиболее существенный вклад вносят такие составляющие времени реакции как: время обработки запросов на передачу файла на сервере, время обработки получаемых в пакетах IP частей файла на клиентском компьютере, время передачи пакетов между сервером и клиентским компьютером по протоколу Ethernet в пределах одного коаксиального сегмента. Можно было бы выделить еще более мелкие этапы выполнения запроса, например, время обработки запроса каждым из протоколов стека TCP/IP на сервере и клиенте.

Для конечного пользователя, таким образом, определенное время реакции является понятным и наиболее естественным показателем производительности сети (размер файла, который вносит некоторую неопределенность в этот показатель, можно зафиксировать, оценивая время реакции при передаче, например, одного мегабайта данных). Однако, сетевого специалиста интересует в первую очередь производительность

собственно сети, поэтому для более точной ее оценки целесообразно вычленить из времени реакции составляющие, соответствующие этапам несетевой обработки данных - поиску нужной информации на диске, записи ее на диск и т.п. Полученное в результате таких сокращений время можно считать другим определением времени реакции сети на прикладном уровне.

Вариантами этого критерия могут служить времена реакции, измеренные при различных, но фиксированных состояниях сети:

А) *Полностью ненагруженная сеть*. Время реакции измеряется в условиях, когда к серверу 1 обращается только клиент 1, то есть на сегменте сети, объединяющем сервер 1 с клиентом 1, нет никакой другой активности - на нем присутствуют только кадры сессии FTP, производительность которой измеряется. В других сегментах сети трафик может циркулировать, главное - чтобы его кадры не попадали в сегмент, в котором проводятся измерения. Так как ненагруженный сегмент в реальной сети - явление экзотическое, то данный вариант показателя производительности имеет ограниченную применимость - его хорошие значения говорят только о том, что программное обеспечение и аппаратура данных двух узлов и сегмента обладают необходимой производительностью для работы в облегченных условиях. Для работы в реальных условиях, когда будет иметь место борьба за разделяемые ресурсы сегмента с другими узлами сети, производительность тестируемых элементов сети может оказаться недостаточной.

В) *Нагруженная сеть.* Это более интересный случай проверки производительности сервиса FTP для конкретных сервера и клиента. Однако при измерении критерия производительности в условиях, когда в сети работают и другие узлы и сервисы, возникают свои сложности - в сети может существовать слишком большое количество вариантов нагрузки, поэтому главное при определении критериев такого сорта - проведение измерений при некоторых типовых условиях работы сети. Так как трафик в сети носит пульсирующий характер и характеристики трафика существенно изменяются в зависимости от времени дня и дня недели, то определение типовой нагрузки - процедура сложная, требующая длительных измерений на сети. Если же сеть только проектируется, то определение типовой нагрузки еще больше усложняется.

Во втором случае критерием производительности сети является время задержки между передачей кадра Ethernet в сеть сетевым адаптером клиентского компьютера 1 и поступлением его на сетевой адаптер сервера 3. Этот критерий также относится к критериям типа "время реакции", но соответствует сервису нижнего - канального уровня. Так как протокол Ethernet - протокол дейтаграммного типа, то есть без установления соединений, для которого понятие "ответ" не определено, то под временем реакции в данном случае понимается время прохождения кадра от узла-источника до узла-получателя. Задержка передачи кадра включает в данном случае время распространения кадра по исходному сегменту, время передачи кадра коммутатором из сегмента A в сегмент B, время передачи кадра маршрутизатром из сегмента B в сегмент C и время передачи кадра из сегмента C в сегмент D повторителем. Критерии, относящиеся к нижнему уровню сети, хорошо характеризуют качества транспортного сервиса сети и являются более информативными для сетевых интеграторов, так как не содержат избыточную для них информацию о работе протоколов верхних уровней.

При оценке производительности сети не по отношению к отдельным парам узлов, а ко всем узлам в целом используются критерии двух типов: *средне-взвешенные* и *пороговые*.

Средне-взвешенный критерий представляет собой сумму времен реакции всех или некоторых узлов при взаимодействии со всеми или некоторыми серверами сети по определенному сервису, то есть сумму вида

$$(\Sigma_i \Sigma_j T_{ij})/(n \times m)$$

где T_{ij} - время реакции i-го клиента при обращении к j-му серверу, n - число клиентов, m - число серверов. Если усреднение производится и по сервисам, то в приведенном выражении добавится еще одно суммирование - по количеству учитываемых сервисов. Оптимизация сети по данному критерию заключается в нахождении значений параметров, при которых критерий имеет минимальное значение или по крайней мере не превышает некоторое заданное число.

Пороговый критерий отражает наихудшее время реакции по всем возможным сочетаниям клиентов, серверов и сервисов:

$$\max_{ijk} T_{ijk},$$

где i и j имеют тот же смысл, что и в предыдущем случае, а k обозначает тип сервиса. Оптимизация также может выполняться с целью минимизации критерия, или же с целью достижения им некоторой заданной величины, признаваемой разумной с практической точки зрения.

Чаще применяются пороговые критерии оптимизации, так как они гарантируют всем пользователям некоторый удовлетворительный уровень реакции сети на их запросы. Средневзвешенные критерии могут дискриминировать некоторых пользователей, для которых время реакции слишком велико притом, что при усреднении получен вполне приемлемый результат.

Можно применять и более дифференцированные по категориям пользователей и ситуациям критерии. Например, можно поставить перед собой цель гарантировать любому пользователю доступ к серверу, находящемуся в его сегменте, за время, не превышающее 5 секунд, к серверам, находящимся в его сети, но в сегментах, отделенных от его сегмента коммутаторами, за время, не превышающее 10 секунд, а к серверам других сетей - за время до 1 минуты.

ВЫВОДЫ

В этой главе произведен анализ методов и программных средств повышающих производительность коммуникационных сетей. А также методы аналитического, имитационного и натурного моделирования. Выбраны инструменты мониторинга и анализа производительности сети.

Рассмотрены основные проблемы, задачи разработки методов и программных средств повышающие производительность коммуникационных сетей. Выявлены критерии эффективности работы сети и время реакции сети.

ГЛАВА 2. МЕТОДЫ МОДЕЛИРОВАНИЯ ПРОИЗВОДИТЕЛЬНОСТИ КОМПЬЮТЕРНЫХ СЕТЕЙ

1. Стохастические методы моделирования производительности компьютерных сетей

Краткий обзор стохастических процессов. Стохастический процесс - это множество случайных переменных $\{X(t) \mid t \in T\}$, определенное на некотором пространстве вероятностей, и индексированное параметром t (обычно принимаемый, как время), который может принимать значения в некотором наборе T.

Величины X(t) определяют состояние системы. Набор всех возможных состояний называется пространством состояний и часто обозначается как I. Если пространство состояний дискретно, то это означает, что мы имеем дело с дискретным стохастическим процессом, который называется «цепью». Для удобства, часто предполагается, что пространство состояний представляет собой I = {0,1,2, · · ·}. Пространство состояний может также быть непрерывным. Тогда это будет классифицироваться как непрерывный стохастический процесс. Та же самая классификация может также использоваться относительно индекса T. Набор T может быть непрерывным, ведя к стохастическому процессу непрерывным временем. В случае, если набор T дискретен, стохастический процесс часто обозначается как $\{N_k]k \in T\}$. Так как имеются две возможности для каждого из двух вовлеченных наборов, возможны следующие четыре различных типа стохастических процессов:

I и T дискретные. Рассмотрим число пачек N_k, которые присутствуют в некоторой станции обслуживания, причем, в настоящее время станцию покидает k-тая пачка. Ясно, в станции обслуживания, могут присутствовать только целое число пачек, таким образом, I = {0,1,2, · · ·}. Аналогично, только после того, как первая пачка покидает, N_k ясно может быть определено во времени. Таким образом, мы имеем T = {1 ,2…}.

I дискретный и T непрерывный. Рассмотрим число пачек N(t), которое присутствует в некоторой станции обслуживания во времени t. В станции обслуживания может находиться только целое число пачек, таким

образом, I = {0,1,2, · · ·}. Можно, однако, наблюдать станцию обслуживания непрерывно. Это подразумевает что IT=R^+.

I непрерывный и T дискретный. Через W_k обозначается время, которое определяет k-тая пачка должна ждать обслуживания. Ясно, что $k \in T$ снова дискретный набор индекса, принимая во внимание, что W_k может принимать любое значение [0, ∞], подразумевая, что I является непрерывным.

I и T непрерывный. Через C_t обозначается общая сумма всех пачек, которым должно быть оказано в станции обслуживания в течение времени t. Ясно, $t \in T$ - непрерывный параметр. Кроме того, C_t может принимать любое значение из [0, ∞], при условии, что I является непрерывным.

Кроме вышеприведенных различий между стохастическими процессами, их возможно классифицировать и другим способом. Предполагается разделение стохастических процессов на процессы с непрерывным временем и непрерывным пространством состояний. Однако, предложенная классификация также применима для трех других упомянутых случаев.

В некоторой установленной точке времени $\bar{t} \in T$, величина X($\bar{t}$) просто случайная переменная, описывающая состояние стохастического процесса. Общая функция плотности (CDF) случайных переменных X($\bar{t}$) называется распределением первого порядка стохастического процесса $\{X(t)/t \in T\}$ и обозначается, как $F(\bar{x},\bar{t}) = \Pr\{X(\bar{t}) \le \bar{x}\}$. Можно обобщить это к случаю совместно распределенных случайных величин n-го порядка. Из стохастического процесса $\{X(t)/t \in T\}$ получится:

$$F(\bar{x},\bar{t}) = \Pr\{X(\bar{t}_1) \le \bar{x}_1, \cdots, X(\bar{t}_n) \le \bar{x}_n\}, \quad (1)$$

где $\bar{x} \in I^n$ и $\bar{t} \in T^n$.

Если все распределения n-ого порядка (n > 1) стохастического процесса $\{X(t)/t \in T\}$ инвариантны для изменений времени для всех возможных значений переменных $\bar{x}$ и $\bar{t}$, то стохастический процесс называют, строго стационарным, то есть, $F(\bar{x},\bar{t}) = F(\bar{x},\bar{t}+r)$, где $\bar{t}$ + r – значение из вектора $(\ldots,\bar{t}_i + r,\ldots.)$.

Стохастический процесс $\{X(t)/t \in T\}$ называется независимым процессом всякий раз, когда его совместное распределение n-ого порядка удовлетворяет следующее условие:

$$F(\tilde{x}, \tilde{t}) = \prod_{i=1}^{n} F(\tilde{x}_i, \tilde{t}_i) = \prod_{i=1}^{n} \Pr\{X(\tilde{t}_i) \leq \tilde{x}_i\}. \quad (2)$$

Пример независимого стохастического процесса - процесс возобновления. Процесс возобновления $\{X_n!n = 1,2,...\}$, является дискретно-временным стохастическим процессом, где $X_1, X_2,...$ независимые тождественно распределенные, неотрицательные случайные переменные.

Процесс возобновления - стохастический процесс, в котором между ее последовательными состояниями существует полная независимость. Во многих ситуациях, однако, некоторая форма зависимости существует между последовательными состояниями стохастического процесса. Возможная минимальная зависимость означает, что следующее состояние, которое будет принято стохастическим процессом, только зависит от текущего состояния стохастического процесса, а не предыдущих состояний. Это называется зависимостью заказа первого порядка или зависимостью Маркова.

Стохастический процесс $\{X(t)/t \in T\}$ называется процессом Маркова, если для любого $t_0\, t_0 \langle ....\langle t_{n\langle} \langle s \langle t$, распределение X (t), получает значения $X(t_0),...X(t_n), X(s)$ только зависят от X (s), то есть:

$$\Pr\{X(t) \leq x | X(t_0) = x_0, \cdots, X(t_n) = x_n, X(s) = x_s\} = \Pr\{X(t) \leq x | X(s) = x_s\}. \quad (3)$$

Уравнение (2.1.3) называется свойством Маркова. Подобные определения можно давать для случаев (процессов) имеющих дискретное состояние и для дискретного времени. Очень часто Марковские процессы, являются инвариантными к изменениям времени, то есть для любого s, t, (s < t), и x, x_s получается:

$$\Pr\{X(t) \leq x | X(s) = x_s\} = \Pr\{X(t-s) \leq x | X(0) = x_s\}. \quad (4)$$

В этих случаях можно говорить о гомогенно-временных процессах Маркова. Важно обратить внимание на то, что следующее состояние только зависит от текущего состояния, а не от того, как долго система находилась в том состоянии.

В качестве примера можно рассмотреть количество коммуникаций в компьютерной сети. Рассмотрим коммуникационную систему с большим

количеством потенциальных пользователей. Начнем с того, что количество пользователей настолько велико, что разумно предположить, что интервалы времени между последовательными запросами распределены по экспоненте. Согласно предположению, это используется для определения количества связей по экспоненте распределенного интервала времени компьютерной сети, которое управляется процессом Маркова. Кроме того, это управляется специальным случаем процесса называемым “рождения-смерти”.

Цепи Маркова с дискретным временем. Цепи Маркова с дискретным временем (DTMCs-Discrete-time Markov chains) или DTMC автоматически имеют свойства всех процессов Маркова, то есть их будущее поведение зависит только от их текущего состояния, а не от предыдущих состояний. Без потери общности можно предположить, что индекс устанавливает T = {0,1,2, · · ·} и что пространство состояний определяется пространством I. Свойство Маркова (3) для этого случая имеет следующую форму:

$$\Pr\{X_n = i_n | X_0 = i_0, \cdots, X_{n-1} = i_{n-1}\} = \Pr\{X_n = i_n | X_{n-1} = i_{n-1}\}, \quad (5)$$

где $i_0,...,i_n \in I$. Представим $p_j(n) = \Pr\{X_n = j\}$, обозначают вероятность "нахождения" в состоянии j в момент времени n. Кроме того, условная вероятность определяется как $p_{i,k}(m,n) = \Pr\{X_n = k! X_m = j\}(0 \le m \le n)$, то есть вероятность перехода от состояния j во время m., к состоянию k. Вероятности перехода зависят только от разницы во времени n-m. Поэтому они обозначаются как $p_{i,j}(n) = \Pr\{X_{m+n} = k! X_m = j\}$. Эти вероятности называются вероятностями перехода n-го шага. Вероятности перехода шага 1 обозначается $p_{j,k}$ (индекс 1 опущен). Вероятности 0 шага определены как $p_{j,k}(0) = 1$ всякий раз, когда j = k и 0 в других случаях. Начальное распределение, p (0) цепи Маркова определено как p (0) = $(p_0(0),...)$ ·

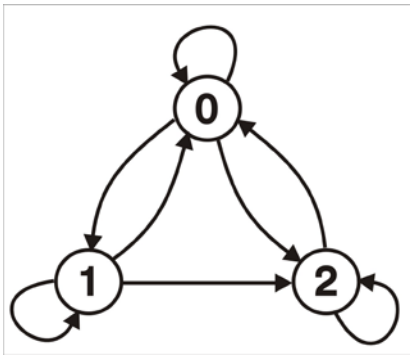

Рис. 1: Диаграмма перехода состояний для DTMC.

Многократным применением правил на условных вероятностях, может легко быть получено

$$\Pr\{X_0 = i_0, X_1 = i_1, \cdots, X_n = i_n\} = p_{i_0}(0)p_{i_0,i_1} \cdots p_{i_{n-1},i_n}. \quad (6)$$

Это подразумевает, что цепь Маркова полностью описана начальными вероятностями и вероятностями с 1 шагами. Вероятности с 1 шагами легко описываются матрицей вероятностей перехода $P=[p_{i,j}]$. Матрица P - стохастическая матрица, все ее элементы $p_{i,j}$ удовлетворяют $0 \le p_{i,j} \le 1 \sum_j p_{i,j} = 1$ и для всех i.

DTMC очень удобно представляется как направленный граф с вершинами элементами из I. Направленное ребро с "весом" $p_{i,j}$ существует между вершинами i и j когда $p_{i,j} > 0$. Такие представления цепей Маркова часто называются диаграммами перехода состояний. В Рис. 1 представлена диаграмма перехода состояний для DTMC с матрицей.

$$\mathbf{P} = \begin{pmatrix} 6/10 & 2/10 & 2/10 \\ 1/10 & 8/10 & 1/10 \\ 6/10 & 0 & 4/10 \end{pmatrix}, \quad (7)$$

и начальный вектор вероятности p (0) = (1,0,0).

