Санкт-Петербургский государственный университет информационных технологий,
механики и оптики.
Кафедра «Компьютерные технологии»

А. Д. Жданов, Т. М. Коломейцева, А. А. Шалыто

Реализация надежного протокола передачи данных (Модификация протокола *TCP*)

Программирование с явным выделением состояний.
Проектная документация

Проект создан в рамках «Движения за открытую проектную документацию »

http://is.ifmo.ru

Санкт-Петербург

2006

Введение

Низкий уровень передачи данных обеспечивает корректную передачу пакета данных, но не гарантирует сохранения последовательности пакетов и факта их доставки. Такие условия неприемлемы для передачи данных в современных сетях. В связи с этим появляется необходимость создания протокола передачи данных, который бы гарантировал, что все переданные в сеть пакеты доставлены принимающей стороне в корректной последовательности. В данной работе рассмотрен вариант реализации помехоустойчивости соединения за счет индексирования всех переданных пакетов и подтверждения получения каждого пакета.

Данный протокол может быть использован как база для разработки протоколов более высокого уровня, как, например, протокола *SMTP (Simple Mail Transfer Protocol)* [1 – 2], используемого для передачи электронной почты.

Для упрощения понимания поведения сторон используется автоматный подход. Проект реализован в виде библиотеки на языке *C++* под операционной системой (ОС) *Linux*®. Для тестирования проекта реализована консольная версия клиента и сервера. Под ОС *Windows*® программа запускается посредством пакета *Cygwin* [3], обеспечивающего эмуляцию *UNIX*®-подобных систем.

1. Постановка задачи

Требуется разработать надежный протокол передачи данных, который будет гарантировать их корректную доставку. Этот протокол предполагается использовать в системах типа *клиент-сервер* и распределенных системах типа *P2P (Peer to peer, Одноранговая сеть)*. В качестве основы был взят протокол *TCP (Transfer Control Protocol)* [4 – 6].

В спецификации *RFC (Request For Comments)* протокол *TCP* описывался одним автоматом (рис. 1), который дает представление о поведении, как клиентской стороны, так и серверной, без их разделения.

