Министерство образования Республики Беларусь
Учреждение образования
«Белорусский государственный университет
информатики и радиоэлектроники»

Факультет инфокоммуникаций

Кафедра инфокоммуникационных технологий

М. Ю. Хоменок

ИМИТАЦИОННОЕ МОДЕЛИРОВАНИЕ В ПРОГРАММНОЙ СРЕДЕ NETWORK SIMULATOR-2 И НЕЧЕТКОЙ ЛОГИКИ FUZZY LOGIC TOOLS. ЛАБОРАТОРНЫЙ ПРАКТИКУМ

Рекомендовано УМО по образованию в области информатики и радиоэлектроники в качестве учебно-методического пособия для специальности 1-45 01 01 «Инфокоммуникационные технологии (по направлениям)»

Минск БГУИР 2018

УДК 004.94(076.5)
ББК 32.973.26я73
Х76

Р е ц е н з е н т ы:

кафедра радио и информационных
технологий учреждения образования
«Белорусская государственная академия связи»
(протокол №7 от 26.02.2018);

главный научный сотрудник государственного научного учреждения
«Объединенный институт проблем информатики
Национальной академии наук Беларуси»
доктор технических наук, доцент С. Ф. Липницкий

Хоменок, М. Ю.

Х76 Имитационное моделирование в программной среде Network Simulator-2 и нечеткой логики Fuzzy Logic Tools. Лабораторный практикум : учеб.-метод. пособие / М. Ю. Хоменок. – Минск : БГУИР, 2018. – 71 с. : ил.

ISBN 978-985-543-426-0.

В учебно-методическом пособии излагаются основы программирования и средства анимации результатов моделирования, реализованного в программной среде Network Simulator-2, а также основы функций пакета Fuzzy Logic Tools для систем нечеткого вывода, используемых для симуляции сетей WLAN и WSN.

Предназначено для студентов, изучающих дисциплину «Беспроводные локальные и сенсорные сети».

УДК 004.94(076.5)
ББК 32.973.26я73

ISBN 978-985-543-426-0

СОДЕРЖАНИЕ

СПИСОК ПРИНЯТЫХ СОКРАЩЕНИЙ

ARED (Adaptive RED) – адаптивный алгоритм RED
Bandwidth – ширина полосы пропускания
BES (Best Effort Service) – негарантированная доставка
CBR (Constant Bit-Rate) – источник трафика с постоянной скоростью пакетов
Delay – задержка при передаче пакета, измеряется в миллисекундах
DiffServ (Differentiated Service) – дифференцированное обслуживание
FIFO (First In – First Out) – первый пришел – первый ушел
FRED (Flow RED) – алгоритм управления состоянием для каждого потока
FTP (File Transfer Protocol) – протокол, предназначенный для передачи файлов
IntServ (Integrated Service) – модель интегрированного обслуживания
Jitter (джиттер) – изменение задержки во времени при передаче пакетов
Lock-Out – блокировка
MAMT (Multiple Average Multiple Threshold) – несколько средних значений, несколько наборов значений границ
MRED (Multi-level RED) – многоуровневый алгоритм RED
NS-2 – ядро симулятора
Nam (Network Animator) – визуализатор результатов моделирования
Otcl (Object Tcl) – объектно-ориентированное расширение языка Tcl
Packet Loss – потеря (отбрасывание) пакетов при передачи данных
QoS (Quality of Service) – способность сети обеспечить необходимый уровень сервиса сетевому трафику при использовании различных технологий
RED (Random Early Detection) – алгоритм активного управления очередями
RIO (RED In & Out) – в этом режиме RED поддерживает очереди различной длины и различные пороговые значения
SAMT (Single Average Multiple Threshold) – одно среднее значение, несколько наборов значений границ
SAST (Single Average Multiple Threshold) – одно среднее значение, один набор значений границ
Tcl (Tool Command Language) – командный язык инструментов, скриптовый язык высокого уровня, который используется в пакете NS-2
Tclcl – интерфейс между C++ и Tcl
TCP (Transmission Control Protocol) – протокол управления передачей
ToS (Type of Service) – тип сервиса (от него зависит приоритет у пакетов)
Tk – библиотека C-функций для языка Tcl
Trace-файл – трассировочный файл
Tracegraph (для Windows) – программа для отображения графических результатов и получения статистики моделирования
UDP (User Datagram Protocol) – протокол пользовательских датаграмм
WRED (Weighed RED) – взвешенный алгоритм RED
Xgraph – программа графического представления результатов моделирования

ВВЕДЕНИЕ

С развитием технологий проведение натурных экспериментов становится нецелесообразным как по экономическим, так и по техническим причинам, однако создание моделей различных сетей с помощью математического аппарата и на неспециализированном ПО становится трудоемкой работой, которая может занимать большое количество времени.

В то же время развитие сетевых технологий, алгоритмов и протоколов, изменение требований к функционированию сетей и, в частности, к ряду сервисных услуг, включающих групповую передачу, защищенность, работу мобильных сетей, систем сетевого управления, требует серьезного изучения динамики процессов в реальных сетях.

Преимуществом имитационного моделирования является возможность подмены процесса событий в исследуемой системе в реальном масштабе времени на ускоренный или замедленный 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. Либо несколько секунд работы сети можно воспроизводить в течение десятков минут, что дает возможность подробно рассмотреть процессы, происходящие в сети.

Результатом работы программ имитационного моделирования являются собранные в ходе наблюдения за протекающими событиями статистические данные о наиболее важных характеристиках сети: времени реакции, коэффициентах использования каналов и узлов, вероятности потери пакетов и т. п.

Одними из таких средств для исследования в области сетевых и инфокоммуникационных технологий являются программный симулятор Network Simulator (NS) и пакет Fuzzy Logic Toolbox (FL) программной среды MATLAB.

С помощью NS можно описать топологию сети, конфигурацию источников и приемников трафика, параметры соединений (полосу пропускания, задержку, вероятность потерь пакетов) и множество других параметров моделируемой системы. При моделировании имеется возможность управления параметрами буферов, мониторинга принятых, отправленных и потерянных пакетов, сбора статистики, а также может быть получена информация о динамике трафика, состоянии соединений и объектов сети, а также работе протоколов.

Инструкция по установке ПО NS-2 на английском языке определяется сетевыми ресурсами: http://www.isi.edu/nsnam/ns/ns-win32-build.html либо http://fiddle.visc.vt.edu/courses/ece5566/lectures/09ns2_install.pdf.

Дополнительная информация работы с языком программирования Tcl представлена в программе-самоучителе для языка Tcl «TclTutor», ссылки на которую размещены по адресу http://www.msen.com/~clif/TclTutor.html.

При установке NS-2 в ОС Windows необходимо иметь эмулятор Unix. Тогда можно инсталлировать пакет NS-2 «все в одном». В качестве эмулятора Unix рекомендуется использовать пакет Cygwin, который является наиболее

мощной оболочкой Unix для ОС Windows. Установку Cygwin можно произвести с официального сайта www.cygwin.com.

В то же время проблемы принятия решений в сложных условиях занимают особое место в информационных технологиях. Теория оптимизации создала совокупность методов, помогающих при использовании ЭВМ эффективно принимать решения при известных и фиксированных параметрах, или когда параметры – случайные величины с известными законами распределения. Существует, однако, ряд задач, которые не поддаются формальному описанию в силу того, что часть параметров представляет собой неточно или качественно заданные величины, для которых переход от «принадлежности к классу» к «непринадлежности» непрерывен. Традиционные методы недостаточно пригодны для решения подобных задач именно потому, что они не в состоянии описать возникающую неопределенность.

Распространение беспроводных сетей (от глобальных до персональных), активный переход на IP6, а также плюс рост популярности облачных технологий и технологий M2M постепенно перемещают концепцию IoT в практическую плоскость. Автоматизация межмашинного обмена физических объектов на основе АСУ ТП эффективно может быть реализована на основе математического аппарата теории нечетких множеств (fuzzy sets) и нечеткой логики (fuzzy logic), которые являются обобщениями классической теории множеств и классической формальной логики.

Пакет Fuzzy Logic Toolbox содержит два типа программных инструментов. Первый из них может быть вызван непосредственно путем набора имени функции в командной строке (command line) или из собственных пользовательских приложений. Большинство из этих функций представляют собой функции в виде *m*-файлов. В этом случае можно посмотреть запрограммированные в этих функциях алгоритмы, а также редактировать и корректировать эти файлы.

Второй тип содержит диалоговые модули, которые обеспечивают доступ к большинству функций через графический интерфейс. Кроме того, эти модули обеспечивают удобную среду для проектирования, исследования и внедрения систем на основе нечеткого логического вывода. Для запуска интерактивных модулей достаточно набрать имя модуля в командной строке.

1 ИМИТАЦИОННОЕ МОДЕЛИРОВАНИЕ В СРЕДЕ NS-2

Цель работы: изучение графического интерфейса сетевого симулятора NS-2; получение навыков моделирования в среде NS-2; изучение принципов функционирования алгоритма активного управления очередями RED.

1.1 Особенности архитектуры и программного обеспечения NS-2

Для инсталляции полной версии пакета NS-2 необходимо установить следующие компоненты, рисунок 1.1:

- Tcl (Tool Command Language) – простой командный язык программирования, который используется при моделировании в пакете NS-2;
- Tk – библиотека Си-функций для языка Tcl;
- Otcl – объектно-ориентированное расширение языка Tcl;
- Tclcl – интерфейс между C++ и Tcl;
- NS-2 – ядро симулятора;
- NAM (Network Animator) – визуализатор результатов моделирования;
- Xgraph (для Unix) или Tracegraph (для Windows) – пакеты для отображения графических результатов и получения статистики моделирования.

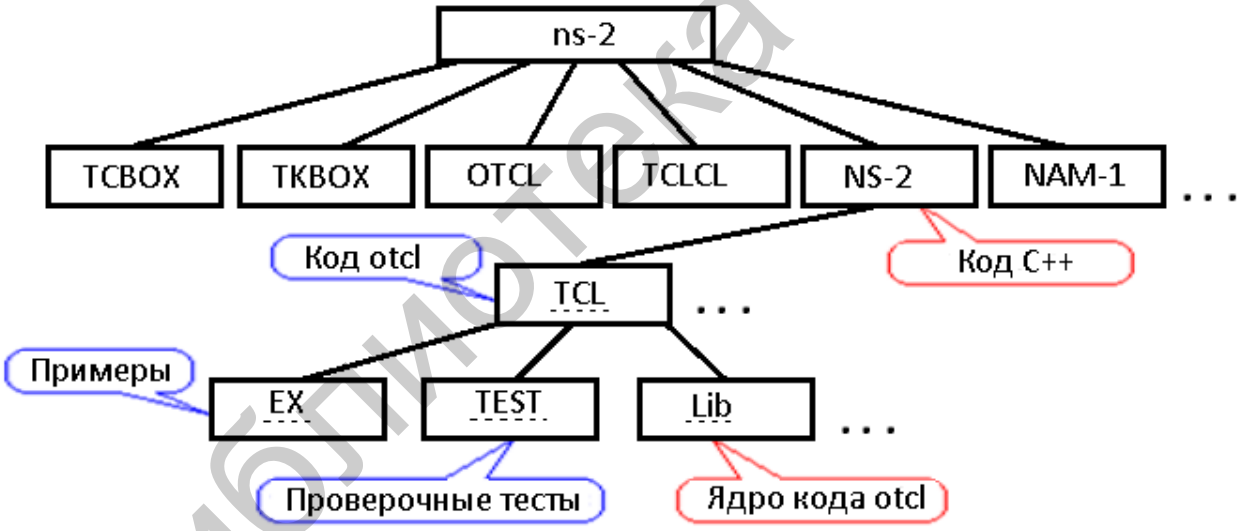


Рисунок 1.1 – Структура каталогов NS-2

NS-2 является объектно-ориентированным программным обеспечением, ядро которого реализовано на языке C++, обеспечивающего:

- высокую производительность;
- работу с пакетами потока на низком уровне абстракции модели;
- модификацию ядра NS-2 с целью поддержки новых функций и протоколов.

В качестве языка программирования высокого уровня абстракции исполь-

зуется язык скриптов (сценариев) OTcl (Object oriented Tool Command Language), который является объектно-ориентированным расширением языка Tcl. OTcl используется в качестве интерпретатора и обеспечивает ряд положительных свойств, присущих языку Tcl (Tool Command Language):

- простота синтаксиса;
- простота построения сценария моделирования;
- возможность соединения воедино блоков, выполненных на системных языках программирования, и простая манипуляция ими.

NS-2 полностью поддерживает иерархию классов C++ (называемую в терминах NS-2 компилируемой иерархией) и подобную иерархию классов интерпретатора OTcl (называемую интерпретируемой иерархией). Обе иерархии обладают идентичной структурой, т. е. существует однозначное соответствие между классом одной иерархии и таким же классом другой.

Использование двух языков программирования в NS-2 позволяет использовать системный язык программирования, обеспечивающий высокую скорость выполнения и способный манипулировать достаточно большими объемами данных, а для удобства пользователя, быстроты реализации и модификации различных сценариев моделирования привлекательнее использовать язык программирования более высокого уровня абстракции. Такой подход является компромиссом между удобством использования и скоростью.

Сетевая топология, время передачи, использование протокола и т. д. определены в файле сценария, реализующего программу имитационного моделирования. Для создания файла сценария моделирования NS-2 использует язык Tcl.

Tcl является интерпретируемым языком программирования по собственным характеристикам и возможностям, схожим с Perl, и состоит из командного интерпретатора и библиотеки. В первую очередь Tcl – простой язык обработки текстовой информации, предназначенный для создания команд для интерактивных программ типа текстовых редакторов, отладчиков и оболочек. Tcl имеет достаточно простой синтаксис и позволяет создавать новые команды для создания нового, более мощного синтаксиса.

Tcl включает библиотеку, которая может быть использована в прикладных программах. Библиотека Tcl содержит синтаксический анализатор для языка Tcl, подпрограммы, реализующие встроенные команды Tcl, и процедуры, которые доступны каждому приложению и расширяют язык Tcl при помощи дополнительных команд, специфичных для каждого приложения. Прикладная программа генерирует команды Tcl и передает их синтаксическому анализатору Tcl для последующего выполнения. Команды могут быть сгенерированы символами от стандартного устройства ввода в командной строке. Когда синтаксический анализатор Tcl принимает команды, он анализирует их и выполняет встроенные команды. Для последующего выполнения команд, декларированных приложением, Tcl обращается к программному коду приложения. Интерпретатор Tcl состоит из набора команд и переменных и поддерживает только

один тип данных – *строки.* Учитывая это, все параметры, передаваемые командам, все команды и все переменные являются строками. В тех случаях, когда команды требуют числовых параметров или возвращают числовые значения, они интерпретируются как строки. Необходимо отметить, что многие команды ожидают параметры в определенном формате, однако анализ и конвертация строк-параметров в необходимый формат является задачей самих команд. Например, параметры команд часто содержат непосредственно команды Tcl, которые должны быть выполнены как часть команды (*set y [set z 10]*).

Существует три общие формы интерпретации, которые принимают строки: команды, выражения и списки.

Команды Tcl

Команда Tcl состоит из одной или нескольких команд, разделенных символом новой строки или символом «точка с запятой» (;).

Каждая из команд является набором полей, разделенных пробелами или символом табуляции.

Первое из полей – обязательно имя команды, дополнительные поля (если определены командой и существуют) – параметры данной команды.

Например, команда *set x 22* устанавливает переменную в значение 22, имеет три поля: первое *set* является названием команды Tcl, остальные два *x* и *22* являются параметрами команды *set*. Названием команды может являться встроенная команда Tcl или команда, созданная приложением при помощи библиотечной процедуры *Tcl_Create Command* или любой пользовательской процедуры *proc*.

Параметры команды являются текстовыми строками. Имена команд Tcl в общем случае должны набираться полностью, без сокращений. Однако, если интерпретатор Tcl не может распознать команду, он вызывает специальную команду *unknown*, которая пытается найти и/или создать команду.

Если первый непробельный символ команды Tcl является знаком «#», то вся введенная информация от данного знака «#» до символа новой строки интерпретируется как комментарий и игнорируется при исполнении:

 # Это комментарий

Во избежание проблем, возникающих с некоторыми редакторами, использующими для идентификации новой строки нестандартные символы, рекомендуется всегда использовать в качестве символа завершения команды «;», а в качестве комментария следующую последовательность:

;# Это комментарий

Обычно параметры команд разделяются пробельными символами и, следовательно, параметры, заданные таким образом, не могут содержать пробельных символов. Tcl предоставляет специальный способ использования пробельных символов внутри параметров. Для таких целей используется символ двойных кавычек – «"». Таким образом, если параметр команды начинается с двойных кавычек, он заканчивается не пробельным символом, а символом двойных кавычек. Необходимо отметить, что двойные кавычки не входят в параметр,

передаваемый команде. Так, например, строка *set x "Параметр с пробельными символами"* передаст два параметра команде *set: x* и *Параметр с пробельными символами*. В пределах двойных кавычек осуществляется подстановка переменных и команд. Кроме того, если первый символ поля – не двойные кавычки, то они не принимают никакую специальную интерпретацию в синтаксическом анализе этого поля: *set x Пар"аметр*.

Изогнутые скобки «{" и "}» могут также использоваться для группировки параметров. Но они имеют два отличия от двойных кавычек. Во-первых, в командах Tcl можно использовать вложенные скобки. Это делает их более легкими для использования для сложных параметров подобно вложенным строкам команды Tcl. Во-вторых, в пределах изогнутых скобок не производится подстановка переменных и команд Tcl. Строка *set x {set x 10}* передает команде *set* два параметра: *x* и *set x 10*. Таким образом, переменная будет установлена в значение *set x 10*.

Если поле параметра начинается со скобок, изогнутых в левую сторону «{», то параметр, передаваемый команде, заканчивается соответствующими скобками, изогнутыми в правую сторону «}». Аналогично двойным кавычкам интерпретатор Tcl удаляет наиболее удаленные изогнутые скобки из значения параметра. Остальная часть поля передается команде без изменений. Так, например, строка *set x {Good {work}}* передаст команде *set* два параметра: *x* и Good {work}.

Также необходимо отметить, что при использовании как двойных кавычек, так и изогнутых скобок элементы, закрывающие поле параметра, необязательно должны быть на той же самой строке, на которой начинается поле. В этом случае символ новой строки будет включен в поле параметра так же, как и любой другой символ в пределах поля между двойными кавычками или изогнутыми скобками.

Так, например, команда *eval* принимает один параметр, который является командной строкой. Команда *eval* вызывает командный интерпретатор Tcl для выполнения переданного параметра.

Команда

```
eval {
set x 22;
set y 33;
}
```

присвоит значение *22* переменной *x* и *33* переменной *y*.

Если первый символ поля – не изогнутые влево скобки, то символы «{» и «}» не принимают никакую специальную интерпретацию в синтаксическом анализе этого поля параметра.

Если в пределах поля параметра появляется левая квадратная скобка «[», в поле происходит подстановка команды (за исключением поля, ограниченного изогнутыми скобками). Весь текст, расположенный от данной левой квадратной скобки до соответствующей закрывающей правой квадратной скобки, интер-

претируется как Tcl-команда и выполняется до исполнения команды, в поле которой она находится. Рассмотрим команду *set x [set y]*.

Команда *set* имеет единственный параметр *set y*. Команда *set y* возвращает содержание переменной *y*. В этом случае, если переменная *y* имеет значение *node*, то указанная команда эквивалентна *set x node*.

Квадратные скобки могут организовывать более сложные структуры. Так, например, если переменная *x* имеет значение *sut*, а переменная *y* имеет значение *ru*, то команда *set name [set x].[set y]* эквивалентна команде *set name sut.ru*.

Команда в квадратных скобках может содержать несколько команд, разделенных символами новой строки или точками с запятой. В этом случае для подстановки используется значение последней команды. Таким образом, команда *set Before[set b 22; expr $b+2]After* эквивалентна команде set *Before24After*.

Если поле параметра ограничено изогнутыми скобками, квадратные скобки в его пределах не интерпретируются. Так команда *set x {[set y]}* присвоит переменной *x* значение *[set y]*.

Знак доллара *$* может использоваться для подстановки значений переменных. Необходимо отметить, что подстановка значения не происходит в пределах полей параметров, ограниченных изогнутыми скобками. Символы после знака *$* до первого символа, не являющегося цифрой, буквой или подчеркиванием, составляют имя переменной. Рассмотрим следующую последовательность команд:

set x node;
set y $x;

что эквивалентно команде *set y node*.

Существует две формы для подстановки переменной. Если следующий символ после названия переменной – открывающая круглая скобка, то переменная является именем массива и все символы между открывающей круглой скобкой и следующей ближайшей закрывающей круглой скобкой являются индексом в массиве. В пределах круглых скобок, задающих индекс массива, может использоваться подстановка переменных.

Например, если переменная *myArray* – массив из двух элементов, индекс первого *first* и значение *Value of the first*, индекс второго *second* и значение *Value of the second*, то команда *set x $myArray(first);* является эквивалентной команде *set x "Value of the first";*

Другой пример:

set y second;
set x $myArray($y);

Здесь проиллюстрировано использование подстановки переменных в пределах круглых скобок, задающих индекс массива.

Вторая форма подстановки переменных имеет место, когда за знаком доллара следует левая изогнутая скобка. В этом случае имя переменной состоит из всех символов до соответствующей закрывающей изогнутой скобки. Необ-

ходимо отметить, что в этом случае переменная должна являться скаляром, но не массивом. Пусть переменная *x* имеет значение *n*. Тогда команда *set y ${x}s2* является эквивалентной команде *set y ns2*. Подстановка переменных не происходит в параметрах, которые включены в изогнутые скобки.

Знак доллара является просто сокращением общей формы *[set var]*. Так, например, запись *$var* является сокращенной формой *[set var]*.

Обычно каждая команда занимает одну строку и заканчивается символом новой строки. Символ точка с запятой «;» позволяет размещать на одной строке несколько команд. Точки с запятой не обрабатываются как разделители команд, если они появляются в пределах изогнутых скобок или двойных кавычек.

Обратный слеш используется для вставки непечатных символов в поля параметров команд. Кроме того, обратный слеш позволяет вставлять специально интерпретируемые символы. Последовательности, реализуемые при помощи обратного слеша и корректно интерпретируемые Tcl, перечислены ниже. В каждом случае обратный слеш заменяется указанным символом:

b – возврат на один символ (0x8);
f – перевод страницы (0xc);
n – новая строка (0xa);
r – перевод каретки (0xd);
t – табуляция;
v – вертикальная табуляция;
{ – левая изогнутая скобка («{»);
} – правая изогнутая скобка («}»);
[– левая квадратная скобка («[»);
] – правая квадратная скобка («]»);
*$* – знак доллара («$»);
sp – пробел (« »);
; – точка с запятой, не ограничивает команду;
" – двойная кавычка;
nl – присоединяет следующую строку к данной;
\\ – обратный слеш («\»);
ddd – восьмеричное представление числа ddd.

Примеры:

1. *set x \{Code\}*. Переменная *x* будет установлена в *{Code}*.
2. *set x \[set y 10\]*. Переменная *x* будет установлена в *[set y10]*.
3. *set x My\tname\tis*. Переменная *x* будет установлена в *My name is*.

Обратный слеш, за которым следуют символы, отличные от перечисленных выше, интерпретируется Tcl как обычная косая черта («\»).

4. *set x \Bonch*. Переменная *x* будет установлена в *\Bonch*.

Если поле параметра ограничено изогнутыми скобками, то последовательности с обратным слешем внутри поля параметра анализируются, однако подстановка не происходит (за исключением пары обратный слеш – новая строка). Последовательности с обратным слешем передаются команде без под-

становок. В частности, в такой ситуации фигурные скобки не означают поиска соответствующей парной скобки, заканчивающей аргумент. Например, в команде *set a {\{abc}* второй аргумент команды *set* будет *\{abc*.

Последовательности с обратным слешем недостаточны для генерации любой структуры поля параметра и учитывают лишь наиболее общие случаи.

В тех случаях, когда последовательностей с обратным слешем недостаточно, необходимо использовать команду *format* вместе с подстановкой команд и переменных.

Выражения Tcl

Выражения Tcl состоят из комбинации операндов, операторов и круглых скобок. Пробельные символы и символы табуляции при анализе выражений игнорируются. Там, где это возможно операнды интерпретируются как целочисленные значения. Целочисленные значения могут быть определены в десятичном, восьмеричном (если первый символ операнда – «0») или в шестнадцатеричном виде (если первые два символа операнда – «0x»). Если операнд не принадлежит ни к одному из целочисленных форматов, приведенных выше, то он будет обработан как число с плавающей запятой (если это возможно).

Операнды могут быть определены одним из следующих способов:

1. Как числовое значение, или целое число или число с плавающей точкой.

2. Как переменная Tcl, используя *«$»* или *[set variable]*. Значение переменной используется как операнд.