Теперь вычисляя вероятности перехода с шагом 2 для DTMC с помощью матрицы вероятностей P с шагом 1, имеем:

$$p_{i,j}(2) = \Pr\{X_2 = j | X_0 = i\} = \sum_{k \in I} \Pr\{X_2 = j, X_1 = k | X_0 = i\}, \quad (8)$$

Переход из состояния i, в состояние j в двух шагах, через любое промежуточное состояние $k \in I$. Теперь, из правил условных вероятностей также из свойств цепей Маркова, можно написать:

$$\begin{aligned} p_{i,j}(2) &= \sum_{k \in I} \Pr\{X_2 = j, X_1 = k | X_0 = i\} \\ &= \sum_{k \in I} \Pr\{X_2 = j | X_1 = k, X_0 = i\} \Pr\{X_1 = k | X_0 = i\} \\ &= \sum_{k \in I} \Pr\{X_2 = j | X_1 = k\} \Pr\{X_1 = k | X_0 = i\} = \sum_{k \in I} p_{k,j} p_{i,k}. \end{aligned} \quad (9)$$

В последнем уравнении присутствует матричное произведение. Вероятности с шагами 2 т.е. $p_{i,j}(2)$ являются элементами матрицы P^2. Вышеупомянутая техника может применяться итеративно для шага n. Вероятности данного шага $p_{i,j}(n)$ элементы матрицы P^n. Для идентичности, матрицу I, можно написать $I=P^0$. Уравнение

$$\mathbf{P}^{m+n} = \mathbf{P}^m \mathbf{P}^n \quad (10)$$

известно как уравнение Чапмана-Колмогорова.

Когда же нужно вычислить $p_j(n)$, можно делать так, просто, обусловливая, что начальная вероятность, то есть $p_j(n) = \Pr\{X_n = j\} = \sum_{i \notin I} \Pr\{X_0 = i\} \Pr\{X_n = j\} ! X_0 = i\} = \sum_{i \in I} p(0) p_{i,j}(n)$. Записав это в векторном/матричном виде, получим

$$p(n) = (p_0(n), p_1(n), ...) \text{ и } p(n) = p(0)P^n \quad (11).$$

Большинство, но определенно не все DTMC, имеют все строки в P^n, и сходятся к общему пределу, при $n \to \infty$, следовательно:

$$v_j = \lim_{n \to \infty} p_j(n) = \lim_{n \to \infty} \Pr\{X_n = j\} = \lim_{n \to \infty} \sum_{i \in I} p_i(0) p_{i,j}(n). \quad (12)$$

Учитывая это, получается:

$$\underline{v} = \lim_{n \to \infty} \underline{p}(n) = \lim_{n \to \infty} \underline{p}(0) \mathbf{P}^n. \quad (13)$$

Но, также получается:

$$\underline{v} = \lim_{n \to \infty} \underline{p}(n+1) = \lim_{n \to \infty} \underline{p}(0) \mathbf{P}^{n+1} = \lim_{n \to \infty} \left(\underline{p}(0) \mathbf{P}^n\right) \mathbf{P} = \underline{v} \mathbf{P}. \quad (14)$$

Всякий раз, когда вероятности ограничения v существуют, они могут быть получены решением систему линейных уравнений v=vP , так как v - вектор вероятности, $\sum_i v_i = 1$ и $0 \le v \le 1$. Уравнение v=vP не имеет полный ранг. Чтобы получить уникальное решение, мы должны использовать факт, что v является вектором вероятности, то есть сумма его элементы должна равняться к 1.

Вектор v часто называется постоянным вектором вероятности или устойчивым распределением состояний цепи Маркова. Для цепей Маркова существует уникальное распределение ограничения, которое является независимым от начальных вероятностей состояний.

Цепи Маркова с непрерывным временем. Рассматривая теорию цепей Маркова с непрерывным временем (CTMCs), принимается без потери общности, что I = {0,1,2, - ·} и что T = [0, ∞]. Для CTMCs общее свойство цепей Маркова должна иметь силу. Это означает, что:

$$\Pr\{X(t) = x | X(t_0) = x_0, \cdots, X(t_{n-1}) = x_{n-1}\} = \Pr\{X(t_n) = x_n | X(t_{n-1}) = x_{n-1}\}. \quad (15)$$

Важно обратить внимание на то, что следующее состояние только зависит от текущего состояния и не зависит от времени, когда цепь находится в текущем состоянии. Действительно, как может показаться, резидентное время (время нахождения в очереди и время обслуживания) в CTMCs распределено по экспоненте.

Таким же образом, как DTMC полностью описывается матрицей P и начальными вероятностями p (0), CTMC описан бесконечной генерирующей матрицей или матрицей $Q = (q_{i,j})$, и вероятностью начального состояние p(0). Обозначая систему в момент времени $t \in T$, как $X(t) \in T$, мы имеем

$$\Pr\{X(t+\delta t) = j | X(t) = i\} = q_{i,j}\delta t + o(\delta t), \quad i \neq j. \quad (16)$$

Величина $q_{i,j}\,(i \neq j)$ может интерпретироваться как показатель, который характеризирует как текущее состояние i переходит в состояние j. Обозначив через $p_i(t)$ вероятность того, что в момент времени t система находиться в состоянии i, т.е. $p_i(t) = \Pr\{X(t) = i\}$. С заданием $p_i(t)$, появляется заинтересованность в развитии цепи Маркова в будущем, то есть:

$$p_i(t+\delta t) = p_i(t)\Pr\{\text{do not depart from } i\} + \sum_{j \neq i} p_j(t)\Pr\{\text{go from } j \text{ to } i\} =$$

$$p_i(t)(1 - \sum_{j \neq i} q_{i,j}\delta t) + (\sum_{j \neq i} p_j(t) q_{j,i})\delta t + o(\delta t).$$

$$q_{i,i} = -\sum_{j \neq i} q_{i,j}, \quad (17)$$

Теперь, определяем, что $q_{i,j} = -\sum_{i \neq j} q_{i,j}$ и получается, что

$$p_i(t+\delta t) = p_i(t) + (\sum_{j \in I} q_{j,i} p_j(t))\delta t + o(\delta t). \quad (18)$$

Преобразовав это и разделив на σt и взяв предел $\sigma t \to 0$ получается

$$p_i'(t) = \lim_{\delta t \to 0} \frac{p_i(t+\delta t) - p_i(t)}{\delta t} = \sum_{j \in I} q_{j,i} p_j(t), \quad (19)$$

Который в матричной форме имеет следующий вид:

$$\underline{p}'(t) = \underline{p}(t)\mathbf{Q}, \quad (20)$$

где $p(t) = (..., p_i(t), ...)$.

Получив эту линейную систему дифференциальных уравнений, можно решить это, используя начальное распределение вероятности $p(0)$ и получая:

$$\underline{p}(t) = \underline{p}(0)e^{-\mathbf{Q}t}. \quad (21)$$

В многих случаях нет заинтересованности в этих, так называемых, переходных поведениях $p(t)$ процесса Маркова. Вообще, вполне достаточно получить вероятности устойчивых состояний $p_i = \lim_{t\to\infty} p'(t)$. Когда предполагается, что устойчивое распределение существует, это подразумевает, что вышеупомянутый предел существует, и таким образом что $\lim_{t\to\infty} p'_i(t) = 0$. Следовательно, чтобы получать вероятности устойчивых состояний нужно решить систему линейных уравнений:

$$\underline{p}\mathbf{Q} = \underline{0}, \quad \sum_{i\in I} p_i = 1. \tag{22}$$

Правая часть добавлена, чтобы утверждать, что полученное решение - действительно вектор вероятности. Левая часть один имеет бесконечно много решений, который после нормализации они представляют тот же самый вектор вероятностей.

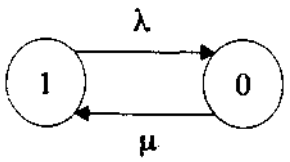

Рис.2: Пример CTMC с 2-состояниями

В практических ситуациях, любое из уравнений системы pQ = 0 может быть заменено уравнением $\sum_{i\in I} p_i = 1$, чтобы получить уникальное решение.

Также, как было возможно для DTMC, CTMC может очень удобно быть изображен в соответствии с диаграммами перехода состояний. Снова, эти диаграммы перехода - графы, вершины которых совпадают с состояниями CTMC. Ребро между вершинами l и j существует всякий раз, когда $q_{i,j} \rangle 0$. Грани в графе маркированы соответствующими нормами (интенсивностями).

2. Общие стохастические модели M|G|1

Рассмотрим единственную станцию обслуживания очереди сервером с неограниченной буферной способностью (вместимостью) и неограниченным потоком заявок (популяцией клиентов, пачек). Пачки, достигающие станции

обслуживания очереди формируют процесс Пуассона с нормой λ. Требование обслуживания пачки - случайная переменная S, распределенное согласно функции распределения B (s), то есть. $B(s) = \Pr\{S \le s\}$. S имеет ожидание E [S] (начальный момент) и второй момент $E[S^2]$.

Снова мы используем примечание E [N] для среднего числа пачек в системе обслуживания очереди, $E[N_q]$ для среднего номера (числа) клиентов в очереди, и $E[N_s]$ для среднего числа клиентов в сервере. При применении закона Литтла для сервера мы имеем: $\rho = E[N_s] = \lambda E[S]$. Происхождение $E[N_q]$ более сложно. Доказательства этих соотношений приводиться в следующих секциях.

Для среднего числа пачек в очереди M|G|1 станции обслуживания очереди, $E[N_q]$ имеем следующее выражение:

$$E[N_q] = \frac{\lambda^2 E[S^2]}{2(1-\rho)} \quad (23)$$

При применении закона Литтла $(E[W] = E[N_q]/\lambda)$, мы имеем

$$E[W] = \frac{\lambda E[S^2]}{2(1-\rho)} \quad (24)$$

Эти два уравнения только представляют часть очереди станции обслуживания очереди. Включением сервера, получают следующие выражения:

$$E[N] = \lambda E[S] + \frac{\lambda^2 E[S^2]}{2(1-\rho)}, \quad (25)$$

$$E[R] = E[S] + \frac{\lambda E[S^2]}{2(1-\rho)}. \quad (26)$$

Результат M|G|1 представлен главным образом в одной из четырех формул выше. Формула (25) часто упоминается как формула Поллачека-Хинчина (ПХ).

Рассмотрев ПХ - формулы видно, что E [N] зависит от первого и второго момента распределения времени обслуживания. Что это подразумевает? С тех пор с первых двух моментов распределения, его разница может быть получена, то есть $\sigma^2{}_S = E[(S - E[S]^2] = E[S^2] - E[S]^2$, видно, что более высокая разница подразумевает более высокое среднее число пачек в системе. С точки зрения обслуживания очереди, не имеющей никакой разницы $(E[S^2] = E[S]^2$ является оптимальной. Отсутствие разницы означает,

что это детерминированные распределения, то есть, что случайности нет вообще. Это общий "закон": чем большее разница (случайность) существует в системе, тем хуже производительность системы. С худшим выполнением работы сервером мы, конечно, подразумеваем более длинные очереди, более длительное время ожидания и т.д.

M|G|1 модели с прибытием партии пачек. Во многих системах связи, пачки не прибывают индивидуально. Во многих системах, пачки, которые должны быть переданы, прибывают от более высокого уровня, в котором ориентируемая на пользователя пачка была раздроблена на множество минипачек, которые соответствуют размеру пачки, используемому на более низких уровнях в стеке потока. Как пример, можно рассмотреть пользователя, который хочет передать файл по некоторой среде передачи данных. Файл не будет вообще иметь размер, который может непосредственно быть передан; это будет должен быть сцепленные пачки допустимого размера. Таким образом, когда пользователь посылает файл, объекты протокола преобразуют это в множество минипачек, которые должны быть переданы. Реферируя из протокола, обрабатывающего время, видно это в момент передачи файла, что фактически множество минипачек должно быть передано на более низком уровне.

Можно моделировать этот вид систем, принимая так называемую партию или взрывное прибытие. Предполагаем, что имеется Пуассоновский процесс прибытия партий минипачек с интенсивностью λ. Прибывающая партия состоит случайное количество H минипачек, характеризующихся дискретными вероятностями h_k, k = 1,2, · · ·. Ожидаемое число минипачек в партии

$$E[H] = \sum_{k=1}^{\infty} k h_k, \qquad (27)$$

И второй факториальный момент равняется

$$E[H(H-1)] = \sum_{k=1}^{\infty} k(k-1) h_k. \qquad (28)$$

Каждая мини пачка требует случайного количества обслуживания S с первым и вторым моментом E [S] и E [S^2] соответственно. Ясно, что $\rho = E[H]E[S]$. Без доказательства, определяется среднее время ожидания E[W] минипачки:

$$E[W] = \frac{\lambda E[H]E[S^2]}{2(1-\rho)} + \frac{E[H(H-1)]E[S]}{2E[H](1-\rho)}. \quad (29)$$

Первое слагаемое действительно среднее время ожидания минипачек в нормальной модели M|G|l, как будто пачки прибыли со средним размером E [H] E [S] и вторым моментом E [H] E [S^2]. Второе слагаемое составляет некоторое дополнительное время ожидания, что пачки, которые - не одиночные, а в партии.

Модель M|G|1 с прибытием партиями часто обозначается как $M^{[H]}$ |G|1, где [H] обозначает распределение размера партии.

M|G|1 модели с приоритетами. Важный аспект стратегий приоритета заключается в том, есть ли они или нет. С приоритетными стратегиями приоритета пачка в обслуживании будет остановлена обслуживаемым сервером, как только прибудет пачка с более высоким приоритетом. В том случае, сначала обслуживается она, а после обслуживание пачки с более низким приоритетом, которая была в обслуживании первоначально, будет возобновлено. С неприоритетными стратегиями приоритета, пачка в обслуживании всегда сначала заканчивается прежде, чем новая пачка будет помещена в обслуживание. Неприоритетные стратегии приоритета только берут пачки из очереди. Однако, неприоритетные стратегии происходят более часто.

Рассмотрим модель с единственным сервером, в которой прибывающие пачки могут классифицироваться по P категориями или классам приоритета, и перечисляются 1 через P. Предположим, что класс 1 имеет самый высокий и класс P самый низкий приоритет.

Пачки класса r = 1, · · ·, P прибывают в станцию обслуживания очереди согласно процессу Пуассона с нормой λ_r. Среднее требование обслуживания для класса r пачек - $E[S_r] = 1/\mu_r$ второй момент обслуживания для класса r, пачки - $E[S^2{}_r]$. В Рис.3. показана модель M|G|1 с многократными приоритетами.

Получим отношение между средним временем ожидания $E[W_r]$ класса r пачек и средним временем ожидания пачек более высоких классов приоритета, т. е. классов 1, · · ·, r -1.

Пачка класса r , прибывающая на станцию обслуживания очереди должна ждать прежде, чем будут обслужены пачки более высоких приоритетов. Это время ожидания состоит из трех компонентов:

- сохранение обслуживания любой пачки находящейся в обслуживании;
- периода, необходимого, чтобы служить, все пачки приоритета класса k = 1, · · ·, r, то есть всех выше и равных приоритетов пачек, которые присутствуют в системе во время по прибытию пачки класса r;
- времени обслуживания пачек более высоких классов приоритета, то есть пачек классов (занятий) k - 1, · · ·, r - 1, которые прибывают в течение ожидаемого периода пачки класса r.

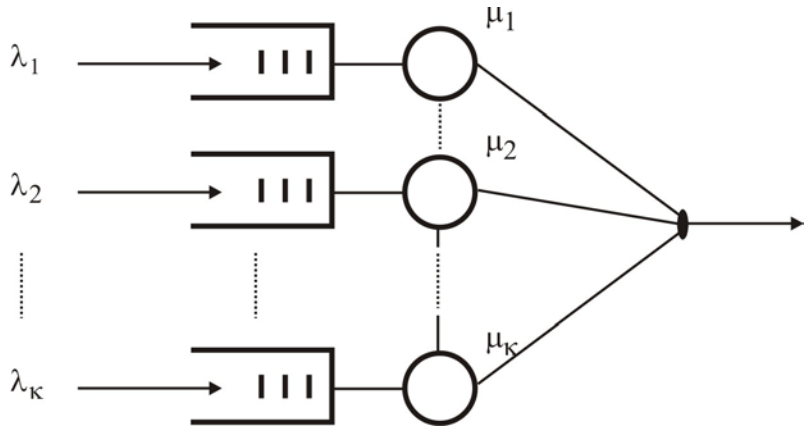

Рис. 2.3: M|G 1 модель с P классами приоритета

В форме уравнения, мы имеем:

$$W_r = T_0 + \sum_{k=1}^{r} T_k + \sum_{k=1}^{r-1} T_k', \qquad (30)$$

где W_r - время ожидания пачек класса r , T_0 время обслуживания сохранения пачки в обслуживании, T_k время необходимое для обслуживания всего класс k клиентов, которые присутствуют во время прибытия класса r клиента, и T_k' время, которое необходимо, чтобы обслужить весь класс k клиентов, которые прибывают в течение W_r. При вычислений ожиданий с обеих сторон получаем:

$$E[W_r] = E[T_0] + \sum_{k=1}^{r} E[T_k] + \sum_{k=1}^{r-1} E[T_k']. \qquad (31)$$

Получим выражения для трех сроков времени в правой стороне этого уравнения.