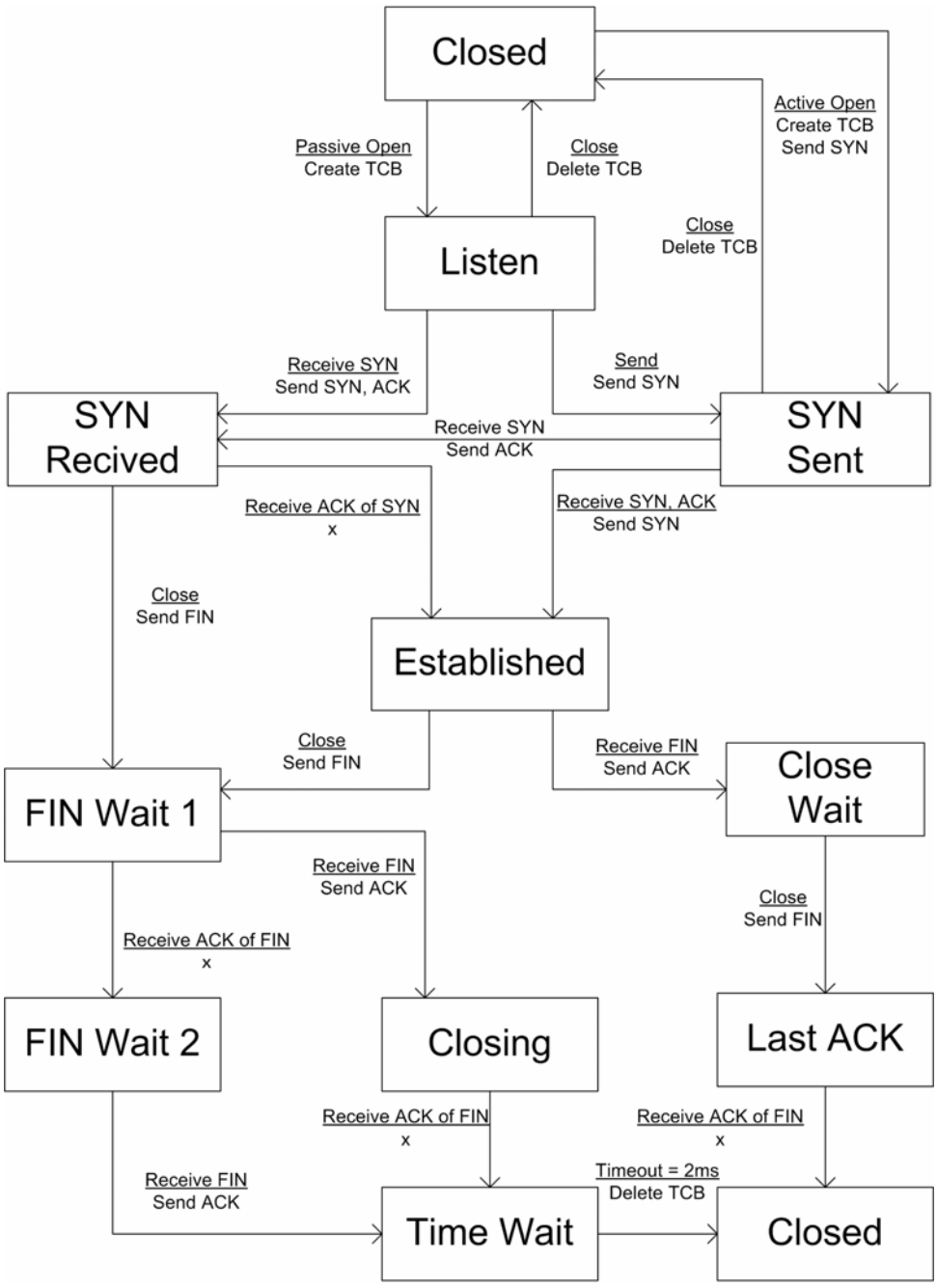

Рис. 1. Описание протокола *TCP* в *RFC*

При этом граф переходов автомата содержит неформальное описание входных и выходных воздействий. Для понимания столь нечетко заданного описания протокола приходится разбираться почти в 80-и страницах текста. Значительную часть спецификации занимают примеры работы данного протока, показывающие как он должен работать в различных ситуациях. Таким образом, понимание его работы строится скорее на этих примерах, чем на формальном описании протокола. При этом отметим, что реализация этого стандарта существует независимо от спецификации, а так как спецификация не формализована, то и реализация выполняется эвристически.

Цель настоящей работы – реализовать протокол *TCP* с небольшими изменениями, предварительно формализовав его спецификацию с помощью системы взаимодействующих конечных автоматов. В изменениях были добавлены дополнительные переходы, которые не описаны в спецификации. Они отвечают за корректное поведение протокола при получении некорректного с точки зрения текущего состояния пакета данных.

При описании протокола в каждом автомате каждое входное и выходное воздействие должно быть четко определено. Входные и выходные воздействия реализуются с использованием протокола более низкого уровня, который не гарантирует корректную доставку данных.

В данной работе предлагается проектировать и реализовывать протокол с использованием *SWITCH*-технологии [7, 8]. При этом спецификация задается формально в виде шести автоматов, а по ней формально и изоморфно строится его реализация.

Для обеспечения надежности каждый передаваемый пакет кроме данных содержит заголовок, в котором хранятся:

- адрес получателя;
- адрес отправителя;
- порт получателя;
- порт отправителя;
- порядковый номер передаваемого пакета;
- порядковый номер пакета, который ожидается следующим на получение;
- набор флагов:
    - запрос на синхронизацию. Этот флаг означает, что сторона инициирует соединение и передает индекс своего первого пакета.
    - запрос на окончание соединения. Этот флаг означает, что сторона закончила сеанс связи и больше не ждет данных.

    - подтверждение получения данных;
    - аварийное завершение соединения;
- размер поля данных.

После передачи пакета данных передающая сторона ожидает подтверждения их получения от принимающей стороны. Если такого подтверждения не было получено в течение максимального времени жизни пакета, то передача повторяется. Если же за несколько таких итераций подтверждение не было получено, то принимающей стороне передается сообщение об ошибке, и соединение разрывается. Кроме того, на принимающей стороне игнорируются все пакеты, которые устарели – уже были получены ранее.

Также на каждой из сторон при получении некорректного для текущего состояния пакета происходит аварийный разрыв соединения с передачей сообщения об этом.

Протокол надежен, так как построенная система гарантирует, что ни один пакет не будет получен дважды и ни один не будет пропущен.