3. Как строка, включенная в двойные кавычки. Синтаксический анализатор выражения осуществит подстановку последовательностей с обратным слешем, переменных и команд и использует полученное значение как операнд.

4. Как строка, включенная в изогнутые скобки. Символы между скобками используются как операнд без предварительных подстановок.

5. Как команда Tcl, заключенная в квадратные скобки. Команда предварительно выполняется, и ее результат далее используется как операнд.

В тех случаях, когда подстановки происходят в соответствии с вышеперечисленными случаями, они выполняются процессором выражений. Но дополнительный уровень подстановки может быть осуществлен анализатором команд.

Списки Tcl

Третьей основной смысловой формой строк в Tcl являются списки. Список – это обычная строка с подобной списку структурой, состоящей из полей, разделенных промежутками. Основная структура списков аналогична структуре командных строк, за исключением того, что символ новой строки служит таким же разделителем, как и пробел с табуляцией. Для списков действуют такие же правила в отношении фигурных скобок, двойных кавычек и обратных слешей, как и для команд. Например, строка *a b\ c {d e {f g h}}* есть список из трех элементов: *a, b c* и *d e {f g h}*. Всегда, когда из списка извлекается элемент, действуют те же правила относительно фигурных скобок, двойных кавычек и об-

ратных слешей, что и для команд. Таким образом, когда из списка в примере будет извлечен третий элемент, результат будет *d e {f g h}* (потому что при извлечении произошло только отбрасывание внешней пары фигурных скобок). В отношении списков никогда не выполняются подстановки команд и переменных (по крайней мере командами обработки списков: список всегда может быть передан интерпретатору Tcl для обработки).

1.2 Графический интерфейс симулятора NS-2

Процесс разработки модели сети для Network Simulator включает несколько этапов. Для начала необходимо создать новый объект класса Simulator. В этом классе содержатся все методы, необходимые для дальнейшего описания модели *set ns [new Simulator]*.

Для создания и уничтожения объектов используются методы *new* и *delete*. Следует отметить, что в описании модели язык Tcl часто смешан с внутренним языком симулятора (в данном примере команды внутри скобок представляют собой внутренние команды класса *Simulator*, а *set ns* – команда языка TCL, используемая для присвоения имени нового объекта типа *Simulator* переменной «*ns*»).

Ко второму этапу можно отнести процесс описания собственно топологии моделируемой сети. Сетевая топология реализуется на основе трех основных функциональных блоков: узлов (*nodes*), соединений (*links*) и агентов (*agents*). У класса *Simulator* имеются методы для создания/конфигурирования каждого из этих строительных блоков. Узлы создаются при помощи метода *node*, и каждому узлу автоматически присваивается уникальный адрес. Соединения между узлами, необходимые для формирования сетевой топологии, могут быть заданы с помощью симплексных или дуплексных методов класса симулятора *simplex-link* и *duplex-link*, в результате чего строятся однонаправленные и двунаправленные линии соединений. Агенты представляют собой объекты, активно управляющие моделированием. Они могут рассматриваться как процессы и/или как транспортные единицы, которые работают на узлах. Примеры агентов – это источники трафика, приемники – *sinks*, различные динамические маршрутизирующие и протокольные модули. Агенты создаются с помощью методов общего класса *Agent* и являются объектами его подкласса, т. е. Agent/type, где *type* определяет вид конкретного объекта. Например, TCP-агент может быть создан с помощью следующей команды:

set tcp [new Agent/TCP]

Когда агенты созданы, они закрепляются за узлами при помощи метода *attach-agent*. Каждому агенту присваивается уникальный адрес порта для заданного узла (аналогично портам TCP и UDP). Некоторые агенты могут иметь закрепленные за ними источники, тогда как другие способны генерировать свои собственные данные. Например, можно прикрепить FTP- или Telnet-источники к TCP-агенту, тогда как CBR-агенты (*Constant Bit-Rate*-источник трафика с по-

стоянной скоростью пакетов) генерируют свои собственные данные. Источники прикрепляются к агентам с помощью методов *attach-source* и *attach-traffic*. Большинство объектов имеют некоторые конфигурационные параметры, ассоциируемые с ними вначале процесса моделирования и которые могут быть модифицированы в дальнейшем. Например, размер окна в TCP-сессии может быть изменен следующим образом:

$tcp set window_ 25

На третьем этапе производится задание различных действий, характеризующих работу сети. Действия разных агентов могут быть назначены в определенные моменты времени с использованием планировщика. Различные действия, в том числе запуск тех или иных процедур OTcL, могут быть выполнены с использованием метода симулятора *at*. Подобная привязка ко времени делает процесс моделирования достаточно гибким и удобным: в определенные моменты времени могут быть задействованы или отключены те или иные источники данных, произведена запись статистики, разрыв либо восстановление соединений, реконфигурация топологии и т. д. Моделирование начинается при помощи метода *run*.

К дополнительным возможностям симулятора можно отнести возможность сетевой эмуляции, которая позволяет наряду с возможностями замкнутого моделирования использовать симулятор совместно с реальными сетями. Специальные объекты класса симулятора позволяют принимать и обрабатывать реальный трафик в самом симуляторе, а также экспортировать моделируемый трафик в реальную сеть. Интерфейс между симулятором и реальной сетью обеспечивается с помощью ряда объектов, используемых для обработки заголовков пакетов, а также сетевых агентов, обеспечивающих доступ к реальной сети (или к *trace*-файлу сетевого трафика). Для работы с реальным трафиком также используется специальный планировщик реального времени, который привязывает выполнение действий симулятора к реальному времени.

Как было отмечено, в состав симулятора входят много агентов, позволяющих реализовывать работу различных протоколов (UDP, RTP/RTCP, несколько разновидностей TCP). С помощью TcL и C++ возможны модификация имеющихся и создание новых протоколов. Имеются также агенты генерации трафика с возможностью задания ряда параметров (размер пакетов, временные распределения трафика и т. д.) В моделируемой сетевой топологии могут быть установлены модули генерации ошибок передачи, позволяющие устанавливать бит ошибки в пакетах или отбрасывать пакеты целиком.

Совместно с симулятором может использоваться ряд дополнительных пакетов, расширяющих его возможности. В первую очередь это средства визуализации, к которым можно отнести *Nam* (*Network Animator*) – сетевой аниматор, средство, используемое для графического отображения свойств моделируемой системы и проходящего через нее трафика. Аниматор имеет возможности отображения сетевой топологии, анимации пакетного уровня, а также различные средства инспекции данных.

Кроме Nam также часто используется пакет *Xgraph* – программа графического представления результатов моделирования.

При выполнении файла сценария результат моделирования будет выводиться в файлах выходных данных симулятора NS-2: *out.tr* и *out.nam*:

- *out.tr* – файл трассировки, который содержит всю информацию о сетевом взаимодействии и источнике передаваемого пакета данных;
- *out.nam* – содержит данные для анимации результата эксперимента.

Этот файл реализуется анимационным программным обеспечением Nam, являющимся средством анимации результатов моделирования в NS-2.

Визуализатор Nam (Network Animator) графически воспроизводит имитационную модель (топология сети, анимация прохождения пакетов по сети, постановки их в очередь и т. д.) и наглядно показывает алгоритмы работы протоколов, дисциплин обслуживания очередей.

Запустить Nam можно с помощью:

- команды Nam <nam-file>, где <nam-file> – имя trace-файла Nam, созданного NS-2;
- команды прямо из скрипта моделирования OTcl.

Интерфейс пользователя содержит зону анимации, несколько меню и кнопок (рисунок 1.2).

Вызов аниматора может быть осуществлен с рядом параметров:

nam [-g geometry] [-t graphInput] [-i inteval] [-p peerName] [-N appName] [-c cashSize] [-f configfile] [-S] tracefile.

Опции:

- *g* – определяет геометрию отображения при запуске;
- *t* – для использования совместно с утилитой *tkgraph* (также определяет входной файл для *tkgraph)*;
- *i* – определяет частоту обновления экрана в миллисекундах, скорость обновления по умолчанию 50 мс (т. е. 20 кадров в секунду);
- *N* – определяет имя приложения для данного запускаемого процесса Nam;
- *c* – максимальный размер кэша, используемого для хранения «активных» объектов при обратном воспроизведении;
- *f* – имя файла инициализации, загружаемого при запуске (в этом файле пользователь может определить функции вызываемые из trace-файла; примеры таких функций – *«link-up»* и *«link-down»* для динамических соединений в ns);
- *S* – дополнительная синхронизация графического вывода с X-Windows в системах Unix;
- *p* – определяет имя приложения дополнительного процесса Nam, работа которого может быть синхронизирована с работой текущего процесса Nam.

Для этого может быть использована следующая последовательность команд:

• *nam-N [name#1] [trace file name #1]* – запуск первого процесса Nam (подчиненного);

• *nam-N [name#2] [trace file name #2]* – запуск второго процесса Nam (главного).

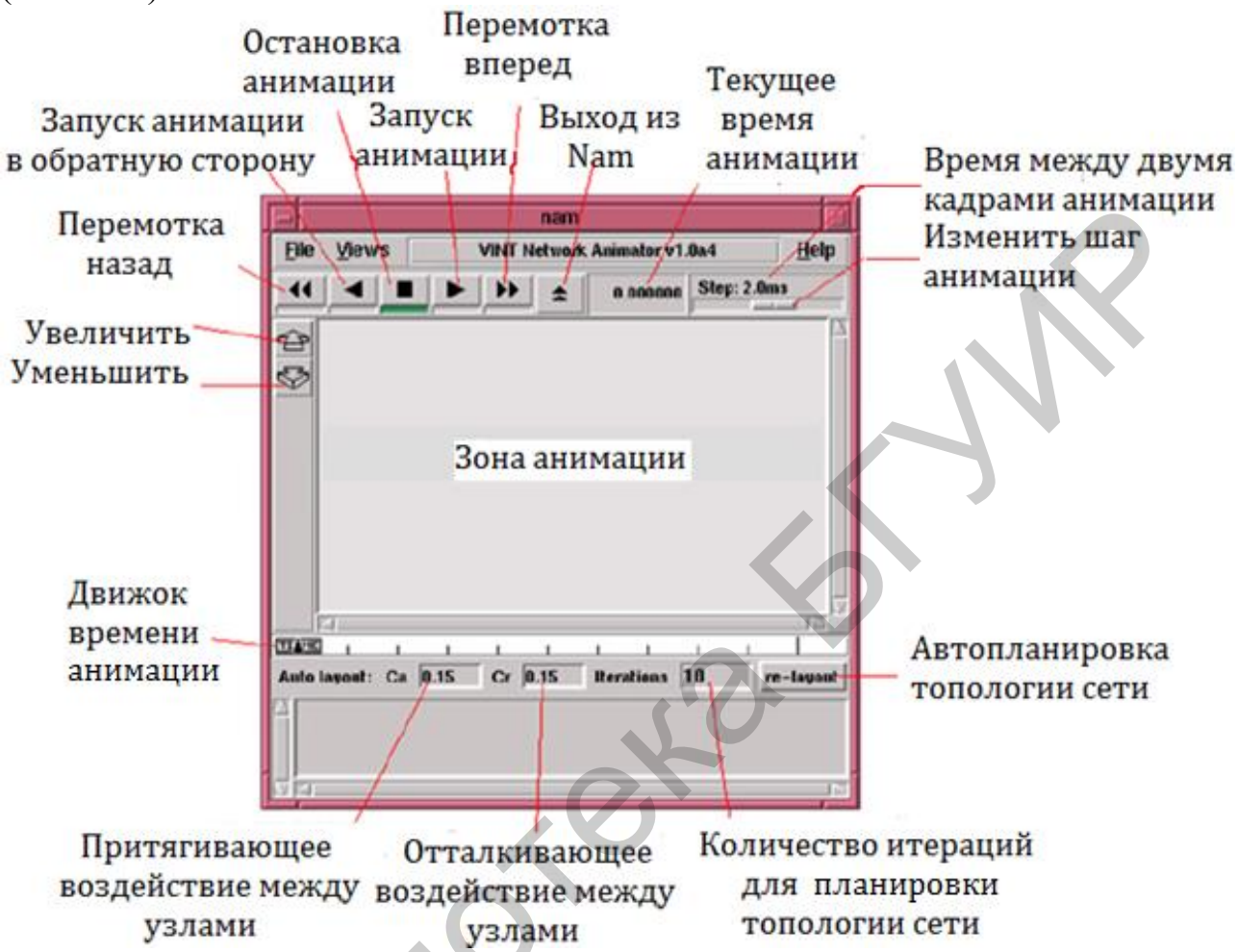


Рисунок 1.2 – Пользовательский интерфейс визуализатора Nam

После этого любой контроль анимации (воспроизведение, остановка, обратное воспроизведение, за исключением других операций наблюдения, таких как интерактивное управление (мониторинг)) будет синхронизирован между двумя процессами.

Окно аниматора имеет пользовательский интерфейс, состоящий из панелей меню (см. рисунок 1.2).

Меню «File»:

1. *Save layout* – запись текущей конфигурации топологии в файл *«savefile»*.

2. *Print layout* – распечатка текущей конфигурации топологии.

3. *Record Animation* – запись процесса анимации.

Auto FastForward – автоматическое воспроизведение анимации в ускоренном режиме.

Quit – выход из аниматора.

Меню «Views»:

1. *New view* – открытие нового окна аниматора. Пользователь может сдвигать и масштабировать изображение.

2. *Edit view* – открытие нового окна аниматора для редактирования.

3. *Show monitors* – если эта опция установлена, то в нижней части окна Nam отображается панель с информацией об объектах, над которыми осуществляется наблюдение.

4. *Show autolayout* – если эта опция установлена, то в нижней части окна Nam отображается панель с элементами управления расположением объектов моделируемой топологии. Эта опция присутствует не всегда. Когда в trace-файле содержится собственная информация о расположении элементов топологии, то опция *Show autolayout* неактивна. Если же информация о расположении элементов топологии неполна или отсутствует, то эта опция доступна.

Show annotation – если уставлено, то в нижней части окна отображается список с комментариями о событиях, происходящих в процессе моделирования.

Меню «Analysis»:

1. *Active Sessions* – открытие окна со списком активных на данный момент сессий.

2. *Legend* – открывает окно с описанием условных обозначений.

Справа от панели меню находится панель с указанием имени используемого trace-файла и меню *«Help»*, содержащее опции справочной информации об использовании Nam – *«Help»* и *«About nam»*.

Дополнительно в Nam могут использоваться кнопки быстрого доступа. *Alt+ «f»* – открытие меню «File», *Alt+ «v»* – меню *«Views»* и *Alt+ «a»* – меню *«Analysis»*.

Под панелью меню находится панель управления процессом моделирования, содержащая шесть кнопок, индикатор времени и бегунок управления скоростью отображения.

Кнопка 1 «[[» – отмотка назад. При нажатии осуществляется переход назад на время, равное 25-и интервалам текущей скорости обновления экрана.

Кнопка 2 «[» – воспроизведение анимации в обратном направлении с текущей скоростью.

Кнопка 3 «квадрат» – остановка (пауза).

Кнопка 4 «]» – воспроизведение анимации.

Кнопка 5 «]]» – перемотка вперед. При нажатии осуществляется переход вперед на время, равное 25-и интервалам текущей скорости обновления экрана.

Кнопка 6 «^» – выход.

Индикатор времени отображает текущее время анимации (время, определенное в trace-файле). Бегунок управления скоростью используется для контроля за скоростью обновления экрана. Текущая скорость отображается сверху над бегунком.

Под панелью управления располагается основное окно, в котором отображается моделируемая топология. Слева от него находится панель с двумя кнопками, позволяющими масштабировать изображение.

Если щелкнуть левой кнопкой мыши по любому объекту в главном окне, то появится всплывающее информационное окно. Для объектов типа пакеты и агенты в этом окне будет присутствовать кнопка *«Monitor»*, с помощью которой можно добавить панель мониторинга для этих объектов. Она появится в нижней части окна Nam. Для объектов типа соединения информационное окно будет содержать кнопку *«Graph»*, при нажатии которой будет вызвано еще одно всплывающее окно, в котором пользователь может выбрать, что будет отображаться на графике для данного соединения: ширина полосы проходящего трафика либо график потерь. Для дуплексного соединения можно выбрать для наблюдения любое направление соединения.

1.3 Формат trace-файла

События, описываемые в trace-файле, могут быть разделены на шесть типов в зависимости от того, какому объекту событие соответствует.

Пакеты

За основным действием, связанным с пакетами, могут располагаться следующие конструкции:

- *«h» – нор* (пакет, который начал передаваться из *src_addr* к *dst_addr*);
- *«r» – receive* (пакет, передача которого завершена и который начал обрабатываться получателем);
- *«d» – drop* (пакет, который был отброшен из очереди или соединения из *src_addr* и *dst_addr*);
- *«+» – enter queue* (пакет, полученный очередью из *src_addr* в *dst_addr*);
- *«-» – leave queue* (пакет, покинувший очередь из *src_addr* и *dst_addr*).

Потеря пакетов в соединении и очереди не различается. Решение принимается по времени отбрасывания.

Флаги имеют следующие значения:

-t [time] – время, когда событие произошло;

-e [extent] – размер пакета (в байтах);

-s [src] – исходный узел;

-d [dst] – узел назначения;

-c [conv] – идентификатор соединения;

-i [id] – идентификатор пакетов соединения 4;

-a [attr] – атрибут пакета (используется для указания цвета).

Дополнительные флаги, используемые некоторыми протоколами:

-P – [pkt_type] ASCII строка, определяющая разделенный запятыми список типов; некоторые величины: *TCP* – пакет *tcp*-данных, *ACK* – пакет подтверждения (acknowledgement), *NACK* – пакет отрицательного подтверждения

(negative acknowledgement), *SRM* – пакет *SRM*-данных;

-n [sequence number] – последовательный номер пакета (sequence number).

Состояние очередей/соединений (Link/Queue)

l -t [time] -s [src] -d [dst] -S [state] [-c [color]] [-r [bw] -D [delay]]

q -t [time] -s [src] -d [dst] -a [attr]

[state] – состояние передачи соединения. Имеет три параметра: *UP* и *DOWN* характеризуют восстановление и разрыв соединения, *COLOR* отмечает изменение цвета соединения. Если *COLOR* указан, то ожидается *-c [color]*, в результате устанавливается новый цвет.

Определение *[-r [bw] -D [delay]]* указывает ширину полосы и задержку соответственно. Оно используется, только когда *Nam* создает соединение, т. е. загружает *trace*-файл.

[attr] – определяет положение отображаемой очереди/соединения, т. е. угол между очередью/соединением и горизонтальной линией.

Состояние узла

n-t[time] -s [src] -S [state] [-c [color]] [-o [color]] [-A [labels]]

Флаги *«-t», «-S»* и *«-c»* имеют те же значения, что и для соединений. Флаг *«-A»* используется для добавления заданной строки к метке узла. Может использоваться для указания состояния узла. Флаг *«-o»* используется для восстановления старых цветов отображения узла.

Маркировка узла

Узел отмечается цветным кругом. Он определяется следующим способом:

m -t [time] -n [mark name] -s [node] -c [color] -h [shape] [-o [color]]

и может быть удален с помощью:

m -t [time] -n [mark name] -s [node] -X

Для уже созданного узла *mark* не способен изменить форму. Имеются следующие варианты форм: *круг, квадрат* и *шестиугольник*.

Состояние протокола

Агенты могут быть созданы следующим способом:

a -t [time] -n [agent name] -s [src] -d [dst]

и могут быть удалены с помощью:

a -t [time] -n [agent name] -s [src] -d [dst] -X

Для отображения переменных состояния протокола используется конструкция *«feature»*, имеющая три типа: таймеры *(timers)*, списки *(lists)* и простые переменные. Обычно используются последние.

Элементы *Features* могут быть добавлены или модифицированы в любое время после создания агента.

f -t [time] -a [agent name] -T [type] -n [var name] -v [value] -o [prevvalue]

[type] имеет значение *«l»* – для списка, *«v»* – для простой переменной, *«s»* – для остановленного таймера, *«u»* – для включенного таймера, *«d»* – для таймера, включенного в обратном направлении.

-v [value] – возвращает новое значение для переменной. Значение переменной представляет простые ASCII-строки, подчиняющиеся форматам строк Tcl (строки в кавычках). Списки должны соответствовать формату списков. Величины типа timer представляют численные ASCII-величины.

-o [prev value] – возвращает старое значение для переменной. Используется для обратного воспроизведения анимации.

Элементы Features могут быть уничтожены с помощью команды:

f -t [time] -a [agent name] -n [var name] -o [prevvalue] -X

Другие объекты

v -t [time] TcL script string – используется для комментариев, может включать обычный Tcl-сценарий, который может быть выполнен в заданное время. Сценарий представляется в виде строки и не может содержать более 256 символов. Порядок флагов и строки-сценария важен.

c -t [time] -i [color id] -n [color name] – определяет цвет. После определения обращение к цвету может осуществляться по его идентификатору.

1.4 Создание Tcl-сценариев

Сетевой фрагмент в NS-2 представлен на рисунке 1.3. Узлы: терминал, маршрутизатор – в реальной сети имеют четыре уровня функциональной модели TCP/IP, два нижних уровня которых формируются автоматически при настройке узла. Транспортный уровень TCP или UDP отображаются как агент. Протоколы FTP, CBR находятся в прикладных слоях. Фактически отправитель установит приложение для создания данных, в то время как получатель не будет, так как нет необходимости имитировать принимаемые данные в прикладном уровне.

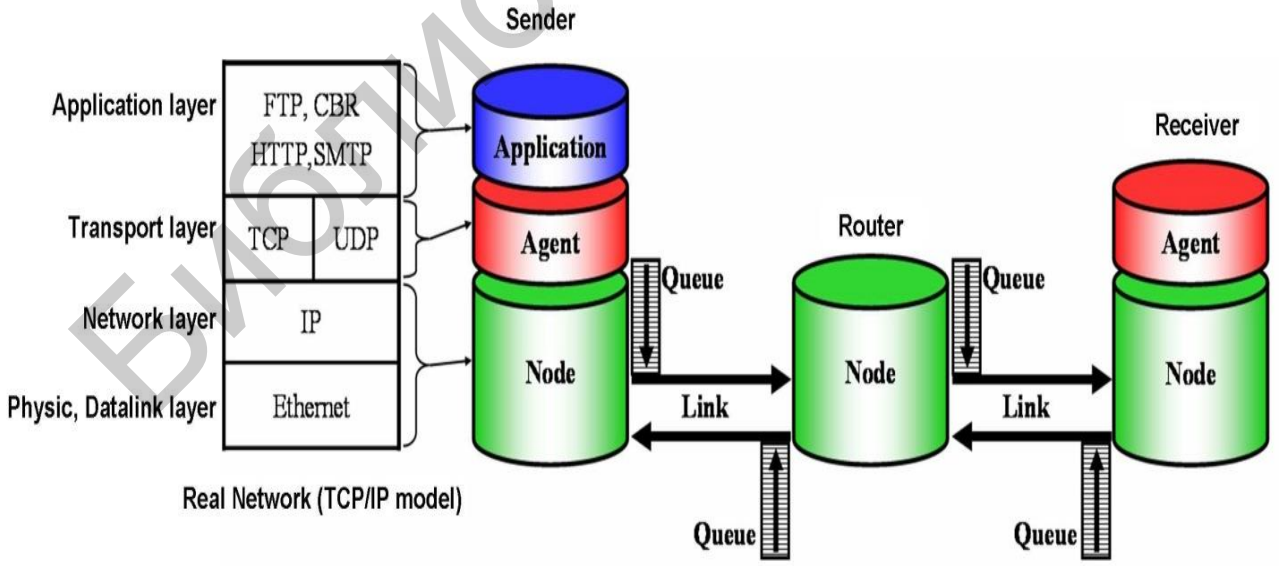


Рисунок 1.3 – Имитационная модель в NS-2

Между двумя узлами для установления связи может быть использовано проводное или беспроводное соединение, реализуемое звеном данных, которое в NS-2 является однонаправленным. Для двунаправленного соединения необходимы две линии. Каждое звено данных имеет очередь в обслуживании поступающих пакетов. Когда очередь пуста, пакет будет отправляться на другой узел через звено данных.

С целью разработки имитационной модели в NS-2 сначала с помощью команды *set ns [new Simulator*] необходимо создать объект *Simulator (модель).*

Далее, если необходимо получить анимационный результат моделирования, нужно открыть файл записи данных трассировки для визуализатора Nam:

set nf [open out.nam w]

$ns namtrace-all$nf

В данном примере имя файла *out.nam*. Имя файла может быть произвольным, но расширение рекомендуется использовать стандартное (*nam*). «*w*» означает, что файл открыт для записи (*write*). Команда *namtrace-all* определяет тип файла трассировки (файл для визуализатора Nam) и содержимое (*all* – должны записываться все события, происходящие в моделируемой сети).

