Федеральное агентство по образованию
Государственное образовательное учреждение
высшего профессионального образования
Петрозаводский государственный университет
Математический факультет

Кафедра информатики
и математического обеспечения

Проект архитектуры для сервера обработки потоков FlowServer

Заказчик:
к. ф.-м. н.,
доцент Корзун Д. Ж.

Последняя модификация:
Чтв Ноя 23 19:17:48 MSK 2006

Рабочая версия

Петрозаводск — 2006

Содержание

1 Высокоуровневая архитектура FlowServer

1.1 Общая схема

Архитектура FlowServer представлена на рисунке 1

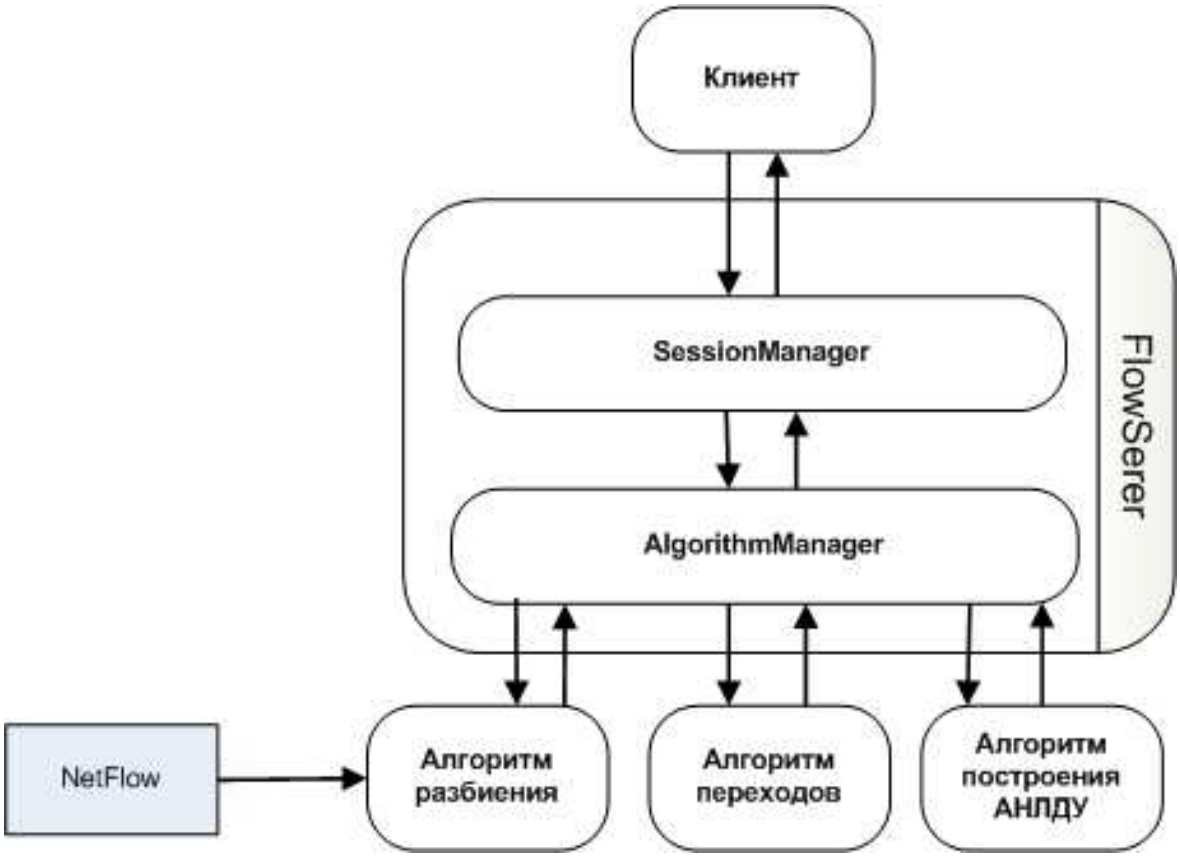

Рис. 1: Схема архитектуры

Модуль Session Manager Взаимодействует с клиентом, получая от него параметры задачи и возвращая ему результат работы модуля. Запускает задачи и контролирует их выполнение.

Модуль AlgorithManager Преобразует данные сетевого трафика в системы одАНЛДУ, используя внешние алгоритмы распределения потоков, построения переходов и генерации систем одАНЛДУ. Управляет выполнением и осуществляет контроль над внешними алгоритмами.

Алгоритм разбиения Распределяет потоки сетевого трафика по временным интервалам.

Алгоритм переходов Строит переходы между состояниями.

Алгоритм построения одАНЛДУ На основе системы переходов строит системы одАНЛДУ.

1.2 Взаимодействие модулей

В модуле SessionManager создается объект класса algorithm_manager(см. ниже), описанный в модуле AlgorithmManager, с помощью метода start() и атрибута params, которого и осуществляется взаимодействие модулей.

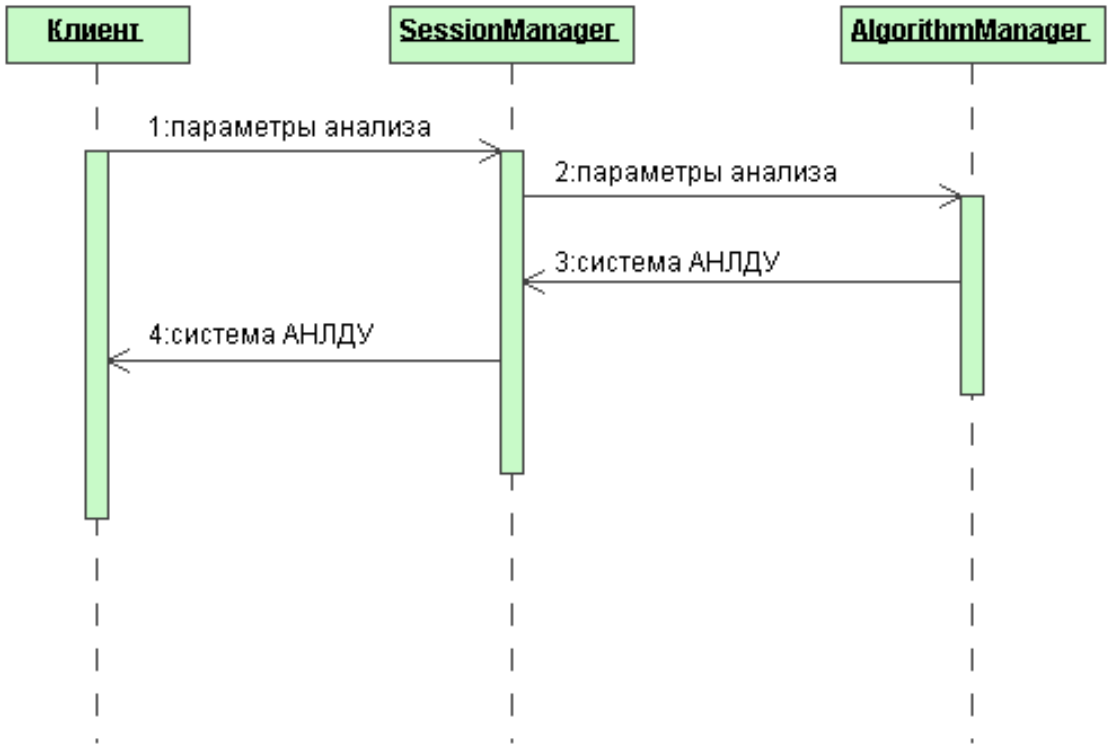

Рис. 2: Схемы работы FlowServer

Схема работы:

1) Клиент передает идентификатор задачи и ее параметры модулю SessionManager

2) Модуль SessionManager создает объект класса algorithm_manager, передает ему параметры задачи

3) Модуль SessionManager вызывает метод класса algorithm_manager из модуля AlgorihmManager, предназначенный для запуска задачи

4) Модуль AlgorihmManager выполняет задачу и передает систему переходов и систему одАНЛДУ модулю SessionManager

5) Модуль SessionManager передает системы переходов и одАНЛДУ клиенту

2 Структуры данных

2.1 Структура anlde_system

Служит для хранения системы диофантовых уравнений

```
struct anlde_system{
int nequ;
int nvar;
int unknowns[ ];
int system[ ][ ];
};
```

Описание полей:

1) *nequ* содержит количество уравнений системы.

2) *nvar* содержит количество переменных системы.

3) *unknowns* описывает левую часть системы АНЛДУ; каждый элемент массива является номером уравнения в системе, в котором находится переменная, соответствующая индексу элемента.

4) *system* описывает правую часть системы АНЛДУ; представляет собой матрицу коэффициентов правой части системы АНЛДУ.

2.2 Структура task_params

Служит для хранения параметров алгоритмов.

```
struct task_params{
int task_id;
int trans_fd;
int window_size;
int nfiles;
char *filenames[ ];
int allot_alg_id;
int allot_prec;
int trans_prec;
int trans_bord;
}
```

Описание полей:

1) *task_id* хранит идентификатор типа поставленной задачи; определяет, требуется ли генерировать систему переходов или она уже задана пользователем.

2) *trans_fd* хранит дескриптор файла с описанием системы переходов

3) *window_size* хранит размер окна алгоритма распределения потоков.

4) *nfiles* хранит количество обрабатываемых файлов NetFlow

5) *filenames[]* хранит список указателей на имена файлов NetFlow, которые подлежат исследованию.

6) *allot_alg_id* хранит идентификатор алгоритма генерации разбиения.

7) *allot_prec* хранит точность округления в алгоритме распределения потоков.

8) *trans_prec* хранит точность определения состояний в алгоритме построения переходов.

9) *trans_bord* минимальное количество появлений одного и того же перехода для того, чтобы он учитывался при построении системы одАНЛДУ. Те, что встречаются реже, отбрасываются.

2.3 Структура client

Служит для организации очереди запросов клиентов.
```
struct client{
int client_socket;
struct client* next;
};
```

Описание полей:

1) *client_socket* - сокет клиента, находящегося в очереди

2) *next* - сслыка на следующего клиента.

3 Описание алгоритмов

3.1 Алгоритм распределения потоков по интервалам

Входные данные:

Набор потоков сетевого трафика и соответствующие им временные метки, отражающие время начала и завершения потоков, во внутреннем формате программного комплекса tcpconan

Результат

Временный файл разбиений. Формат файла смотри ниже.

Параметры

1) Размер окна анализа (либо иной параметр, в зависимости от конкретного алгоритма)

2) Точность разбиения

3.1.1 Описание алгоритма

1) Получение первого потока;

2) Создание интервала с временными границами первого потока;

3) Занесение информации о потоке в интервал;

4) Если потоков больше нет, то переход к пункту 11;

5) Получение очередного потока;

6) Поиск интервалов конца и начала потока;

7) Создание ненайденных интервалов;

8) Занесение информации о потоке во все интервалы между интервалом начала и конца;

9) Если произошел выход за пределы окна, смещение окна;

10) Переход на пункт 4;

11) Вывод информации о потоках в пределах окна.

3.1.2 Комментарии

Упомянутое окно представляет собой окно анализа, его длина равна заданной максимальной продолжительности потока. Смысл использования окна состоит в следующем: так как на маршрутизаторе всегда установлена максимальная длина потока, обычно равная 30 минутам, и все потоки поступают в порядке их окончания, то нет смысла хранить и обрабатывать информацию об интервалах, которые старше обрабатываемого потока на заданное максимальное время потока.

3.2 Алгоритм построения системы переходов между состояниями

Входные данные

Временный файл разбиений. Его формат смотри ниже.

Результат

Текст на языке задания систем переходов. Синтаксис языка описан ниже.

Параметры

1) Точность определения состояния

2) Нижняя граница значимости переходов

3.2.1 Описание алгоритма

Алгоритм строит переходы между состояниями. Под состоянием в данном случае понимается определенный с заданной точностью объем потоков на некотором временном интервале. Каждому из интервалов с одинаковым (с некоторой точностью) суммарным объемом потоков на нем соответствует одно и то же состояние. В работе алгоритма можно выделить следующие этапы:

1) Получение очередного (текущего) состояния;

2) Определение наличия перехода (сравнение текущего состояния с предыдущим);

3) Обработка обнаруженного перехода (добавление в буфер или увеличение счетчика его появлений);

4) Упорядочивание следования переходов (в случае нарушения порядка);

5) Если остались необработанные объемы потоков, переход на пункт 1;

6) Отсеиваие переходов, значение счетчика появлений которых ниже заданной границы значимости переходов;

7) Вывод результата.

3.2.2 Комментарии

Каждому из переходов соответствует структура, содержащая информацию о том, от какого состояния к какому произошел переход, и сколько раз он повторялся.

3.3 Алгоритм построения системы одАНЛДУ

Входные данные

Текст, задающий систему переходов.

Результат

Временный файл системы одАНЛДУ. Формат смотри ниже.

3.3.1 Описание алгоритма

Алгоритм получает на вход систему переходов, которая определяет грамматику. Каждый переход определяет правило. Каждому переходу соответствует переменная, а каждому состоянию уравнение. Схема работы алгоритма следующая:

1) Получение количеств состояний (nstates) и переходов (ntrans);
2) Объявление матрицы инцидентности I[nstates][ntrans] и вектора, описывающего левую часть системы, V[ntrans];
3) Получение очередного перехода t из состояния i в состояние j;
4) I[i][t]=1, I[j][t]=-1;
5) Если остались переходы, переход к пункту 3;
6) Заполнение вектора V: если I[i][j]=1, то V[j]=i;
7) Заполнение матрицы S правой части системы: если I[i][j]=-1, то S[i][j]=1, иначе S[i][j]=0;
8) Вывод результатов;

3.3.2 Комментарии

Алгоритм представляет собой транслятор текста, задающего систему переходов, в систему одАНЛДУ. Реализация алгоритма предполагает использование утилит flex и bison.