Время обслуживания пачек, находящихся в обслуживании явно зависит от класса пачки, которая находится в обслуживании. Пачка класса r находится в обслуживании с вероятностью ρ_r, где $\rho_r = \lambda_r E[S_r]$, утилизация , вызванное классом r пачки. Сохранение, необходимого времени обслуживания класса r пачка равняется $E[S_r^2]/2E[S_r]$. В итоге получается:

$$E[T_0] = \sum_{k=1}^{P} \rho_k \frac{E[S_k^2]}{2E[S_k]} = \frac{1}{2}\sum_{k=1}^{P} \lambda_k E[S_k^2]. \quad (32)$$

Величина $E[T_k]$ явно зависит от числа пачек в каждого класса в станции обслуживания очереди по прибытию нового класса r пачки. Из свойства PASTA и закона Литтла известно, что в среднем есть $E[N_{q,k}] = \lambda_k E[W_k]$ пачек класса k, прибывающих в очередь. Так как, они требуют в среднем $1/\mu_k$ количества времени обслуживания каждой пачки, мы имеем $E[T_k] = E[N_{q,k}]/\mu_k = \lambda_k E[W_k]/\mu_k = \rho_k E[W_k]$.

Наконец, нужно вычислить $E[T_k^{'}]$. В течение $E[W_r]$ единицы времени пачка класса r а должна ждать, в среднем $\lambda_k E[W_r]$ пачки прибывают, каждое требование $1/\mu_k$ времени обслуживания. Таким образом, мы имеем $E[T_k^{'}] = \lambda_k E[W_r]/\mu_k = \rho_k E[W_r]$. Подставляя эти результаты в уравнении (33), получается:

$$\begin{aligned} E[W_r] &= \sum_{k=1}^{P} \frac{\lambda_k}{2} E[S_k^2] + \sum_{k=1}^{r} \rho_k E[W_k] + \sum_{k=1}^{r-1} \rho_k E[W_r] \\ &= \sum_{k=1}^{P} \frac{\lambda_k}{2} E[S_k^2] + \sum_{k=1}^{r-1} \rho_k E[W_k] + \sum_{k=1}^{r} \rho_k E[W_r] \\ &= \sum_{k=1}^{P} \frac{\lambda_k}{2} E[S_k^2] + \sum_{k=1}^{r-1} \rho_k E[W_k] + E[W_r] \times (\sum_{k=1}^{r} \rho_k). \end{aligned} \quad (34)$$

Обеспечение всего $E[W_r]$ и определяющий $\sigma_r = \sum_{k=1}^{r} \rho_k$, дает:

$$\begin{aligned} E[W_r](1 - \sum_{k=1}^{r} \rho_k) &= \sum_{k=1}^{P} \frac{\lambda_k}{2} E[S_k^2] + \sum_{k=1}^{r-1} \rho_k E[W_k] \Rightarrow \\ E[W_r] &= \frac{\sum_{k=1}^{P} \frac{\lambda_k}{2} E[S_k^2] + \sum_{k=1}^{r-1} \rho_k E[W_k]}{1 - \sigma_r}. \end{aligned} \quad (35)$$

Таким образом, установлено отношение, которое выражает $E[W_r]$ в терминах $E[W_{r-1}],...E[W_1]$. В случае, когда r = 1, получается:

$$E[W_1] = \frac{E[T_0]}{(1 - \sigma_1)} = \frac{E[T_0]}{1 - \rho_1} = \frac{1}{2}\frac{\sum_{k=1}^{P} \lambda_k E[S_k^2]}{1 - \rho_1}. \quad (36)$$

Для случая r = 2:

$$E[W_2] = \frac{E[T_0] + \rho_1 E[W_1]}{(1-\sigma_2)} = \cdots = \frac{E[T_0]}{(1-\sigma_2)(1-\sigma_1)}. \quad (37)$$

При продолжении этого процесса получается следующее отношение:

$$E[W_r] = \frac{E[T_0]}{(1-\sigma_r)(1-\sigma_{r-1})}, \quad (38)$$

с

$$E[T_0] = \sum_{k=1}^{P} \lambda_k E[S_k^2]/2. \quad [3]$$

3. Имитационное моделирование производительности компьютерных сетей

Идея имитационного моделирования. Пусть необходимо получить область под кривой $y = f(x) = x^2$ из $x = 0$ до $x = 1$. Через A обозначается результат вычисления, который будет получен. Получение этой области - простая проблема, которая может легко быть решена аналитически:

$$A = \int_0^1 x^2 dx = \frac{1}{3} x^3 |_{x=0}^{x=1} = \frac{1}{3}. \quad (39)$$

Ясно, в этом случае, расчетная величина, точно та же самая как реальное значение A. Рассматривая проблему, несколько более сложную, можно излагать тот же самый вопрос когда $f(x) = x^{\sin x}$. Теперь уже нельзя решить проблему аналитически. Однако, можно обратиться к числовым методам типа правила трапеций. Можно разбивать интервал [0,1] в n последовательные интервалы $[x_0, x_1]$, $[x_1, x_2]$, · · ·, $[x_{n-1}, x_n]$. Область под кривой тогда равняется

$$A = \sum_{i=1}^{n} \frac{1}{2}(x_i - x_{i-1})(f(x_i) + f(x_{i-1})). \quad (40)$$

Делая, интервалы достаточно маленькими в желании приближается с любым уровнем желательной точности. Этот пример показывает числовую технику решения.

Возможно, несколько неожиданно, но можно также получить результаты посредством стохастического моделирования. От простого

анализа известно, что всякий раз, когда $x \in [0,1]$ функция $f(x) = [0,1]$. Теперь, берется два случайных образца x_i и y_i от однородного распределения на [0,1]. Можно интерпретировать это как выбор случайной точки в квадрате площади $\{(x, y) / 0 \le x \le 1, 0 \le y \le 1\}$. Когда берется N таких пар образцов, Через n обозначается число типовых пар, для которых неравенство $y_i \le f(x_i)$ дает, то есть $n = \sum_{i=1}^{N} I_i$ с $I_i = 1\{y_i \le f(x_i)\}$. Следовательно, как n - число случайных точек той области ниже кривой $y = f(x)$, фракция $A = n / N$ дает оценку для области α.

Это может быть замечено следующим образом. Всякий раз, когда выполняется упомянутое неравенство, случайные типовые пары под кривой $y = f(x)$. При взятии достаточно типовых пар, фракция $A = n / N$ будет сходиться к α.

Пробуя получить α, посредством моделирования Монте-Карло, мы должны держать след полученных результатов. Начиная с А получено, поскольку функция случайных переменных, реализация случайной переменной X. Мы предполагаем, что X и $E[X] = \alpha$. От моделирования можно также получить величину для разницы X, то есть.

$$S^2 = \frac{1}{N-1} \sum_{i=1}^{N} (I_i - \frac{n}{N})^2. \quad (41)$$

Применяя неравенство Чебышева:

$$\Pr\{|E[X] - A| > \beta\} \le \frac{\mathrm{var}[X]}{\beta^2}. \quad (42)$$

При переписывании этого уравнения, при замене S^2 для $\mathrm{var}[X]$, и при взятии $\delta = 1 - S^2 / \beta^2$ получается

$$\Pr\{|E[X] - A| \le \frac{S}{\sqrt{1-\delta}}\} \ge \delta. \quad (43)$$

Это уравнение сообщает, что с вероятностью, по крайней мере δ оценок является ближе к реальной величине $E[X] = \alpha$, тогда $S / \sqrt{1-\delta}$. При выборе δ относительно большой, то есть 0.90 или 0.95, можно быть весьма уверены о правильности оценки А.

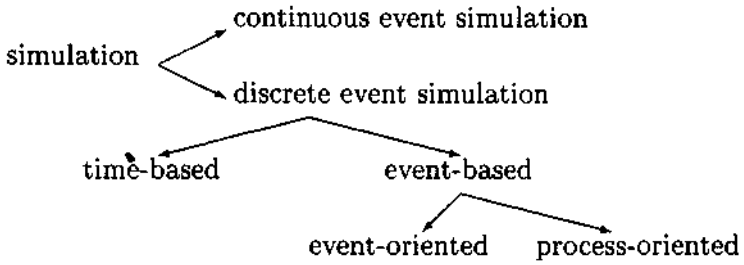

Рис.4: Классификация моделирования

Важно обратить внимание на то, что в случае, если есть аналитическое решение, доступное для специфической проблемы, это аналитическое решение дает гораздо большее количество понимания в поведении, чем числовые ответы, которые получены моделированием или числовыми средствами. Числовые величины, полученные моделированием или числовыми средствами, только дают информацию о решении проблемы, и по диапазону возможных решений, и при этом они не дают понимание в чувствительности ответа на изменения (замены) в один или большее количество параметров системы. Однако есть много случаев, когда моделирование – единственное жизнеспособная альтернатива.

Классификация моделирования. Моделирование имеет различные виды. В этой секции проводится классификация моделирования согласно двум критериям, являющимся их пространством и их развитием во времени.

В непрерывных случаях моделирования, изучены системы, в которых состояние непрерывно изменяется со временем. Как правило, эти системы - физические процессы, которые могут быть описаны системами дифференциальных уравнений с начальными или граничными условиями.

Числовое решение такой системы дифференциальных уравнений иногда называется моделированием. В физическом времени систем обычно - непрерывный параметр, хотя можно также наблюдать системы только в предопределенных случаях времени, выдавая дискретный параметр времени.

Большее количество соответствующего нашим целям - дискретные моделирования случая. В дискретном моделировании случая состояние системы, которую мы моделируем, представлено одной или более переменными. Снова можно или рассматривать время как непрерывный или

как дискретный параметр. В зависимости от применения под рукой, один из этих двух могут быть более или менее подходящий.

Дискретное моделирование случая. Перед анализом детали дискретного моделирования случая сначала нужно определить некоторую терминологию. Моделирование времени - значение параметра "время", которое используется в программе моделирования, то есть величина времени, которое бы имело силу в реальной системе.

Время выполнения - время, которое рассматривается как время исполнения программы моделирования.

В дискретной системе случая, состояние изменяется во времени. Причина переменного изменения состояния называется случаем. Очень часто изменения состояний также называются событиями. Эта терминология практически не запутывает. Так как предполагается, что события происходят по очереди, можно говорить о дискретном моделировании случая.

Концепция событий - очень важна. Это - фактически рассмотрение событий и их относительного времени, чем мы заинтересованы, потому что точно это описывает выполнение работы системы. В программе моделирования будут имитироваться все события также как и случаи их возникновения. Храня администрацию всех этих событий и их времен, возможно получить различные меры типа среднего времени меж случая или среднего времени между определенными парами событий и т.д.

Можно отличать два основных пути, которыми дискретное моделирование случая может быть осуществлено: моделирования на основе времени и моделирование на основе случая.

В моделировании на основе времени (также часто называется установленным моделированием приращения времени или синхронным моделированием) главным является, что моделирование управляет продвижением времени в постоянных шагов. В начале этого управления образовывают интервалы $[t, t+\Delta t]$. Тогда важно случились ли любые события в интервале времени $[t, t+\Delta t]$. Если так, эти события будут выполнены, то есть состояние будет изменено согласно этим событиям, перед следующим циклом. Принимается, что события в пределах интервала $[t, t+\Delta t]$ не имеют важность, и что эти события являются независимыми. Число событий, которые случились в интервале $[t, t+\Delta t]$, может изменяться время от времени.

Когда t увеличивается выше некоторого максимального времени моделирования, остановок программы.

Моделирование на основе времени легко осуществить. Выполнение близко напоминает выполнение, например, метода Эйлера решения дифференциальных уравнений. Однако, есть также некоторые недостатки. Оба предположение, что происхождение событий в пределах интервала $[t, t+\Delta t]$ - не важны и предположение, что эти события являются независимыми, требуют, чтобы Δt были достаточно маленькими, чтобы минимизировать возникновение взаимно зависимых событий. Очень часто, однако, по этим причинам надо брать Δt настолько маленьким, что рассматриваемое моделирование становится очень неэффективным. Много шагов времени должны быть сделаны для любого случая, встречающегося вообще. По этим причинам моделированием на основе времени не часто пользуются.

Было моделирование на основе времени в установленные шаги времени, но число событий в шаге времени на основе случая (также часто такое моделирование называется переменным моделированием приращения времени или асинхронным моделированием). Тогда имеем дело с изменяющимися шагами времени, но есть всегда точно один случай в шаге времени. Так что моделирование управляется возникновением "следующих событий". Это очень эффективно, так как шаги времени - теперь только достаточно велики, чтобы продолжить оптимальное моделирование и только достаточно коротки, чтобы исключить возможность наличия больше чем один случай в шаге времени, таким образом, обходящий возможность наличия зависимых событий в одном шаге времени.

Всякий раз, когда происходит случай, это провоцирует новые события в будущем. Например, прибытие пачки в очереди. Это случай, который заставляет состояние системы изменяться. Все будущие события вообще собираются в списке случая. Начало этого списка - следующий случай, который может произойти. Хвост этого списка содержит будущие события, в их порядок возникновения. Всякий раз, когда первый случай моделируется обработанным, это принято от списка. В моделировании этого случая, могут быть созданы новые события. Эти новые события вставлены в список случая

в соответствующих местах. В Рис.5. мы показана диаграмма потока действий, которые будут выполнены.

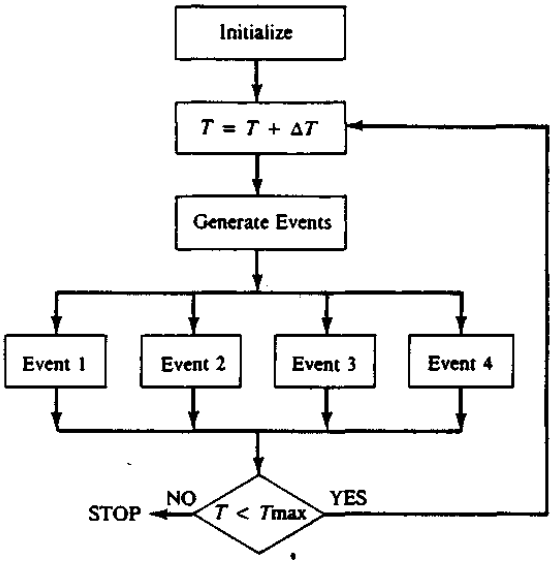

Рис.5.: Действия в моделировании на основе времени

Выбрав для подхода на основе случая к моделированию, можно все еще иметь различные формы выполнения. Выполнение может или ориентироваться на случай или ориентироваться на процесс. Прежнее выполнение может быть сделано с любым императивом, программирующим язык. Для последнего варианта, определение процессов, которые являются псевдопараллельными, необходимо.

С ориентируемым на случай выполнением есть процедура P_i определенный для каждого типа случая i, который может происходить. В имитации список случая считается определенным. После того, как инициализация этого случая вносит в список главные запуски петли управления. Первый случай принят от списка (обычно этот случай будет представлен головой список). Время моделирования увеличено к значению, в которое "текущий" случай произошел. Тогда, если этот случай имеет тип i, процедура P_i вызывается. В этой процедуре глобальное состояние изменено согласно возникновению случая, i, и новые события произведены и вставлены в список в соответствующих местах. После того, как процедура заканчивается "измерения" могут быть собраны, и затем, если некоторый критерий остановки еще не выполнен, главная петля управления продолжается с обработкой нового первого случая. После главных концов петли управления, статистика рассчитана и выпуск в подходящей форме.

Однако также может быть выбрана другая точка зрения. Можно связываться с каждым случаем, связывается процесс, и рассматривают действие системы как выполнения процессов случая, которые конечно связываются друг с другом. Планирование событий в моделировании тогда становится неявным в планировании процессов случая. Последний случай может быть сделан операционной системой или системой языка времени выполнения. Предпосылка для этого подхода то, что некоторая форма псевдопараллельного программирования поддержана языком выполнения. Типично требуемые запросы - подобно активизируют (обрабатывают) во времени, задержка (времени), и пассивность().

В обоих вариантах выполнения формируется критерий остановки, который может иметь различные формы. Нужно принять во внимание число событий, которые произошли, законченное-ли моделирование или управляли временем или шириной интервалов секретности, которые получены. Конечно, возможно множество комбинаций.