Также необходимо открывать стандартный файл трассировки, так как именно он в дальнейшем используется для обработки и анализа результатов моделирования:

set f [open out.tr w]

$ns trace-all$f

Здесь для записи создается файл *out.tr*, в котором регистрируются все события. В качестве расширения для таких файлов рекомендовано использовать «*tr*».

Следующий шаг – это описание процедуры «*finish*», которая закрывает файлы трассировки и запускает визуализатор Nam:

```
proc finish {} {
global ns f nf # описание глобальных переменных
$ns flush-trace # прекращение трассировки
close $f # закрытие файлов трассировки
close $nf # закрытие файлов трассировки
exec nam out.nam & # запуск nam в фоновом режиме
exit 0
}
```

Для запуска процедуры «*finish*» необходимо добавить at-событие для планировщика событий:

$ns at 5.0 "finish"

В данном случае процедура запускается через 5 с после начала моделирования.

В конце сценария прописывается строка, запускающая модель *$ns run.*

Данный сценарий ничего не выполняет, но является шаблоном практически для всех сценариев, которые предназначены для моделирования в NS-2.

При этом команды для описания топологии необходимо вставлять перед строкой «*$ns run*» (стилистически даже лучше перед описанием процедуры «*finish*»).

Так, сетевой фрагмент на базе этого шаблона в окне аниматора из двух узлов и одного звена, которое их соединяет, задается следующими командами:

- команда для создания двух новых узлов:

set n0 [$ns node] set n1 [$ns node];

- команда для соединения двух узлов одним звеном:

$ns duplex-link$n0 $n1 2Mb 10ms DropTail.

В этой строке указано, что между узлами *n0* и *n1* создано дуплексное звено с полосой пропускания 2 Мбит/с, задержкой 10 мс и очередью с механизмом обслуживания *DropTail*.

В NS-2 данные посылаются от одного агента к другому. Следующий шаг – создание агента, который посылает данные на узле *n0* и другого агента, который принимает данные на узле *n1*.

#Создание агента UDP и подключение его к узлу n0
set udp0 [new Agent/UDP]
$ns attach-agent$n0 $udp0
 # Создание источника трафика CBR (constant bit rate) и подключение его к агенту udp0
set cbr0 [new Application/Traffic/CBR]
$cbr0 set packetSize_ 500
$cbr0 set interval_ 0.005
$cbr0 attach-agent$udp0

При помощи этих строк создается агент UDP и присоединяется к узлу *n0*. Но в NS-2 агент сам не может генерировать трафик, он лишь реализует протоколы и алгоритмы транспортного уровня. Поэтому к агенту присоединяется приложение. В данном случае – это источник с постоянной скоростью, который каждые 5 мс посылает пакет длиной 500 байт. Таким образом, скорость источника: $R = 500 \cdot 8/0{,}005 = 800\,000$ бит/с.

Также необходимо создать агент-приемник и прикрепить его к узлу *n1*:

set null0 [new Agent/Null]
$ns attach-agent$n1 $null0

В случае использования UDP агент-приемник называется Null (пустой). Известно, что функции транспортного агента UDP заключаются только в приеме информации. В случае моделирования TCP агент называется TCPSink (приемник TCP).

Далее агенты соединяются друг с другом:

$ns connect $udp0 $null0

Теперь для запуска и остановки приложения CBR необходимо добавить at-события в планировщик событий.

$ns at 0.5 "$cbr0 start"
$ns at 4.5 "$cbr0 stop"

В данном примере запуск приложения происходит через 0,5 с модельного времени и останавливается на 4,5 с.

После запуска файла создадутся два trace-файла: *out.nam* и *out.tcl* и запустится визуализатор Nam, как это показано на рисунке 1.4.

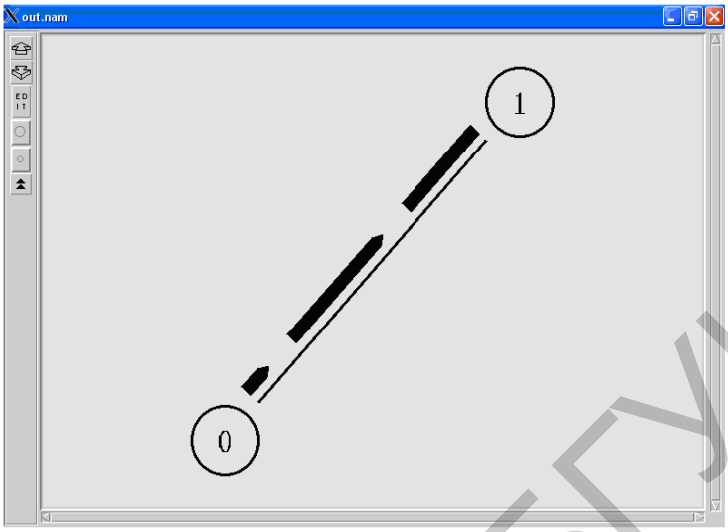


Рисунок 1.4 – Визуализатор Nam при анимации сетевого фрагмента: два узла и одно звено

Рисунок 1.5 иллюстрирует анимацию сетевой топологии на основе четырех узлов, один из которых является маршрутизатором.

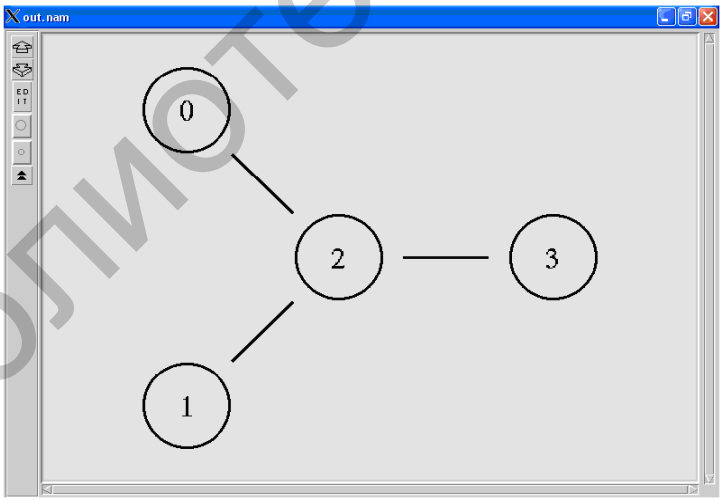


Рисунок 1.5 – Топология сети при анимации результатов моделирования

Создание четырех узлов:

set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]

Эти строчки можно заменить циклом *for*:

for {set i 0} {$i < 4} {incr i} {
set n($i) [$ns node]
}

Далее создаются три дуплексных звена

$ns duplex-link $n0 $n2 2Mb 10ms DropTail
$ns duplex-link $n1 $n2 2Mb 10ms DropTail
$ns duplex-link $n3 $n2 2Mb 10ms DropTail

Для того чтобы данная топология была наглядной, при анимации в визуализаторе Nam необходимо для каждого звена указать направление:

$ns duplex-link-op $n0 $n2 orient right-down
$ns duplex-link-op $n1 $n2 orient right-up
$ns duplex-link-op $n2 $n3 orient right

Создадим два агента – один агент UDP с прикрепленным к нему источником CBR, второй – агент TCP с прикрепленным к нему приложением FTP.

#Создание агента UDP и присоединение его к узлу n0
set udp0 [new Agent/UDP]
$ns attach-agent $n0 $udp0
 # Создание источника CBR-трафика и присоединение его к агенту udp0
set cbr0 [new Application/Traffic/CBR]
$cbr0 set packetSize_ 500
$cbr0 set interval_ 0.005
$cbr0 attach-agent$udp0
#Создание агента TCP и присоединение его к узлу n1
set tcp1 [new Agent/TCP]
$ns attach-agent $n1 $tcp1
 # Создание приложения FTP и присоединение его к агенту tcp1
set ftp [new Application/FTP]
$ftp attach-agent $tcp1
#Создание агента-получателя для udp0
set null0 [new Agent/Null]
$ns attach-agent $n3 $null0
#Создание агента-получателя для tcp1
set sink1 [new Agent/TCPSink]
$ns attach-agent $n3 $sink1

Далее необходимо соединить агенты *udp0* и *tcp1* и их получатели:

$ns connect $udp0 $null0
$ns connect $tcp1 $sink1

Для того чтобы программа начала работать, необходимо добавить at-события:

$ns at 0.5 "$cbr0 start"
$ns at 1.0 "$ftp start"
$ns at 4.0 "$ftp stop"
$ns at 4.5 "$cbr0 stop"

Оба потока в визуализаторе Nam изображены черным цветом, и невозможно отличить один тип пакетов от другого на отрезке между узлами *n2* и *n3*. Но в сценарий возможно вставить строки, которые позволяют изображать потоки от разных источников разными цветами:

$udp0 set class_ 1

$tcp1 set class_ 2

Но перед этим необходимо описать цвет каждого класса, например:

$ns color 1 Blue, $ns color 2 Red

При этом цвет tcp-пакетов будет красный, а цвет udp-пакетов – синий, а для того чтобы осуществить мониторинг очереди, необходимо добавить строку

$ns duplex-link-op$n2 $n3 queuePos 0.5,

которая определяет место положения изображения очереди между узлами *n2* и *n3* в зоне анимации визуализатора Nam (рисунок 1.6).

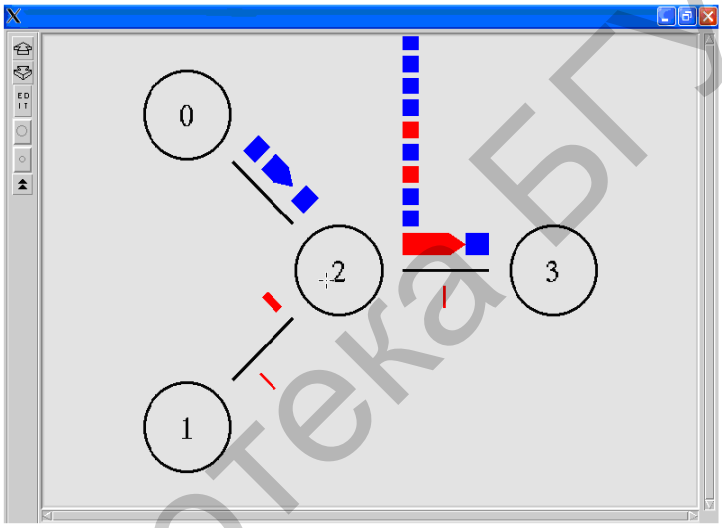


Рисунок 1.6 – Мониторинг очереди в визуализаторе Nam

Таким образом, данный пример позволяет полностью увидеть процессы, происходящие в сети. При запуске этой программы на маршрутизаторе будут наблюдаться потери.

Во избежание этого можно изменить механизм обслуживания очереди с *DropTail*, например, на *SFQ* (Stochastic Fair Queuing – стохастическая справедливая очередь) или *RED* (Random Early Detection). Это можно сделать с помощью переопределения описания соединения между узлами *n2* и *n3*:

$ns duplex-link$n2 $n3 1Mb 10ms SFQ

Также можно задать размер очереди:

$ns queue-limit$n2 $n3 20

Если какие-либо параметры не заданы пользователем, то они остаются установленными по умолчанию. Значения переменных, установленные по умолчанию, записаны в файле *ns-default.tcl*, который находится в каталоге *ns-allinone-2.29/ns-2.29/tcl/lib*. Например, размер очереди определяется в строке

Queue set limit_ 50

Названия данных переменных находятся в соответствующих процедурах, описанных в файле *ns-lib.tcl*, который находится в той же папке.

Для размера очереди необходимо найти процедуру *queue-limit*:

```
Simulator instproc queue-limit { n1 n2 limit } {
$self instvar link_
[$link_([$n1 id]:[$n2 id]) queue] set limit_ $limit
}
```

Из процедуры видно, что *queue-limit* – это метод объекта Simulator, который использует три параметра – два узла, которые описывают звено, и размер очереди. Отсюда видно, что величина размера очереди находится в переменной *limit*.

В качестве приложений в NS-2 можно использовать различные типы трафика (Парето, экспоненциальный и др.). Каждый генератор трафика в NS-2 моделирует трафик, имеющий две величины, распределенные по определенному закону: интервал между пачками пакетов и длина пакета.

Для мониторинга очередей объекты трассировки записывают время прибытия пакетов в то место, где они расположены. Несмотря на то что симулятор отражает достаточно большой объем информации из trace-файла, пользователю может быть нужна информация о том, что происходит внутри некоторых видов очередей.

Например, пользователю, заинтересованному в поведении очереди RED (Random Early Detection), могут понадобиться измерения динамики изменения среднего размера очереди и текущего размера очереди. Мониторинг очереди может быть осуществлен при использовании объектов мониторинга очереди и объектов слежения за очередью.

Когда прибывает пакет, объект слежения за очередью извещает диспетчера очереди об этом событии. Используя эту информацию, диспетчер очереди осуществляет мониторинг очереди. Объекты слежения за очередью могут использоваться одновременно с объектами трассировки.

1.5 Алгоритм активного управления очередями RED. Качество обслуживания QoS

Качество обслуживания QoS (Quality of Service) в области компьютерных сетей означает вероятность того, что сеть связи соответствует заданному соглашению о трафике, или же в ряде случаев неформальное обозначение вероятности прохождения пакета между двумя точками сети.

Для большинства случаев качество связи определяется четырьмя параметрами:

1. Полоса пропускания (*Bandwidth*) – описывает номинальную пропускную способность среды передачи информации и определяется шириной канала. Измеряется в бит/с (bit/s или bps), кбит/с (kbit/s или Kbps), Мбит/с (Mbit/s или Mbps), Гбит/с (Gbit/s или Gbps).

2. Задержка при передаче пакета (*Delay*) – измеряется в миллисекундах.

3. Колебания (дрожание) задержки при передаче пакетов – джиттер.

4. Потеря пакетов (*Packet loss*) – определяет количество пакетов, потерянных в сети во время передачи.

Когда в силу ограниченной пропускной способности передача данных сталкивается с проблемой «бутылочного горлышка» для приема и отправки пакетов на маршрутизаторах, то обычно используется метод *FIFO (First In – First Out)*: первый пришел – первый ушел. При интенсивном трафике это создает заторы, которые разрешаются крайне простым образом: все пакеты, не вошедшие в буфер очереди FIFO (на вход или на выход), игнорируются маршрутизатором и соответственно теряются безвозвратно.

Однако более рационально использовать «умную» очередь, в которой приоритет у пакетов зависит от типа сервиса – *ToS* (*Type of Service*). Необходимое условие – пакеты должны уже нести метку типа сервис для создания «умной» очереди.

На сетях используются различные модели QoS:

1. *Негарантированная доставка – Best Effort Service (BES)*. Наличие марки ToS Best Effort Service не является механизмом тонкого регулирования и является признаком простого увеличения пропускной способности без какого-либо выделения отдельных классов трафика и регулирования.

2. *Интегрированный сервис grated Service (IntServ).* Согласно RFC 1633, модель интегрированного обслуживания обеспечивает сквозное (*End-to-End*) качество обслуживания, гарантируя необходимую пропускную способность. IntServ использует для своих целей протокол резервирования сетевых ресурсов *RSVP*, который обеспечивает выполнение требований ко всем промежуточным узлам. В отношении *IntServ* часто используется термин «резервирование ресурсов» (*Resource reservation*).

3. *Дифференцированное обслуживание – Differentiated Service* (*DiffServ*). Описана в RFC 2474 и RFC 2475. Обеспечивает QoS на основе распределения ресурсов в ядре сети и определенных классификаторов, и ограничений на границе сети, комбинируемых с целью предоставления требуемых услуг. В этой модели вводится разделение трафика по классам, для каждого из которых определяется свой уровень QoS. DiffServ состоит из управления формированием трафика (классификация пакетов, маркировка, управление интенсивностью) и управления политикой (распределение ресурсов, политика отбрасывания пакетов). DiffServ является наиболее подходящим примером «умного» управления приоритетом трафика.

Алгоритм RED (Random Early Detection), или «Вероятностное заблаговременное обнаружение перегрузки» заложил целое направление работ, посвященных управлению перегрузками в сетях передачи данных посредством модификации и/или разработки новых алгоритмов управления очередями. RED позволяет контролировать нагрузку в рамках очереди маршрутизатора и при обнаружении перегрузки или состоянии, близком к перегрузке, осуществлять

вероятностный сброс пакетов, соблюдая принцип «справедливого распределения ресурсов» и соответственно избегая возникновения проблемы «*Lock-Out*» – *Блокировка*.

Пример функционирования алгоритма RED представлен на рисунках 1.7 и 1.8. Алгоритм RED ориентирован на работу с протоколом TCP, поэтому сброс пакета позволит источнику нагрузки уменьшить размер окна и, таким образом, понизить нагрузку.

Как следует из функциональной схемы (рисунок 1.8), алгоритм RED состоит из двух частей: оценка среднего размера очереди и принятие решения о сбросе вновь поступившего пакета.

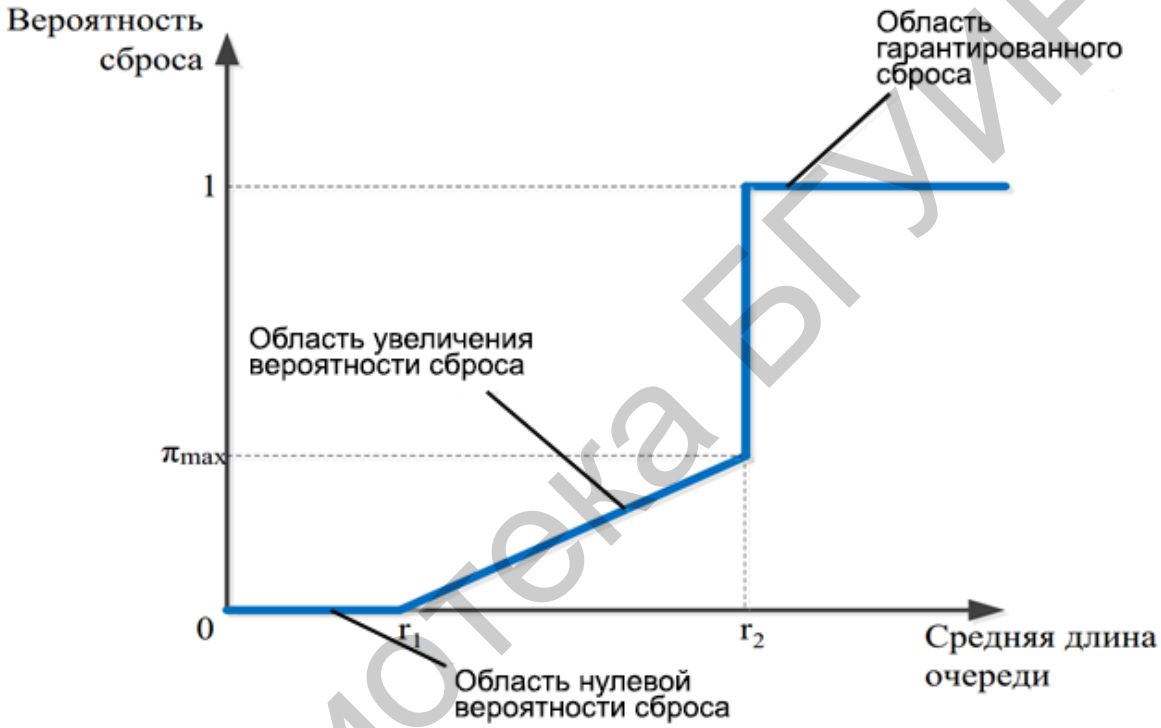


Рисунок 1.7 – Графическая модель алгоритма RED

При поступлении каждого нового пакета RED определяет средний (an average) размер очереди avg. Далее значение avg сравнивается со значениями ранее определенных границ: верхней max_th и нижней min_th, которые определяют степень нагрузки в очереди.

MAX_TH – максимальный порог сброса, MIN_TH – минимальный порог сброса, P – вероятность сброса, q – средняя длина очереди.

В случае если значение avg принадлежит интервалу [0; min_th], то поступающий пакет помещается в очередь, если же значение avg принадлежит интервалу [min_th; max_th], то вычисляется вероятность сброса пакета и либо с вероятностью P_o осуществляется сброс поступающего пакета, либо с вероятностью $1 – P_o$ поступающий пакет помещается в очередь.

Если же значение среднего размера очереди avg превышает значение

max_th, то поступающий пакет обязательно сбрасывается и так как вероятность сброса пакета является функцией от переменной avg, то с повышением нагрузки повышается вероятность сброса поступающего пакета.

Используемые разновидности алгоритма представлены на рисунке 1.9: адаптивный алгоритм RED (*Adaptive RED – ARED*), многоуровневый алгоритм MRED (*multi-level RED*) для поддержки архитектуры *DiffServ*, алгоритм управления состоянием для каждого потока FRED (*Flow RED*).

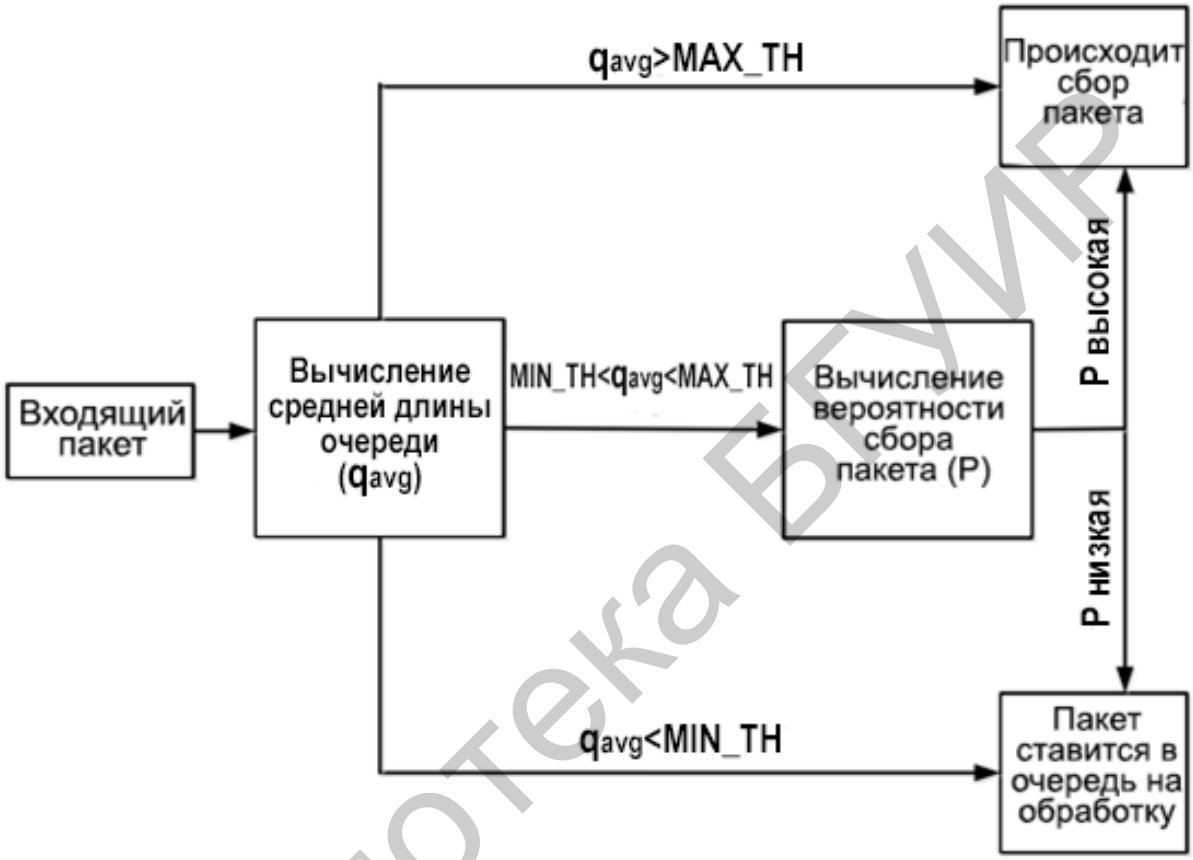


Рисунок 1.8 – Принцип действия алгоритма RED

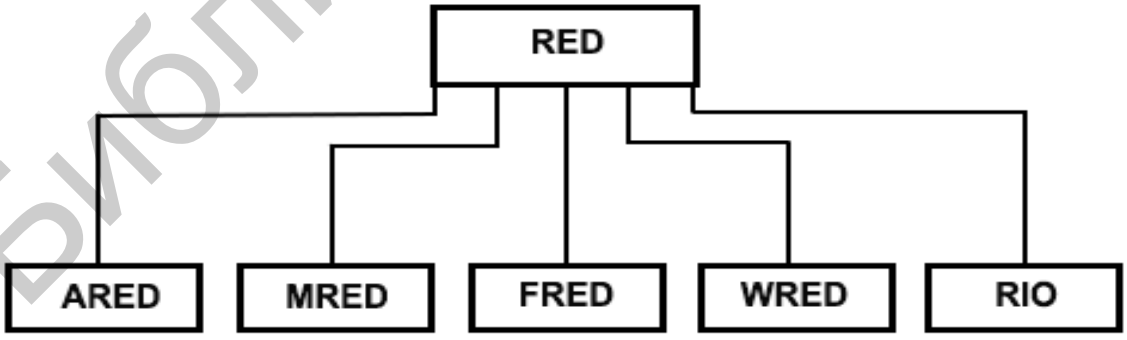


Рисунок 1.9 – Разновидности алгоритма RED

В то же время алгоритмы класса MRED могут быть разбиты на три категории (рисунок 1.10):

1. Категория SAST (*Single Average Single Threshold* – «одно среднее значение, один набор значений границ») представлена базовым алгоритмом RED.