4 Форматы обмена данными

Форматы обмена данными представлены в видоизмененной форме Бакуса-Наура (БНФ), а также в виде краткого словесного описания.

Данная БНФ включает в себя следующие конструкции:

имя = определение

Имя правила отделяется от его определения знаком равенства. Имена основных (бызовых) правил пишутся в верхнем регистре. Угловые скобки используются, когда необходимо словесно описать использование имени правила.

правило{N}

Данная конструкция показывает, что “правило” входит в определение в точности N раз. В качестве N может выступать числовая константа, арифметическое

выражение или выражение “$n”, которое интерпретируется как семантическое значение n-того компонента определения. Таким образом, $1 - семантическое значение 1-кого компонента определения. Семантическое значение выражения в данном случае должно быть числовым.

правило*

Символ “*” стоящий после какого-либо элемента определения, говорит о том, что данный элемент может повторяться 0 и более раз.

[правило]

Квадратные скобки заключают необязательные элементы определения.

правило1 | правило2

Элементы, разделенные символом “|” представляют собой альтернативу друг другу. Таким образом, может быть выбран лишь один из этих элементов.

(правило1 правило2)

Элементы, заключенные в круглые скобки, считаются за один элемент. Таким образом, “(elem (foo | bar) elem)” позволяет задать любую из следующих последовательностей токенов: “elem foo elem” и “elem bar elem”.

“строка”

Набор символов, заключенных в двойные кавычки интерпретируется буквально, как строка.

; комментарий

Двоеточие начинает комментарий, который продолжается до конца строки. Это позволяет включать пояснения параллельно спецификации.

4.1 Основные правила

NUM = <любая 32-битная последовательность данных>
CR = <carriage return, возврат коретки>
LF = <line feed, перевод строки>
CRLF = CR LF
ALPH = <ASCII-символ, латинская буква или символ подчеркивания>
CHAR = <ASCII-символ с кодом в диапазоне 0 - 127> DIGIT = <ASCII-символ

с кодом в диапазоне 48 - 58>
OCTET = <любая 8-битная последовательность данных>

4.2 Формат представления разбиения временного интервала

Формат

```
allot_format = nintervals interv_vol{$1}
nintervals = NUM
interv_vol = NUM
```

Описание

1) *nintervals* - количество интервалов, на которые разбит исследуемый временной диапазон, 32-битное число;

2) *interv_ vol* - суммарный объем потоков на каждом из интервалов.

4.3 Спецификация языка задания систем переходов

Формат

```
trans_format = [ta] transitions
transitions = ; отсутствие переходов
| transitions line
line = CRLF
| expr ; выражение, задающее переходы
| comment ; комментарий
expr = list “->” list [“:” tr_attributes] “;”
list = [list “,”] state
state = ALPH [ALPH | DIGIT]*
ta = “ta” “{” attr “}”
attr = [attr] atype “=” <название данного атрибута> “;”
atype = “str” | “int”
tr_attributes = [tr_attributes “,”] tr_attribute
tr_attribute = <название атрибута> “=” <значение атрибута>
comment = “♯” OCTET* CRLF
```

Описание

Система переходов будет задаваться на языке, специально разработанном для этой цели. Текст на этом языке будет генерироваться алгоритмом построения системы переходов и подаваться на вход алгоритма генерация систем одАНЛДУ. Таким образом, алгоритм генерации систем одАНЛДУ будет реализован как транслятор

текста на данном языке в систему одАНЛДУ.
Переходы задаются двумя списками состояний, разделенных оператором “->”. Каждый из списков представляет собой имена состояний, разделенных запятой. Переходы осуществляются из каждого из состояний, присутствующих в списке, находящемся слева от “->”, в каждое из состояний, находящихся в правом списке. Каждый из переходов может иметь некоторые атрибуты. Секция, описывающая эти атрибуты должна располагаться до задания самих переходов. Атрибуты могут быть числового или строкового типа. Значения атрибутов могут задаваться в строке, задающей переходы.

4.4 Формат представления сгенерированной системы одАНЛДУ

Формат

```
anlde_format = nequations nvariables equ{$2} coef{$1 * $2}
nequations = NUM
nvariables = NUM
equ = NUM
coef = NUM
```

Описание

1) *nequations* - количество уравнений в ситеме одАНЛДУ
2) *nvariables* - количество переменных в системе одАНЛДУ
3) *equ* - номер уравнения, в котором находится каждая из переменных (описывает левую часть уравнения)
4) *coef* - коэффициент при каждой из переменных в каждом из уравнений (описывает правую часть уравнения)

4.5 Описание формата конфигурационного файла

4.5.1 Формат

```
config_format = [allot]* [trans] [parse] [tcpconan] [data] [queue]
[timeout] [params]
allot = “allot” path name “;”
trans = “trans” path “;”
parse = “parse” path “;”
tcpconan = “tcpconan” path “;”
data = “data” path “;”
queue = “queue” size “;”
timeout = “timeout” time “;”
```

```
params = [field_name ‘‘=’’ value‘‘;’’]*
field_name = ‘‘window_size’’ | ‘‘allot_alg_id’’ | ‘‘allot_prec’’ |
‘‘trans_prec’’ | ‘‘trans_bord’’
path = <путь к файлу>
name = CHAR*
size = NUM
time = NUM
value = min max default
min = NUM
max = NUM
default = NUM
```

4.5.2 Описание

1) `allot` - путь к алгоритму распределения потоков и имя алгоритма; этих алгоритмов может быть несколько

2) `trans` - путь к алгоритму построения системы переходов

3) `parse` - путь к алгоритму построения системы одАНЛДУ

4) `tcpconan` - путь к модулю machine системы TCPConan

5) `data` - путь к каталогу, содержащему файлы NetFlow. Все файлы из этого каталога будут рассматриваться как файлы NetFlow и будут предложены клиенту для анализа.

6) `queue` - максимальный размер очереди

7) `timeout` - максимальное время, в течении которого может выполнятся задача.

8) `params` - параметры алгоритмов (минимальное, максимальное и значение по умолчанию)

4.6 Описание формата протокол взаимодействия сервера потоков с клиентом

4.6.1 Команды клиента

Описание команд:

1) HELLO
   Команда установления соединения. Обязательно должна быть послана в самом начале диалоге клиента с сервером.
   После этого сообщения к клиенту может вернуться сообщения: READY, WAIT, FAILED (см. ниже).

2) FULL_EXEC params
   Команда выполнения полного алгоритма преобразования. Запуск выполнения задачи преобразования данных сетевого трафика в систему АНЛДУ.
   Параметр команды: params . параметры задачи преобразования данных сетевого трафика (см п.)
   После этого сообщения к клиенту может вернуться сообщения: SOLVING, DONE, FAILED.