2. Методика формализации автоматов

Как было сказано выше, описание протокола в его спецификации дано не в лучшем для понимания и реализации виде. Для улучшения его понимания и упрощения реализации описанного в *RFC* протокола был применен следующий подход:

- На первом шаге требовалось разделить протокол на три этапа работы:
  - установка соединения;
  - передача данных;
  - завершение соединения.
- Далее в каждом из этапов была выделена активная и пассивная стороны. Активная сторона отвечала за инициализацию процесса, в то время как пассивная только отвечала на запросы. Покажем как разделены эти стороны в различных этапах работы.
  - Перед установкой соединения одна из сторон находится в пассивном состоянии, ожидая соединение (так называемая серверная сторона). В этот момент к ней возможно подключение стороны, перешедшей в активный режим работы (так называемая клиентская сторона).
  - Находясь в состоянии с установленным соединением, обе стороны находятся в пассивном состоянии до тех пор, пока одной из них не требуется передать данные или завершить соединение. Тогда она переходит в активное состояние и инициализирует процесс передачи данных.
  - Для завершения соединения одна из сторон переходит в состояние активного разъединения и инициализирует процесс. В этом случае вторая сторона меняет своё состояние с пассивного ожидания данных на пассивное разъединение.

Таким образом, получено шесть различных возможных состояний каждой из сторон системы, поведение стороны в которых можно описать с помощью шести автоматов. После этого стало возможно, формально реализовав автоматы, с первого раза получить корректно работающее приложение.

3. Интерфейс

Для демонстрации работы разработанного модуля, применяются две программы. Первая выступает в роли клиента, который отправляет переданное ей сообщение по указанному адресу. Вторая – в роли сервера, который принимает сообщение и отвечает повторением текста полученного сообщения (*Echo Server*). На рис. 2, 3 приведен внешний вид этих приложений.