2. Категория SAMT (*Single Average Multiple Threshold* – «одно среднее значение, несколько наборов значений границ») представлена алгоритмом WRED (Weighed RED, взвешенный RED).

3. Категория MAMT (*Multiple Average Multiple Threshold* – «несколько средних значений, несколько наборов значений границ») представлена алгоритмом RIO (*RED In & Out*).

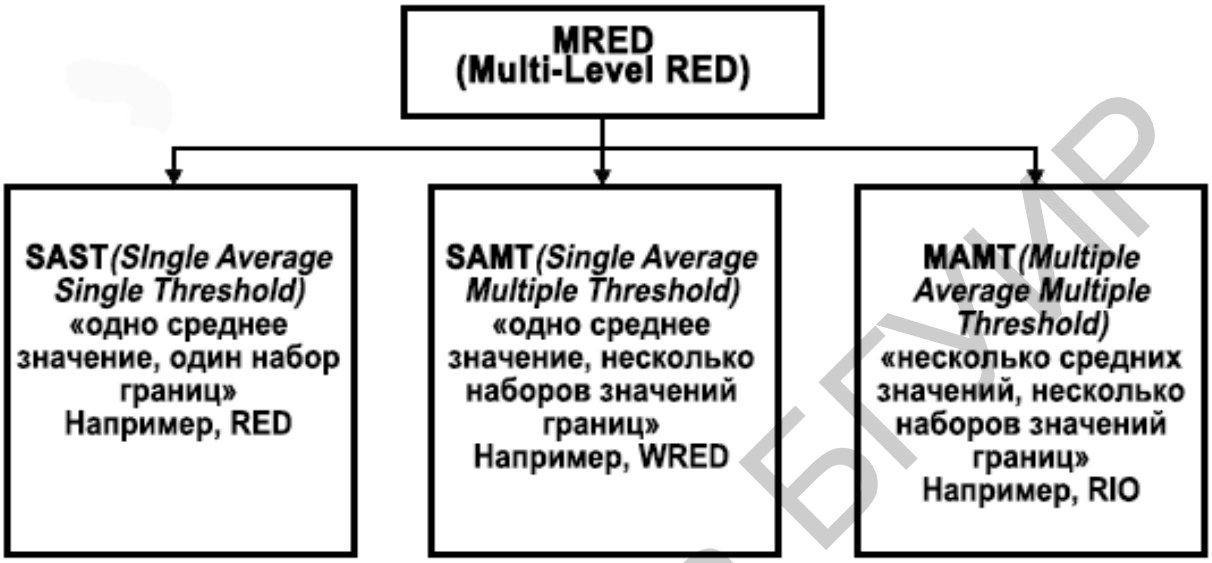


Рисунок 1.10 – Классификация MRED

Алгоритм управления очередью RIO (*RED In & Out*) полностью аналогичен алгоритму RED, за исключением того, что он параметризируется двумя наборами значений параметров – каждый набор для отдельного класса пакетов. Предполагается, что существует два класса пакетов – немаркированные (*in-profile*), т. е. когда пакет принадлежит потоку, значения параметров которого не превышают заранее определенных величин, и маркированные (*out-profile*), т. е. когда пакет принадлежит потоку, значения параметров которого превышают заранее определенные величины.

Для исследования влияния нагрузки UDP на соблюдение принципа «справедливого распределения ресурсов» определены три базовых типа нагрузки в сети Интернет:

1. «Неадаптивная» нагрузка (*Non-adaptive*): нагрузка этого типа забирает столько сетевых ресурсов, сколько для него требуется (или если количество ресурсов меньше, чем ей требуется, то все оставшиеся), при наступлении перегрузки источник не снижает скорость передачи и, соответственно, игнорирует сброс пакетов. В качестве примера можно привести некоторые аудио- и видеоприложения.

2. «Устойчивая» нагрузка (*robust*) – соединения TCP с малым значением RTT. Эти соединения достаточно быстро функционируют, и скорость реакции на возникающие перегрузки высока. Количество пакетов, находящихся в буфере, достаточно велико. Примером может служить трафик HTTP.

3. «Хрупкая» или «неустойчивая» нагрузка (*fragile*) – соединения TCP с большим значением RTT и/или малой скоростью. Время реакции этих соединений на перегрузку велико и количество пакетов, находящихся в буфере, сравнительно мало. Трафик, генерируемый интерактивными приложениями, можно отнести к этому типу нагрузки.

Таким образом, в реальной сети очередь любого маршрутизатора в некоторый момент времени содержит определенное число пакетов, относящихся ко всем трем типам нагрузок – неадаптивной, устойчивой и хрупкой. При наступлении перегрузки интенсивность неадаптивной нагрузки не уменьшится, в то время как интенсивность устойчивой и хрупкой нагрузок понизится.

Суммарная нагрузка упадет, а неадаптивный трафик, пока нагрузка от TCP-соединений мала, начнет занимать ресурсы и вероятность сброса для этого типа нагрузки временно существенно понизится. Однако увеличение размеров окон соединений TCP, относящихся к устойчивой и хрупкой нагрузке, приведет к новой перегрузке. Причем сложится ситуация, когда неадаптивная нагрузка будет занимать достаточно большое количество ресурсов, устойчивая нагрузка – меньше, чем неадаптивная, а хрупкая – совсем мало по причине достаточно медленного обновления размеров окон.

Вероятность сброса для поступающего пакета высока, так как значение среднего размера очереди также высоко, т. е. пакеты соединений TCP будут сбрасываться, даже если нагрузка, создаваемая ими, низкая. В этом случае хрупкая нагрузка подвержена большим потерям, нежели устойчивая, в связи с тем, что время реакции на перегрузку высоко. В данном примере принцип «справедливого распределения ресурсов» не выполняется.

Для решения проблемы несоблюдения справедливого распределения ресурсов при функционировании RED была разработана его модификация FRED (*Flow RED*).

Алгоритм FRED позволяет эффективно изолировать неадаптивную нагрузку от TCP-соединений, а также обеспечить защиту пакетов низкоскоростных соединений TCP от «несправедливого» сброса.

1.6 Разработка файла сценария сетевого фрагмента в NS-2

1.6.1 Основные этапы по созданию файла сценария

Шаг 0: объявить симулятор и установить выходной файл.
Шаг 1: настройка узлов и звеньев канального уровня.
Шаг 2: настройки транспортного уровня.
Шаг 3: настройка уровня приложения.
Шаг 4: установка времени и графиков моделирования.
Шаг 5: объявить окончание выполнения программы.
Модель сетевого фрагмента представлена на рисунке 1.11.

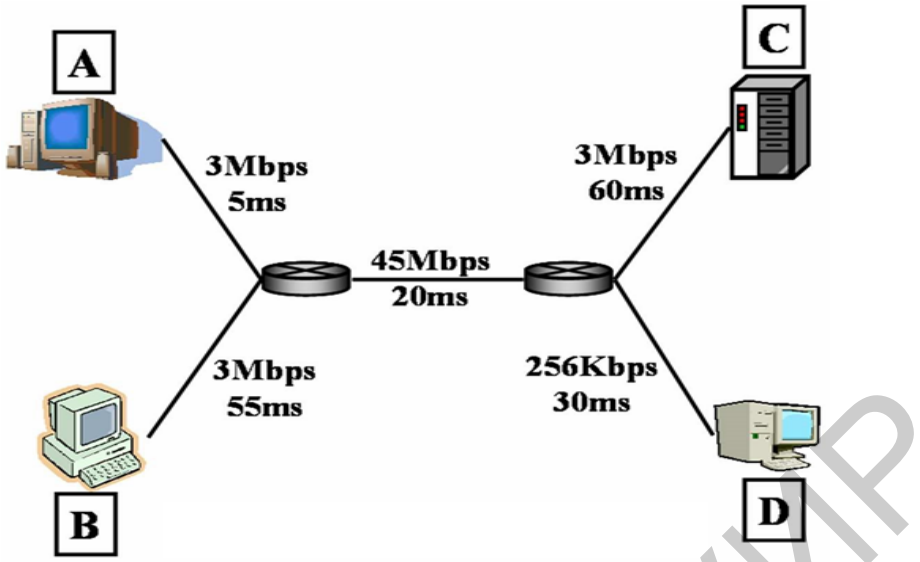


Рисунок 1.11 – Имитационная модель сетевого фрагмента

Шаг 0: объявить симулятор и установить выходной файл

Объявить симулятор и установить выходной файл, как показано ниже.

```
-------------------------------------------------
set ns [new Simulator]
set file [open out.tr w]
$ns trace-all $file
set namfile [open out.nam w]
$ns namtrace-all $namfile
set tcpfile [open out.tcp w]
Agent/TCP set trace_all_oneline_ true
-------------------------------------------------
```

Шаг 1: установка узла и звена данных

Установка узла показана ниже.

```
-------------------------------------------
set n0 [$ns node]
-------------------------------------------
```

После этого узел, который имеет имя n0, готов к использованию (в файле Tcl он будет ссылаться на $n0). Узел пронумерован как 0. Ссылка объявлена ниже.

Установка звена данных показана ниже.

```
-------------------------------------------------
$ns duplex-link $n0 $n2 3Mb 5ms DropTail
$ns duplex-link-op $n0 $n2 orient right-down
-------------------------------------------------
```

В строке 1 связь между двумя узлами n0 и n2 имеет ширину полосы 3 Мбит/с и задержку 5 мс. Объявляется двунаправленное соединение между узлами n0 и n2. DropTail – это тип очереди ожидания. Если имеет место перепол-

нение очереди, то новый введенный пакет будет удален. Строка 2 устанавливает позицию узла и звена данных для Nam. Это не влияет на результат моделирования.

Длина очереди задается следующим образом:

```
-----------------------------------------------
$ns queue-limit $n2 $n3 20
$ns duplex-link-op $n2 $n3 queuePos 0.5
-----------------------------------------------
```

В строке 1 длина очереди по ссылке от n2 до n3 составляет 20 [пакетов]. В строке 2 переменная queuePos 0.5 задана для выполнения результата моделирования в Nam, 0.5 – угол между звеном и очередью, равный (0.5π).

Шаг 2: агент настройки

UDP-агент

Используя UDP для моделирования, отправитель устанавливает агент как агент UDP, в то время как приемник установлен на Null Agent, который осуществляет прием пакета. Установка агента выполняется как

```
-----------------------------------------------
set udp [new Agent/UDP]
$ns attach-agent $n0 $udp
set null [new Agent/Null]
$ns attach-agent $n3 $null
$ns connect $udp $null
$udp set fid_ 0
$ns color 0 blue
-----------------------------------------------
```

В первых четырех строках udp, null агенты заданы для n0, n3. Строка 5 объявила передачу между udp и null. Строка 6 задает номер для потока данных udp. Этот номер будет записан для всего пакета, который отправляется из udp. Используя этот номер, можно легко наблюдать поток, к которому принадлежит пакет, путем поиска файла трассировки. Аналогично в строке 7 отмечается цвет дискретного пакета для отображения результата в Nam.

TCP-агент

Используя TCP для симуляции, отправитель устанавливает агент как агент TCP, в то время как получатель настроен на агента TCPSink. При получении пакета агент TCPSink ответит пакетом подтверждения (ACK). Настройка агента для TCP похожа на UDP:

```
-----------------------------------------------
set tcp [new Agent/TCP]
$ns attach-agent $n1 $tcp
set sink [new Agent/TCPSink]
$ns attach-agent $n3 $sink
$ns connect $tcp $sink
$tcp set fid_ 1
```

```
$ns color 1 red
```

Необходимо убедиться, что **fid_** и цвет отличаются от udp.
Файл трассировки TCP написан следующим образом:

```
$tcp attach-trace $tcpfile
$tcp trace cwnd_
```

Шаг 3: настройка приложения

В общем агент **UDP** использует приложение **CBR**, в то время как агент **TCP** использует приложение **FTP**.

```
set cbr [new Application/Traffic/CBR]
$cbr attach-agent $udp
set ftp [new Application/FTP]
$ftp attach-agent $tcp
```

Шаг 4: установка графика времени для моделирования

Временной график моделирования задается следующим образом:

```
$ns at 1.0 "$cbr start"
$ns at 1.5 "$ftp start"
$ns at 3.0 "$ftp stop"
$ns at 3.5 "$cbr stop"
$ns at 4.0 "finish"
proc finish {} {
global ns file namfile tcpfile
$ns flush-trace
close $file
close $namfile
close $tcpfile
exit 0
}
```

CBR передают данные от 1,0 до 3,5 с, а FTP передает данные с 1,5 до 3,0 с. Функция «finish » используется для файла выходных данных, который всегда находится в конце моделирования.

Шаг 5: объявить завершение

После завершения настройки declare finish записывается в конце файла.

1.6.2 Выполнение программы симуляции и запуск Nam

Путем запуска командной строки начнется выполнение программы.

---

ns sample.tcl

После запуска финишной обработки Nam начнет работу и покажет анимацию моделирования.

---

nam out.nam

1.6.3 Просмотр файла трассировки (out.tr)

Нижние три строки находятся отдельно от файла трассировки.

+ 1.825127 2 3 tcp 1040 --A-- 1 1.0 3.1 30 303
- 1.825367 2 3 cbr 210 -------- 0 0.0 3.0 203 274
r 1.825580 3 2 ack 40 --------- 1 3.1 1.0 29 302

Каждая строка показывает информацию о пакете, а значение каждого столбца в строке имеет следующий смысл.

Столбец 1. Название события:

+: пакет входит в очередь;

-: пакет завершает очередь;

r: пакет принимается узлом;

d: пакет снижается.

Столбец 2. Время для события в секундах.

Столбец 3, 4. Позиция события (звено данных). Число в столбце 3, 4 – это номера узлов, в которых происходят события, связанные с отправкой данных с номера узла в столбце 3 на номер узла в столбце 4.

Столбец 5. Тип пакета:

cbr: пакет, созданный приложением CBR, отправленный агентом UDP;

tcp: пакет, созданный приложением FTP, отправленный агентом TCP Agent;

ack: пакет, созданный агентом TCP Sink (ACK).

Столбец 6. Размер пакета в байтах.

Столбец 7. Флаг, используемый для первого пакета повторной передачи в ретрансляционном управлении, А является значением флага. Фактически флаг не отображается для всех пакетов повторной передачи.

Столбец 8. Номер потока, к которому относится пакет.

Столбец 9. Отправитель пакетов.

Например, если значение равно 1.0, это означает, что узел, у которого есть ID, равный 1, отправляет данные из порта 0.

Столбец 10. Приемник пакетов.

Например, если значение равно 3.1, это означает, что узел с идентификатором, равным 3, принимает данные по порту 1.

Столбец 11. Номер передаваемого пакета.

Столбец 12. Номер идентификатора этого пакета. Каждый пакет имеет индивидуальный (ID).

Примечание – В реальной сети компьютер должен получать данные из многих служб (Web, Mail ...). В NS-2 будет установлено множество агентов. По этой причине, если пользоваться только идентификатором отправителя и получателя, будет оставаться неизвестным, какой пакет принадлежит агенту. Использование другого порта для каждого агента это – способ решить эту проблему. Фактически в сети для каждого сервиса используется отдельный порт.

Столбец 11. Последовательный номер передаваемого пакета

Столбец 12. Идентификационный номер этого пакета. Каждый пакет имеет индивидуальный идентификационный номер.

Далее в следующих двух строках

+ 1.70342 2 3 tcp 1040 ------- 1 1.0 3.1 30 231
d 1.70342 2 3 tcp 1040 ------- 1 1.0 3.1 30 231

В момент времени 1.70342 с пакет входит в очередь, а затем отбрасывается одновременно. Это означает, что пакет пытается войти в очередь и будет утерян, потому что очередь переполнена.

1.6.4 Файл трассировки (out.tcp)

Ниже приведена часть файла трассировки TCP (фактически это одна строка).

---

time: 1.57005 saddr: 1 sport: 0 daddr: 3 dport: 1 maxseq: 2 hiack: 1 seqno: 3
cwnd: 3.000 ssthresh: 20 dupacks: 0 rtt: 0.040 srtt: 0.030 rttvar: 0.015 bkoff: 1

Каждый параметр TCP имеет свое назначение, когда пакет отправляется (таблица 1.1).

Таблица 1.1 – Назначение TCP-параметров

Название параметра	*Значение параметра*
1	2
Time [Packet send time]	Время отправки пакета
Saddr [ID number of sender]	ID номер отправителя
Sport [Port number of sender]	Номер порта отправителя
Daddr [ID number of receiver]	Идентификационный номер приемника
Dport [Port number of receiver]	Номер порта приемника
Maxseq [Maximum sequence number after sending]	Максимальный порядковый номер после отправки
Hiack [Maximum sequence number after receiving]	Максимальный порядковый номер после получения

1	2
Seqno [Sending sequence number]	Порядковый номер отправления
Cwnd [Congestion Window Size]	Размер окна перегрузки
Ssthresh [Slow Start Threshold value]	Пороговое значение мягкого старта
Dupacks [Number of duplicate ACK]	Количество дубликатов ACK
Rtt [Round trip time delay]	Время задержки в оба конца
Srtt [Smoothing of rtt]	Сглаживание rtt
Rttvar [Average deviation of srtt]	Среднее отклонение srtt
Bkoff [times of exponential back-of]	Время экспоненциального отступления

1.6.5 Программный код NS-2 сетевого фрагмента

```
##### Declare Simulator
set ns [new Simulator]
##### Setting output file
set file [open out.tr w]
$ns trace-all $file
set namfile [open out.nam w]
$ns namtrace-all $namfile
set tcpfile [open out.tcp w]
Agent/TCP set trace_all_oneline_ true

##### Setting Node
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]

##### Setting Link
$ns duplex-link $n0 $n2 3Mb 5ms DropTail
$ns duplex-link $n1 $n2 3Mb 5ms DropTail
$ns duplex-link $n2 $n3 1.5Mb 10ms DropTail
$ns duplex-link-op $n0 $n2 orient right-down
$ns duplex-link-op $n1 $n2 orient right-up
$ns duplex-link-op $n2 $n3 orient right
$ns duplex-link-op $n2 $n3 queuePos 0.5

##### Setting Queue Length
$ns queue-limit $n2 $n3 20

##### Setting UDP Agent
set udp [new Agent/UDP]
$ns attach-agent $n0 $udp
set null [new Agent/Null]
```

```
$ns attach-agent $n3 $null
$ns connect $udp $null
$udp set fid_ 0
$ns color 0 blue
##### Setting CBR Application
set cbr [new Application/Traffic/CBR]
$cbr attach-agent $udp

##### Setting TCP Agent
set tcp [new Agent/TCP]
$ns attach-agent $n1 $tcp
set sink [new Agent/TCPSink]
$ns attach-agent $n3 $sink
$ns connect $tcp $sink
$tcp set fid_ 1
$ns color 1 red
### Setting output file of TCP Agent
$tcp attach-trace $tcpfile
$tcp trace cwnd_

##### Setting FTP Application
set ftp [new Application/FTP]
$ftp attach-agent $tcp

##### Setting time schedule of simulation
$ns at 1.0 "$cbr start"
$ns at 1.5 "$ftp start"
$ns at 3.0 "$ftp stop"
$ns at 3.5 "$cbr stop"
$ns at 4.0 "finish"
proc finish {} {
    global ns file namfile tcpfile
    $ns flush-trace
    close $file
    close $namfile
    close $tcpfile
    exit 0
}
##### Finish setting and start simulation $ns run
```

1.7 Разработка файла сценария лабораторного задания в NS-2

1.7.1 Топология сетевого фрагмента

Сетевой фрагмент включает пять функциональных узлов: два компьютера, которые являются оконечными объектами схемы (Source1/Destination2 и Source2/Destination1), и три маршрутизатора, два из которых являются пограничными (Edge1 и Edge2) и один центральный (Core) (рисунок 1.12).

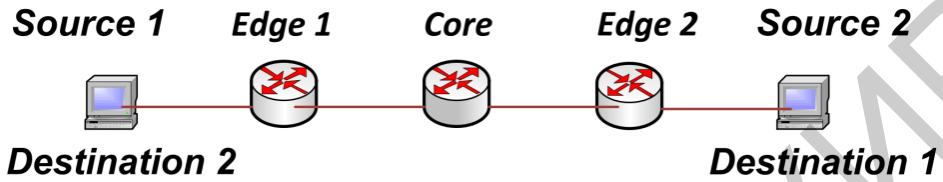


Рисунок 1.12 – Сетевой фрагмент в NS-2

1.7.2 Разработка программного кода с двумя CBR-потоками

Определение топологии сети в NS-2:

```
set s1 [$ns node] // Задание первого оконечного узла
set e1 [$ns node] // Задание первого оконечного маршрутизатора
set core [$ns node] // Задание центрального маршрутизатора
set e2 [$ns node] // Задание второго оконечного маршрутизатора
set dest [$ns node] // Задание второго оконечного узла
```

Установление связи между узлами:

```
$ns duplex-link $s1 $e1 10Mb 5ms DropTail
$ns simplex-link $e1 $core 10Mb 5ms dsRED/edge
$ns simplex-link $core $e1 10Mb 5ms dsRED/core
$ns simplex-link $core $e2 5Mb 5ms dsRED/core
$ns simplex-link $e2 $core 5Mb 5ms dsRED/edge
$ns duplex-link $e2 $dest 10Mb 5ms DropTail
```

Таким образом, устанавливается дуплексный канал между узлами s1 и e1 и между узлами dest и e2 с использованием алгоритма DropTail на этих участках. Между маршрутизаторами e1 и core, а также между e2 и core установлены два симплексных канала с использованием алгоритма RED.

```
$ns duplex-link-op $s1 $e1 orient right
$ns duplex-link-op $e1 $core orient right
$ns duplex-link-op $core $e2 orient right
$ns duplex-link-op $e2 $dest orient right
```

Данными строками устанавливается для визуализатора Nam ориентация элементов сети в одну линию. В определяемой топологии сети трафик будет идти от source1 до destination1 и от source2 до destination2.

Исходя из этого буферы виртуальных очередей определяются строками

```
set qE1C [[$ns link $e1 $core] queue]
set qE2C [[$ns link $e2 $core] queue]
set qCE1 [[$ns link $core $e1] queue]
set qCE2 [[$ns link $core $e2] queue]
```

Параметры алгоритма RED для маршрутизаторов задаются строками

```
$qE1C meanPktSize $packetSize
$qE1C set numQueues_1 // Определяется количество физических очередей
$qE1C setNumPrec 2 // Определяется количество виртуальных очередей
$qE1C addPolicyEntry [$s1 id] [$dest id] TokenBucket 10 $cir0 $cbs0
$qE1C addPolicyEntry [$dest id] [$s1 id] TokenBucket 10 $cir1 $cbs1
$qE1C addPolicerEntry TokenBucket 10 11
$qE1C addPHBEntry 10 0 0
$qE1C addPHBEntry 11 0 1
$qE1C configQ 0 0 20 40 0.02
$qE1C configQ 0 1 10 20 0.10
```

В последних двух строках задаются параметры виртуальных очередей:

$qE1C configQ 0 0 20 40 0.02 – указывается, что если количество пакетов в очереди будет от 20 до 40, то вероятность отбрасывания будет равна 0.02.

$qE1C configQ 0 1 10 20 0.10 – указывается, что если число пакетов в очереди будет от 10 до 20, то вероятность отбрасывания пакета будет равна 0.1.

Для настройки трафика в сети необходимо создать два агента: TCP и UDP, а после этого прикрепить к ним CBR-трафик.

Создание агента UDP:

```
set udp0 [new Agent/UDP] // Создание агента UDP
$ns attach-agent $s1 $udp0 // Прикрепление агента UDP к узлу s1
```

Создание источника CBR-трафика и присоединение его к агенту udp0:

```
set cbr0 [new Application/Traffic/CBR] // Создание источника трафика
$cbr0 set packetSize_500
$cbr0 set interval_0.005
$cbr0 attach-agent $udp0 // Присоединение трафика к агенту
```

Создание агента TCP:

```
set tcp1 [new Agent/TCP] // Создание TCP-агента
$ns attach-agent $dest $tcp1 // Прикрепление агента к узлу Dest
```

Создание источника CBR-трафика и присоединение его к агенту tcp1:

```
set cbr1 [new Application/Traffic/CBR] // Создание источника трафика
$cbr1 set packetSize_500
$cbr1 set interval_0.005
```

```
$cbr1 attach-agent $tcp1 // Присоединение трафика к агенту
```

Создание агента-получателя для udp0

```
set null0 [new Agent/Null]
$ns attach-agent $dest $null0
```

Создание агента-получателя для tcp1

```
set sink1 [new Agent/TCPSink]
$ns attach-agent $s1 $sink1
```

В результате выполнена настройка трафика в сетевом фрагменте с двумя CBR-потоками. Один идет от s1 до dest с помощью UDP, а второй идет от dest до s1 с помощью TCP.