3) PART_EXEC trans
   Команда выполнения частичного алгоритма преобразования. Запуск выполнения задачи преобразования системы переходов в систему АНЛДУ.
   Параметр команды: trans . параметры задачи преобразования системы переходов (см п.)
   После этого сообщения к клиенту может вернуться сообщения: SOLVING, DONE, FAILED.

4) RESET
   Команда отмены выполнения задачи.
   После этого сообщения к клиенту может вернуться сообщения: FAILED.

5) GET_LIST
   Команда получения списка файлов с данным сетевого трафика на сервере.
   После этого сообщения к клиенту может вернуться сообщения: SEND_LIST, FAILED.

Формат параметров:

```
params=nfiles file_name{$1} [field_name -"value]*
nfiles=NUM
file_name=string
field_name="window_size "allot_alg_id "allot_prec "trans_prec-
"trans_bord"
value=NUM

trans=text
text=string | string CRLF text
string=CHAR | CHAR string
```

Описание:

1) string - строка

2) text - текст

3) nfiles - кол-во файлов

4) file_name - имя файла

5) field_name - имя поля

6) value - значение поля
   возможные значения поля:

   (a) "window size размер окна

   (b) "allot_alg_id идентификатор алгоритма разбиения

   (c) "allot_prec точность алгоритма разбиения

   (d) "trans_prec точность алгоритма построения переходов

   (e) "trans_bord минимальное кол-во появлений одного и того же перехода для того, чтобы он учитывался при построении системы АНЛДУ. Те, что встречаются реже, отбрасываются.

4.6.2 Команды сервера

Описание команд:

1) READY
   Команда подтверждения готовности сервера начать взаимодействие с клиентом.

2) DONE result
   Команда передачи результата выполнения задачи клиенту.
   Параметр: result . результат работы сервера: система АНЛДУ и система переходов.

3) SENDLIST filenames
   Команда отправки списка файлов с данными сетевого трафика, находящимися на сервере.
   Параметр: filenames . список файлов.

4) FAILED msg
   Команда отправки сообщения о ошибке, возникшей во время выполнения задачи.
   Параметр: msg . идентификатор ошибки.

5) WAIT
   Команда сообщающая клиенту о том, что сервер занят.

6) SOLVING
   Команда подтверждающая принятие задачи на выполнение

Формат параметров:

```
filenames=nfiles file_name{$1}
nfiles=NUM
file_name=string
string=CHAR | CHAR string
```

```
msg="File not found"
| "Unknown error"
| "Trans system is wrong"
```
<уточнить все ошибки!>

```
result=anlde_system tran
text=string | string CRLF text
```

Описание:

1) anlde_system - система АНЛДУ (см п.)
2) trans - система переходов
3) string - строка
4) nfiles - кол-во файлов
5) file_name - имя файла

4.6.3 Описание работы протокола

Клиент инициализирует соединение командой HELLO. И ожидает ответа сервера. Если сервер готов, то сервер отсылает READY. В случае, когда сервер занят, он отвечает WAIT, что приводит клиента в состояние ожидания, как только сервер освободится и сможет принять запрос клиента, он посылает команду READY. После получения READY клиент передает параметры задачи преобразования с помощью FULL_EXEC либо PART_EXEC в зависимости от требуемой задачи. После получения такого сообщения сервер начинает выполнение задачи и клиенту возвращается сообщение SOLVING, уведомляющее о синтаксической корректности всех параметров и начале работы над выполнением преобразования. Клиент может прервать работу сервера командой RESET, после этой команды сервер останавливает, удаляет данные о задаче и возвращает сообщение FAILED с параметром .Aborted..После выполнения клиенту посылается сообщение DONE содержащее результат работы сервера. После этого соединение разрывается.

Для получения списка файлов на сервере клиентом посылается сообщение GET_LIST, получив его, сервер отсылает весь список файлов с данными сетевого трафика командой SEND_LIST. Эта процедура не требует изначального инициирования соединения клиентом командой HELLO.

4.6.4 Ошибки и их коды

1) Aborted by user
2) Time is out
3) NetFlow file not found

4) Algorithm not found

5) Invalid parameter

6) Internal server error

7) Unknown error

5 Проектирование подсистем

5.1 Модуль SessionManager

5.1.1 Общая архитектура модуля

Этот модуль предназначен для работы с клиентами сервера потоков и вызова алгоритма выполнения задачи преобразования данных сетевого трафика в системы АНЛДУ.

Архитектура модуля представлена на рисунке 3.

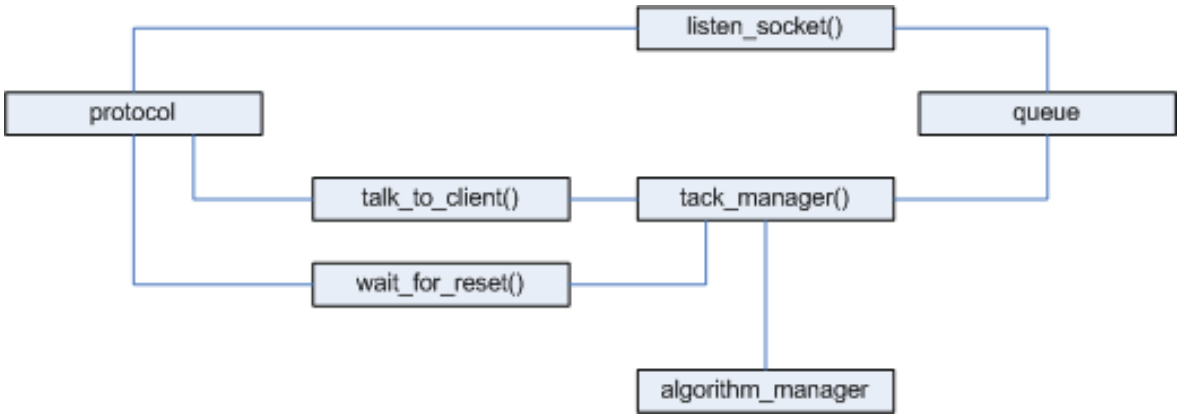

Рис. 3: Архитектура модуля SessionManager

Модуль состоит из 2 классов:

1) queue - класс очереди клиентов,

2) protocol - класс, содержащий функции для общения с клиентом по средством разработанного протокола,

а также из 4 функций:

1) listen_socket() - функция первоначального общения с клиентом,

2) task_manager() - функция, управляющая вызовом алгоритма выполнения задачи и общением с клиентом,

3) talk_to_client() - функция общения с клиентом,

4) wait_to_reset() - функция ожидания отмены задачи от клиента во время её выполнения.