ВЫВОДЫ

В этой главе рассмотрены стохастические методы моделирования производительности компьютерных сетей. Произведен краткий обзор стохастических процессов. Также рассмотрены общие стохастические модели M|G|1 и их модели с прибытием партии пачек и модели с приоритетами. Рассмотрены также имитационное моделирование производительности компьютерных сетей. Идея имитационного моделирования, классификация моделирования и дискретное моделирование случая.

ГЛАВА 3. СТРУКТУРА ПРОГРАММНОГО КОМПЛЕКСА ПОВЫШЕНИЯ ПРОИЗВОДИТЕЛЬНОСТИ КОМПЬЮТЕРНЫХ СЕТЕЙ

1. Архитектура программного комплекса повышения производительности компьютерных сетей на основе стохастических моделей

Принципиальная структура программного комплекса. Данный программный комплекс повышения производительности коммуникационных сетей на основе стохастических моделей представляет собой интегрированную среду, предоставляющую доступ к любому разделу данной магистерской диссертации, а именно:

- общей теоретической информации, посвященной обзору текущей ситуации в сфере оптимизации коммуникационных сетей и их составных элементов;
- подробной технической документации (преимущественно посвященной оптимизации работы серверов коммуникационных сетей), на которой и основывается данная диссертация;
- расчетной части данной работы, включающей в себя 5 разделов по различны тематикам и предоставляющей возможность эмпирическим путем убедиться в том или ином прогнозируемом поведении сети при наборе определенных параметров;
- демонстрационной части программного комплекса, выполненной на основе 2-х и 3-х мерной анимации и дающей наглядное представление об основных принципах той проблемы, которой посвящены ее части;
- подробной теоретической информации, посвященной обзору состояния современных средств разработки программного обеспечения и вспомогательных средств создания и внедрения 2-х и 3-х мерной анимации, которые были использованы при создании рассматриваемого программного комплекса;
- обобщенной информации, раскрывающей принципиальную структуру данного программного комплекса и некоторые его специфические особенности.

Ниже приведена принципиальная схема данного программного комплекса, позволяющая в общих чертах увидеть взаимозависимость модулей и возможность переходов между ними.

2. Выбор и обоснование программных средств для реализации программного комплекса повышения производительности компьютерных сетей на основе стохастических моделей

Комплекс визуального объектно–ориентированного программирования C++Builder v5.0

Общие характеристики и особенности применения. Новейшая система объектно-ориентированного программирования C++ Builder производства корпорации Borland предназначена для операционных систем Windows NT. Интегрированная среда C++ Builder обеспечивает скорость визуальной разработки, продуктивность повторно используемых компонент в сочетании с мощью языковых средств C++, усовершенствованными инструментами и разномасштабными средствами доступа к базам данных.

C++ Builder может быть использован везде, где требуется дополнить существующие приложения расширенным стандартом языка C++, повысить быстродействие и придать пользовательскому интерфейсу качества профессионального уровня.

Рассмотрим отличительные характеристики, которые выдвигают Borland C++ Builder на авангардные позиции систем объектно-ориентированного программирования для быстрой разработки современного математического обеспечения персональных компьютеров. Borland C++ Builder выпускается в трех вариантах: Standard, Professional и Client/Server Suite.

Характеристики	*Standard*	*Professional*	*Client/Server Suite*
Язык C++ с поддержкой расширений стандартов ANSI/ISO	*V*	*V*	*V*
Высокопроизводительный 32-разрядный оптимизирующий	*V*	*V*	*V*

Характеристики	*Standard*	*Professional*	*Client/Server Suite*
компилятор			
Быстрый инкрементальный компоновщик приложений	*V*	*V*	*V*
Интегрированная среда разработки IDE	*V*	*V*	*V*
Механизмы двунаправленной разработки	*V*	*V*	*V*
Интегрированный отладчик CPU View и инструменты командной строки	*V*	*V*	*V*
Создание библиотек DLL, LIB и исполняемых программных файлов EXE	*V*	*V*	^
Полный комплект общих элементов управления Windows 95	*V*	*V*	*V*
Объекты модулей данных	*V*	*V*	*V*
Полный доступ к Windows API	*V*	*V*	*V*
Хранилище объектов	*V*	*V*	*V*
Визуальное наследование форм	*V*	*V*	*V*
Контроллеры и серверы OLE Automation	*V*	*V*	*V*
Библиотека Визуальных Компонент VCL	*V*	*V*	*V*
Мастер для помощи в создании компонент	*V*	*V*	*V*
Компоненты для работы с базами данных	*V*	*V*	*V*
Расширенная справочная служба on-line	*V*	*V*	*V*

Характеристики	*Standard*	*Professional*	*Client/Server Suite*
Проводник баз данных	*V*	*V*	*V*
Компоненты для создания отчетов	*V*	*V*	*V*
Исходные тексты VCL		*V*	*V*
Масштабируемый словарь баз данных		*V*	*V*
Поддержка соединений ODBC		*V*	*V*
Компонента визуализации и редактирования баз данных Multi-Object Grid		*V*	*V*
32-разрядный однопользовательский сервер Local InterBase		*V*	*V*
Полный комплект печатной документации		*V*	*V*
Генератор дистрибутивов InstallShield Express		*V*	*V*
Internet Solutions Pack для разработки Web-приложений		*V*	*V*
WinSight32 для мониторинга сообщений Windows		*V*	*V*
Открытые инструменты API		*V*	*V*
Расширенный набор компонент ActiveX		*V*	*V*
Драйверы SQL Links для баз данных Oracle, Sybase, Informix, DB2, Microsoft SQL Server и InterBase			*V*
SQL Database Explorer			*V*
SQL Monitor			*V*
Visual Query Builder			*V*
Интегрированная система контроля			*V*

Характеристики	*Standard*	*Professional*	*Client/Server Suite*
версий в коллективных проектах Intersolv PVCS			
InterBase NT			*V*
Механизм кэшируемых обновлений			*V*

Скорость визуальной разработки. *Интегрированная среда разработки* объединяет Редактор форм. Инспектор объектов. Палитру компонент. Администратор проекта и полностью интегрированные Редактор кода и Отладчик - инструменты быстрой разработки программных приложений, обеспечивающие полный контроль над кодом и ресурсами.

Профессиональные средства языка C++ интегрированы в визуальную среду разработки. C++Builder предоставляет быстродействующий компилятор с языка Borland C++, эффективный инкрементальный загрузчик и гибкие средства отладки как на уровне исходных инструкций, так и на уровне ассемблерных команд - в расчете удовлетворить высокие требования программистов-профессионалов.

Конструирование по способу "drag-and-drop " позволяет создавать приложение простым перетаскиванием захваченных мышью визуальных компонент из Палитры на форму приложения. Инспектор объектов предоставляет возможность оперировать со свойствами и событиями компонент, автоматически создавая заготовки функций обработки событий, которые наполняются кодом и редактируются в процессе разработки.

Механизмы двунаправленной разработки (two-way-tools) устраняют барьеры между программистом и его кодом. Технология двунаправленной разработки обеспечивает контроль за вашим кодом посредством гибкого, интегрированного и синхронизированного взаимодействия между инструментами визуального проектирования и Редактором кода.

Свойства, методы и события - это именно те элементы языка, которые обеспечивают быструю разработку приложений в рамках объектно-ориентированного программирования. *Свойства* позволяют легко устанавливать разнообразные характеристики объектов. *Методы* производят определенные, иногда довольно сложные, операции над объектом. *События*

связывают воздействия пользователя на объекты с кодами реакции на эти воздействия. События могут возникать при таких специфических изменениях состояния объектов как обновление данных в интерфейсных элементах доступа к базам данных. Работая совместно, свойства, методы и события образуют среду *RAD (Rapid Application Development)* быстрого и интуитивного программирования надежных приложений для Windows.

Визуальное наследование форм воплощает важнейший аспект объектно-ориентированного программирования в удобном для пользования инструменте визуального проектирования. Характеристики новой формы приложения могут быть унаследованы от любой другой существующей формы, что обеспечивает централизованную репродукцию изменений пользовательского интерфейса, облегчает контроль за кодом и уменьшает временные затраты на введение новых качественных атрибутов.

Испытание прототипа позволяет без труда переходить от прототипа приложения к полностью функциональному, профессионально оформленному программному продукту, действуя в пределах интегрированной среды. Чтобы удостовериться, что ваша программа производит ожидаемые результаты, раньше приходилось многократно проходить по циклу редактирование => компиляция => сборка, непроизводительно расходуя время. C++Builder объединяет три этапа разработки в единый производственный процесс. В результате удается строить приложения, базирующиеся на текущих требованиях заказчика, вместе с тем гибкие настолько, чтобы быстро адаптировать их к новым запросам пользователей.

Мастер инсталляции руководит созданием унифицированных дистрибутивных пакетов для разработанных приложений.

Исходные тексты Библиотеки Визуальных Компонент облегчают разработку новых компонент на базе готовых примеров.

Отрытые инструменты API могут быть непосредственно интегрированы в визуальную среду системы. Вы сможете подключить привычный текстовый редактор или создать собственного мастера для автоматизации выполнения повторяющихся процедур.

Расширенная математическая библиотека содержит дополнительные унифицированные функции статистических и финансовых вычислений.

Продуктивность компонент. *Библиотека Визуальных Компонент VCL* приобрела статус нового промышленного стандарта и в настоящее время применяется более чем полумиллионом пользователей, существенно ускоряя разработку надежных приложений любой степени сложности. VCL содержит около 100 повторно используемых компонент, которые реализуют все элементы пользовательского интерфейса операционной системы Windows. Кроме того, VCL предоставляют в распоряжение программистов такие оригинальные объекты, как записные книжки с закладками, табличные сетки для отображения содержимого баз данных и даже органы управления устройствами мультимедиа. Находясь в среде объектно-ориентированного Программирования C++Builder, компоненты можно использовать непосредственно, менять их свойства, облик и поведение или порождать производные элементы. обладающие нужными отличительными характеристиками.

Хранилище объектов является инструментом новой методики хранения и повторного использования модулей данных, объектов, форм и программной бизнес-логики. Поскольку построение нового приложения на существующем фундаменте значительно экономит временные затраты, хранилище объектов предоставляет для повторного использования готовые структуры: формы и законченные программные модули. Создавая прототип нового приложения, вы можете наследовать, ссылаться или просто копировать существующую структуру - точно так же архитектор приступает к проектированию нового здания.

Компонента ChartFX обеспечивает немедленное построение на вашей форме разнообразных графиков, диаграмм, таблиц и предусматривает проверку правописания на многих языках. В варианте C++Builder Standard эта компонента является единственным представителем группы ActiveX.

Интеграция компонент ActiveX позволяет расширить Библиотеку Визуальных Компонент, включив компоненты стандарта ActiveX для разработки приложений в сети Internet.

Мощность языковых средств C++. *Оптимизирующий 32-разрядный компилятор* построен по проверенной ведущей компиляторной технологии корпорации Borland, обеспечивающей исключительно падежную и быструю оптимизацию как длины выходного исполняемого кода, так и расходуемой памяти.

Новые элементы стандарта ANSI/ISO языка C++ представлены шаблонами, пространствами имен, исключениями, информацией о типах времени выполнения (RTTI), наряду с расширением набора ключевых слов bool, explicit, mutable, typename. automated и др.

Инкрементальный линкер осуществляет быструю и надежную сборку приложении в формате EXE файлов сравнительно меньшего размера. Автоматически устраняя повторную сборку не изменившихся исходных объектных файлов и подключение неиспользуемых функций, инкрементальный линкер строит эффективную выполняемую программу с минимальными потерями времени.

Чистый и доступный код приложений, которые C++Builder строит на основе предоставляемых разработчику компонентных свойств, событий и методов, исключает скрытые и трудные в отладке макросы.

Поддержка промышленных стандартов ActiveX, OLE, COM, MAPI, Windows Sockets TCP/IP, ISAPI. NSAPI, ODBC, Unicode и MBCS.

Отладчик низкого уровня CPU View позволяет проникнуть в специфику работы вашего приложения на уровне машинных кодов. Окно отладчика разделено на пять панелей. Панель ассемблерных команд интерпретирует исполнение исходной C++ программы. Панель памяти показывает содержимое блока памяти, доступного загруженному и исполняемому в данный момент модулю. Панель стека отображает текущее содержимое верхушки программного стека. Панель регистров и панель флагов показывают текущие значения регистров и служебных битов центрального процессора. Каждая панель включает собственное меню, управляющее ее видом и поведением.

Инструменты командной строки включены в систему по требованию профессионалов, которые всегда стремятся сохранить детальный контроль над процессами компиляции и сборки своих программных файлов.

Создание DLL, LIB, и EXE файлов предоставляет свободу выбора формата целевого приложения в соответствии с требованиями конкретного проекта.

Прямое обращение к системным функциям Windows NT дает возможность программистам, работающим в среде C++Builder. при необходимости воспользоваться всеми усовершенствованиями современных операционных систем.

Механизм OLE Automation предоставляет вашему приложению возможность управлять другими типовыми программными комплексами для Windows (такими как Microsoft Word, Excel, Visual Basic, Lotus 1-2-3, dBASE и Paradox) по схеме сетевого взаимодействия контроллер/сервер.

Масштабируемые соединения с базами данных. *Разработка по способу "drag-and-drop"* многократно упрощает и ускоряет обычно трудоемкий процесс программирования СУБД в архитектуре клиент/сервер. Широкий выбор компонент управления визуализацией и редактированием позволяет легко изменять вид отображаемой информации и поведение программы. C++Builder использует *Проводник баз данных (Database Explorer)* и масштабируемый *Словарь данных (Data Dictionary),* чтобы автоматически настроить средства отображения и редактирования применительно к специфике вашей информации.

Проводник баз данных предоставляет графический способ проводки пользователя по содержимому базы данных, обеспечивая создание и модификацию таблиц, иерархических указателей и псевдонимов.

Словарь данных поддерживает целостность изменяющейся информации о содержимом таблиц баз данных. Пользователь может динамически модифицировать состав Словаря. Словарь содержит информацию о расширенных атрибутах полей в записях: минимальные и максимальные значения, свойства отображения, маски редактирования и т.п.

Живые данные (live data) предоставляются разработчику в процессе визуального проектирования прототипов и при испытании приложений баз данных. Вам не потребуется более писать тестовые ловушки или многократно перетранслировать и запускать приложение - данные на стадии проектирования будут точно такими же и представлены точно так же, как их увидит пользователь законченной программы.

Механизм BDE (Borland Database Engine) поддерживает высокопроизводительный 32-разрядный доступ к базам данных dBASE, Paradox: Sybase. Oracle, DB2. Microsoft SQL Server. Informix, InterBase и Local InterBase. C++Builder использует контроллер *ODBC (Open Database Connectivity)* производства Microsoft для связи с серверами баз данных Excel, Access, FoxPro и Btrieve. Являясь фундаментом любого приложения базы данных, BDE тесно связан с Хранилищем объектов и Модулями данных.

Объекты Модулей данных действуют как связующий каркас приложения - они определяют источники и бизнес-логику базы данных, фиксируют взаимосвязи компонент. В централизованной модели доступа к данным бизнес-логика отделена от разработки графического интерфейса с пользователем (GUI). Любое изменение бизнес-логики вашей базы данных сказывается на поведении только соответствующего Модуля данных, а результаты изменения проявляются немедленно во всех приложениях, использующих данный модуль. Работая с модулями данных, вы однократно устанавливаете связи вашего приложения с адресуемой базой данных, а затем по способу "drag-and-drop" можете перетаскивать поля записей на новые формы - в любой узел вашей сети. Никакого дополнительного кодирования при этом не требуется.

Фильтры поля ссылок устанавливают ограничения поиска и отображения информации базы данных простым нажатием кнопки. Изменяя значения свойства Filter в компонентах доступа, можно специфицировать некоторое подмножество интересующих вас данных. Ссылки обеспечивают автоматическое отображение данных из нескольких таблиц.

Копируемые обновления (cached updates) заметно ускоряют отклик SQL сервера за счет уменьшения общего числа сетевых обменов с клиентом. Будучи упакованными, множественные коммуникации проявляют себя как одиночные транзакции, тем самым, снижая загруженность сервера и улучшая производительность вашего приложения.

Отчеты Quick Reports позволяют визуально конструировать стилизованные отчеты по данным, поставляемым любым источником, включая таблицы и запросы компонент доступа к базам данных. Отчеты могут содержать поля заголовков, колонтитулов, сносок и итогов. Quick Reports предоставляют мощные средства отображения отчетов в разных видах, автоматического подведения итогов и подсчета полей - на любом уровне группировки данных.