Лабораторное задание

1. Задать источники трафика согласно номеру варианта и построить сеть в NS-2
2. Реализовать визуализацию схемы в Nam.
3. Получить графики изменения задержки и джиттера во времени.

Таблица 1.2 – Варианты выполнения лабораторного задания

1	2	3	4	5	6	7	8	9	0
CBR1	Exp	Exp	CBR1	CBR2	Парето	Парето	Парето	Exp	CBR2
Exp	CBR1	Exp	CBR1	Парето	Exp	Парето	CBR2	Парето	CBR2

CBR1 (Constant Bit Rate) – источник трафика с постоянной скоростью 800 000 бит/с.

CBR2 (Constant Bit Rate) – источник трафика с постоянной скоростью 1 600 000 бит/с .

Exp – источник трафика с распределением по экспоненциальному закону.

Парето – источник трафика с распределением по закону Парето.

Содержание отчета

1. Цель работы.
2. Топология фрагмента сети.
3. Фрагмент программного кода.
4. Графики джиттера и задержки.
5. Выводы по работе.

Контрольные вопросы

1. Поясните логическую структуру симулятора.
2. Поясните назначение аниматора и основные опции.
3. Определите логическую структуру файла сценария.
4. Поясните элементы Tcl-сценариев, необходимые при создании узлов и связей между ними.

5. Перечислите основные параметры линии связи между узлами и поясните, как они задаются в Tcl-сценарии.

6. Каким образом можно вручную размещать компоненты на схеме сети? Приведите примеры.

7. Что такое агенты и какие функции они выполняют?

8. Перечислите известные агенты и опишите их основные характеристики.

9. Какие параметры агентов учитываются при моделировании сети и каким образом?

10.Что такое CBR-генераторы и как они участвуют в моделировании?

11. Какие параметры CBR-генераторов могут быть заданы при моделировании?

12. Дайте определение понятию QoS. Какими параметрами он характеризуется?

13. Какие виды организации очереди используются в NS-2? Приведите примеры.

14. Поясните графическую модель алгоритма RED и других алгоритмов, производные от RED: ARED, MRED, WRED, FRED, алгоритм RIO.

15. Поясните графики экспериментальных исследований для сетевого фрагмента с заданным типом трафика.

Литература

1. The ns Manual / ed. Kannan Varadhan [Электронный ресурс]. – Режим доступа : http://www.isi.edu>nsnam/ns/doc/ns_doc.pdf.

2. Altman, E. NS for Beginners Lecture Notes / E. Altman, T. Jimenes // Univ. de Los Andes, Merida, Venezuela, Sept. 2002 [Электронный ресурс]. – Режим доступа : http: // www-sop.inria.fr/maestro/personnel/Eitan.Altman/COURS-NS/n3.pdf/.

3. Уэлш, Б. Практическое программирование на Tcl и Tk / Б. Уэлш, К. Джонс, Дж. Хоббс. – М. : Вильямс, 2004.

4. Петровский, А. И. Командный язык программирования Tcl (Tool Command Language) / А. И. Петровский. – М. : Майор, 2001.

5. Кучерявый, Е. А. Управление трафиком и качество обслуживания в сети Internet / Е. А. Кучерявый. – СПб. : Наука и техника, 2004.

2 ИМИТАЦИОННОЕ МОДЕЛИРОВАНИЕ В ПАКЕТЕ FUZZY LOGIC TOOLBOX

Цель работы: изучение метода нечеткой логики как средства отображения неопределенностей при моделировании и проектировании систем управления; изучение основных функций пакета FUZZY LOGIC TOOLBOX программной среды MATLAB; получение навыков проектирования систем нечеткого вывода в пакете FUZZY LOGIC TOOLBOX; получение навыков построения нечеткой аппроксимирующей системы в пакете FUZZY LOGIC TOOLBOX.

2.1 Нечеткие множества и основные операции над ними

После того как компьютеры перестали быть единичными и уникальными изделиями, началась массовая автоматизация по двум практически независимым направлениям: автоматизация бизнес-процессов на основе информационных технологий (IT – Information Technology) и автоматизация технологических процессов на основе операционных технологий (OT – Operational Technology), которые представляют комплекс аппаратного и программного обеспечения, предназначенного для контроля и управления физическими процессами. Такие автоматизированные системы управления технологическими процессами являются элементами систем искусственного интеллекта (ИИ) и лежат в основе концепции Industrial Internet of Things (IIoT), выполняющего взаимодействие по технологии M2M (machine-to-machine).

В рамках систем искусственного интеллекта разработан ряд методов, позволяющих решать различные задачи smart-управления без использования сложной и дорогостоящей разработки математических моделей управляемых объектов.

Технологии ИИ включают в себя искусственные нейронные сети (ИНС), экспертные системы (ЭС), нечеткую (fuzzy) логику (НЛ), генетические алгоритмы (ГА) и т. д.

Например, ИНС-сети обладают способностью к обучению, ЭС-системы принимают решения на основе наборов правил и опыта экспертов, а системы с нечеткой логикой оперируют такими понятиями, как неопределенность и частичная/приблизительная истина. Данные методы предназначены для решения очень сложных нелинейных задач, которые либо превышают возможности общепринятых алгоритмических методов, либо требуют для своего решения слишком больших материальных и временных затрат.

В обычной теории множеств существуют несколько способов задания множества. Одним из них является задание множеств с помощью характеристической функции, определяемой следующим образом. Пусть U – так называемое универсальное множество, из элементов которого образованы все остальные множества, рассматриваемые в настоящей задаче, например: множество всех целых чисел, множество всех гладких функций, заданных на действительной оси, и т. д. В качестве универсального в проводимом анализе будет ис-

пользовано множество всех действительных чисел. Характеристическая функция множества $A \subseteq U$ – это функция μ_A, значения которой указывают, является ли $x \in U$ элементом множества A:

$$\mu_A(x) = \begin{cases} 1, & \text{если } x \in A, \\ 0, & \text{если } x \notin A. \end{cases} \quad (2.1)$$

Особенностью этой функции является бинарный характер ее значений – 1 или 0. С точки зрения характеристической функции нечеткие множества являются естественным обобщением обычных множеств, когда мы отказываемся от бинарного характера этой функции и предполагаем, что она может принимать любые значения из отрезка [0,1]. В теории нечетких множеств характеристическая функция называется функцией принадлежности, а ее значение $\mu_A(x)$ – степенью принадлежности элемента x нечеткому множеству A.

Более точно нечетким множеством A называется совокупность пар

$$\forall\, x \in U \quad \{(x; \mu_A(x))\},$$

где μ_A – функция принадлежности: $\mu_A: U \longrightarrow [0,1]$.

Пусть, например,

$$U = \{a, b, c, d, e\}, A = \{(a; 0), (b; 0{,}1), (c; 0{,}5), (d; 0{,}9), (e; 1)\}.$$

Будем говорить тогда, что элемент $\boldsymbol{a}$ не принадлежит множеству A, элемент $\boldsymbol{b}$ принадлежит ему в малой степени, элемент $\boldsymbol{c}$ более или менее принадлежит, элемент $\boldsymbol{d}$ принадлежит в значительной степени, $\boldsymbol{e}$ является элементом A. В различных приложениях используются различные функции принадлежности, приведем несколько типичных примеров таких функций для множеств, заданных словесно. Пусть A – множество чисел, близких к 10, тогда можно принять

$$\mu_A(x) = (1 + k|x - 10|^m)^{-1}, \; k > 0, \quad m = 3. \quad (2.2)$$

В зависимости от требуемой степени близости к 10 показатель степени m может быть взят равным 1, 2, 4 и т. д. Например, для описания множества чисел, очень близких к 10, можно положить $m = 4$; для множества чисел, не очень далеких от 10, $m = 2$. Учет более тонких нюансов может быть произведен за счет коэффициента k или использования дробной степени показателя m. Если B – множество чисел, значительно больших 10, то в качестве функции принадлежности может быть взята функция

$$\mu_B(x) = \begin{cases} (1 + k(x - 10)^{-2})^{-1}, & x > 10, k > 0, \\ 0, & x \le 10. \end{cases} \quad (2.3)$$

Если C – множество чисел, которые не должны значительно выходить за пределы интервала (4, 8), то $\mu_C(x) = (1 + k(x - 6)^2)^{-1}, \; k > 0$.

Приближенные графики рассмотренных функций приведены на рисунке 2.1, а.

Конкретный вид функции принадлежности (и значения входящих в нее параметров) носит в значительной мере субъективный характер. Уменьшить степень этой субъективности можно, используя метод экспертных оценок, суть которого состоит в том, что как вид функции принадлежности, так и значения

соответствующих параметров являются результатом коллективного творчества группы специалистов в рассматриваемой области – экспертов.

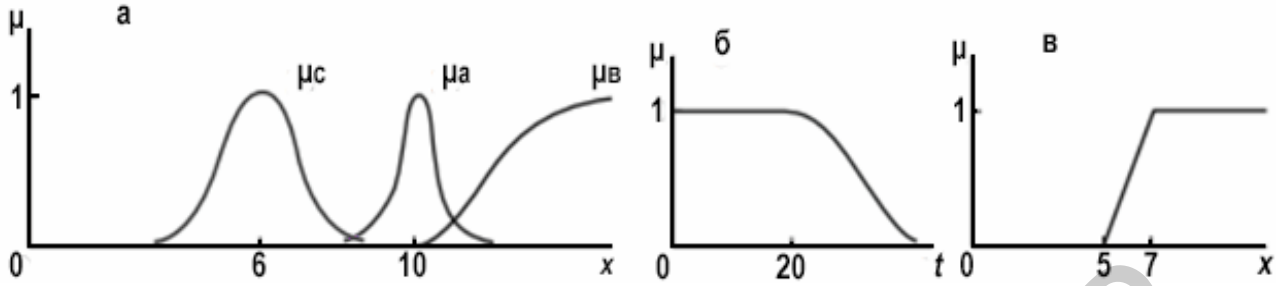


Рисунок 2.1 – Примеры функций принадлежности

Пусть, например, решается задача определения значения некоторого параметра α. Тогда каждым из n экспертов назначается свое значение этого параметра $-\alpha_1, \alpha_2, \ldots, \alpha_n$, эти числа усредняются:

$$\bar{\alpha} = \frac{1}{n}(\alpha_1 + \alpha_2 + \cdots + \alpha_n) \qquad (2.4)$$

и полученный результат используется в качестве значения параметра α.

В соответствии со степенью опытности экспертов им могут быть присвоены веса $\upsilon_1, \upsilon_2, \ldots, \upsilon_n$, с учетом которых предыдущая формула несколько усложняется:

$$\bar{\alpha} = \frac{1}{n\upsilon}(\alpha_1\upsilon_1 + \alpha_2\upsilon_2 + \cdots + \alpha_n\upsilon_n), \qquad (2.5)$$

где $\upsilon = \upsilon_1 + \upsilon_2 + \cdots + \upsilon_n$.

Для выполнения операций над нечеткими множествами вводится понятие лингвистической переменной. Ее можно определить как переменную, значениями (термами) которой являются не числа, а слова или предложения естественного (или формального) языка. Например, лингвистическая переменная «возраст» может принимать следующие значения: очень молодой, молодой, среднего возраста, старый, очень старый и др. – в зависимости от требуемой степени детальности описания.

Ясно, что переменная «возраст» будет обычной переменной, если ее значения – точные числа; лингвистической она становится, будучи использованной в нечетких рассуждениях человека. Каждому терму лингвистической переменной соответствует определенное нечеткое множество со своей функцией принадлежности, которая описывает совместимость этого терма с различными числовыми значениями. Так, лингвистическому значению «молодой» может соответствовать функция принадлежности, приведенная на рисунке 2.1, б.

Другой вид функции принадлежности приведен на рисунке 2.1, в, применительно к рассуждению, например, сколько предметов составляют кучу: один, два, …, пять – не куча, семь и больше – куча, а шесть?

Функция принадлежности, которая описывает результат таких рассуждений, приведена на рисунке 2.1, в. Для наглядности ее график изображен сплошной линией.

Следовательно, тройка параметров $\langle A, U, \mu_A(x)\rangle$ определяет понятие нечеткой переменной, где A – наименование (имя) нечеткой переменной; U – универсальное множество (область определения); $\mu_A(x)$ – ограничения на возможные значения (смысл) переменной A.

Таким образом, нечеткая переменная – это поименованное нечеткое множество.

Лингвистическая переменная представляет собой пятерку параметров, где:

- A – наименование (имя) лингвистической переменной;
- $T(A)$ – терм-множество лингвистической переменной A, т. е. множество ее лингвистических значений;
- U – универсальное множество, в котором определяются значения лингвистической переменной A;
- G – синтаксическое правило, порождающее значения лингвистической переменной A (часто имеет форму грамматики);
- M – семантическое правило, которое ставит в соответствие каждому элементу $T(A)$ его «смысл» как нечеткое подмножество U.

Для задания правила G обычно задаются *базовые* значения (например, маленький, средний, большой), *модификаторы* (не-, очень-, слегка- и т. п.) и правила образования элементов терм-множества (небольшой, очень маленький, слегка большой, не очень маленький т. д.).

При задании правила M предполагается, что функции принадлежности базовых терминов заданы, а также известно, каким образом модификаторы воздействуют на функции принадлежности базовых значений. Например, если известна функция принадлежности $\mu_A(x)$ некоторого базового значения A, то функции принадлежности лингвистических значений «*не A*», «*очень A*» могут вычисляться следующим образом:

$$\mu_{\text{очень } A(x)} = \mu_A^2(x). \qquad (2.6)$$

Таким образом, предполагается, что, задав функции принадлежностей нескольких базовых значений и выбрав формализации логических операций «*и*», «*или*», «*не*», можно «вычислять» функции принадлежности любых элементов терм-множества $T(A)$. Более того, можно «вычислять смысл» любого высказывания, производя соответствующие операции над функциями принадлежности понятий различных лингвистических переменных. Этот «смысл» может быть представлен либо в виде функции принадлежности, либо в виде лингвистического значения, наилучшим способом аппроксимирующем результирующую функцию принадлежности.

Над нечеткими множествами можно производить различные операции, при этом необходимо определить их так, чтобы в частном случае, когда нечеткое множество является четким (обычным), эти операции переходили в обыч-

ные операции теории множеств, т. е. операции над нечеткими множествами должны обобщать соответствующие операции над обычными множествами. При этом обобщение может быть реализовано различными способами, из-за чего какой-либо операции над обычными множествами может соответствовать несколько операций в теории нечетких множеств.

Так, пусть A и B – нечеткие множества. Будем говорить, что A содержится в B, и обозначать $A \subseteq B$, если

$$\forall x \in U, \quad \mu_A(x) \leq \mu_B(x). \tag{2.7}$$

Например, если A – множество чисел, очень близких к 10, а B – множество чисел, близких к 10, то $A \subseteq B$.

Формально это можно проверить, используя функции принадлежности, описанные выше. Если A и B – обычные множества, а μ_A и μ_B – характеристические функции, то из неравенства (2.1) следует, что если некоторый элемент x принадлежит A, т. е. $\mu_A(x) = 1$, то он принадлежит и B, поскольку $\mu_B(x) = 1$. Таким образом, определение (2.1) корректно в том смысле, что в частном случае оно переходит в известное.

Два нечетких множества A и B равны в том и только том случае, если равны их функции принадлежности.

Объединением нечетких множеств A и B называется нечеткое множество, обозначаемое $A \cup B$, функция принадлежности которого определяется следующим образом:

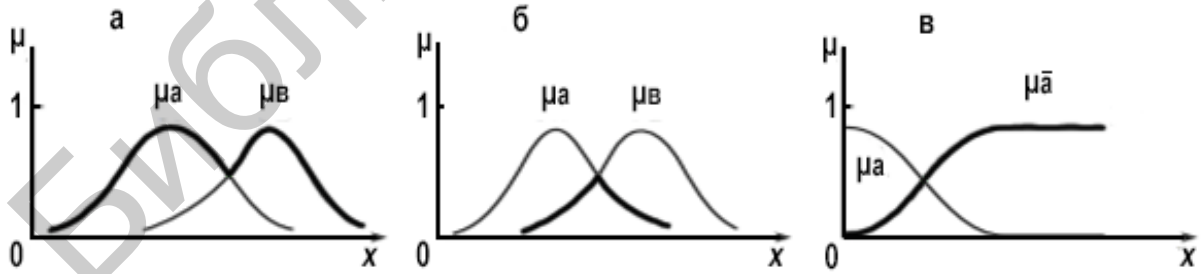
$$\forall x \in U, \ \mu_{A \cup B}(x) = \max\{\mu_A(x), \mu_B(x)\}. \tag{2.8}$$

Пересечение множеств $A \cap B$ определяется функцией принадлежности

$$\forall x \in U, \ \mu_{A \cap B}(x) = \min\{\mu_A(x), \mu_B(x)\}. \tag{2.9}$$

Дополнение $\bar{A}$ нечеткого множества A имеет функцию принадлежности $\mu_{\bar{A}}(x) = 1 - \mu_A(x)$. На рисунке 2.2 приведены функции принадлежности соответствующих множеств. Алгебраическое произведение множеств A и B – это множество $A \cdot B$ с функцией принадлежности

$$\mu_{A \cdot B} = \mu_A(x)\mu_B(x). \tag{2.10}$$



а – операция объединения; б – операция пересечения;
в – операция дополнения

Рисунок 2.2 – Функции принадлежности для некоторых операций над нечеткими множествами

Нетрудно проверить, что в случае обычных множеств эта операция переходит в пересечение множеств, которое, таким образом, имеет в теории нечетких множеств два обобщения: алгебраическое произведение и пересечение.

Аналогично обстоит дело и с операцией объединения – в теории нечетких множеств ей соответствуют операции объединения и алгебраической суммы $A + B$ с функцией принадлежности:

$$\mu_{A+B}(x) = \mu_A(x) + \mu_B(x) - \mu_A(x)\mu_B(x). \quad (2.11)$$

Для операций объединения, пересечения и дополнения нечетких множеств остаются справедливыми практически все свойства соответствующих обычных операций: коммутативность, ассоциативность, дистрибутивность. Читателю предлагается самостоятельно убедиться в справедливости правил де Моргана:

$$\overline{A \cup B} = \overline{A} \cap \overline{B} \text{ и } \overline{A \cap B} = \overline{A} \cup \overline{B}. \quad (2.12)$$

При этом, однако, $A \cap \overline{A} \neq \emptyset$ и $A \cup \overline{A} \neq U$. Например, если A – множество целых неотрицательных чисел, не очень далеких от 0, то функция μ_A может иметь вид

$$\mu_A = \{(0;1), (1;1), (2;1), (3;0{,}9), (4;0{,}8), \dots, (9;0{,}3), \dots\}. \quad (2.13)$$

По определению, однако, $\overline{A}$ – множество целых, не очень далеких от 0 и одновременно не очень близких к 0:

$$\mu_{A\cap\overline{A}} = \{(0;0), (1;0), (2;0), (3;0{,}1), (4;0{,}2), \dots, (9;0{,}3), \dots\}. \quad (2.14)$$

Операции алгебраического произведения и алгебраического произведения и алгебраической суммы обладают более ограниченным набором свойств, для них не выполняется дистрибутивность, но правила де Моргана остаются в силе: $\overline{A + B} = \overline{A} \cdot \overline{B}$ и $\overline{A \cdot B} = \overline{A} + \overline{B}$. Благодаря использованию категории нечеткости можно по-новому взглянуть на решение задач оптимизации. Этот вывод иллюстрируется на примере определения наибольшего значения функции при ограничении на переменную. Из рисунка 2.3 видно, что при заданном виде целевой функции решение достигается в точке x^*.

Ограничение иногда носит объективный характер (например, обусловлено законами природы, ограниченными ресурсами и т. д.) и, безусловно, должно соблюдаться. Нередко же его наличие связано с субъективными причинами, в частности нежеланием лица, поставившего задачу, выйти за определенные пределы (в данном случае превысить число).

В этом случае более реалистичным является подход, когда ограничение формулируется нечетким образом: «переменная» не должна быть существенно больше числа. Соответствующая функция принадлежности приведена на рисунке 2.3, на котором также приведена нормированная функция как

$$f^* = \frac{f(x)}{\max(f(x))}. \quad (2.15)$$

В интервале, где $f^*(x) \geq 0$, функцию можно рассматривать в качестве функции принадлежности некоторого нечеткого множества. По условию задачи необходимо найти максимальное значение $f^*(x)$ при выполнении нечеткого ограничения на переменную x. Это отвечает операции пересечения соответствующих нечетких множеств и дает функцию принадлежности μ^*, отобража-

емую на рисунке 2.3 составной утолщенной линией из двух пересечений кривых μ^* и f^*. Ее можно рассматривать как нечеткую инструкцию решения задачи. С позиции нечеткого подхода этой инструкции наилучшим образом соответствует ее наибольшее значение, достигаемое при $x = x^*$, что можно считать решением задачи; ясно, что

$$f(x^*) > f(c). \qquad (2.16)$$

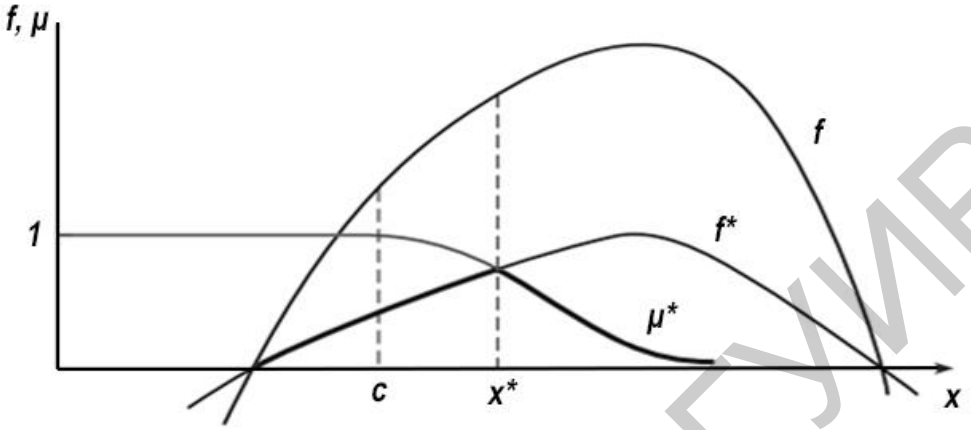


Рисунок 2.3 – Иллюстрация к решению задачи оптимизации

Данную задачу можно также рассматривать как многокритериальную, поскольку, с одной стороны, требуется найти наибольшее значение функции f(или f^*), а с другой – наилучшим образом удовлетворить нечеткому ограничению, что также приводит к необходимости поиска наибольшего значения функции принадлежности μ. Известно, что такие задачи, как правило, не имеют решения, если не привлекать дополнительные соображения, приводящие, в частности, к переходу от многих критериев к одному тем или иным способом (так называемая свертка критериев). Одним из таких способов является рассмотренный переход к функции принадлежности μ^*.

Еще одной из обширных областей приложения нечетких концепций является теория управления техническими системами. При этом имеются в виду не только сложные системы, существенной составляющей которых является человек с его нечеткими представлениями, но и системы, работающие без вмешательства человека. Для многих таких систем известны весьма точные модели, которые позволяют строить оптимальные алгоритмы управления. Тем не менее даже в этом случае использование нечетких представлений может дать заметный эффект.

Дело в том, что для построения оптимальных алгоритмов управления необходимо знать точные значения параметров системы. Последние нередко известны только приближенно, причем в процессе работы системы они могут изменяться в широких пределах. Кроме того, при функционировании системы на нее действуют различные внешние факторы, которые имеют случайный характер и не всегда могут быть адекватно отражены в алгоритме управления.

Вышесказанное приводит к тому, что эффективность алгоритма управления (выраженная каким-либо критерием) весьма высокая для расчетных значе-

ний параметров системы, резко падает при изменении этих параметров (кривая 1 на рисунке 2.4). Один из путей преодоления этого недостатка состоит в использовании нечетких моделей и нечетких алгоритмов управления, которые имеют меньшую эффективность для расчетных значений параметров, но сохраняют ее почти постоянной в широком диапазоне изменения этих значений (кривая 2 на рисунке 2.4).

Рисунок 2.4 – Иллюстрация к эффективности алгоритмов управления

Реализация этого подхода состоит из трех основных этапов:

1) фаззификация – переход от точных исходных данных решаемой задачи к нечетким на основе входным функциям принадлежности;

2) решение задачи с использованием нечетких рассуждений (нечеткой логики);

3) дефазификация – переход от нечетких инструкций к четким на основе выходным функциям принадлежности.