5.1.2 Описание схемы работы

Клиент инициирует соединение сообщением HELLO, которое принимает функция listen_socket() через слушающий сокет. После эта функция принимает соединение на этом сокете и полученный сокет для общения с клиентом помещает в очередь. Если же был запрос на получение списка файлов сетевого трафика, то функция сама вызывает команду протокола и отвечает клиенту, обращаясь к классу protocol.
Функция task_manager() работает в отдельном потоке относительно функции listen_socket(), постоянно проверяет очередь queue и, как только там что-то появляется или заканчивается предыдущая задача, получает первый элемент из очереди. После вызывается функция talk_to_client() для общения с клиентом, которая использует функции класса protocol. Получив все параметры задачи task_manager() запускает функцию wait_to_reset() в отдельном потоке и инициирует работу algorithm_manager'a. Во время работы последнего, функция wait_to_reset() следит за сообщениями клиента, ожидая отмены, если произошла отмена, то устанавливается соответствующий флаг, который просматривает algorithm_manager и в случае ошибки отменяет выполнение задачи.
После остановки или завершения выполнения задачи algorithm_manager'ом функция task_manager() завершает выполнение функции wait_for_reset(), с помощью класса protocol отсылает результат клиенту либо сообщение об ошибке, после чего, либо берет новое задание, если очередь не пуста, либо если очередь пуста, то ждет её пополнения.

5.1.3 Описание класса queue

Класс представляет собой очередь клиентов сервера потоков. Сама очередь представляет собой связный список.

```
class queue{
private:
client *first;
client *last;
int count;
public:
queue();
int put(int socket);
int get();
int delete(int socket);
int get_count();
```

```
}
```
Описание методов:

1) `int put(int socket)` - функция добавления клиента в очередь
   Входные параметры:
   int socket - сокет клиента, которого добавляем в конец очереди.
   Результат: код возврата, в случае ошибки равен 1, в случае положительного результата операции - 0.

2) `int get()` - функция получения клиента из очереди
   Результат: сокет очередного клиента, находящегося в начале очереди

3) `int delete(int socket)` - функция удаления клиента из очереди
   Входные параметры:
   int socket - сокет клиента, которого хотим удалить из очереди.
   Результат: код возврата, в случае ошибки равен 1, в случае положительного результата операции - 0.

4) `int get_count()` - функция получения размера очереди
   Результат: количество клиентов в очереди.

Описание полей:

1) `client *first` - ссылка на первый клиент в очереди

2) `client *last` - ссылка на последнего клиента в очереди

3) `int count` - текущий размер очереди

5.1.4 Описание класса protocol

Класс представляет собой набор функций по работе с клиентом при помощи разработанного протокола.

```
class protocol{
private:
int socket;
sockaddr *addr;
socklen_t *addrlen;
task_params* parse_params_to_struct(int file);
int parse_system_from_struct(andle_system* system);

public:
protocol(int socket);
int set_socket(int socket);
int sent_wait();
int sent_ready();
int sent_list();
int sent_failed(message msq);
```

```
int sent_solving();
int sent_done(andle_system* system, int file);
task_params* get_params();
}
```

Описание открытых методов:

1) `int set_socket(int socket)` - функция установления сокета
Входные параметры:
1. `int socket` - сокет клиента, с которым будет идти взаимодействие по протоколу.
Результат: код возврата, в случае ошибки равен 1, в случае положительного результата операции - 0.

2) `int sent_wait()`- функция отправки сообщения WAIT клиенту
Результат: код возврата, в случае ошибки равен 1, в случае положительного результата операции - 0.

3) `int sent_ready()` - функция отправки сообщения READY клиенту
Результат: код возврата, в случае ошибки равен 1, в случае положительного результата операции - 0.

4) `int sent_list()` - функция отправки списка файлов с данными сетевого трафика, доступных для преобразования в системы АНЛДУ сервером.
Результат: код возврата, в случае ошибки равен 1, в случае положительного результата операции - 0.

5) `int sent_failed(int code)` - функция отправки сообщения об ошибке.
Входные параметры:
1. `int code` - код сообщения об ошибке.
Результат: код возврата, в случае ошибки равен 1, в случае положительного результата операции - 0.

6) `int sent_solving()`- функция отправки сообщения SOLVING клиенту.
Результат: код возврата, в случае ошибки равен 1, в случае положительного результата операции - 0.

7) `int sent_done(andle_system* system, int file)` - функция отправки результата выполнения задачи клиенту.
Входные параметры:
1. `andle_system* system` - система АНЛДУ, полученная в результата выполнения задачи и которая должна быть отправлена клиенту
2. `int file` - дескриптор файла, содержащего системы переходов, полученную в результата выполнения задачи и которая должна быть отправлена клиенту
Результат: код возврата, в случае ошибки равен 1, в случае положительного результата операции - 0.

8) `task_params* get_params()` - функция получения параметров задачи от клиента.
   Результат: параметры задачи, полученные от клиента.

Описание закрытых методов:

1) `task_params* parse_params_to_struct(int file)` - функция преобразования параметров задачи, полученных по протоколу, в структуру.
   Входные параметры:
   1. `int file` - файл содержащий параметры задачи, полученные от клиента
   Результат: структура параметров задачи, полученных от клиента

2) `int parse_system_from_struct(andle_system* system)` - функция преобразования системы АНЛДУ из системы в формат, передаваемый протоколом.
   Входные параметры:
   1. `andle_system* system` - структура, содержащая систему АНЛДУ
   Результат: дескриптор файла, содержащего системы АНЛДУ в формате, используемом в протоколе.

Описание полей:

1) `int socket` - сокет клиента

2) `sockaddr *addr` - это указатель на структуру, в которой хранится адрес клиента, в том виде, в каком он известен на уровне коммуникации.

3) `socklen_t *addrlen` - действительная длина адреса в байтах

5.1.5 Описание функции listen_socket()

Функция ожидания запросов от клиента. Работает в отдельном потоке. Функция предназначена для получения запросов от клиента, содержит слушающий сокет, через который поступают запросы от клиентов, для которых, после получения запроса, создается отдельный сокет для общения.

Работает по следующему алгоритму:

1) Слушает сокет, пока не будет запроса от пользователя.

2) Принимает соединение на сокете

3) Если пришло сообщение HELLO, то полученный сокет для клиента добавляет в очередь queue. Переход на шаг 1.

4) Если пришло сообщение GET_LIST, то с помощью функции sent_list() класса protocol отсылает клиенту список файлов сетевого трафика, доступных для сервера, клиенту. Переход на шаг 1.

5) Игнорировать сообщение от клиента, удалить полученный сокет и переход на шаг 1.

5.1.6 Описание функции task_manager()

Функция, управляющая вызовом алгоритма выполнения задачи и общением с клиентом. В её задачи входит управление выполнением задачи и отправке результата клиенту.