Палитра компонент - краткий обзор. Компоненты вкладки Standard. Компоненты этой вкладки осуществляют включение в ваше приложение следующих типовых интерфейсных элементов Windows:

TMainMenu	Создает панель команд главного меню для формы.
TPopUpMerm	Создает "выскакивающее" меню для формы или для другой компоненты.
TLabel	Отображает на форме текст названия, который нельзя редактировать.
TEdit	Отображает область редактируемого ввода одиночной строки информации на форме.
TIVlemo	Отображает область редактируемого ввода множественных строк информации на форме.
TButton	Создает кнопку с надписью.
TCheckBox	Создает элемент управления с двумя состояниями.
TRadioButton	Создает элемент управления с двумя состояниями.
TListBox	Отображает область списка текстовых строк.
TComboBox	Создает комбинацию области редактирования и выпадающего списка текстовых строк.
TScrollBar	Создает линейку прокрутки для просмотра содержимого окна, формы, списка или диапазона значений.
TGroupBox	Создает контейнер, объединяющий на форме логически связанную группу некоторых компонент.
TRadioGroup	Создает контейнер, объединяющий на форме группу логически взаимоисключающих радио-кнопок.
TPanel	Создает панель инструментов или строк состояния.

Компоненты вкладки Win. Компоненты этой вкладки осуществляют включение в ваше приложение следующих типовых интерфейсных элементов Windows:

TTabControl	Отображает набор полей, имеющих вид частично перекрывающих друг друга картотечных вкладок.
TPageControl	Отображает набор полей, имеющих вид частично перекрывающих друг друга картотечных вкладок, для организации многостраничного диалога.
TTreeView	Отображает древовидный перечень элементов - заголовков документов, записей в указателе, файлов или каталогов на диске.
TListView	Отображает древовидный перечень элементов в различных видах - по столбцам с заголовками, вертикально, горизонтально, с пиктограммами.
TImageList	Создает контейнер для группы изображений.
THeaderControl	Создает контейнер для заголовков столбцов.
TRichEdit	Отображает область редактируемого ввода множественных строк информации в формате RTF.
TStatusBar	Создает строку панелей состояния для отображения статусной информации.
TTrackBar	Создает шкалу с метками и регулятором текущего положения.
TProgressBar	Создает индикатор процесса выполнения некоторой процедуры в приложении.
TUpDown	Создает спаренные кнопки со стрелками "вверх" и "вниз". Нажатие этих кнопок вызывает увеличение или уменьшение значения свойства Position.
THotKey	Используется для установки клавиш быстрого вызова во время выполнения программы.

Компоненты вкладки Additional. Компоненты этой вкладки осуществляют включение в ваше приложение следующих специализированных интерфейсных элементов Windows:

TBitBtn	Создает кнопку с изображением битового образа.
TSpeedButton	Создает графическую кнопку быстрого вызова.
TMaskEdit	Создает область редактируемого ввода данных специфического формата.
TStringGrid	Создает сетку для отображения строк по строкам или столбцам.
TDrawGrid	Создает сетку для отображения графических данных по строкам или столбцам.
TImage	Создает на форме контейнер для отображения битового образа, пиктограммы или метафайла.
TShape	Рисует простые геометрические фигуры.
TBevel	Создает линии и рамки с объемным видом.
TScrollBox	Создает контейнер переменного размера с линейками прокрутки, если это необходимо

Использование и создание визуальных компонент. Библиотека Визуальных Компонент была впервые введена системой программирования Delphi 1.0 на языке Объектный Паскаль и модифицирована в Delphi для поддержки 32-разрядных приложений. Delphi оказалась наиболее популярной на рынке систем быстрой разработки программных приложений, однако, многие потребители высказывали интерес к подобной системе для языка C++, которая в конце концов и воплотилась в C++Builder. C++Builder унаследовал версию Библиотеки Визуальных Компонент Delphi 2.0 без каких-либо изменений.

Библиотека VCL интегрирована в среду C++Builder, что, в отличие от других систем программирования, позволяет манипулировать классами визуальных компонент при проектировании приложения, на стадии создания его прототипа. Поведение и вид ваших компонент определяются по мере

разработки приложения, хотя можно модифицировать их и в процессе выполнения программы.

Назначение и устройство VCL. Библиотека Визуальных Компонент позволяет программистам визуально создавать программные приложения, не прибегая более к кодированию классов "вручную", или кодированию в рамках стандартных библиотек MFC (Microsoft Foundation Class), или OWL (Object Windows Library).

C++ программистам теперь не надо создавать или манипулировать объектами интерфейса с пользователем путем написания соответствующего кода. Подавляющее большинство приложений вы будете разрабатывать визуально с помощью Редактора форм C++Builder, добавляя лишь несколько строк к .обработчикам ключевых событии компонент. Используйте объекты всегда, когда это возможно; твердо сопротивляйтесь позыву написать новый код то тех пор, пока все другие возможности не будут исчерпаны.

VCL для прикладных программистов. Программист создает законченное приложение посредством интерактивного взаимодействия с интегрированной визуальной средой C++Builder, используя компоненты VCL для создания интерфейса программы с пользователем и с другими управляющими элементами: обслуживания баз данных, контролируемого ввода параметров и т.д. Характерная для C++Builder методика визуального стиля разработки программного обеспечения не применяется множеством других систем программирования.

Программисты должны знать свойства, методы и события, присущие используемым компонентам. Более того, понимание архитектуры VCL позволяет совершенствовать вашу программу в тех местах, где ощущается необходимость развития существующих или создания новых компонент. Прежде, чем изобретать новый элемент, удостоверьтесь, как принято, не создал ли уже кто-то компоненту с нужными вам характеристиками.

VCL для системных программистов. Системные программисты развивают существующую Библиотеку — либо добавляя в нее новые элементы, либо расширяя функциональность уже имеющихся компонент. Разработчики компонент должны иметь более глубокие знания о внутреннем устройстве VCL, нежели прикладные программисты. Нужно четко представлять себе, какой прием быстрее приведет к поставленной цели: развитие имеющейся или написание новой компоненты. Написание

компонент представляет собой более традиционную задачу программирования и сопряжено с большими условностями, нежели визуальное создание приложений.

Варианты C++Builder Professional и C++Builder Client/Server Suite поставляются вместе с исходными текстами VCL. Наличие исходных текстов облегчает задачу программистов, которые занимаются разработкой новых компонент и расширением функциональных возможностей уже имеющихся компонент Библиотеки.

Для создания новых компонент можно с одинаковым успехом пользоваться средствами C++Builder или Delphi, однако если разработанные компоненты предлагаются для внешнего применения, автор обязан удостовериться, что они работают в рамках обеих систем.

Компоненты VCL. Компоненты — это строительные кирпичи, из которых конструируется интерфейс программы с пользователем, с помощью которых "здание" программы приобретает новый внешний облик и скрытые особенности. Для прикладного программиста любая компонента VCL представляет собой объект, который можно "перетащить" из вкладок Палитры компонент (Рис. 6.1) на форму создаваемого приложения. Поместив компоненту на форму, можно манипулировать ее свойствами (посредством Редактора форм) и кодом (с помощью Редактора кода), придавая компоненте специфическое поведение.

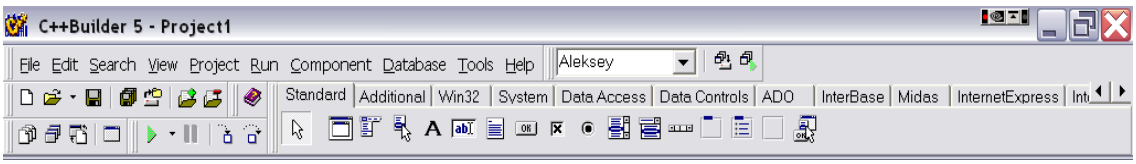

Рис. 1. Палитра компонент.

3. Основные показатели повышения производительности компьютерных сетей и алгоритмы их реализации

Всякий раз, когда должна быть построена новая система, или реконструирована уже имеющаяся, анализ таких сетей может использоваться для предсказывания влияния архитектурных или других изменений на

характер выполнения работ и протекания процессов в коммуникационных сетях.

В зависимости от модели, можно непосредственно вычислять важные параметры производительности выполнения работ коммуникационных сетей, используя так называемые неявные формулы. Эти аналитические модели конечно очень удобны, но большинство реальных систем не могут быть смоделированы таким способом.

Из наиболее важных количественных показателей, характеризующих производительность компьютерных сетей, можно выделить в первую очередь:

- Время между последовательным прибытием запросов клиентов, требующих обслуживание. Эти интервалы часто принимаются, что они подчиняются некоторому стохастическому распределению.
- Характеристика потока клиентов. Мы имеем дело с бесконечным или с конечным потоком клиентов?
- Число одновременно прибывающих клиентов. Клиенты прибывают один за другим или имеют взрывной характер?
- Количество обслуживания запросов клиента. Это вообще также описывается некоторой стохастической переменной.
- Количество обслуживающих устройств (звеньев, серверов и.т.д.), которое является доступным. Мы имеем дело с единственным обслуживающим устройством или с мультиустройством?
- Путь, по которым поступают клиенты, чтобы получить их требуемое обслуживание. Обычно используемые принципы планирования:
  - FCFS: First Come, First Served: (Первый прибывший обслуживается первым)
  - PS: Processor Sharing: (Разделяющийся Процессор)
  - LCFS (PR): Last Come, First Served or without Preemption: (Последний прибывший обслуживается первым или без высвобождения)
  - IS: Infinite Servers: Бесконечные серверы;
  - PRIO:Prioritititized scheduling: (Расположенное по приоритетам планирование).

- объем буфера ожидания, который является доступным для клиентов, которые непосредственно не могут обслуживаться в данный момент.

В разделах 2-й главы данной диссертации приводятся основные алгоритмы, т.е. формулы, характеризующие стохастические процессы, по которым рассчитываются вышеприведенные показатели эффективности работы компьютерных сетей. В частности подробно рассмотрены такие процессы, как:

- Цепи Маркова с дискретным временем
- Цепи Маркова с непрерывным временем
- Процесс “рождения и смерти”
- M|M|1: Постоянные нормы интенсивности
- M|M|l|m: Системы с потерями
- M|M|m: Мультисерверные модели
- M|M| ∞ : Модели с бесконечным числом серверов
- M|M|1|K: Конечное число пачек
- Модели с циклическим сервером
- Циклический сервер модели token ring
- Общие стохастические модели M|G|1.
- M|G|1 модели с прибытием партии пачек
- M|G|1 модели с приоритетами

Когда изучаемые системы очень сложны, они не могут быть смоделированы единственными отдельными FCFS станциями обслуживания.

В этих случаях, например, в случае циклической системы обслуживающих звеньев-серверов каждый сервер обслуживает множество очередей, следовательно, систему обслуживания с прерываниями можно моделировать с помощью реальных систем. В ряд сложных можно отнести систему обслуживания с приоритетами.

5. Описание программного комплекса

Данный программный комплекс повышения производительности коммуникационных сетей на основе стохастических моделей состоит из 8 модулей, связанных между собой взаимной зависимостью:

- 6 расчетных модулей;
- 2 демонстрационных модуля;
- Файлы в формате html, представляющие собой соответствующие разделы данной магистерской диссертации;

Расчетные модули

Цепи Маркова с дискретным временем (DTMC). Экранная форма, отвечающая за расчеты параметров цепей Маркова с дискретным временем содержит:

- поля ввода соответствующих данных;
- кнопку начала расчетов по введенным пользователем параметрам;
- кнопку возврата в главное меню;
- кнопку выхода из программы;
- поля вывода рассчитанных предварительных значений;
- поля вывода рассчитанных окончательных значений;
- анимированные графические объекты, конкретизирующие, к какому классу относятся данные расчеты.

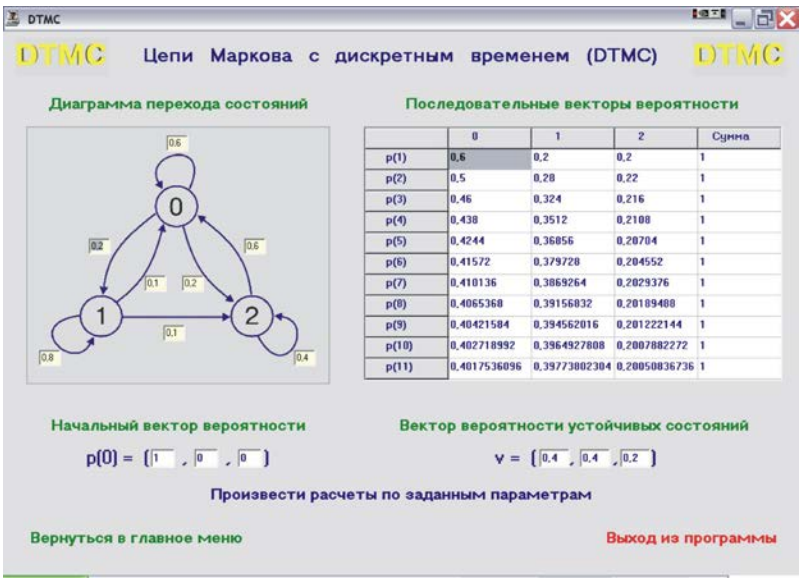

Рис. 2. Цепи Маркова с дискретным временем (DTMC)

M|M|1 – Постоянные нормы интенсивности. Экранная форма, отвечающая за расчеты вероятности одновременного нахождения в обслуживании определенного количества пачек, содержит:

- поля ввода соответствующих данных;

- кнопку начала расчетов по введенным пользователем параметрам;
- кнопку возврата в главное меню;
- кнопку выхода из программы;
- поля вывода рассчитанных предварительных значений;
- поля вывода рассчитанных окончательных значений;
- анимированные графические объекты, конкретизирующие, к какому классу относятся данные расчеты.

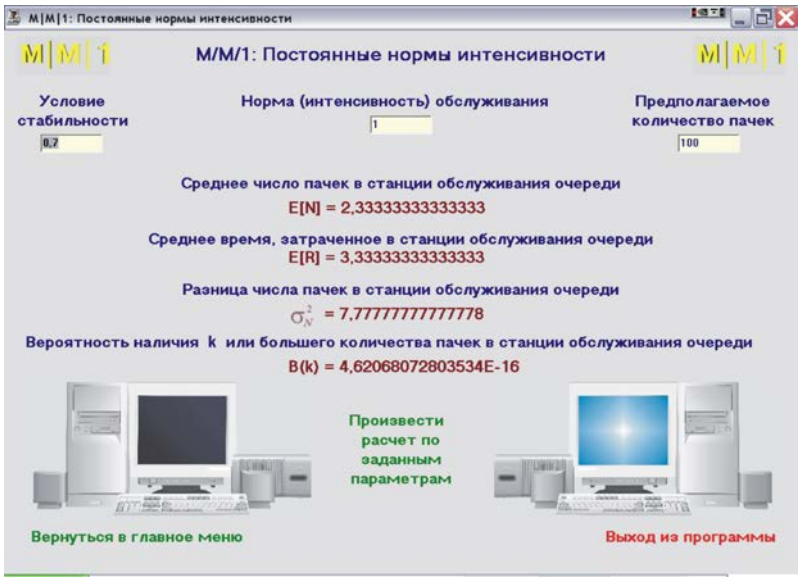

Рис. 3. M|M|1 – Постоянные нормы интенсивности

M|M|m – Мультисерверные модели. Экранная форма, отвечающая за расчеты параметров цепей Маркова модели M|M|m|m с m серверов, содержит:

- поля ввода соответствующих данных;
- кнопку начала расчетов по введенным пользователем параметрам;
- кнопку возврата в главное меню;
- кнопку выхода из программы;
- поля вывода рассчитанных предварительных значений;
- поля вывода рассчитанных окончательных значений;
- анимированные графические объекты, конкретизирующие, к какому классу относятся данные расчеты;

- бегущую строку, содержащую инструкции пользователю при использовании данного расчетного модуля;
- диаграмму гистограммного характера, более наглядно демонстрирующую полученные результаты и дающую возможность их немедленного сравнения.