Проиллюстрировать этот подход можно на простом примере управления уличным движением. Входным точным значением здесь является плотность потока ρ автомашин на некоторой дороге (предполагается, что $\rho \leq 1$); выходным – длительность зеленого сигнала τ (для простоты рассуждений плотность движения на пересекаемой дороге принимается примерно постоянной). Введем две лингвистические переменные: «плотность потока автомашин» с термами «малая», «средняя», «большая» и «длительность зеленого сигнала» с термами «короткая», «средняя», «большая». Соответствующие входные и выходные функции принадлежности приведены на рисунке 2.5.

Нечеткая инструкция управлением этой системой может сводиться к трем простым правилам:

1) при малой плотности длительность зеленого сигнала должна быть короткой;

2) средней плотности должна соответствовать средняя длительность;

3) в случае большой плотности требуется большая длительность.

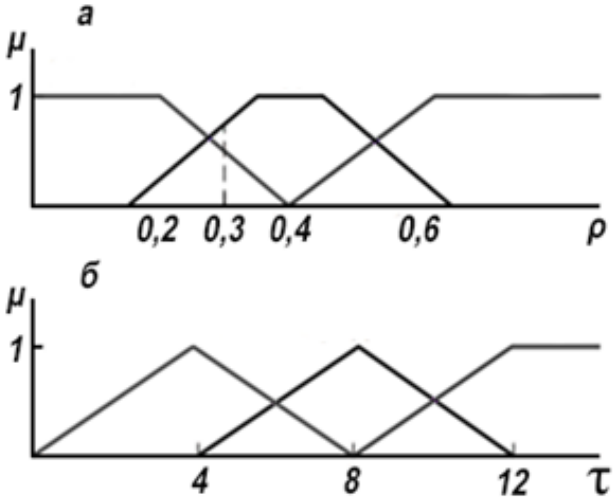


Рисунок 2.5 – Входные (а) и выходные (б) функции принадлежности

Как следует из рисунка 2.5, а, области определения входных функций принадлежности различных термов пересекаются, поэтому выходная нечеткая инструкция будет представляться некоторой комбинацией приведенных правил. При этом естественно предположить, что в эту комбинацию отдельные правила будут входить с коэффициентами, определяемыми степенями принадлежности точных исходных данных соответствующим термам.

Пусть зафиксированное в некоторый момент значение плотности равно 0,3. Процедура фаззификации показывает, что эта плотность со степенью 0,5 принадлежит терму «малая плотность» и со степенью 0,75 – терму «средняя плотность». Согласно принятому правилу комбинирования нечетких выходных инструкций, длительность зеленого сигнала должна быть на 0,5 короткой и на 0,75 средней.

Конкретные правила дефаззификации могут быть различными. Например, можно вычислять искомую длительность следующим образом:

$$\tau^* = \frac{0{,}5\tau_k^* + 0{,}75\tau_c^*}{0{,}5+0{,}75}, \qquad (2.17)$$

где τ_k^* и τ_c^* – длительности зеленого сигнала, при которых выходные функции принадлежности соответствующих термов («короткая» и «средняя») принимают максимальные значения.

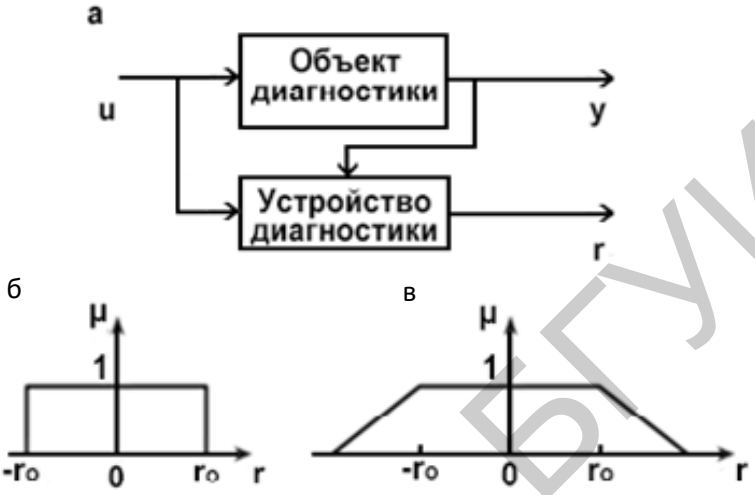
В данном случае

$$\tau^* = 6{,}4 \text{ с}. \qquad (2.18)$$

Теория нечетких множеств и нечеткого логического вывода имеет перспективу использования при разработке алгоритмов работы автоматизированных систем управления технологическими процессами (АСУ ТП), например, в системах контроля и управления состоянием физических объектов. В общей постановке задача может быть сформулирована следующим образом.

Имеется некоторая система (объект диагностирования (ОД)), выполняющая ответственную функцию (например, система управления ядерным или химическим реактором, робот и т. д.). Требуется разработать устройство диагно-

стирования, которое должно работать одновременно с ОД. На него поступают два типа сигналов: во-первых, те же управляющие сигналы $u(t)$, что и на ОД, во-вторых, сигналы $y(t)$ с датчиков, установленных на ОД, которые несут информацию о происходящих в нем процессах (рисунок 2.6, а). Устройство диагностирования непрерывно обрабатывает поступающую на него информацию и формирует заключение о состоянии ОД – появились в нем дефекты, существенно влияющие на качество его функционирования, или нет, и если появились, то какие.



а – схема реализации процесса диагностирования; б – четкий порог;
в – нечеткий порог

Рисунок 2.6 – Иллюстрация к принятию решений при диагностировании

Во многих случаях такое заключение принимается весьма просто: если $r = 0$, то дефектов нет. Ненулевое значение r говорит о появлении дефекта, причем по величине r можно с большей или меньшей точностью сказать, какой дефект возник. Однако нередко из-за действующих помех, неучтенных факторов и прочих причин сигнал r может принимать ненулевые значения даже при отсутствии дефектов. Самый простой выход из положения здесь – ввести порог r_0 и принимать решение очевидным образом: $|r| \le r_0$ – дефекта нет, $|r| > r_0$ – дефект появился (рисунок 2.6, б).

Приведенное простое бинарное правило имеет недостатки, один из которых состоит в том, что при небольших значениях r_0 будет велика вероятность ложной тревоги (дефекта нет, а порог превышен). При больших значениях r_0 существенно возрастет вероятность того, что возникший дефект останется незамеченным.

Выход здесь может быть найден за счет использования нечеткого порога (рисунок 2.6, в) и разработки «умной» АСУ либо включения человека-оператора в процесс принятия решения. В качестве информации для принятия решения оператору сообщается величина $1 - \mu$, характеризующая уровень тревоги. Она может быть представлена в форме звукового сигнала различной интенсивности, уровня яркости красного цвета на экране или иным способом, согласующимся с тем, как оператору сообщается другая информация об объекте.

2.2 Назначение пакета Fuzzy Logic Toolbox

В вычислительную среду MATLAB интегрированы десятки пакетов инженерных и математических программ, одним из них является пакет Fuzzy Logic Toolbox (пакет нечеткой логики).

Пакет Fuzzy Logic Toolbox – это совокупность прикладных программ, относящихся к теории размытых или нечетких множеств и позволяющих конструировать так называемые нечеткие экспертные и/или управляющие системы, а также поддерживать все стадии разработки нечетких систем: их синтез, исследование, проектирование, моделирование, внедрение в режим реального времени. Встроенные GUI-модули (Graphical user interface moduls) обеспечивают понятную среду с графическим интерфейсом.

Функции пакета реализуют большинство современных нечетких технологий, включая нечеткий логический вывод, нечеткую кластеризацию и адаптивную нейро-нечеткую настройку (ANFIS-adaptive neuro-fuzzy inference system). Fuzzy Logic Toolbox открыт для пользователя, можно просмотреть алгоритмы, просмотреть исходный код, добавить собственные функции принадлежности или процедуры дефаззификации.

Ключевыми особенностями этого пакета являются:

- специализированные GUI-модули для создания нечеткого вывода;
- реализация алгоритмов нечеткого вывода Сугено и Мамдани;
- библиотека функций принадлежности;
- настройка функций принадлежности ANFIS-алгоритмом;
- возможность внедрения систем нечеткого вывода в Simulink через модуль Fuzzy Logic Controller;
- С-код алгоритмов, позволяющий использовать спроектированные системы вне среды MATLAB.

Fuzzy Logic Toolbox содержит функции, которые вызываются из рабочей области программы (командная строка) и несколько графических модулей, позволяющих настраивать систему визуально в виде диалогов. Последнее удобнее, но возможности этих модулей ограниченные.

Fuzzy Logic Toolbox включает следующие GUI-модули:

1. Fuzzy Inference System Editor – редактор общих свойств системы нечеткого вывода. Позволяет установить количество входов и выходов системы, выбрать тип системы (Мамдани или Сугено), метод дефаззификации или вызвать другие GUI-модули.

2. Membership Function Editor – редактор функций принадлежности. Выводит на экран графики функций принадлежности термов входных и выходных переменных. Позволяет выбрать количество этих термов, задать их тип и параметры для каждого терма.

3. Rule Editor – редактор нечеткой базы знаний. Позволяет задать и редактировать правила нечеткой базы знаний.

4. Rule Viewer – браузер нечеткого вывода. Визуально показывает выполнение нечеткого вывода по каждому правилу в виде графиков.

5. Surface Viewer – браузер поверхности «входы – выход». Выводит поверхность зависимости выходной переменной от любых двух входных переменных.

6. ANFIS EDITOR – редактор нейро-нечеткой сети. Позволяет синтезировать и настраивать нейро-нечеткие сети по выборке данных вход – выход (обучающей выборке). Для настройки используется метод обратного распространения ошибки или его комбинация с методом наименьших квадратов.

7. Find cluster – инструмент субтрактивной кластеризации (Clustering), позволяющий найти центры кластеров данных, которые используются для экстракции нечетких правил.

Все GUI-модули, кроме Find cluster, динамически обмениваются данными друг с другом и могут быть вызваны один из другого.

Пакет позволяет работать:

- в режиме графического интерфейса;
- в режиме командной строки;
- с использованием блоков и примеров пакета Simulink.

2.3 Проектирование систем нечеткого вывода в пакете Fuzzy Logic Toolbox

Порядок выполнения тестового задания:

I. Подготовить нечеткие правила в соответствии с графическим представлением функций задания и вариантов**.**

II. Трехмерное изображение выбранной зависимости построить, воспользовавшись программными средствами MATLAB.

При этом можно руководствоваться программой нижеприведенного примера, которая моделирует зависимость $y=x_1^2 \cdot \sin(x_2-1)$.

```
% Построение графика функции y=x1^2*sin(x2-1)
% в области x1 = [-6, 4], x2=[-4.4, 1.7].
n=15; % количество точек дискретизации
x1=linspace(-6, 4, n); x2=linspace(-4.4, 1.7, n);
y=zeros(n, n);for j=1:n
y(j,:)=x1^2*sin(x2(j)-1); end
surf(x1, x2, y)
xlabel('x_1'); ylabel('x_2'); zlabel('y');
title('Искомая зависимость')
```

Сформируем нечеткие правила, соответствующие зависимости выходной переменной от входных переменных, по полученной поверхности (рисунок 2.7).

Можно воспользоваться и возможностями пакета *Symbolic Math Toolbox,* введя, например, следующие команды:

symsxy % создаем символьные переменные
z=*x*^2*sin(*y*-1);
ezsurf(*z*,[-2,2])

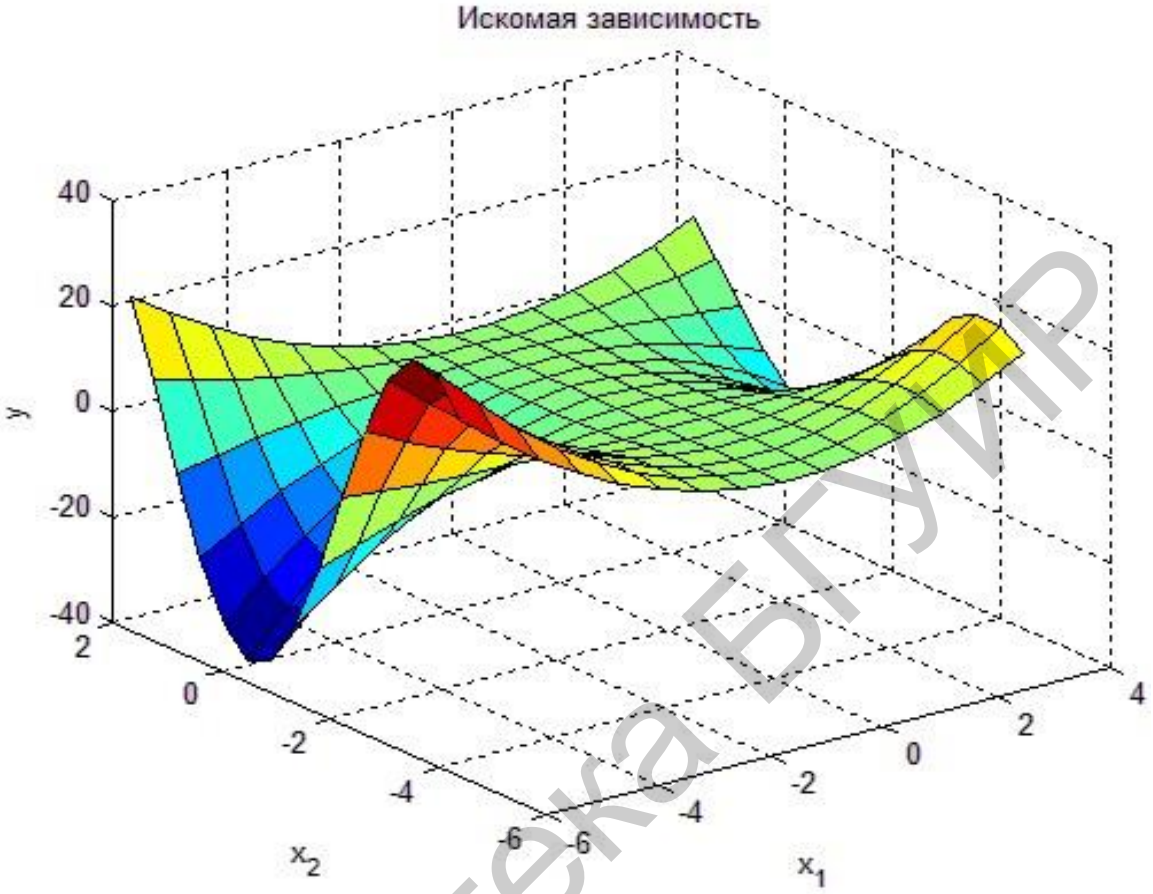


Рисунок 2.7 – Графическое представление функции $y=x_1^2\sin(x_2-1)$

Для лингвистической оценки входных переменных x_1 и x_2 выберем по три терма – *низкий, средний, высокий*; для выходной *y* пять термов – *низкий, ниже среднего, средний, выше среднего, высокий*. Получим, например, следующие правила:

1) ЕСЛИ x_1 = *низкий* И x_2 = *низкий*, ТО *y* = *высокий*;
2) ЕСЛИ x_1 = *низкий* И x_2 = *средний*, ТО *y* = *низкий*;
3) ЕСЛИ x_1 = *низкий* И x_2 = *высокий*, ТО *y* = *высокий*;
4) ЕСЛИ x_1= *средний,* ТО *y* = *средний*;
5) ЕСЛИ x_1 = *высокий* Иx_2 = *низкий*, ТО *y* = *выше среднего*;
6) ЕСЛИ x_1 = *высокий* И x_2 = *средний*, ТО *y* = *ниже среднего*;
7) ЕСЛИ x_1 = *высокий* И x_2 = *высокий*, ТО *y* = *выше среднего*.

III. Спроектировать нечеткую систему, выполнив следующую последовательность шагов:

1. Откроем *FIS*-редактор, напечатав слово *fuzzy* в командной строке.

2. В появившемся графическом окне *FIS Editor* добавим вторую входную переменную. Для этого в меню *Edit* выберем команду *Add input*.

3. Переименуем первую входную переменную. Для этого сделаем щелчок левой кнопкой мыши на блоке *Input 1*, введем новое обозначение x_1 в поле редактирования имени текущей переменной и нажмем <*Enter*>.

4. Переименуем вторую входную переменную, введя x_2 на блоке *Input 2*.

5. Переименуем выходную переменную. Для этого щелкнем мышью на блоке *Output 1*. Введем новое обозначение *y* в поле редактирования имени текущей переменной; нажмем <*Enter*>.

6. Зададим имя системы. Для этого в меню *File* выберем в подменю *Export* команду *To File* и введем имя файла, например *Lab_2*.

7. Перейдем в редактор функций принадлежности. Для этого сделаем двойной щелчок левой кнопкой мыши на блоке x_1 и зададим диапазон изменения переменной x_1, вводя -6 4 в поле *Range* (рисунок 2.8).

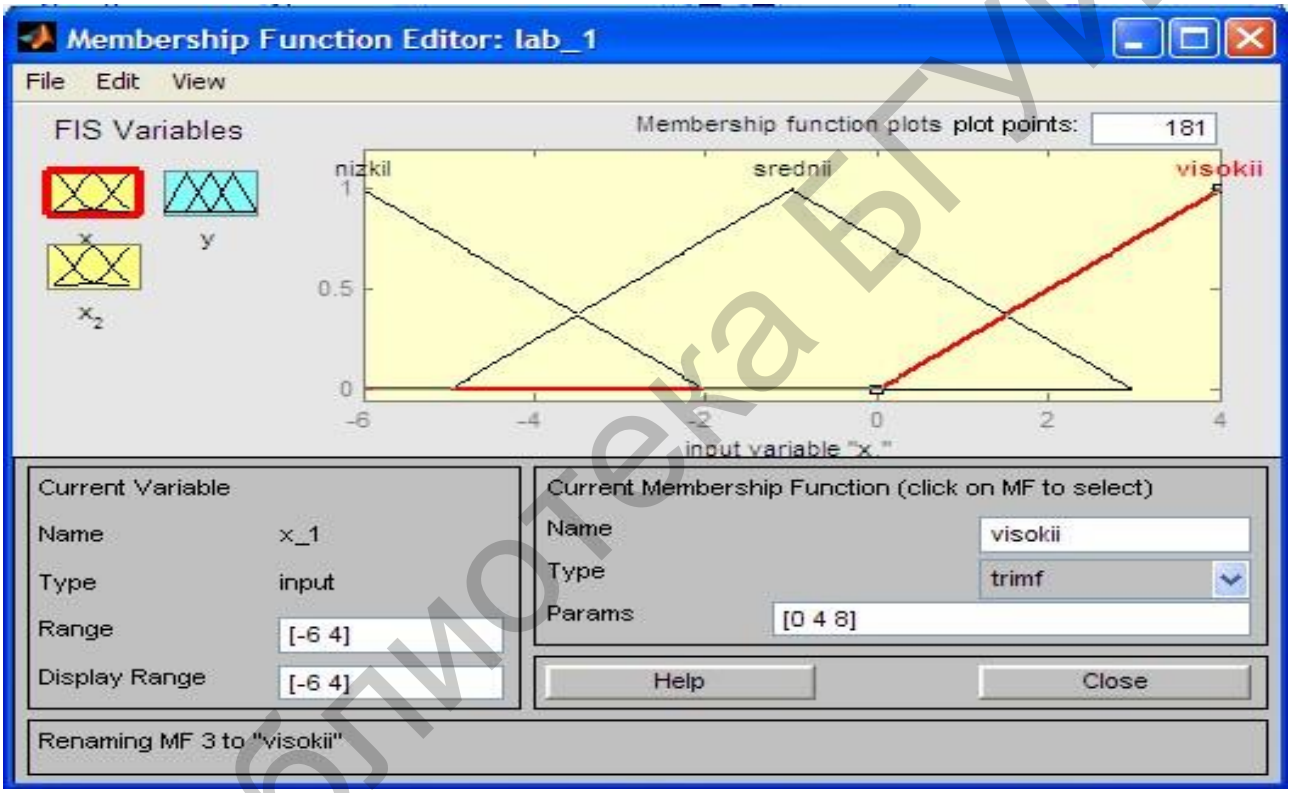


Рисунок 2.8 – Функции принадлежности переменной x_1

8. Зададим функции принадлежности переменной x_1. Для лингвистической оценки этой переменной будем использовать три терма с треугольными функциями принадлежности. Эти функции установлены по умолчанию, поэтому перейдем к следующему шагу.

Зададим наименования термов переменной x_1. Для этого щелкнем мышью на графике первой функции принадлежности (см. рисунок 2.8). График активной функции принадлежности выделится красной жирной линией. Затем введем наименование терма *Низкий* в поле *Name* и нажмем <*Enter*>. Щелкнем мышью на графике второй функции принадлежности, введем наименование терма *Средний* в поле *Name* и нажмем <*Enter*>. Щелкнем мышью на графике третьей

функции принадлежности, введем наименование терма *Высокий* в поле *Name* и нажмем <*Enter*>.

9. Зададим функции принадлежности переменной x_2. Для этого активизируем переменную x_2 щелчком мыши на блоке. Зададим диапазон изменения переменной x_2. Для этого напечатаем -4.4 1.7 в поле *Range* и нажмем <*Enter*>. Для лингвистической оценки этой переменной будем использовать, как и ранее, три терма с треугольными функциями принадлежности. Они установлены по умолчанию, поэтому перейдем к следующему шагу.

10. По аналогии с шагом 9 зададим следующие наименования термов переменной x_2: *Низкий, Средний, Высокий*.

11. Зададим функции принадлежности переменной *y*. Для лингвистической оценки этой переменной будем использовать пять термов с гауссовыми функциями принадлежности. Для этого щелчком мыши на блоке *y* активизируем переменную *y*. Зададим диапазон изменения переменной *y*. Для этого введем -50 50 в поле *Range* (рисунок 2.9) и нажмем <*Enter*>.

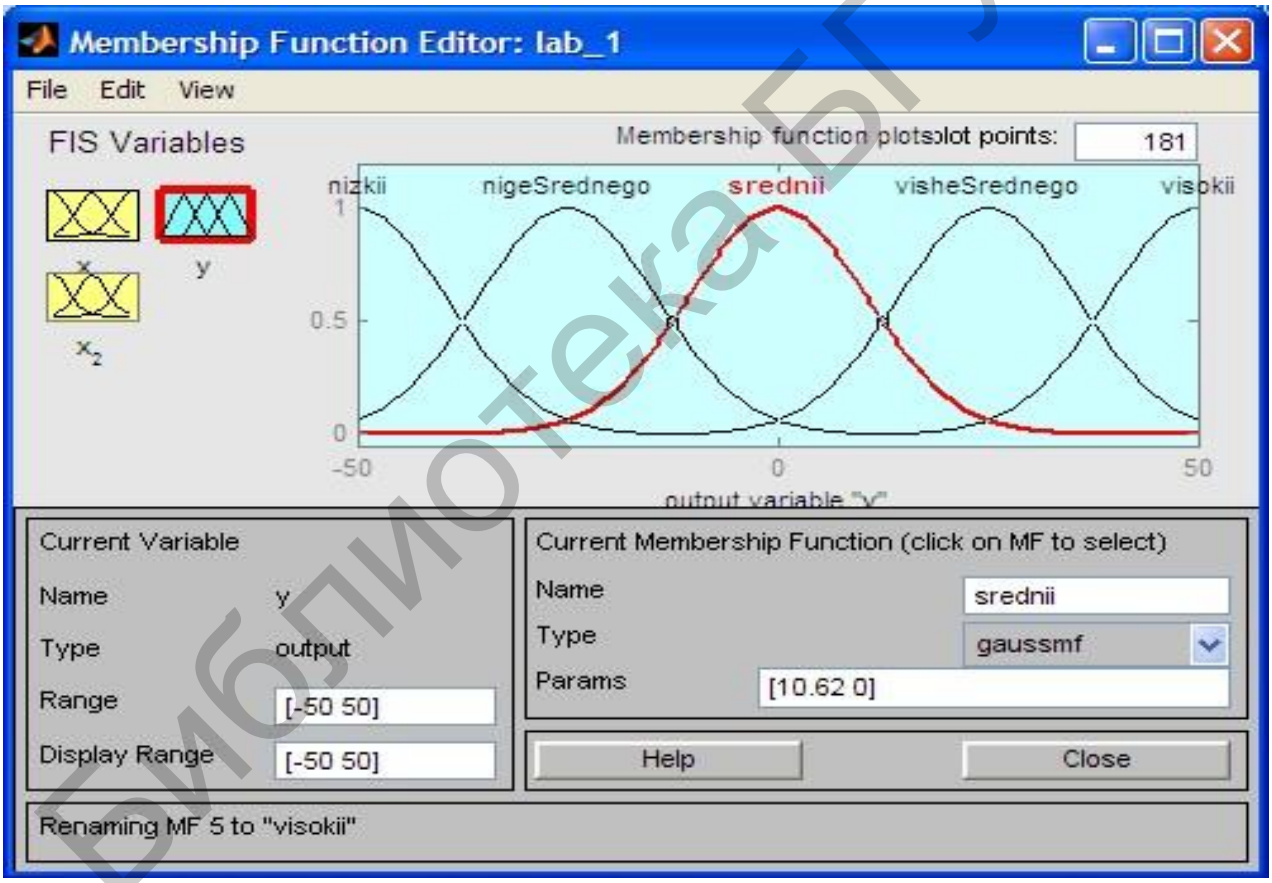


Рисунок 2.9 – Функции принадлежности переменной *y*

Затем в меню *Edit* выберем команду *Remove All MFs* для удаления установленных по умолчанию функций принадлежности. После этого в меню *Edit* выберем команду *Add MF*. В появившемся диалоговом окне выберем тип функции принадлежности *gaussmf* в поле *MF type* и пять термов в поле *Number MFs*. После ввода функций принадлежности редактор активизирует первую входную

переменную, поэтому для продолжения работы щелкнем мышью на пиктограмме *y*.