Функция работает по следующему алгоритму:

1) Если очередь queue пуста, то ждать определенное время и переход на шаг 1.
2) Получить очередного клиента с очереди queue.
3) Вызов функции talk_to_client() для общения с клиентом и получения параметров задачи
4) Если параметры не переданы, то переход на шаг 1.
5) Создание потока и запуск в нем функции wait_for_reset().
6) Создание класса algorithm_manager и передача ему параметров задачи
7) Завершение функции wait_for_reset().
8) Если были ошибки задачи или пользователь отменил выполнение задачи, то отправка сообщения об ошибке с помощью функции sent_failed() класса protocol, и переход на шаг 1.
9) Отправка результата выполнения задачи клиенту с помощью функции sent_done() класса protocol, и переход на шаг 1.

5.1.7 Описание функция talk_to_client()

Функция общения с клиентом для получения параметров задачи. Отсылает клиенту сообщение READY, получает параметры задачи и отсылает клиенту сообщение о начале выполнения его задачи.

Функция работает по следующему алгоритму:

1) С помощью функции sent_ready() класса protocol отсылает клиенту сообщение READY
2) С помощью функции get_params() класса protocol ожидает и получает параметры задачи.
3) С помощью функции sent_ready() класса protocol отсылает клиенту сообщение SOLVING
4) Возвращает структуру с параметрами задачи

5.1.8 Описание функция wait_for_reset()

Функция ожидание отмены задачи от клиента во время её выполнения. Работает в отдельном потоке. В случае получения отмены, устанавливает соответствующий флаг.

Работает по следующей схеме:

1) Слушает сокет, пока не будет сообщения от клиента
2) Если это сообщение RESET, то устанавливает соответствующий флаг и завершается.
3) Игнорирует сообщение и переход на шаг 1.

5.2 Модуль AlgorithmManager

Этот модуль предназначен для управления и вызова алгоритмов преобразования данных сетевого трафика в системы АНЛДУ. Модуль содержит класс algorithm_manager, управляющий выполнением алгоритмов, а также набор классов, соответствующих алгоритмам:

1) класс allot_alg
2) класс trans_alg
3) класс parse_alg

Все классы, соответствующие алгоритмам, наследуются от абстрактного класса *algorithm* (см. рис. 3), который содержит методы:

1) *exec()* - запуск алгоритма на выполнение
2) *wait()* - ожидание завершения алгоритма

Архитектура и схема работы модуля AlgorithmManager изображена на рисунке 4 и 5 соответственно.

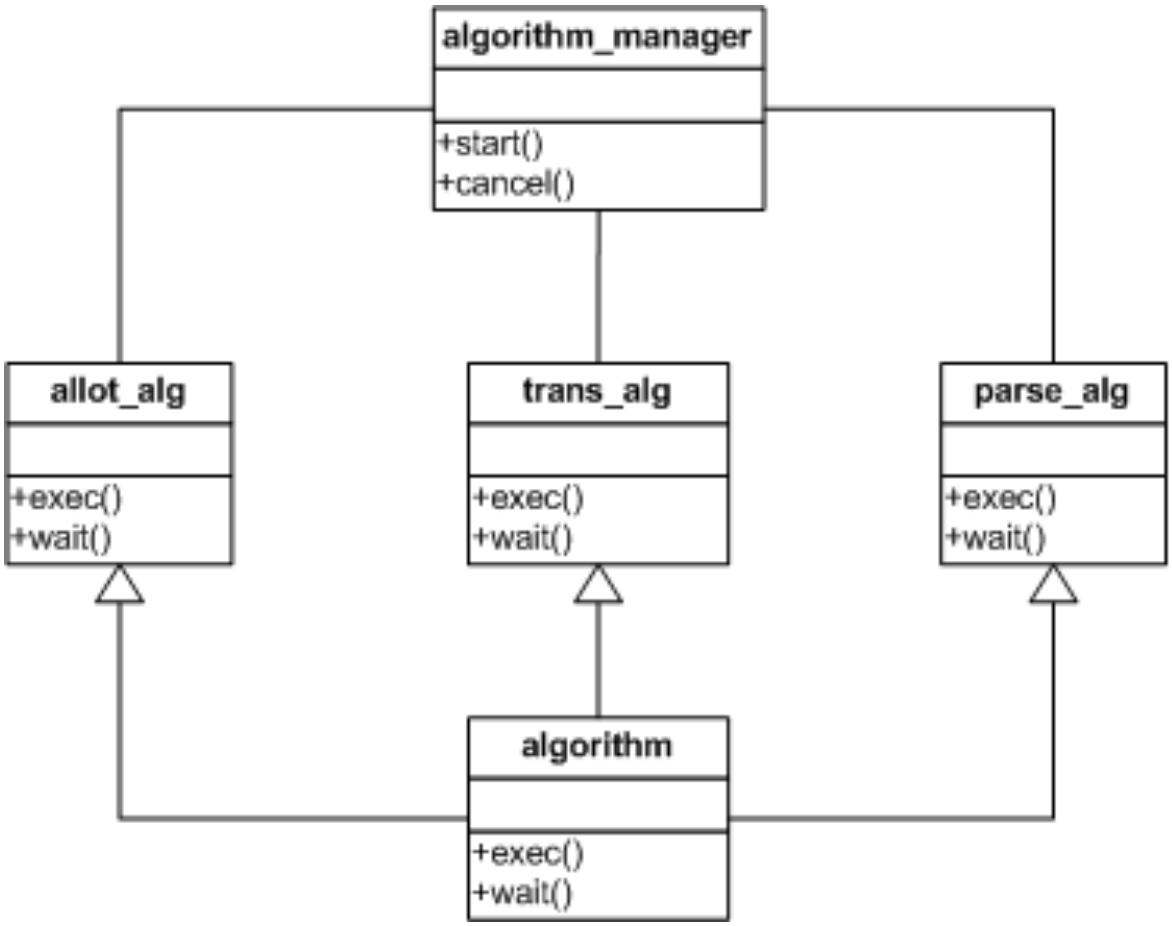

Рис. 4: Диаграмма классов модуля AlgorithmManager

5.2.1 Класс algorithm_manager

Этот класс осуществляет управление процессами, которые реализуют алгоритмы, и координирует их действия. Класс AlgrithmManager предоставляет классу SessionManager интерфейс взаимодействия с алгоритмами.

```
class algorithm_manger{
public:
algotithm_manager( task_params params );
void start();
void cancel();
anlde_system get_anlde_system();
int get_res_code();
}
```

Описание конструкторов:

algorithm_ manager(task_ params params) создает объект класса algorithm_manager, передавая созданному экземпляру параметры алгоритмов.