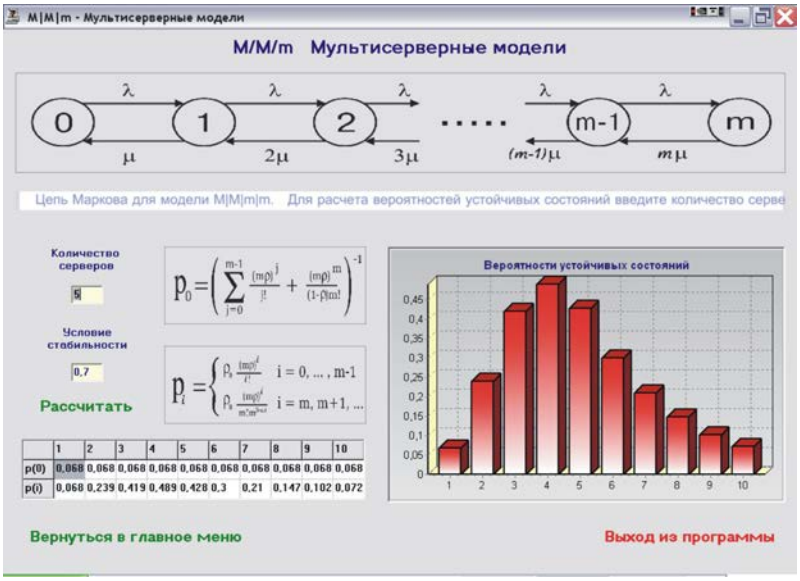

Рис. 4. M|M|m|m – Мультисерверные модели

M|G|1 –Модели с прибытием пачек. Экранная форма, отвечающая за расчеты среднего времени ожидания пачек до их обслуживания содержит:

- поля ввода соответствующих данных;
- кнопку начала расчетов по введенным пользователем параметрам;
- кнопку возврата в главное меню;
- кнопку выхода из программы;
- поля вывода рассчитанных предварительных значений;
- поля вывода рассчитанных окончательных значений;
- анимированные графические объекты, конкретизирующие, к какому классу относятся данные расчеты;
- диаграмму гистограммного характера, более наглядно демонстрирующую полученные результаты и дающую возможность их немедленного сравнения.

Рис. 5. M|G|1 –Модели с прибытием партии пачек

M|G|1 –Модели с приоритетами. Экранная форма, отвечающая за расчеты среднего времени ожидания пачек до их обслуживания в зависимости от имеющегося системного приоритета содержит:

- поля ввода соответствующих данных;
- кнопку начала расчетов по введенным пользователем параметрам;
- кнопку возврата в главное меню;
- кнопку выхода из программы;
- поля вывода рассчитанных предварительных значений;
- поля вывода рассчитанных окончательных значений;
- анимированные графические объекты, конкретизирующие, к какому классу относятся данные расчеты;
- диаграмму гистограммного характера, более наглядно демонстрирующую полученные результаты и дающую возможность их немедленного сравнения.

Рис. 6. M|G|1 –Модели с приоритетами

5. Инструкции по применению программного комплекса

Инструкции пользователю

Данный программный комплекс повышения производительности коммуникационных сетей на основе стохастических моделей представляет собой интегрированную среду, предоставляющую доступ к любому разделу данной магистерской диссертации.

Необходимо отметить, что в данной программе отсутствуют кнопки, привычные пониманию обычного пользователя. Они заменены аналогами Web-стиля, то есть каждая надпись является ссылкой на документ, если при наведении курсора в ее область он видоизменяется и принимает вид указателя Web – руки с отогнутым указательным пальцем.

В поля ввода данных необходимо вводить ТОЛЬКО числовые значения, иначе будет сгенерирована ошибка и вычисления не будут произведены. Ввод чисел с плавающей запятой производится с учетом особенностей вычислительной системы компьютера пользователя – через точку или запятую.

Инструкции программисту

Данный программный комплекс разработан при использовании принципов объектно-ориентированного программирования и не содержит четкой логической структуры, поскольку основа любых действий – событийность.

Программный комплекс состоит из следующих модулей и разделов:

- DTMCpr.cpp – модуль проекта;
- DTMC.cpp – модуль расчетной части «Цепи Маркова с дискретным временем»;
- M_G_1.cpp – модуль расчетной части «M|G|1 – модели с прибытием пачек»;
- M_G_1_Prior.cpp - модуль расчетной части «M|G|1 – модели с приоритетами»;
- M_M_1_Const.cpp - модуль расчетной части «M|M|1 – постоянные нормы интенсивности»;

- M_M_m.cpp - модуль расчетной части «M|M|m – Мультисерверные модели»;
- Menu.cpp – модуль главного меню;
- TSTULan.cpp – модуль вывода корпоративной сети передачи данных;
- Start.exe – заставка программного комплекса;
- Menu.exe – файл запуска программного модуля;
- Директория «Html» - набор всех документов, предоставляемых в данной диссертации;
- Директория «Photos» - набор всех статических графических объектов, используемых в данном программном комплексе.

6. Рекомендации по использованию программного комплекса повышения производительности компьютерных сетей

Описанный в данной диссертации разработанный программный комплекс повышения производительности компьютерных сетей представляет собой мощный вспомогательный инструментарий, созданный для облегчения работ, связанных с проведением расчетов, направленных на прогнозирование поведения компьютерной сети и в частности ее серверов в различные моменты времени при наступлении условий, заданных конкретными значениями основных параметров, наиболее сильно влияющих на уровень производительности.

Таким образом, первое рекомендуемое применение данного программного обеспечения – это использование его в качестве вспомогательного средства, предназначенного преимущественно для проектировщиков и компьютерных сетей. Включенные в программный комплекс расчетные части дают возможность существенно сократить время, затрачиваемое на расчет основных параметров проектируемой компьютерной сети.

Программный комплекс и разработанные алгоритмы позволяют имитировать в реальном масштабе времени численные показатели производительности при определенном наборе значений параметров, влияющих на производительность компьютерных сетей:

- Интенсивность поступления заявок (запросов);
- Производительность (интенсивность обработки) серверного оборудования;
- Количество серверного оборудования;
- Объем буферного пространства серверов;
- Пропускная способность коммуникационной среды;
- Приоритеты поступающих заявок и т.д.

Более того, встроенные в комплекс диаграммы гистограммного типа помогают прямо в процессе работы наглядно оценить конкретные результаты от внесенных изменений и сравнить эффективность работы сети до и после внесенных изменений. Параллельно с этим пользователь получит не только усредненные значения параметров, но и конкретные значения, высчитанные с точностью до 8-9 знака, то есть до одной миллиардной доли, что представляет собой большую ценность при учете экономии времени работ.

Представленное программное обеспечение может быть использовано не только как расчетное средство, но также и в качестве демонстрационно-обучающего комплекса.

Основанием для этого является то, что в данный комплекс встроены анимированные объекты, наглядно демонстрирующие сущность того или иного процесса, а также предоставлены все формулы, по которым производятся соответствующие расчеты.

Более того, в процессе работы пользователь может получить доступ к любому разделу данной диссертационной работы, поскольку все пункты работы конвертированы в формат Web – страниц - html.

В связи с этой особенностью предоставляется дополнительная возможность использования этого комплекса в качестве электронного учебника с возможностью использования его при дистанционном образовании.

Таким образом, данный программный комплекс является многофункциональной и многоцелевой системой с возможностью применения в различных направлениях указанной сферы деятельности специалистов данного профиля.

ВЫВОДЫ

Эта глава полностью посвящена описанию программного комплекса, инструкции по применению программного комплекса.

А также даны рекомендации по использованию программного комплекса повышающего производительность компьютерных сетей.

ЗАКЛЮЧЕНИЕ

1. Проанализировано большое количество методов и программных средств повышения производительности коммуникационных сетей, а также выработаны основные задачи разработки этих методов и программных средств;
2. Изучены и разработаны стохастические методы моделирования производительности компьютерных сетей, основанные на Марковских моделях типов M|M|1 и M|G|1.
3. Для более широкого обзора рассматриваемого вопроса были также рассмотрены проблемы **и**митационного моделирования производительности компьютерных сетей.
4. Разработана структура программного комплекса повышения производительности компьютерных сетей на основе стохастических моделей, подробно описана его архитектура, а также приведены доказательства, на основании которых был сделан выбор программных средств для реализации данного программного комплекса.
5. Выделены основные показатели повышения производительности компьютерных сетей, такие как:
   - Интенсивность поступления заявок на обработку;
   - Интенсивность работы серверов (циклических, параллельных и т.д.);
   - Размер буферного пространства серверного оборудования;
   - Среднее время ожидания заявки в буфере;
   - Среднее время обработки заявки…

   и алгоритмы реализации;
6. Составлено описание всех составляющих модулей программного комплекса и выработаны подробные инструкции пользователям и программистам для облегчения их работы с разработанным программным обеспечением;
7. Выработаны рекомендации по использованию программного комплекса повышения производительности компьютерных сетей в повседневной жизни.

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Закон Республики Узбекистан "Об информатизации" от 11 декабря 2003 г. // www.ictcouncil.gov.uz\
2. Указ Президента Республики Узбекистан "О дальнейшем развитии компьютеризации и внедрении информационно-коммуникационных технологий"30.05. 2002 г. №УП-3080 //"Собрание законодательства Республики Узбекистан ", 2002 г., N 10, ст.77
3. Джордж Ф. Люггер «Искусственный интеллект: стратегии и методы решения сложных проблем», Издательский дом «Вильямс», 2005г.
4. Сетевое программирование для профессионалов-Кравчик.Э,Кумар.В , Лагари.Н,Мунгале.А,Паркер.Т
5. Microsoft.NET Remoting- Маклейн.С, Вильямс.К, Нафтел.Д
6. K. Kant, Introduction to Computer System Performance Evaluation, McGraw-Hill, 2008.
7. L. Kleinrock, Queuing Systems; Vol. 1 and 2, John Wiley and Sons,2005.
8. Н.А. Олифер, В.Г. Олифер, Средства анализа и оптимизации локальных сетей, © Центр Информационных Технологий, 1998.
9. Р. Сардарян, Macromedia Flash, Москва, 2001.
10. www.citforum.ru - Словарь сетевых терминов.
11. В. Олифер, Н. Олифер., Высокоскоростные технологии ЛВС, © Центр Информационных Технологий, 1999.
12. В. Олифер, Н. Олифер., Транспортная подсистема неоднородных сетей, © Центр Информационных Технологий, 2001.
13. www.cisco.com - CISCO Internetworking Technology Overview.
14. М. Федотов, Системы сетевого/системного управления: принципы создания, Москва, 2003.
15. www.citforum.ru – Сетевые операционные системы.
16. www.citforum.ru – Локальные сети на основе коммутаторов.
17. www.microsoft.com - TCP/IP: protocol IPv.6

18. А. Архангельский, Программирование в C++Builder 5.0, Москва, 2001.
19. www.citforum.ru – VCL
20. J. Kiting, Flash MX, New York, 2003.
21. Болтаев Р.Д. “Администрирование брандмауэра” Актуальные вопросы в области технических и социально-экономических наук. Республиканский межвузовский сборник. Ташкент 2010г.
22. Бабамухамедова М.З., Болтаев Р.Д.-“Операцион тизимлар ва офис иловалари” фанини ўқитишда мультимедиа воситаларидан фойдаланиш. Алоқа ва ахборотлаштириш сохаси учун кадрлар тайёрлаш сифатини ошириш муаммолари. Тошкент 2012й.
23. Бабамухамедова М.З., Болтаев Р.Д.-“Анализ производительности компьютерных сетей”. Проблемы информационных технологий и телекоммуникаций. Сборник докладов. Ташкент 2012г.

ПРИЛОЖЕНИЕ

DTMC.cpp

```
//---------------------------------------------------------
#include <vcl.h>
#include <math.h>
#include <stdlib.h>
#pragma hdrstop
#include "ShellAPI.h"
#include "DTMC.h"
#include "M_M_m.h"
#include "M_G_1.h"
#include "Menu.h"
//---------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TDTMC_Form *DTMC_Form;
//---------------------------------------------------------
__fastcall TDTMC_Form::TDTMC_Form(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------
double m11,m12,m13,m21,m22,m23,m31,m32,m33,
      p01,p02,p03,p1,p2,p3,d,v1,v2,v3,ost;
void __fastcall TDTMC_Form::FormCreate(TObject *Sender)
{
DTMC_Form->StringGrid1->Cells[1][0] = "       0";
DTMC_Form->StringGrid1->Cells[2][0] = "       1";
DTMC_Form->StringGrid1->Cells[3][0] = "       2";
DTMC_Form->StringGrid1->Cells[4][0] = "    Сумма";
DTMC_Form->StringGrid1->Cells[0][1] = "      p(1)";
DTMC_Form->StringGrid1->Cells[0][2] = "      p(2)";
DTMC_Form->StringGrid1->Cells[0][3] = "      p(3)";
DTMC_Form->StringGrid1->Cells[0][4] = "      p(4)";
DTMC_Form->StringGrid1->Cells[0][5] = "      p(5)";
DTMC_Form->StringGrid1->Cells[0][6] = "      p(6)";
```

```
DTMC_Form->StringGrid1->Cells[0][7] = "      p(7)";
DTMC_Form->StringGrid1->Cells[0][8] = "      p(8)";
DTMC_Form->StringGrid1->Cells[0][9] = "      p(9)";
DTMC_Form->StringGrid1->Cells[0][10] = "      p(10)";
DTMC_Form->StringGrid1->Cells[0][11] = "      p(11)";
}
//---------------------------------------------------------
void __fastcall TDTMC_Form::Label8Click(TObject *Sender)
{
m11 = StrToFloat(DTMC_Form->Edit5->Text);
m12 = StrToFloat(DTMC_Form->Edit1->Text);
m13 = StrToFloat(DTMC_Form->Edit3->Text);
m21 = StrToFloat(DTMC_Form->Edit2->Text);
m22 = StrToFloat(DTMC_Form->Edit6->Text);
m23 = StrToFloat(DTMC_Form->Edit8->Text);
m31 = StrToFloat(DTMC_Form->Edit4->Text);
m32 = 0;
m33 = StrToFloat(DTMC_Form->Edit7->Text);
p01 = StrToFloat(DTMC_Form->Edit9->Text);
p02 = StrToFloat(DTMC_Form->Edit10->Text);
p03 = StrToFloat(DTMC_Form->Edit11->Text);
for (int i=1; i<12; i++) {
if (floor(p1+p2+p3)!=1)ShowMessage(AnsiString("Неверный набор параметров!"));
DTMC_Form->StringGrid1->Cells[1][i] = p1;
DTMC_Form->StringGrid1->Cells[2][i] = p2;
DTMC_Form->StringGrid1->Cells[3][i] = p3;
DTMC_Form->StringGrid1->Cells[4][i] = p1+p2+p3;
p01 = p1; p02 = p2; p03 = p3;
}
v2 = ceil(p2*100)/100;
v3 = 1-v1-v2;
DTMC_Form->Edit12->Text = v1;
DTMC_Form->Edit13->Text = v2;
DTMC_Form->Edit14->Text = v3;
}
//---------------------------------------------------------
void __fastcall TDTMC_Form::Label9Click(TObject *Sender)
```

```
{
DTMC_Form->Close();
Main_menu->Show();
}
//---------------------------------------------------
void __fastcall TDTMC_Form::Label10Click(TObject *Sender)
{
Main_menu->Close();
}
//---------------------------------------------------
void __fastcall TDTMC_Form::FormActivate(TObject *Sender)
{
DTMC_Form->Animate1->FileName = "AVI\\DTMC-1.avi";
DTMC_Form->Animate2->FileName = "AVI\\DTMC-2.avi";
DTMC_Form->Animate1->Active = true;
DTMC_Form->Animate2->Active = true;
}
//---------------------------------------------------
```

DTMC.h

```
//---------------------------------------------------------------------------
#ifndef DTMCH
#define DTMCH
//---------------------------------------------------------------------------
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
#include <Grids.hpp>
#include <Chart.hpp>
#include <Series.hpp>
#include <TeEngine.hpp>
#include <TeeProcs.hpp>
#include <Buttons.hpp>
#include <ComCtrls.hpp>
//---------------------------------------------------------------------------
```

```
class TDTMC_Form : public TForm
{
__published: // IDE-managed Components
        TImage *Image1;
        TEdit *Edit1;
        TEdit *Edit2;
        TEdit *Edit3;
        TEdit *Edit4;
        TEdit *Edit5;
        TEdit *Edit6;
        TLabel *Label1;
        TLabel *Label2;
        TLabel *Label3;
        TEdit *Edit9;
        TEdit *Edit10;
        TEdit *Edit11;
        TLabel *Label4;
        TStringGrid *StringGrid1;
        TLabel *Label5;
        TLabel *Label6;
        TLabel *Label7;
        TEdit *Edit12;
        TEdit *Edit13;
        TEdit *Edit14;
        TLabel *Label8;
        TLabel *Label9;
        TAnimate *Animate1;
        TAnimate *Animate2;
        void __fastcall FormCreate(TObject *Sender);
        void __fastcall Label8Click(TObject *Sender);
        void __fastcall Label9Click(TObject *Sender);
        void __fastcall Label10Click(TObject *Sender);
        void __fastcall FormActivate(TObject *Sender);
private:        // User declarations
public:                 // User declarations
        __fastcall TDTMC_Form(TComponent* Owner);
};
```

```
//---------------------------------------------------------------------------
extern PACKAGE TDTMC_Form *DTMC_Form;
//---------------------------------------------------------------------------
#endif
```