12. По аналогии с шагом 9 зададим следующие наименования термов переменной *y*: *Низкий, Ниже среднего, Средний, Выше среднего, Высокий*. В результате будет сформировано графическое окно, изображенное на рисунке 2.9.

13. Перейдем в редактор базы знаний *Rule Editor*. Для этого в меню *Edit* выберем команду *Rules*. Для ввода правила выберем в меню соответствующую комбинацию термов и нажмем кнопку *Add rule*.

14. На рисунке 2.10 изображено окно редактора базы знаний после ввода всех семи правил. В конце правил в скобках указаны весовые коэффициенты.

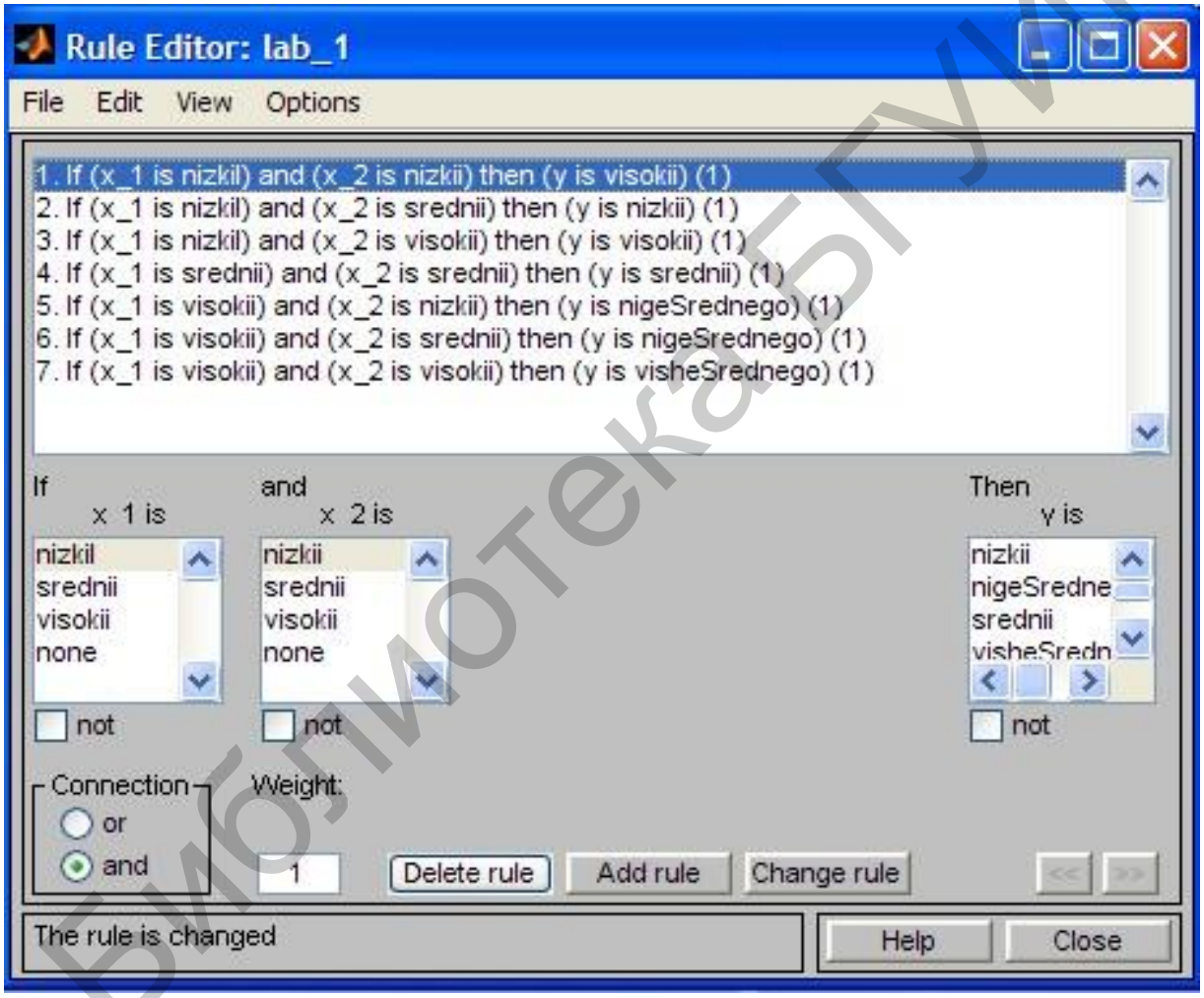


Рисунок 2.10 – Нечеткая база знаний *Mamdani*

15. Окно визуализации нечеткого вывода приведено на рисунке 2.11. Окно активизируется командой *Rules* меню *View*. В поле *Input* указываются значения входных переменных, для которых выполняется нечеткий логический вывод.

16. На рисунке 2.12 приведена поверхность «входы – выход», соответствующая синтезированной нечеткой системе. Окно выводится по команде *Surface* меню *View*.

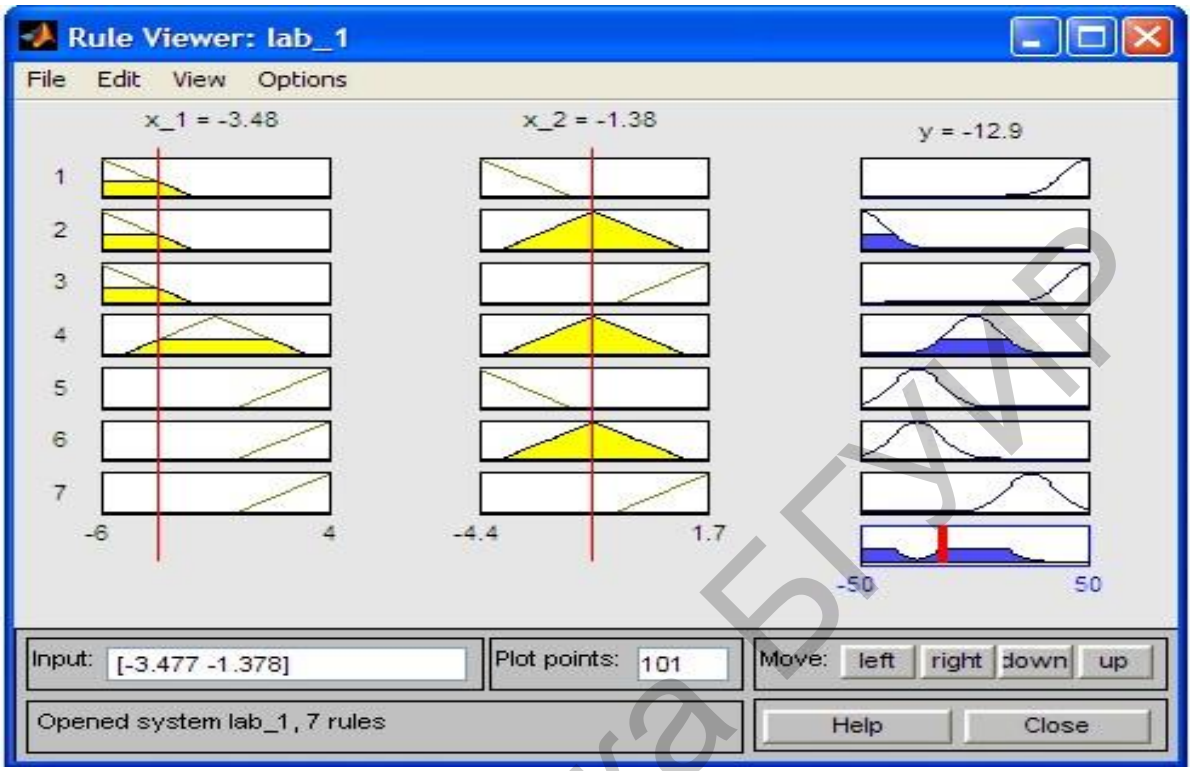


Рисунок 2.11 – Визуализация нечеткого вывода *Mamdani*

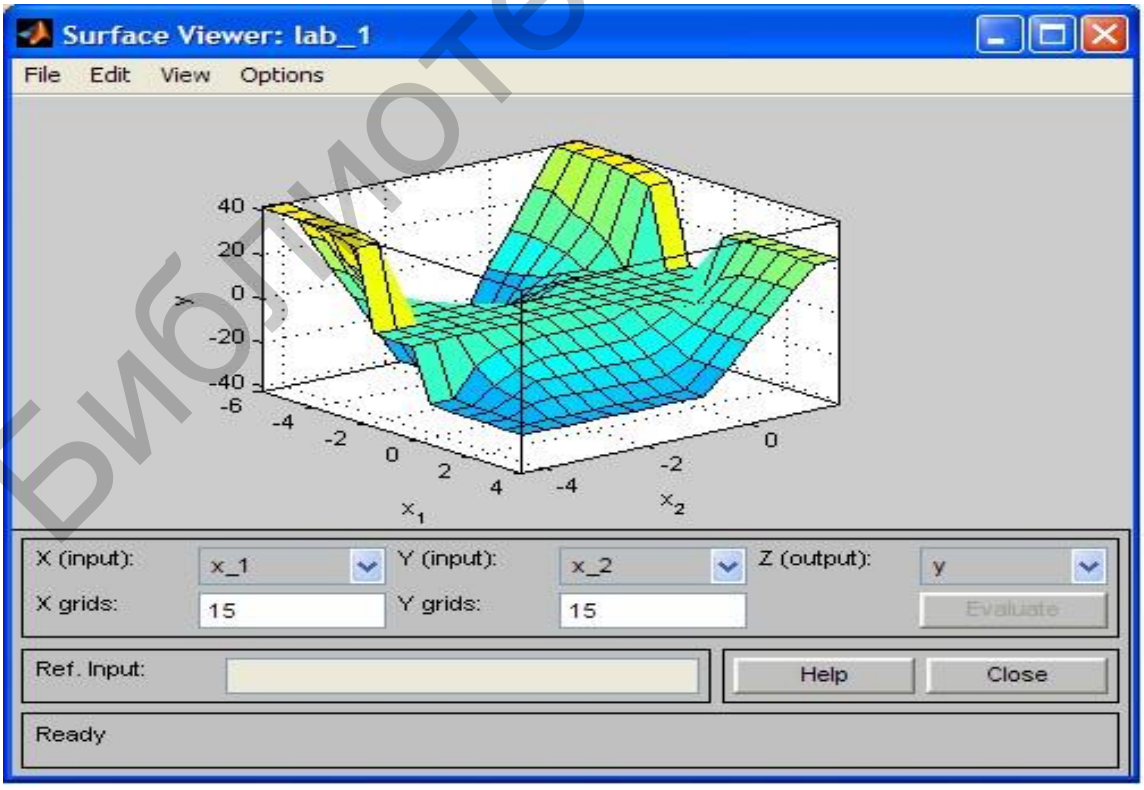


Рисунок 2.12 – Поверхность «входы – выход» для базы знаний в *SurfaceViewer*

17. Для сохранения созданной системы необходимо в меню *File* выбрать в подменю *Export* команду *To File.*

При сравнении поверхностей на рисунках 2.7 и 2.12 можно сделать выводы относительно качества описания нечеткими правилами моделируемой нелинейной зависимости.

Лабораторное задание

Спроектировать систему нечеткого вывода согласно вариантам задания, определяемым таблицей 2.1

Таблица 2.1 – Варианты заданий

Номер варианта	Вид зависимости	Диапазон значений [*x*, *y*]
1	*Z* = *x*^2-*y*^2	[-1, 1]
2	*Z* = *x*^3+*y*^2	[-1, 1]
3	*Z* = exp(-*x*^2-*y*^2)	[-1, 1]
4	*Z* = exp(-*x*^2+*y*^2)	[-2, 2]
5	*Z* = *x***y**sin(*x*^2+*y*^2)	[-2, 2]
6	*Z* = *x*^2*sin(*y*-1)	[-2, 2]
7	*Z* = *y*^2*sin(*x*)	[-2, 2]
8	*Z* = *y*^2*cos(*x*)	[-2, 2]
9	Z = *y*^2*cos(*x*)^2	[-1, 1]
10	*Z* = 4*cos(*x*)/*y*	[0.5, 3.14]
11	*Z* = (*x*-*y*)/(*x*+*y*)	[1, 10]
12	*Z* = exp(-*x*^2)+exp(-*y*^2)	[-1, 1]
13	*Z* = *x*^2+*y*^2	[-1, 10]
14	*Z* = 5**x*^2*cos(*y*)	[-1, 1]
15	*Z* = - *x***y*+*y*^2	[0, 5]
16	Z = 5**x*^2*sin(*y*)	[-1, 1]
17	*Z* = *y*^2*sin(*x*)^2	[-1, 1]
18	*Z* = 2·*x*^2-(*y*-1)^2	[-1, 1]
19	*Z* = *y*^2*cos(*x*)	[-2, 2]
20	*Z* = *x*^3+*y*^2	[-1,1]

Содержание отчета

1. Исходная функция варианта задания и ее графическое представление.
2. Лингвистические правила решений.
3. Описание последовательности действий при проектировании нечеткой системы и анализ результатов нечеткого вывода.
4. Оценка качества описания нечеткими правилами моделируемой нелинейной зависимости.

2.4 Построение нечеткой аппроксимирующей системы в пакете Fuzzy Logic Toolbox

Порядок выполнения тестового задания:

1. Командой (функцией) *Fuzzy* из режима командной строки запустим основную интерфейсную программу пакета *Fuzzy Logic* – редактор нечеткой системы вывода (*Fuzzy Inferen System Editor, FIS Editor, FIS*-редактор).

Вид открывающегося при этом окна приведен на рисунке 2.13.

Главное меню редактора содержит позиции:

- *File* – работа с файлами моделей (их создание, сохранение, считывание и печать);
- *Edit* – операции редактирования (добавление и исключение входных и выходных переменных);
- *View* – переход к дополнительному инструментарию.

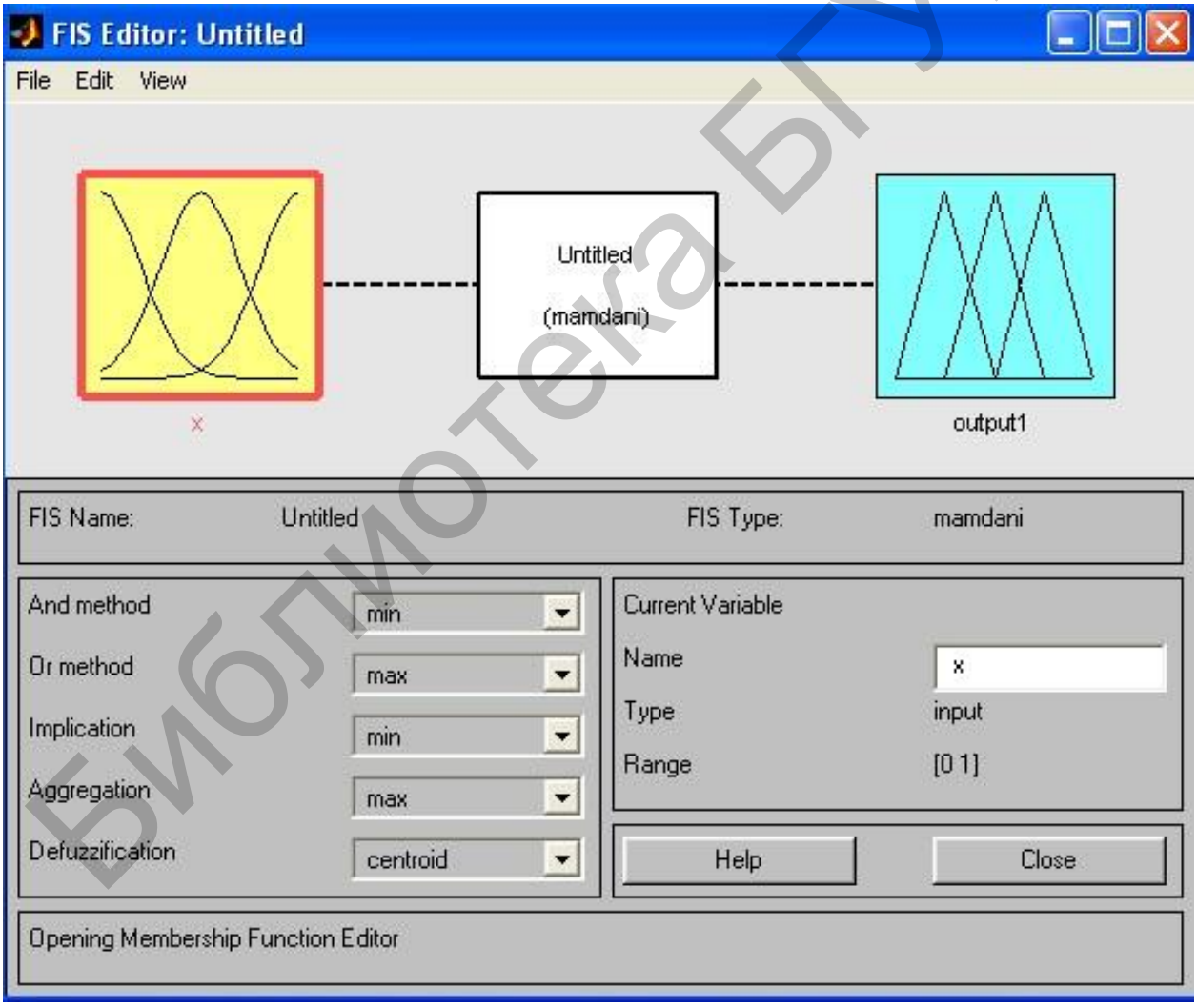


Рисунок 2.13 – Вид окна *FIS Editor*

2. Спроектируем нечеткую систему, отображающую зависимость $y = x^2$ между переменными x и y, заданную с помощью таблицы 2.2.

Таблица 2.2 – Значения *x* и *y*

x	-1	0.6	0	0.4	1
y	1	0.36	0	0.16	1

3. Для выполнения задания необходимо следовать последовательности шагов, представленных ниже.

Шаг 1. В позиции меню *File* выбираем опцию *New Sugeno FIS* (новая система типа *Sugeno*), при этом в блоке, отображаемом белым квадратом, в верхней части окна редактора появится надпись *Untitled2* (*Sugeno*).

Шаг 2. Щелкнем левой кнопкой мыши на блоке, озаглавленном inputl (вход 1). Затем в правой части редактора в поле, озаглавленном *Name* (Имя), вместо inputl вводится обозначение используемого аргумента, т. е. *x*. Обратим внимание, что если теперь сделать где-нибудь (вне блоков редактора) однократный щелчок мыши, то имя отмеченного блока изменится на *x*; то же достигается нажатием после ввода клавиши Enter.

Шаг 3. Дважды щелкнем на этом блоке. Перед нами откроется окно редактора функций принадлежности – *Membership Function Editor* (рисунок 2.14).

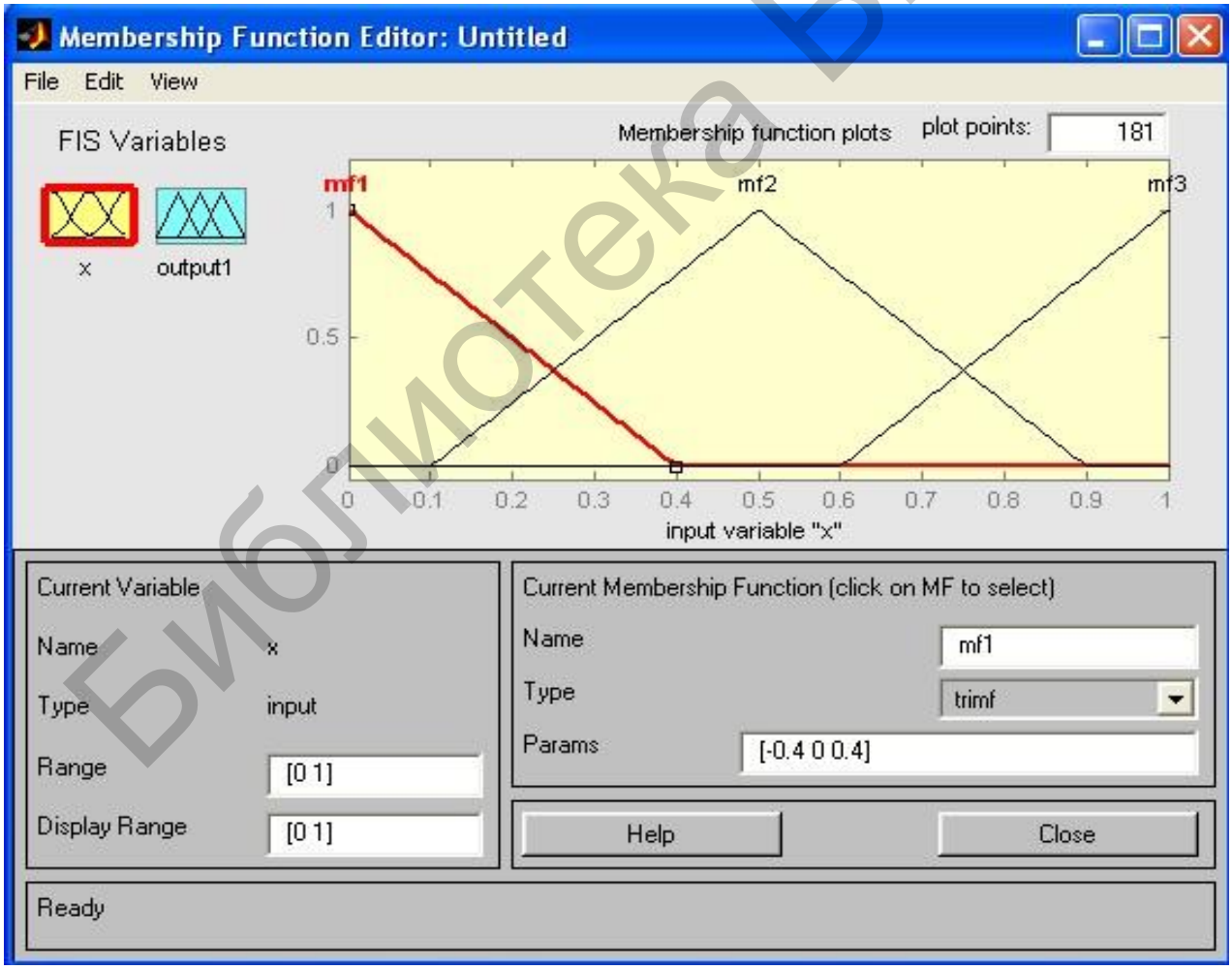


Рисунок 2.14 – Окно редактора функций принадлежности

Войдем в позицию меню Edit данного редактора и выберем в нем опцию Add MFs (Add Membership Functions – добавить функции принадлежности).

При этом появится диалоговое окно (рисунок 2.15), позволяющее задать тип (*MF type*) и количество (*Number of MFs*) функций принадлежности (в данном случае все относится к входному сигналу, т. е. к переменной *x*). Выберем гауссовы функции принадлежности (*gaussmf*), а их количество зададим равным пяти – по числу значений аргумента в таблице 2.1. Подтвердим ввод информации нажатием кнопки OK, после чего произойдет возврат к окну редактора функций принадлежности.

Рисунок 2.15 – Диалоговое окно задания
типа и количества функций принадлежности

В поле *Range* (Диапазон) установим диапазон изменения *x* от -1 до +1, т. е. диапазон, соответствующий таблице 2.1. Затем щелкнем левой кнопкой мыши в любой точке поля редактора (или нажмем клавишу ввода Enter). Обратим внимание, что после этого произойдет соответствующее изменение диапазона в поле *Display Range* (Диапазон дисплея).