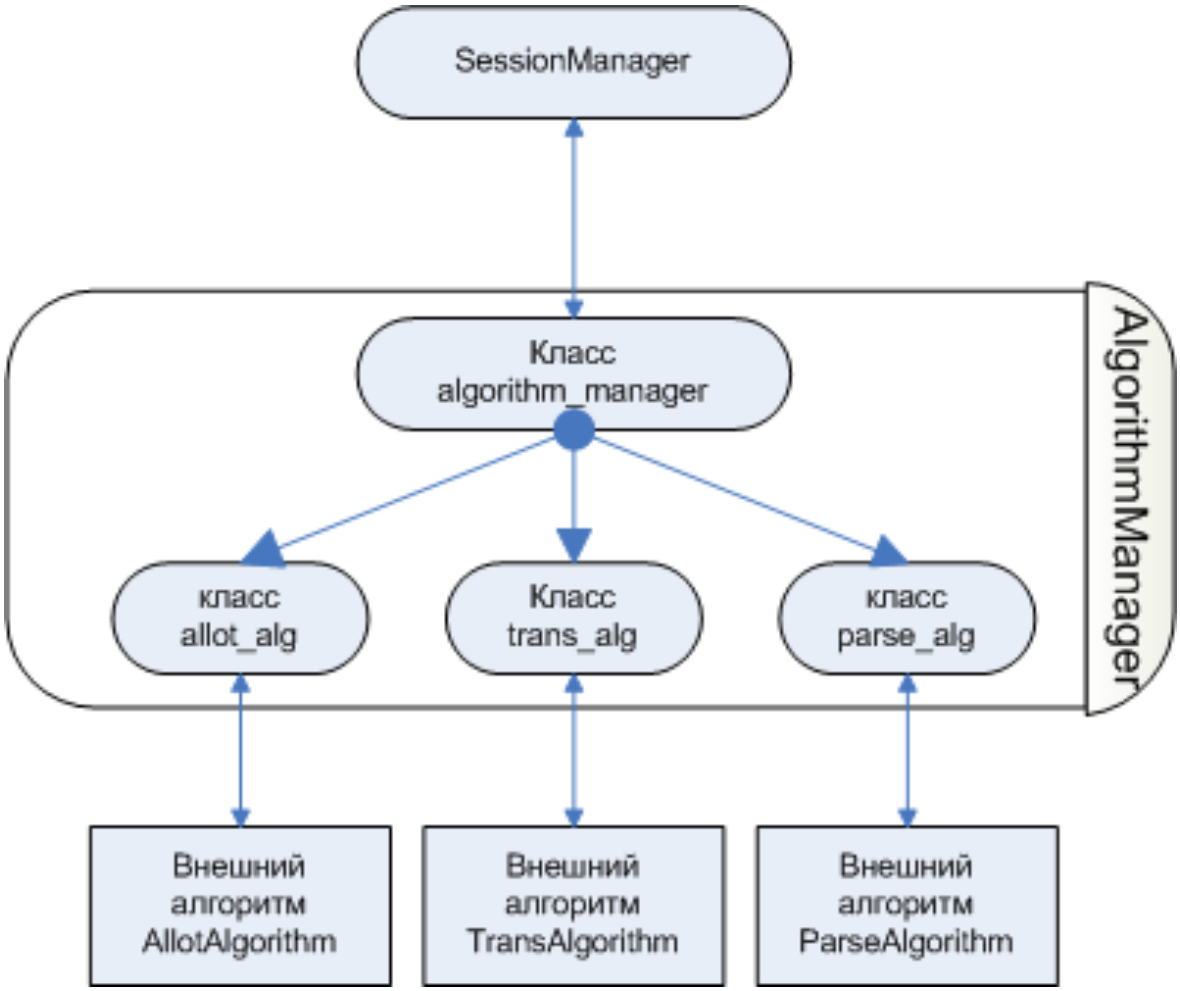

Рис. 5: Схема архитектуры

Описание методов:

1) *void start()* заполняет структуру anlde_system, при этом метод:

   (a) Создает объекты классов allot_alg, trans_alg, parse_alg и объявляет указатели на массивы для хранения входных параметров каждого из алгоритмов;

   (b) Создает временный файл, затем выполняет разбор структуры params, разделяя между собой входные параметры каждого из алгоритмов;

   (c) Пользуясь методами классов, соответствующих алгоритмам, последовательно запускает каждый из алгоритмов
   При возникновении ошибки, менеджер алгоритмов возвращает код ошибки и завершает выполнение задачи;

   (d) Заполняет структуру anlde_system;

   (e) Выполняет зачистку (удаляет созданные объекты, удаляет временный файл и т.п.).

2) *void cancel()* прекращает выполнение задания.

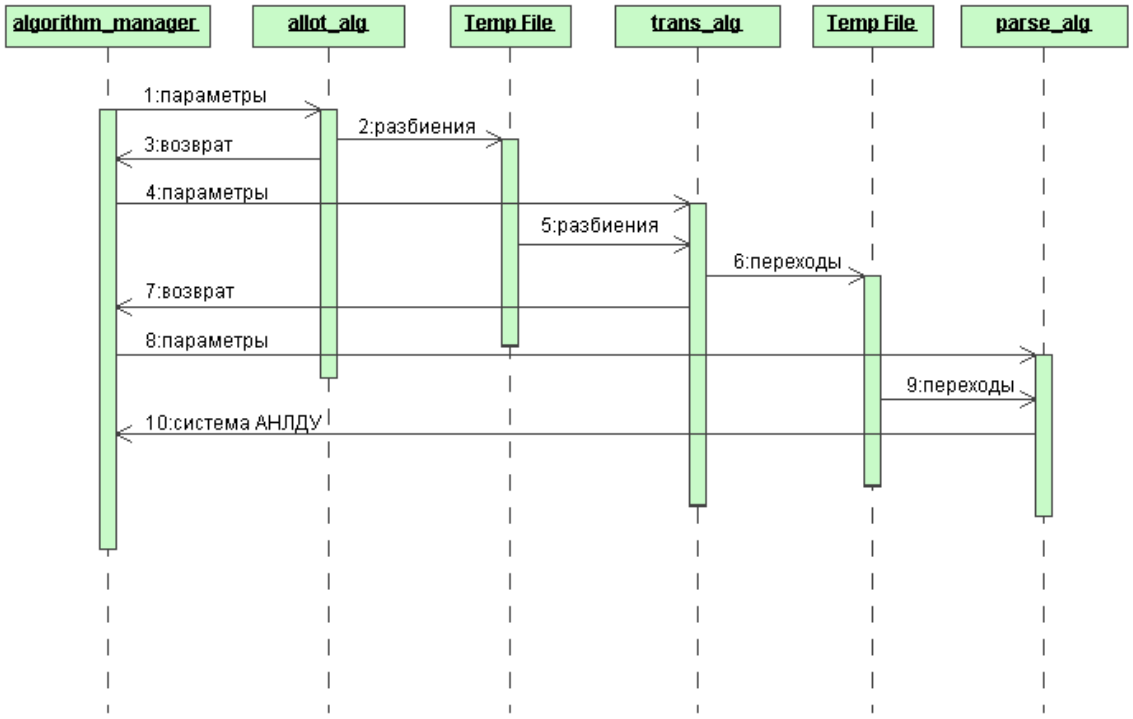

Рис. 6: Схема работы модуля AlgorithmManeger

3) *anlde_ system get_ anlde_ system()* возвращает систему АНЛДУ.