DTMCpr.cpp

```
//---------------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop
USERES("DTMCpr.res");
USEFORM("DTMC.cpp", DTMC_Form);
USEFORM("M_M_m.cpp", M_M_m_Form);
USEFORM("M_G_1.cpp", M_G_1_Form);
USEFORM("M_M_1_Const.cpp", M_M_1_Const_Form);
USEFORM("TstuLan.cpp", TstuLan_Form);
USEFORM("M_G_1_Prior.cpp", M_G_1_Prior_Form);
USEFORM("Sound.cpp", Music_Form);
//---------------------------------------------------------------------------
WINAPI WinMain(HINSTANCE, HINSTANCE, LPSTR, int)
{
      try
      {
            Application->Initialize();
            Application->Title = "Оптимизация сервера.";
            Application->CreateForm(__classid(TMain_menu), &Main_menu);
            Application->CreateForm(__classid(TDTMC_Form), &DTMC_Form);
            Application->CreateForm(__classid(TM_M_m_Form), &M_M_m_Form);
            Application->CreateForm(__classid(TM_G_1_Form), &M_G_1_Form);
            Application->CreateForm(__classid(TM_M_1_Const_Form),
&M_M_1_Const_Form);
            Application->CreateForm(__classid(TTstuLan_Form), &TstuLan_Form);
            Application->CreateForm(__classid(TM_G_1_Prior_Form),
&M_G_1_Prior_Form);
            Application->CreateForm(__classid(TMusic_Form), &Music_Form);
            Application->Run();
      }
```

```
        catch (Exception &exception)
        {
                Application->ShowException(&exception);
        }
        return 0;
}
//---------------------------------------------------------------------------
```

M_G_1.cpp

```
//---------------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop
#include "M_G_1.h"
#include "DTMC.h"
#include "math.h"
#include "Menu.h"
#include "M_M_m.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TM_G_1_Form *M_G_1_Form;
//---------------------------------------------------------------------------
double lm,ea,eh,es,p,v,t,w,ew1,ew2;
__fastcall TM_G_1_Form::TM_G_1_Form(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------------------------
void __fastcall TM_G_1_Form::Label7Click(TObject *Sender)
{
M_G_1_Form->Series1->Clear();
lm = StrToFloat(Edit1->Text);
ea = StrToFloat(Edit2->Text);
v = StrToFloat(Edit3->Text);
t = StrToFloat(Edit4->Text);
w = 1-exp(-v/1000);
```

```
eh = 1/w;
Label8->Caption = eh;
Label5->Caption = ew1;
Label6->Caption = ew2;
M_G_1_Form->Series1->Add(ew1,"E[W]-1",clRed);
M_G_1_Form->Series1->Add(ew2,"E[W]-2",clRed);
}
//---------------------------------------------------------------------------
void __fastcall TM_G_1_Form::Label13Click(TObject *Sender)
{
M_G_1_Form->Close();
Main_menu->Show();
}
//---------------------------------------------------------------------------
void __fastcall TM_G_1_Form::Label14Click(TObject *Sender)
{
Main_menu->Close();
}
//---------------------------------------------------------------------------
void __fastcall TM_G_1_Form::FormActivate(TObject *Sender)
{
M_G_1_Form->Animate1->FileName = "AVI\\MG1-1.avi";
M_G_1_Form->Animate2->FileName = "AVI\\MG1-2.avi";
M_G_1_Form->Animate1->Active = true;
M_G_1_Form->Animate2->Active = true;
}
//---------------------------------------------------------------------------
```

M_G_1.h

```
//---------------------------------------------------------------------------
#ifndef M_G_1H
#define M_G_1H
//---------------------------------------------------------------------------
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
```

```
#include <Forms.hpp>
#include <Chart.hpp>
#include <ExtCtrls.hpp>
#include <TeEngine.hpp>
#include <TeeProcs.hpp>
#include <Buttons.hpp>
#include <Series.hpp>
#include <jpeg.hpp>
#include <ComCtrls.hpp>
//---------------------------------------------------------------------------
class TM_G_1_Form : public TForm
{
__published: // IDE-managed Components
        TEdit *Edit1;
        TLabel *Label1;
        TLabel *Label2;
        TEdit *Edit2;
        TLabel *Label3;
        TEdit *Edit3;
        TLabel *Label4;
        TEdit *Edit4;
        TLabel *Label5;
        TLabel *Label6;
        TChart *Chart1;
        TLabel *Label8;
        TLabel *Label10;
        TBarSeries *Series1;
        TLabel *Label9;
        TLabel *Label11;
        TLabel *Label12;
        TImage *Image1;
        TImage *Image2;
        TLabel *Label7;
        TLabel *Label13;
        TLabel *Label14;
        TAnimate *Animate1;
        TAnimate *Animate2;
```

```
        void __fastcall Label7Click(TObject *Sender);
        void __fastcall Label13Click(TObject *Sender);
        void __fastcall Label14Click(TObject *Sender);
        void __fastcall FormActivate(TObject *Sender);
        void __fastcall FormCreate(TObject *Sender);
private:        // User declarations
public:                 // User declarations
        __fastcall TM_G_1_Form(TComponent* Owner);
};
//---------------------------------------------------------------------------
extern PACKAGE TM_G_1_Form *M_G_1_Form;
//---------------------------------------------------------------------------
#endif
```

M_G_1_Prior.cpp

```
//---------------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop
#include "M_G_1_Prior.h"
#include "Menu.h"
#include "Math.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TM_G_1_Prior_Form *M_G_1_Prior_Form;
//---------------------------------------------------------------------------
double ET0,lmbd1,lmbd2,ES1,ES2,sig1,sig2,
      lmbd,ro1,ro2,ro0,ES0,W,l1,l2;
__fastcall TM_G_1_Prior_Form::TM_G_1_Prior_Form(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------------------------
void __fastcall TM_G_1_Prior_Form::Label8Click(TObject *Sender)
{
Main_menu->Close();
```

```
}
//---------------------------------------------------------------------------
void __fastcall TM_G_1_Prior_Form::Label7Click(TObject *Sender)
{
M_G_1_Prior_Form->Close();
Main_menu->Show();
}
//---------------------------------------------------------------------------
void __fastcall TM_G_1_Prior_Form::FormActivate(TObject *Sender)
{
e0=e1=e2=0;
M_G_1_Prior_Form->Animate1->FileName = "AVI\\MG1-1.avi";
M_G_1_Prior_Form->Animate2->FileName = "AVI\\MG1-2.avi";
M_G_1_Prior_Form->Animate1->Active = true;
M_G_1_Prior_Form->Animate2->Active = true;
}
//---------------------------------------------------------------------------
void __fastcall TM_G_1_Prior_Form::FormCreate(TObject *Sender)
{
M_G_1_Prior_Form->StringGrid1->Cells[0][0] = "            p";
M_G_1_Prior_Form->StringGrid1->Cells[0][1] = "          0.1";
M_G_1_Prior_Form->StringGrid1->Cells[0][2] = "          0.2";
M_G_1_Prior_Form->StringGrid1->Cells[0][3] = "          0.3";
M_G_1_Prior_Form->StringGrid1->Cells[0][4] = "          0.4";
M_G_1_Prior_Form->StringGrid1->Cells[0][5] = "          0.5";
M_G_1_Prior_Form->StringGrid1->Cells[0][6] = "          0.6";
M_G_1_Prior_Form->StringGrid1->Cells[0][7] = "          0.7";
M_G_1_Prior_Form->StringGrid1->Cells[0][8] = "          0.8";
M_G_1_Prior_Form->StringGrid1->Cells[0][9] = "          0.9";
M_G_1_Prior_Form->StringGrid1->Cells[1][0] = "            L";
M_G_1_Prior_Form->StringGrid1->Cells[2][0] = "           Lc";
M_G_1_Prior_Form->StringGrid1->Cells[3][0] = "           Ld";
M_G_1_Prior_Form->StringGrid1->Cells[4][0] = "         E[W]";
M_G_1_Prior_Form->StringGrid1->Cells[5][0] = "        E[Wc]";
M_G_1_Prior_Form->StringGrid1->Cells[6][0] = "        E[Wd]";
}
//---------------------------------------------------------------------------
```

```
void __fastcall TM_G_1_Prior_Form::Label4Click(TObject *Sender)
{
Series1->Clear();
lmbd = StrToFloat(Edit2->Text);
lmbd1 = lmbd*StrToFloat(Edit8->Text); // control
l1 = StrToFloat(Edit6->Text);
lmbd2 = lmbd*StrToFloat(Edit7->Text); // data
l2 = StrToFloat(Edit5->Text);
EH1 = 1/W1;
EH2 = 1/W2;
ES0 = (l1+l2)/(StrToFloat(Edit1->Text));
ES1 = l1/StrToFloat(Edit1->Text);
ES2 = l2/StrToFloat(Edit1->Text);
for (int i = 1; i<10; i++) {
ro0 = i/10;
EW0 = lmbd*ES0*ES0/(2*(1-ro0)*W) + (1-W)*ES0/((1-ro0)*W);
EW0 = ceil(EW0*10000)/10000;
StringGrid1->Cells[1][i] = "              "+ FloatToStr(lmbd);
StringGrid1->Cells[2][i] = "            "+ FloatToStr(lmbd1);
StringGrid1->Cells[3][i] = "            "+ FloatToStr(lmbd2);
StringGrid1->Cells[4][i] = "          "+ FloatToStr(EW0);
if (EW1<0) EW1 = -EW1;
EW2 = ET0/((1-sig1)*(1-sig2));
if (EW2<0) EW2 = -EW2;
StringGrid1->Cells[5][i] = "          "+ FloatToStr(EW2);
StringGrid1->Cells[6][i] = "          "+ FloatToStr(EW1);
}
Series1->Add(e0/9,"E[W]",clRed);
Series1->Add(e2/9,"E[Wc]",clRed);
Series1->Add(e1/9,"E[Wd]",clRed);
}
//---------------------------------------------------------------------------
```

M_G_1_Prior.h

```
//---------------------------------------------------------------------------
#ifndef M_G_1_PriorH
```

```
#define M_G_1_PriorH
//---------------------------------------------------------------------------
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
#include <Grids.hpp>
#include <ComCtrls.hpp>
#include <Chart.hpp>
#include <ExtCtrls.hpp>
#include <TeEngine.hpp>
#include <TeeProcs.hpp>
#include <Series.hpp>
//---------------------------------------------------------------------------
class TM_G_1_Prior_Form : public TForm
{
__published: // IDE-managed Components
        TLabel *Label1;
        TEdit *Edit1;
        TLabel *Label2;
        TEdit *Edit2;
        TLabel *Label3;
        TEdit *Edit3;
        TLabel *Label5;
        TEdit *Edit5;
        TLabel *Label6;
        TEdit *Edit6;
        TStringGrid *StringGrid1;
        TLabel *Label7;
        TLabel *Label8;
        TLabel *Label9;
        TEdit *Edit7;
        TLabel *Label10;
        TEdit *Edit8;
        TLabel *Label11;
        TAnimate *Animate1;
        TAnimate *Animate2;
```

```
        TLabel *Label4;
        TChart *Chart1;
        TBarSeries *Series1;
        void __fastcall Label8Click(TObject *Sender);
        void __fastcall Label7Click(TObject *Sender);
        void __fastcall FormCreate(TObject *Sender);
        void __fastcall Label4Click(TObject *Sender);
private:        // User declarations
public:                 // User declarations
        __fastcall TM_G_1_Prior_Form(TComponent* Owner);
};
//---------------------------------------------------------------------------
extern PACKAGE TM_G_1_Prior_Form *M_G_1_Prior_Form;
//---------------------------------------------------------------------------
#endif
```

M_M_1_Const.cpp

```
//---------------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop
#include "M_M_1_Const.h"
#include "Menu.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TM_M_1_Const_Form *M_M_1_Const_Form;
//---------------------------------------------------------------------------
__fastcall TM_M_1_Const_Form::TM_M_1_Const_Form(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------------------------
double cro,ck,cmu,cbk = 0;
void __fastcall TM_M_1_Const_Form::Label1Click(TObject *Sender)
{
M_M_1_Const_Form->Close();
```

```
Main_menu->Show();
}
//---------------------------------------------------------------------------
void __fastcall TM_M_1_Const_Form::Label2Click(TObject *Sender)
{
Main_menu->Close();
}
//---------------------------------------------------------------------------
void __fastcall TM_M_1_Const_Form::FormActivate(TObject *Sender)
{
M_M_1_Const_Form->Animate1->FileName = "AVI\\MM1-1.avi";
M_M_1_Const_Form->Animate2->FileName = "AVI\\MM1-2.avi";
M_M_1_Const_Form->Animate1->Active = true;
M_M_1_Const_Form->Animate2->Active = true;
}
//---------------------------------------------------------------------------
void __fastcall TM_M_1_Const_Form::Label14Click(TObject *Sender)
{
double r = 1;
cro = StrToFloat(Edit1->Text);
ck = StrToFloat(Edit2->Text);
cmu = StrToFloat(Edit3->Text);
Label6->Caption = "E[N] = " + FloatToStr(cro/(1-cro));
Label8->Caption = "E[R] = " + FloatToStr(1/(cmu*(1-cro)));
Label10->Caption = "= " + FloatToStr(cro/((1-cro)*(1-cro)));
for (int i=1; i<ck; i++)
{r*=cro;}
Label12->Caption = "B(k) = " + FloatToStr(r);
}
//---------------------------------------------------------------------------
void __fastcall TM_M_1_Const_Form::Timer1Timer(TObject *Sender)
{
Timer1->Enabled = false;
Image4->BringToFront();
Image1->BringToFront();
Timer2->Enabled = true;
}
```

```
//---------------------------------------------------------------------------
void __fastcall TM_M_1_Const_Form::Timer2Timer(TObject *Sender)
{
Timer2->Enabled = false;
Image2->BringToFront();
Image5->BringToFront();
Timer1->Enabled = true;
}
//---------------------------------------------------------------------------
```

M_M_1_Const.h

```
//---------------------------------------------------------------------------
#ifndef M_M_1_ConstH
#define M_M_1_ConstH
//---------------------------------------------------------------------------
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
#include <ExtCtrls.hpp>
#include <jpeg.hpp>
#include <ComCtrls.hpp>
//---------------------------------------------------------------------------
class TM_M_1_Const_Form : public TForm
{
__published: // IDE-managed Components
        TLabel *Label1;
        TLabel *Label2;
        TEdit *Edit1;
        TLabel *Label3;
        TLabel *Label4;
        TLabel *Label5;
        TLabel *Label6;
        TLabel *Label7;
        TLabel *Label8;
        TLabel *Label9;
```

```
        TLabel *Label10;
        TLabel *Label11;
        TLabel *Label12;
        TLabel *Label13;
        TEdit *Edit2;
        TLabel *Label14;
        TImage *Image1;
        TImage *Image2;
        TAnimate *Animate1;
        TAnimate *Animate2;
        TEdit *Edit3;
        TLabel *Label15;
        TImage *Image3;
        TImage *Image4;
        TImage *Image5;
        TTimer *Timer1;
        TTimer *Timer2;
        void __fastcall Label1Click(TObject *Sender);
        void __fastcall Label2Click(TObject *Sender);
        void __fastcall FormActivate(TObject *Sender);
        void __fastcall Timer1Timer(TObject *Sender);
        void __fastcall FormCreate(TObject *Sender);
private:        // User declarations
public:                 // User declarations
        __fastcall TM_M_1_Const_Form(TComponent* Owner);
};
//---------------------------------------------------------------------------
extern PACKAGE TM_M_1_Const_Form *M_M_1_Const_Form;
//---------------------------------------------------------------------------
#endif
```