4. Обратимся к графикам используемых функций принадлежности, изображенным в верхней части окна редактора функций принадлежности. Заметим, что для успешного решения поставленной задачи необходимо, чтобы ординаты максимумов этих функций совпадали с заданными значениями аргумента *x*.

Для левой, центральной и правой функций такое условие выполнено, но две другие необходимо «подвинуть» вдоль оси абсцисс. «Передвижка» делается весьма просто: подводим курсор к нужной кривой и щелкаем левой кнопкой мыши. Кривая выбирается, окрашиваясь в красный цвет, после чего с помощью указателя мыши ее можно подвинуть в нужную сторону (более точную установку можно провести, изменяя числовые значения в поле *Params* (Параметры) – в данном случае каждой функции принадлежности соответствуют два параметра, при этом первый определяет размах кривой, а второй – положение ее центра). Для выбранной кривой, кроме этого, в поле Name можно изменять имя (завершая ввод каждого имени нажатием клавиши Enter).

5. Проделаем требуемые перемещения кривых и зададим всем пяти кривым новые имена, например: самой левой – ***bn***, следующей – ***n***, центральной – ***z***, следующей за ней справа – ***p***, самой правой – ***bp***.

Нажмем кнопку *Close* и выйдем из редактора функций принадлежности, возвратившись при этом в окно редактора нечеткой системы (*FIS Editor*).

6. Сделаем однократный щелчок левой кнопкой мыши на голубом квадрате (блоке), озаглавленном *output*l. В окошке *Name* заменим имя *output*l на *y*.

7. Дважды щелкнем на отмеченном блоке и перейдем к программе – редактору функций принадлежности. В позиции меню *Edit* выберем опцию *Add MFs*. Появляющееся диалоговое окно (см. рисунок 2.15) позволяет задать теперь в качестве функций принадлежности только линейные (*linear*) или постоянные (*constant*) – в зависимости от того, какой алгоритм *Sugeno* (1-го или 0-го порядка) выбирается. Если в компьютере установлена версия, в которой нет данных функций принадлежностей, то можно оставить по умолчанию *trimf*. Это, конечно, повлияет на результат, поэтому можно поэкспериментировать, изменяя тип функций принадлежности.

8. Обратим внимание, что здесь диапазон изменения (*Range*), устанавливаемый по умолчанию [0, 1], менять не нужно. Изменим лишь имена функций принадлежности (их графики при использовании алгоритма *Sugeno* для выходных переменных не приводятся), например, задав их как соответствующие числовые значения y, т. е. 0, 0.16, 0.36, 1; одновременно эти же числовые значения вводятся в поле *Params* (рисунок 2.16). Затем закроем окно нажатием кнопки *Close* и вернемся в окно *FIS*-редактора.

9. Дважды щелкнем левой кнопкой мыши на среднем (белом) блоке, при этом раскроется окно еще одной программы – редактора правил (*Rule Editor*). Необходимо ввести соответствующие правила. При вводе каждого правила необходимо обозначить соответствие между каждой функцией принадлежности аргумента *x* и числовым значением *y.* Кривая, обозначенная как *bn*, соответствует $x = -1$, т. е. $y = 1$. Выберем в левом поле – с заголовком *x is bn*, а в правом – 1 и нажмем кнопку *Add rule* (Добавить правило). Введенное правило появится в окне правил и будет представлять собой запись

If (*x* is *bn*) *then* (*y* is 1)

Аналогично поступим для всех других значений *x*, в результате чего сформируется набор из пяти правил (рисунок 2.17). Закроем окно редактора правил и возвратимся в окно *FIS*-редактора. Построение системы закончено и можно начать эксперименты по ее исследованию. Заметим, что большинство опций выбиралось по умолчанию.

В рассматриваемой задаче необходимо выбрать постоянные функции принадлежности с общим числом 4 (по числу различных значений *y* в таблице 2.2). Подтвердим введенные данные нажатием кнопки ОК, после чего произойдет возврат в окно редактора функций принадлежности.

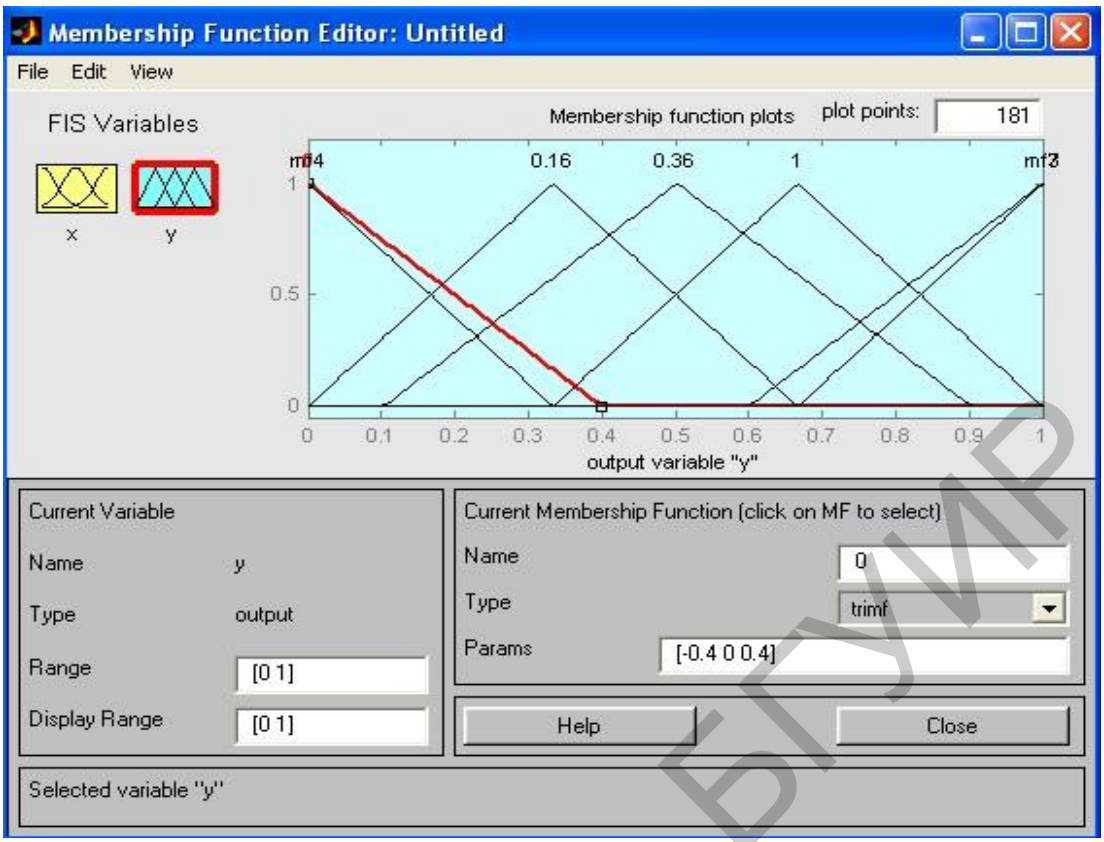


Рисунок 2.16 – Параметры функций принадлежности переменной *y*

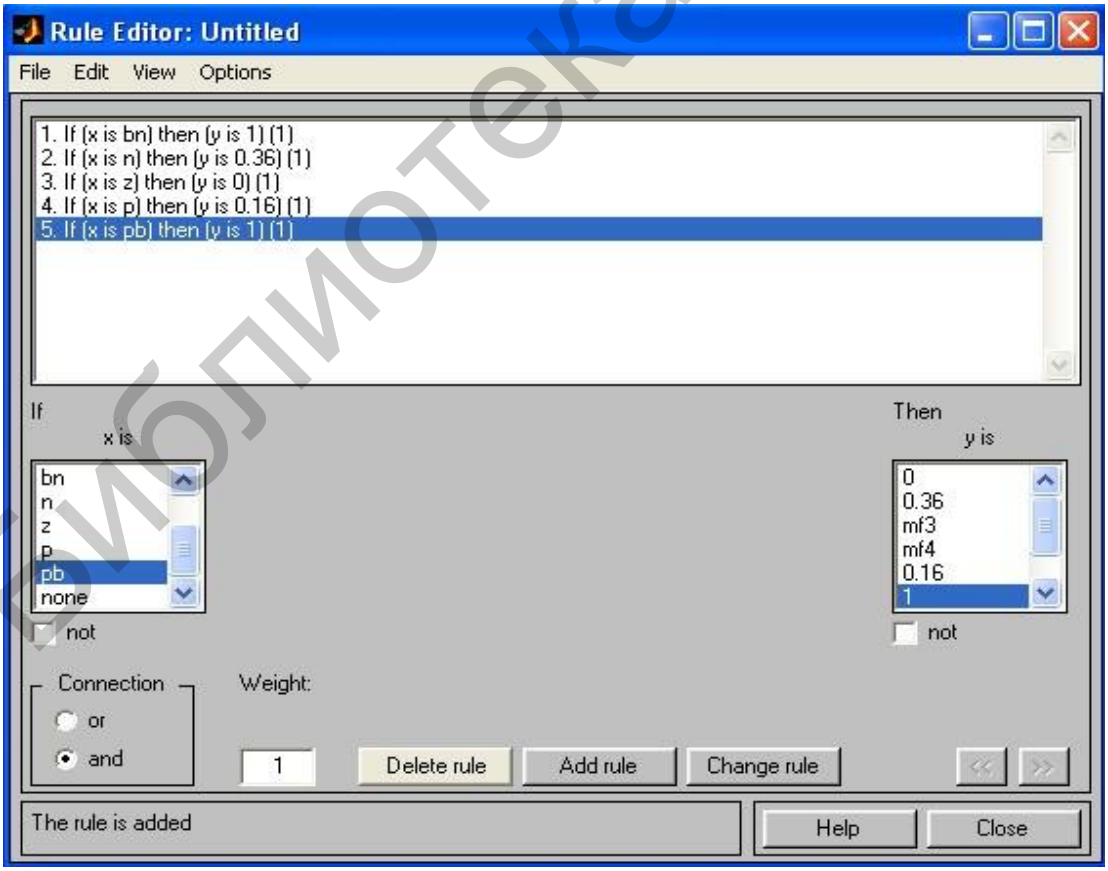


Рисунок 2.17 – Окно редактора правил

10. Предварительно сохраним на диске (используя пункты меню *File/Save to disk as...*) созданную систему под каким-либо именем, например *Proba*.

11. Выберем позицию меню *View*. Как видно из выпадающего при этом подменю, с помощью пунктов *Edit membership functions* и *Edit rules* можно совершить переход к двум вышерассмотренным программам – редакторам функций принадлежности и правил (то же можно сделать и нажатием клавиш *Ctrl*+2 или *Ctrl*+3). Но сейчас интерес представляют два других пункта – *View rules* (Просмотр правил) и *View surface* (Просмотр поверхности). Выберем пункт *View rules*, при этом откроется окно (рисунок 2.18) еще одной программы просмотра правил (*Rule Viewer*).

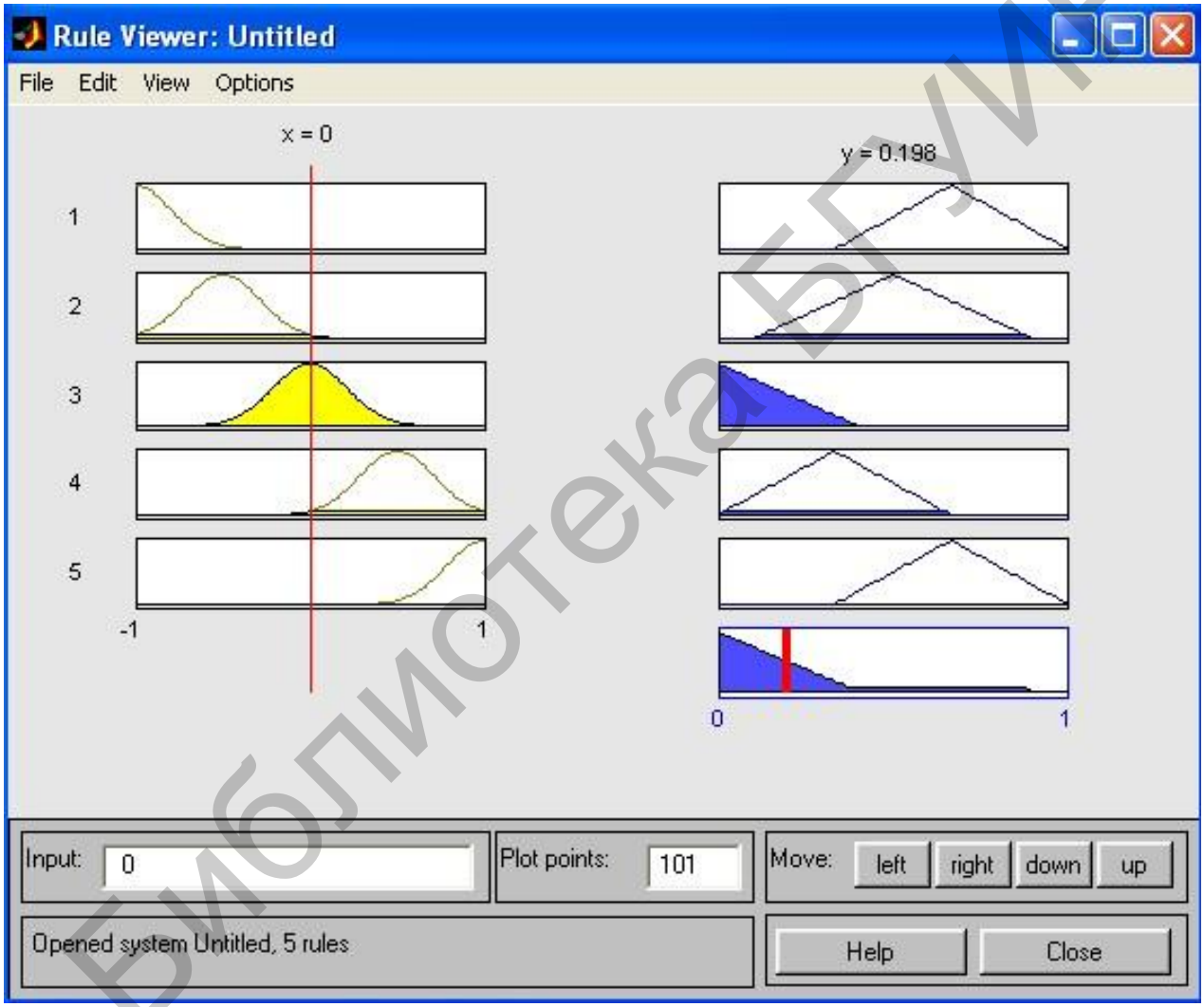


Рисунок 2.18 – Окно просмотра правил

12. В правой части окна в графической форме представлены функции принадлежности аргумента *x*, в левой – переменной выхода *y* с пояснением механизма принятия решения. Красная вертикальная черта, пересекающая графики в правой части окна, которую можно перемещать с помощью курсора, позволяет изменять значения переменной входа (это же можно делать, задавая числовые значения в поле *Input* (Вход)), при этом соответственно изменяются

значения *y* в правой верхней части окна. Зададим, например, *x* – 0.5 в поле *Input* и нажмем затем клавишу ввода (*Enter*). Значение *y* сразу изменится и станет равным 0.202. Таким образом, с помощью построенной модели и окна просмотра правил можно решать задачу интерполяции, т. е. задачу, решение которой и требовалось найти. Изменение аргумента путем перемещения красной вертикальной линии очень наглядно демонстрирует, как система определяет значения выхода.

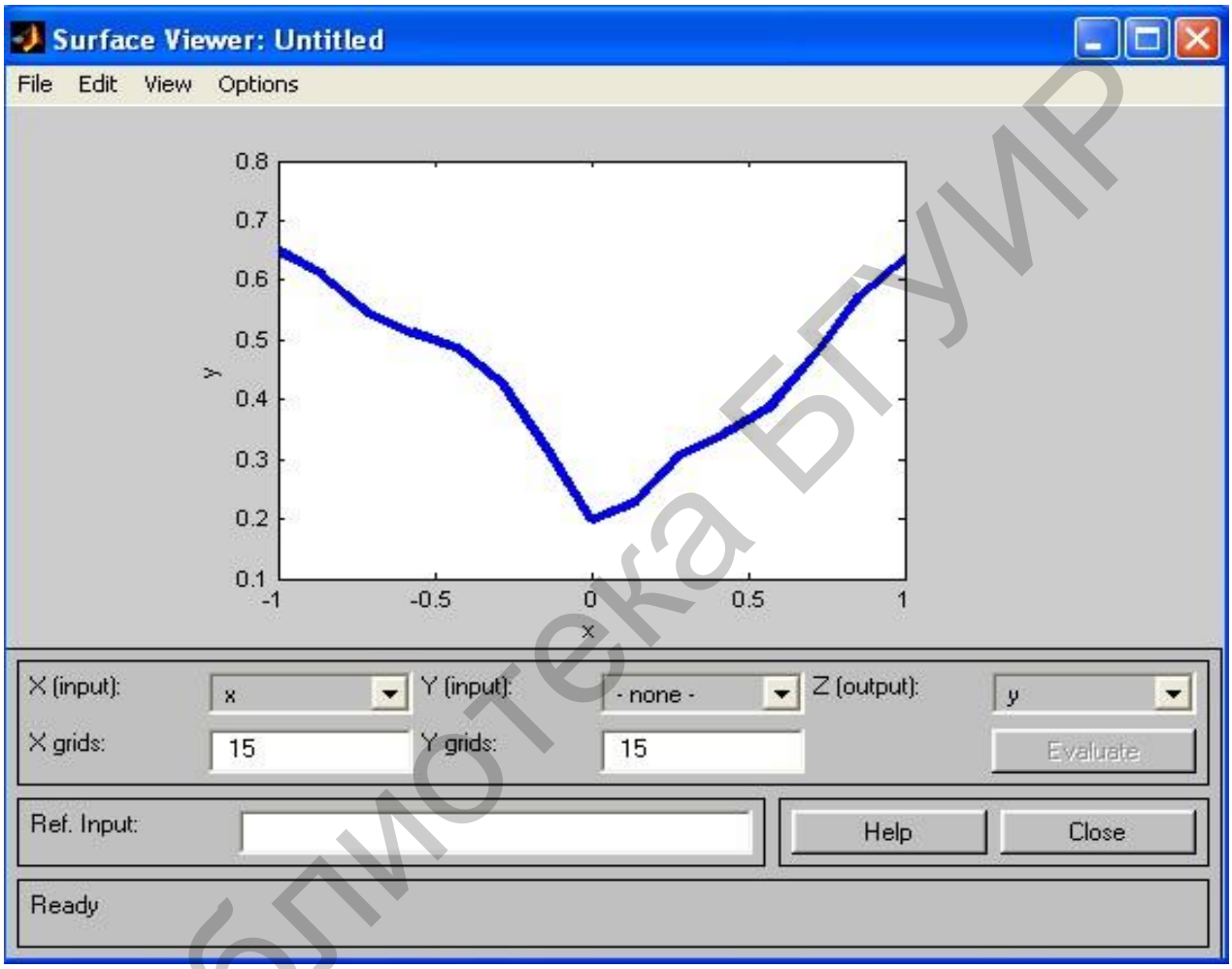


Рисунок 2.19 – Окно просмотра поверхности отклика

13. Закроем окно просмотра правил и, выбрав пункт меню *View/View surface*, перейдем к окну просмотра поверхности отклика (выхода), в рассматриваемом случае – к просмотру кривой *y*(*x*) (см. рисунок 2.19). Видно, что смоделированное системой по таблице данных (см. таблицу 2.2) отображение не очень напоминает функцию x^2. Это объясняется тем, что число экспериментальных точек невелико и параметры функций принадлежности (для *x*) выбраны, вероятно, неоптимальным образом.

Следует отметить, что с помощью вышеуказанных программ-редакторов на любом этапе проектирования нечеткой модели в нее можно внести необхо-

димые коррективы вплоть до задания какой-либо особенной пользовательской функции принадлежности.

Из опций, устанавливаемых в FIS-редакторе по умолчанию при использовании алгоритма Sugeno, можно отметить:

- логический вывод организуется с помощью операции умножения (*prod*);
- композиция – с помощью операции логической суммы (вероятностного ИЛИ, *probor*);
- приведение к четкости – дискретным вариантом центроидного метода (взвешенным средним, *wtaver*).

Используя соответствующие поля в левой нижней части окна FIS-редактора, данные опции можно при необходимости изменить.

Лабораторное задание

Сконструировать нечеткую систему, отображающую зависимость между переменными *x* и *y*, заданную таблицей 2.3. По результатам работы определить тип кривой.

Таблица 2.3 – Исходные данные контрольного задания

Варианты	Значение аргумента и функции					
1	2					
1	*x*	-1	-0.5	0	0.2	1
	y	1	0.25	0	0.4	1
2	*x*	-1	-0.6	0.2	0.4	1
	y	-1	-1.67	5	2.5	1
3	*x*	-1	-0.5	0	0.3	1
	y	-1	-0.13	0	0.27	1
4	*x*	-1	-0.6	0	0.3	1
	y	0	0.8	1	0.95	0
5	*x*	-1	-0.5	0	0.2	1
	y	1	-0.125	0	0.008	1
6	*x*	-1	-0.6	0.2	0.4	1
	y	0	-0.64	-0.96	-0.84	0
7	*x*	-1	-0.5	0	0.3	1
	y	-3	-2	-1	-0.4	1
8	*x*	-1	-0.6	0	0.3	1
	y	0.5	0.09	0	0.0225	0.5
9	*x*	-1	-0.5	0	0.2	1
	y	0.5	0.03125	0	0.0008	0.5
10	*x*	-1	-0.6	0.2	0.4	1
	y	-1	2.78	25	6.25	1
11	*x*	-1	-0.5	0	0.3	1
	y	-1	1.8	3	3.6	5

1	2					
12	*x*	-1	-0.6	0	0.3	1
	y	-1	-0.216	0	0.027	1
13	*x*	-1	-0.5	0	0.2	1
	y	-2	-1.5	-1	-0.8	0
14	*x*	-1	-0.6	0.2	0.4	1
	y	-2	-1.2	0.4	2.5	1
15	*x*	-1	-0.5	0	0.3	1
	y	0	0.25	0.5	0.65	1
16	*x*	-1	-0.5	0	0.3	1
	y	-1	-1.75	-1	-0.31	0
17	*x*	-1	-0.5	0	0.2	1
	y	0	0.25	1	1.44	4
18	*x*	-1	-0.6	0.2	0.4	1
	y	-0.67	-0.2	0.07	0.13	0.67
19	*x*	-1	-0.5	0	0.3	1
	y	0.67	-0.34	0.5	0.18	1
20	*x*	-1	-0.6	0	0.3	1
	y	-2	-1.6	0	0.7	0

Содержание отчета

1. Краткое описание действий по всем пунктам программы.
2. Графики и выводы.

Контрольные вопросы

1. Каковы состав и назначение основных программ графического интерфейса Fuzzy Logic Toolbox?
2. В чем отличие функций принадлежностей четкой и нечеткой логике. Назовите области применения.
3. Поясните процедуры фаззификации и дефаззификации. Приведите примеры.
4. Поясните определение лингвистической переменной и алгоритм нечеткого вывода.
5. Поясните структуру контроллера нечеткой логики.
6. Поясните топологию сенсорных сетей и методы кластеризации.

Литература

1. Fuzzy Logic Toolbox – Проектирование систем управления [Электронный ресурс]. – Режим доступа : http://matlab.exponenta.ru/fuzzylogic.
2. Штовба, С. Д. Проектирование нечетких систем средствами MATLAB / С. Д. Штовба. – М. : Горячая линия – Телеком, 2007.

Св. план 2018, поз. 48

Учебное издание

Хоменок Михаил Юлианович

ИМИТАЦИОННОЕ МОДЕЛИРОВАНИЕ В ПРОГРАММНОЙ СРЕДЕ NETWORK SIMULATOR-2 И НЕЧЕТКОЙ ЛОГИКИ FUZZY LOGIC TOOLS. ЛАБОРАТОРНЫЙ ПРАКТИКУМ

УЧЕБНО-МЕТОДИЧЕСКОЕ ПОСОБИЕ

Редактор *Е. И. Костина*
Корректор *Е. Н. Батурчик*
Компьютерная правка, оригинал-макет *В. М. Задоля*

Подписано в печать 31.10.2018. Формат 60×84 1/16. Бумага офсетная. Гарнитура «Таймс».
Отпечатано на ризографе. Усл. печ. л. 4,3. Уч.-изд. л. 4,0. Тираж 30 экз. Заказ 102.

Издатель и полиграфическое исполнение: учреждение образования
«Белорусский государственный университет информатики и радиоэлектроники».
Свидетельство о государственной регистрации издателя, изготовителя,
распространителя печатных изданий №1/238 от 24.03.2014,
№2/113 от 07.04.2014, №3/615 от 07.04.2014.
ЛП №02330/264 от 14.04.2014.
220013, Минск, П. Бровки, 6