4) *get_ res_ code()* возвращает код состояния, в котором находится экземпляр класса.

5.2.2 Класс allot_alg

Этот класс служит для управления процессом генерации разбиения.

```
claass allot_alg{
public:
allot_alg();
int start(char **argv);
void wait();
}
```

Описание Конструкторов:
allot_ alg() создает объект класса alot_alg.

Описание методов:

1) *public: int start (char **argv)* запускает процесс, реализующий алгоритм. Возвращает идентификатор процесса.

2) *public: void wait ()* метод ждет завершения процесса.

5.2.3 Класс trans_alg

Этот класс служит для управления процессом построения переходов.

```
class trans_alg{
public:
trans_alg();
int start(char **argv);
void wait();
}
```

Описание Конструкторов:
trans_ alg() создает объект класса trans_alg.

Описание методов:

1) *public: int start (char **argv)* запускает процесс, реализующий алгоритм. Возвращает идентификатор процесса.

2) *public: void wait ()* метод ждет завершения процесса

5.2.4 Класс parse_alg

Этот класс служит для создания системы диофантовых уравнений на основании построенной системы переходов.

```
class parse_alg{
public:
parse_alg();
int start(char **argv);
void wait();
}
```

Описание Конструкторов:
parse_ alg() создает объект класса parse_alg.

Описание методов:

1) *public: int start (char **argv)* запускает процесс, реализующий алгоритм. Возвращает идентификатор процесса.

2) *public: void wait ()* метод ждет завершения процесса.

5.2.5 Утилиты, реализующие алгоритмы:

1) Утилита построения разбиения потоков сетевого трафика по времени.
Входные параметры:

argv[0] имя программы;
argv[1] дескриптор временного файла;
argv[2] дескриптор читаемого конца канала;
argv[3] размер окна (или другой параметр, в зависимости от конкретного алгоритма)
argv[4] точность разбиения

Утилита посредством канала взаимодействует с модулем machine программного комплекса tcpconan. Модуль machine обрабатывает переданные ему файлы NetFlow, приводя содержащуюся в них информацию о потоках сетевого трафика к внутреннему формату комплекса tcpconan, после чего пишет полученные данные в записываемый конец канала. Необходимость преобразования формата обусловлена тем, что утилита построения разбиения сгенерирована при помощи комплекса tcpconan и, как следствие, работает с его внутренним внутренним форматом данных.

После получения информации о потоках утилита построения разбиения разбивает полученные потоки по временным интервалам, используя свой алгоритм разбиения. Вообще говоря, на настоящий момент разработано несколько алгоритмов разбиения, каждый из которых будет реализован в виде отдельной утилиты. Каждая из этих утилит работает по одной схеме, различаясь лишь алгоритмом построения разбиений. Какой именно алгоритм использовать, определяет пользователь, передавая идентификатор алгоритма серверу обработки потоков. Менеджер алгоритмов на основании идентификатора определяет, какую именно утилиту запустить.

После того как разбиение построено, утилита записывает во временный файл количество полученных интервалов и последовательность объемов сетевого трафика на каждом из интервалов, после чего завершает свою работу.

2) Утилита построения системы переходов.
Входные параметры:
argv[0] имя программы
argv[1] дескриптор временного файла
argv[2] точность определения состояния
argv[3] нижняя граница значимости перехода

Утилита читает из временного файла информацию об объемах потоков, относя каждый объем к одному из состояний с определенной точностью. Затем строится система переходов между состояниями. Переход между двумя состояниями происходит тогда, когда эти два состояния различны. Два перехода равны, если равны их соответствующие состояния (состоянию, из которого происходит один переход состветствует состояние, из которого происходит другой переход, то же самое и для состояний, в которые прои-

ходит переход). Учитывается только один из равных переходов, остальные лишь увеличивают его счетчик появлений.

После построенииния системы переходов из нее отбрасываются те переходы, счетчик появления которых имеет значение меньшее, чем определенная нижняя граница значимости переходов.

Затем на основании построенных переходов генерируется текст на языке задания систем переходов. Текст пишется во временный файл.

3) Утилита построения системы неотрицательных линейных диофантовых уравнений, ассоциированной с грамматикой.
Входные параметры:
argv[0] имя программы
argv[1] дескриптор временного файла

Утилита представляет собой транслятор текста, задающего систему переходов, в систему одАНЛДУ. Текст берется из временного файла, полученного либо в результате работы утилиты построения системы переходов, либо от клиента. Система переходов определяет грамматику, по которой строится система одАНЛДУ. Каждому переходу соответствует переменная, а каждому состоянию соответствует уравнение.

5.2.6 Функции основной (управляющей) части менеджера алгоритмов.

start()

Сначала менеджер алгоритмов определяет, какая задача решается: задача построения системы переходов и системы одАНЛДУ на основании заданных пользователем параметров алгоритмов, или задача построения лишь системы одАНЛДУ на основании отредактированной пользователем системы переходов. В случае, построения системы переходов с помощью алгоритмов менеджер алгоритмов создает временный файл для записи результатов работы каждого из алгоритмов и канал для взаимодействия модуля machine программного комплекса tcpconan и процесса построения разбиения.
Затем он производит разбор структуры входных параметров, определяя алгоритм построения разбиения и имена файлов NetFlow, подлежащих обработке.
После этого менеджер алгоритмов формирует массивы входных параметров каждого из процессов, и запускает их, следуя следующему порядку:

1) Запуск модуля machine. Модулю передаются имена файлов NetFlow и дескриптор конца канала для записи (Скорее всего, дескриптор конца канала передаваться не будет; вместо этого он будет подменять собой стандартный вывод).

2) Запуск процесса разбиения. Передаваемые параметры указаны выше.

3) Ожидание завершения процесса разбиения.

4) Запуск процесса построения системы переходов.

5) Ожидание завершения процесса построения системы переходов.

6) Запуск процесса построения системы АНЛДУ.

7) Ожидание завершения процесса построения системы АНЛДУ.

Если статус завершения любого из процессов отличен от нуля, выполнение программы прекращается, и генерируется соответствующее сообщение. То же происходит и при возниконовении других ошибок.

После завершения процесса построения системы АНЛДУ менеджер алгоритмов читает из временного файла результат работы алгоритма и на основании полученных данных заполняет струткту, предназначеную для хранения системы АНЛДУ. Эта структура посредством метода класса доступна из функции, в которой был инициализирован менеджер алгоритмов.

После всего этого менеджер алгоритмов закрывает открытые им дескрипторы файлов, освобождает динамическую память, удаляет временный файл.

cancel ()

Менеджер алгоритмов посылает сигнал (SIGINT) запущенным в настоящий момент процессам (их может быть несколько, если работают machine и процесс разбиения) для их завершения. Затем закрывает дескрипторы файлов, освобождает динамическую память и удаляет временный файл.