M_M_m.cpp

```
//---------------------------------------------------------------------------
#include <vcl.h>
#include <math.h>
#pragma hdrstop
```

```
#include "M_M_m.h"
#include "DTMC.h"
#include "Menu.h"
#include "M_G_1.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TM_M_m_Form *M_M_m_Form;
//---------------------------------------------------------------------------
double fact(int f) {
double z=1;
for (int k=1; k<(f+1); k++) {
z*=k; }
return z;
}
double sm,m,ro,ro0,roi;
void __fastcall TM_M_m_Form::FormCreate(TObject *Sender)
{
for (int i=1; i<11; i++) {
M_M_m_Form->StringGrid1->Cells[i][0]=i;}
M_M_m_Form->StringGrid1->Cells[0][1]="p(0)";
M_M_m_Form->StringGrid1->Cells[0][2]="p(i)";
}
//---------------------------------------------------------------------------
void __fastcall TM_M_m_Form::FormShow(TObject *Sender)
{
Animate1->FileName = "AVI\\text.avi";
Animate1->Active = true;
}
//---------------------------------------------------------------------------
void __fastcall TM_M_m_Form::Label4Click(TObject *Sender)
{
M_M_m_Form->Series1->Clear();
sm=ro0=roi=0;
m = StrToFloat(M_M_m_Form->Edit1->Text);
ro = StrToFloat(M_M_m_Form->Edit2->Text);
for (int ij=1; ij<11; ij++){
```

```
M_M_m_Form->StringGrid1->Cells[ij][1]=ro0; }
for (int ii=0; ii<m; ii++) {
M_M_m_Form->StringGrid1->Cells[ii+1][2]=roi;
M_M_m_Form->Series1->Add(roi,IntToStr(ii+1),clRed);
}
for (int ik=m; ik<10; ik++) {
M_M_m_Form->StringGrid1->Cells[ik+1][2]=roi;
M_M_m_Form->Series1->Add(roi,IntToStr(ik+1),clRed);
}
}
//---------------------------------------------------------------------------
void __fastcall TM_M_m_Form::Label5Click(TObject *Sender)
{
M_M_m_Form->Close();
Main_menu->Show();
}
//---------------------------------------------------------------------------
void __fastcall TM_M_m_Form::Label6Click(TObject *Sender)
{
Main_menu->Close();
}
//---------------------------------------------------------------------------
void __fastcall TM_M_m_Form::FormActivate(TObject *Sender)
{
M_M_m_Form->Animate2->FileName = "AVI\\MMm-1.avi";
M_M_m_Form->Animate3->FileName = "AVI\\MMm-2.avi";
M_M_m_Form->Animate2->Active = true;
M_M_m_Form->Animate3->Active = true;
}
//---------------------------------------------------------------------------
```

M_M_m.h

```
//---------------------------------------------------------------------------
#ifndef M_M_mH
#define M_M_mH
//---------------------------------------------------------------------------
```

```
#include <Classes.hpp>
#include <Controls.hpp>
#include <Forms.hpp>
#include <Chart.hpp>
#include <Grids.hpp>
#include <jpeg.hpp>
#include <TeEngine.hpp>
#include <TeeProcs.hpp>
#include <Series.hpp>
#include <TeeFunci.hpp>
#include <MPlayer.hpp>
#include <ComCtrls.hpp>
//---------------------------------------------------------------------------
class TM_M_m_Form : public TForm
{
__published: // IDE-managed Components
        TLabel *Label1;
        TEdit *Edit1;
        TEdit *Edit2;
        TImage *Image1;
        TStringGrid *StringGrid1;
        TLabel *Label2;
        TLabel *Label3;
        TChart *Chart1;
        TImage *Image3;
        TBarSeries *Series1;
        TImage *Image2;
        TAnimate *Animate1;
        TLabel *Label5;
        TLabel *Label6;
        TAnimate *Animate2;
        TAnimate *Animate3;
        void __fastcall FormCreate(TObject *Sender);
        void __fastcall FormShow(TObject *Sender);
        void __fastcall Label4Click(TObject *Sender);
        void __fastcall Label5Click(TObject *Sender);
        void __fastcall Label6Click(TObject *Sender);
```

```
        void __fastcall FormActivate(TObject *Sender);
private:        // User declarations
public:                 // User declarations
        __fastcall TM_M_m_Form(TComponent* Owner);
};
//---------------------------------------------------------------------------
extern PACKAGE TM_M_m_Form *M_M_m_Form;
//---------------------------------------------------------------------------
#endif
```

Menu.cpp

```
//---------------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop
#include "Menu.h"
#include "DTMC.h"
#include "M_G_1.h"
#include "M_G_1_Prior.h"
#include "M_M_m.h"
#include "M_M_1_Const.h"
#include "TstuLan.h"
#include "Sound.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma link "SHDocVw_OCX"
#pragma resource "*.dfm"
TMain_menu *Main_menu;
//---------------------------------------------------------------------------
void __fastcall TMain_menu::Label2Click(TObject *Sender)
{
ShellExecute(Handle,NULL,"html\\1.1.1..htm",NULL,NULL,SW_SHOWMAXIMIZED
);
}
//---------------------------------------------------------------------------
void __fastcall TMain_menu::Label3Click(TObject *Sender)
{
```

```
ShellExecute(Handle,NULL,"html\\1.1.2..htm",NULL,NULL,SW_SHOWMAXIMIZED
);
}
//---------------------------------------------------------------------------
void __fastcall TMain_menu::Label4Click(TObject *Sender)
{
ShellExecute(Handle,NULL,"html\\1.1.3..htm",NULL,NULL,SW_SHOWMAXIMIZED
);
}
//---------------------------------------------------------------------------
void __fastcall TMain_menu::Label6Click(TObject *Sender)
{
ShellExecute(Handle,NULL,"html\\1.1.3.2..htm",NULL,NULL,SW_SHOWMAXIMIZE
D);
}
//---------------------------------------------------------------------------
void __fastcall TMain_menu::Label7Click(TObject *Sender)
{
ShellExecute(Handle,NULL,"html\\1.1.3.3..htm",NULL,NULL,SW_SHOWMAXIMIZE
D);
}
//---------------------------------------------------------------------------
void __fastcall TMain_menu::Label9Click(TObject *Sender)
{
ShellExecute(Handle,NULL,"html\\1.1.4.1..htm",NULL,NULL,SW_SHOWMAXIMIZE
D);
}
//---------------------------------------------------------------------------
void __fastcall TMain_menu::Label10Click(TObject *Sender)
{
ShellExecute(Handle,NULL,"html\\1.1.4.2..htm",NULL,NULL,SW_SHOWMAXIMIZE
D);
}
//---------------------------------------------------------------------------
void __fastcall TMain_menu::Label12Click(TObject *Sender)
{
```

```
ShellExecute(Handle,NULL,"html\\1.1.5..htm",NULL,NULL,SW_SHOWMAXIMIZED
);
}
//---------------------------------------------------------------------
void __fastcall TMain_menu::Label13Click(TObject *Sender)
{
ShellExecute(Handle,NULL,"html\\1.1.5.1..htm",NULL,NULL,SW_SHOWMAXIMIZE
D);
}
//---------------------------------------------------------------------
void __fastcall TMain_menu::Label20Click(TObject *Sender)
{
ShellExecute(Handle,NULL,"html\\1.1.7..htm",NULL,NULL,SW_SHOWMAXIMIZED
);
}
//---------------------------------------------------------------------
void __fastcall TMain_menu::Label22Click(TObject *Sender)
{
ShellExecute(Handle,NULL,"html\\1.1.8.1..htm",NULL,NULL,SW_SHOWMAXIMIZE
D);
}
//---------------------------------------------------------------------
void __fastcall TMain_menu::Label23Click(TObject *Sender)
{
ShellExecute(Handle,NULL,"html\\1.1.8.2..htm",NULL,NULL,SW_SHOWMAXIMIZE
D);
}
//---------------------------------------------------------------------
void __fastcall TMain_menu::Label24Click(TObject *Sender)
{

ShellExecute(Handle,NULL,"html\\1.1.9..htm",NULL,NULL,SW_SHOWMAXIMIZED
);
}
//---------------------------------------------------------------------
void __fastcall TMain_menu::Label25Click(TObject *Sender)
{
```

```
ShellExecute(Handle,NULL,"html\\1.1.10..htm",NULL,NULL,SW_SHOWMAXIMIZE
D);
}
//---------------------------------------------------------------------------
void __fastcall TMain_menu::Label27Click(TObject *Sender)
{
ShellExecute(Handle,NULL,"html\\1.2.0..htm",NULL,NULL,SW_SHOWMAXIMIZED
);
}
//---------------------------------------------------------------------------

void __fastcall TMain_menu::Label28Click(TObject *Sender)
{
ShellExecute(Handle,NULL,"html\\1.2.1..htm",NULL,NULL,SW_SHOWMAXIMIZED
);
}
//---------------------------------------------------------------------------
void __fastcall TMain_menu::Label29Click(TObject *Sender)
{
ShellExecute(Handle,NULL,"html\\1.2.2..htm",NULL,NULL,SW_SHOWMAXIMIZED
);
}
//---------------------------------------------------------------------------
void __fastcall TMain_menu::Label31Click(TObject *Sender)
{
ShellExecute(Handle,NULL,"html\\1.2.4..htm",NULL,NULL,SW_SHOWMAXIMIZED
);
}
//---------------------------------------------------------------------------
void __fastcall TMain_menu::Label32Click(TObject *Sender)
{
ShellExecute(Handle,NULL,"html\\1.2.5..htm",NULL,NULL,SW_SHOWMAXIMIZED
);
}
//---------------------------------------------------------------------------
void __fastcall TMain_menu::Label33Click(TObject *Sender)
{
```

```
ShellExecute(Handle,NULL,"html\\1.2.6..htm",NULL,NULL,SW_SHOWMAXIMIZED
);
}
//---------------------------------------------------------------------
void __fastcall TMain_menu::Label35Click(TObject *Sender)
{
ShellExecute(Handle,NULL,"html\\1.2.6.2..htm",NULL,NULL,SW_SHOWMAXIMIZE
D);
}
//---------------------------------------------------------------------
void __fastcall TMain_menu::Label37Click(TObject *Sender)
{
ShellExecute(Handle,NULL,"html\\3.2.1.1..htm",NULL,NULL,SW_SHOWMAXIMIZE
D);
}
//---------------------------------------------------------------------
void __fastcall TMain_menu::Label38Click(TObject *Sender)
{
ShellExecute(Handle,NULL,"html\\3.2.1.2..htm",NULL,NULL,SW_SHOWMAXIMIZE
D);
}
//---------------------------------------------------------------------
void __fastcall TMain_menu::Label39Click(TObject *Sender)
{
ShellExecute(Handle,NULL,"html\\1.2.6.4..htm",NULL,NULL,SW_SHOWMAXIMIZE
D);
}
//---------------------------------------------------------------------
void __fastcall TMain_menu::Label40Click(TObject *Sender)
{
ShellExecute(Handle,NULL,"html\\1.2.6.5..htm",NULL,NULL,SW_SHOWMAXIMIZE
D);
}
//---------------------------------------------------------------------
void __fastcall TMain_menu::Label42Click(TObject *Sender)
{
```

```
ShellExecute(Handle,NULL,"html\\1.2.6.6..htm",NULL,NULL,SW_SHOWMAXIMIZE
D);
}
//---------------------------------------------------------------------------
void __fastcall TMain_menu::Label43Click(TObject *Sender)
{
ShellExecute(Handle,NULL,"html\\1.2.6.7..htm",NULL,NULL,SW_SHOWMAXIMIZE
D);
}
//---------------------------------------------------------------------------
void __fastcall TMain_menu::Label44Click(TObject *Sender)
{
ShellExecute(Handle,NULL,"html\\1.2.6.8..htm",NULL,NULL,SW_SHOWMAXIMIZE
D);
}
//---------------------------------------------------------------------------
void __fastcall TMain_menu::Label45Click(TObject *Sender)
{
ShellExecute(Handle,NULL,"html\\1.3.0..htm",NULL,NULL,SW_SHOWMAXIMIZED
);
}
//---------------------------------------------------------------------------
void __fastcall TMain_menu::Label47Click(TObject *Sender)
{
ShellExecute(Handle,NULL,"html\\1.3.2..htm",NULL,NULL,SW_SHOWMAXIMIZED
);
}
//---------------------------------------------------------------------------
void __fastcall TMain_menu::Label48Click(TObject *Sender)
{
ShellExecute(Handle,NULL,"html\\1.3.3..htm",NULL,NULL,SW_SHOWMAXIMIZED
);
}
//---------------------------------------------------------------------------
void __fastcall TMain_menu::Label49Click(TObject *Sender)
{
Main_menu->Close();
```

```
}
//---------------------------------------------------------------------------
void __fastcall TMain_menu::Timer1Timer(TObject *Sender)
{
Timer1->Enabled = false;
Image3->BringToFront();
Timer2->Enabled = true;
}
//---------------------------------------------------------------------------
void __fastcall TMain_menu::FormShow(TObject *Sender)
{
Timer1->Enabled = true;
}
//---------------------------------------------------------------------------
void __fastcall TMain_menu::Timer2Timer(TObject *Sender)
{
Timer2->Enabled = false;
Image4->BringToFront();
Timer1->Enabled = true;
}
//---------------------------------------------------------------------------
void __fastcall TMain_menu::Label75Click(TObject *Sender)
{
Main_menu->Close();
}
//---------------------------------------------------------------------------
void __fastcall TMain_menu::Label52Click(TObject *Sender)
{
ShellExecute(Handle,NULL,"html\\2.1.1..htm",NULL,NULL,SW_SHOWMAXIMIZED
);
}
//---------------------------------------------------------------------------
void __fastcall TMain_menu::Label53Click(TObject *Sender)
{
ShellExecute(Handle,NULL,"html\\2.1.2..htm",NULL,NULL,SW_SHOWMAXIMIZED
);
}
```

```
//---------------------------------------------------------------------------
void __fastcall TMain_menu::Label54Click(TObject *Sender)
{
ShellExecute(Handle,NULL,"html\\2.1.3..htm",NULL,NULL,SW_SHOWMAXIMIZED
);
}
//---------------------------------------------------------------------------
void __fastcall TMain_menu::Label56Click(TObject *Sender)
{
ShellExecute(Handle,NULL,"html\\2.1.5..htm",NULL,NULL,SW_SHOWMAXIMIZED
);
}
//---------------------------------------------------------------------------
void __fastcall TMain_menu::Label57Click(TObject *Sender)
{
ShellExecute(Handle,NULL,"html\\2.1.7..htm",NULL,NULL,SW_SHOWMAXIMIZED
);
}
//---------------------------------------------------------------------------
void __fastcall TMain_menu::Label58Click(TObject *Sender)
{
ShellExecute(Handle,NULL,"html\\2.1.8..htm",NULL,NULL,SW_SHOWMAXIMIZED
);
}
//---------------------------------------------------------------------------
void __fastcall TMain_menu::Label59Click(TObject *Sender)
{
ShellExecute(Handle,NULL,"html\\2.1.9..htm",NULL,NULL,SW_SHOWMAXIMIZED
);
}
//---------------------------------------------------------------------------
void __fastcall TMain_menu::Label61Click(TObject *Sender)
{
ShellExecute(Handle,NULL,"html\\2.1.11..htm",NULL,NULL,SW_SHOWMAXIMIZE
D);
}
//---------------------------------------------------------------------------
```

```
void __fastcall TMain_menu::Label63Click(TObject *Sender)
{
ShellExecute(Handle,NULL,"html\\2.1.12.1..htm",NULL,NULL,SW_SHOWMAXIMIZ
ED);
}
//---------------------------------------------------------------------------
void __fastcall TMain_menu::Label68Click(TObject *Sender)
{
ShellExecute(Handle,NULL,"html\\2.2.0..htm",NULL,NULL,SW_SHOWMAXIMIZED
);
}
//---------------------------------------------------------------------------
void __fastcall TMain_menu::Label69Click(TObject *Sender)
{
ShellExecute(Handle,NULL,"html\\2.2.1..htm",NULL,NULL,SW_SHOWMAXIMIZED
);
}
//---------------------------------------------------------------------------
void __fastcall TMain_menu::Label72Click(TObject *Sender)
{
ShellExecute(Handle,NULL,"html\\2.3.1..htm",NULL,NULL,SW_SHOWMAXIMIZED
);
}
//---------------------------------------------------------------------------
void __fastcall TMain_menu::Label73Click(TObject *Sender)
{
ShellExecute(Handle,NULL,"html\\2.3.2..htm",NULL,NULL,SW_SHOWMAXIMIZED
);
}
//---------------------------------------------------------------------------
void __fastcall TMain_menu::Label74Click(TObject *Sender)
{
ShellExecute(Handle,NULL,"html\\2.3.3..htm",NULL,NULL,SW_SHOWMAXIMIZED
);
}
//---------------------------------------------------------------------------
void __fastcall TMain_menu::Label78Click(TObject *Sender)
```

```
{
Main_menu->Hide();
M_M_m_Form->Show();
}
//---------------------------------------------------------------------------
void __fastcall TMain_menu::Label77Click(TObject *Sender)
{
Main_menu->Hide();
M_G_1_Form->Show();
}
//---------------------------------------------------------------------------
void __fastcall TMain_menu::Label79Click(TObject *Sender)
};
//---------------------------------------------------------------------------
extern PACKAGE TTstuLan_Form *TstuLan_Form;
//---------------------------------------------------------------------------
#endif
```