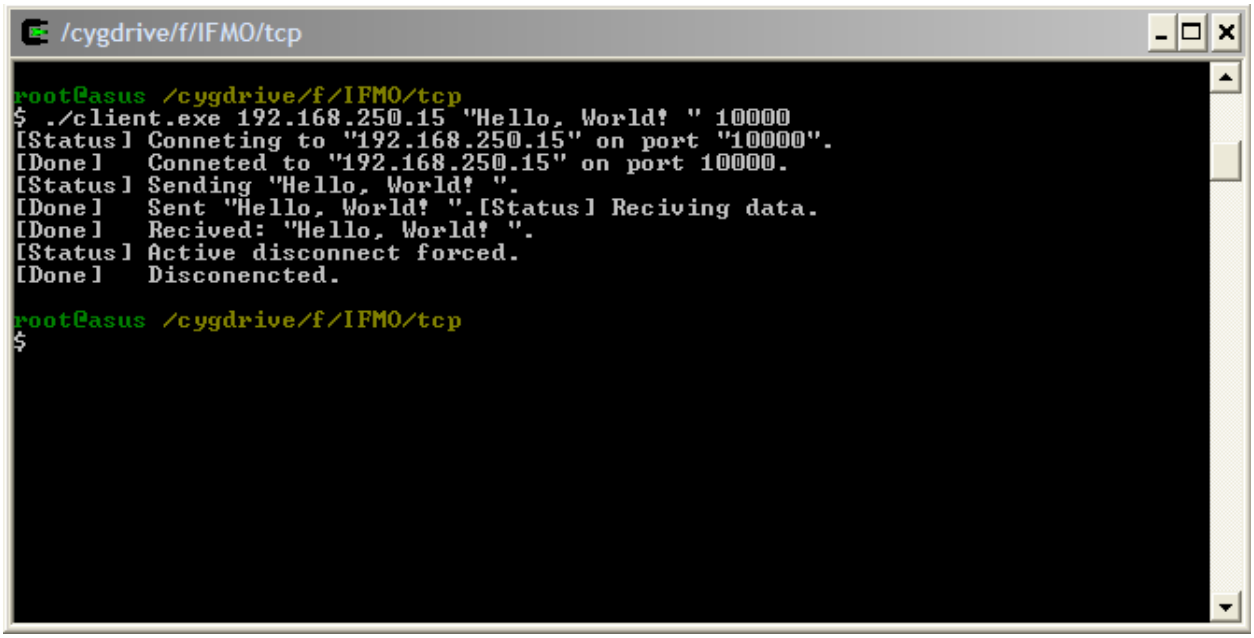

Рис. 2. Внешний вид клиента

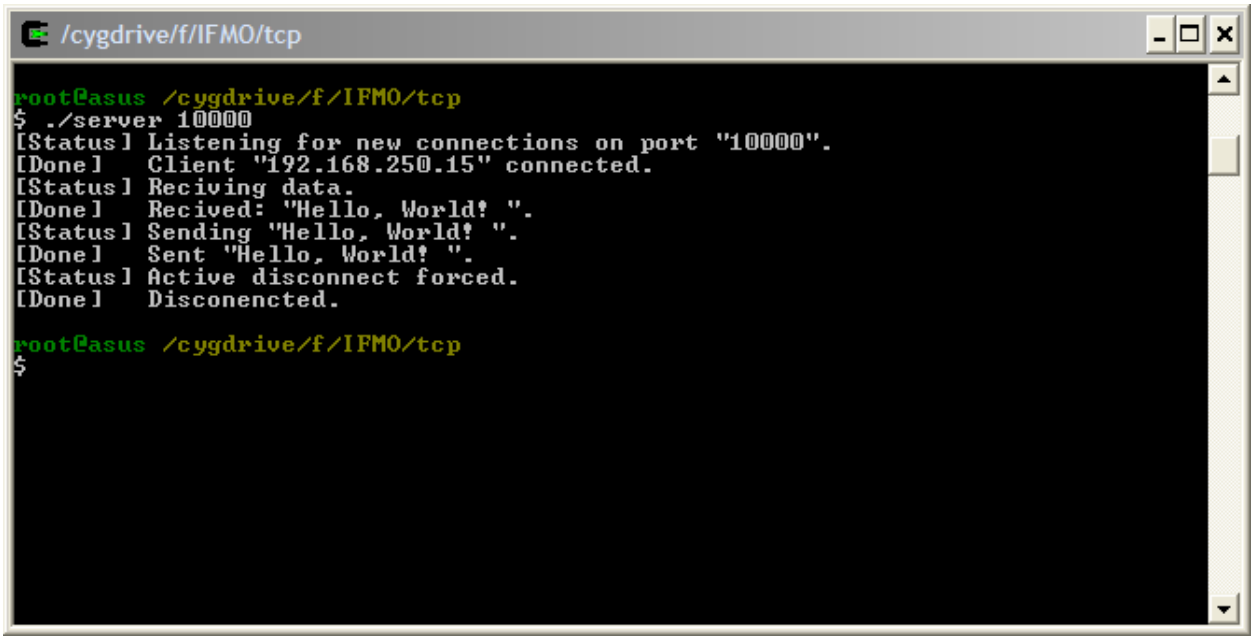

Рис. 3. Внешний вид сервера

В приведенном примере происходит отправка сообщения «Hello, World!» от клиента с адресом 192.168.250.15 серверу с адресом 192.168.250.15. Рассмотрим последовательно, что происходит при отправке сообщения с клиентом и сервером:

- инициализируется сервер

  Сервер:

  ```
  [Status] Listening for new connections on port 10000.
  ```

- клиент инициализирует соединение с сервером

  Клиент:

  ```
  [Status] Connecting to “192.168.250.15” on port 10000.
  ```

- соединение успешно устанавливается

  Сервер:

  ```
  [Done]   Client “192.168.250.15” connected.
  ```

  Клиент:

  ```
  [Done]   Connected to “192.168.250.15” on port 10000.
  ```

- сервер ожидает данных от клиента

  Сервер:

  ```
  [Status] Receiving data.
  ```

- клиент передает данные

  Клиент:

  ```
  [Status] Sending "Hello, World! ".
  ```

- передача успешно завершается

  Сервер:

  ```
  [Done]   Received: "Hello, World! ".
  ```

  Клиент:

  ```
  [Done]   Sent "Hello, World! ".
  ```

- клиент ожидает ответ от сервера

  Клиент:

  ```
  [Status] Receiving data.
  ```

- сервер передает ответ

  Сервер:

  ```
  [Status] Sending "Hello, World! ".
  ```

- передача ответа успешно завершается

  Сервер:

  ```
  [Done]   Sent "Hello, World! ".
  ```

Клиент:

```
[Done]   Received: "Hello, World! ".
```

- клиент и сервер одновременно вызывают окончание соединения

  Сервер, Клиент:

  ```
  [Status] Active disconnect forced.
  ```

- соединение завершается

  Сервер, Клиент:

  ```
  [Done]   Disconnected.
  ```

4. Реализация

Для алгоритмизации и программирования задач логического управления была предложена *SWITCH-технология*, которая называется также «автоматное программирование» или «программирование с явным выделением состояний». Первоначально применявшийся для создания аппаратных схем управления оборудованием, этот подход был применен при решении данной задачи. Кроме того, отметим, что обычно при описании работы протоколов используются автоматы.

Исходя из требований надежности, специфики задачи, связанной с работой в реальном времени программа построена с применением шести автоматов. Каждый автомат реализован в виде функции на языке С.

На рис. 4 – 6 представлена схема взаимодействия автоматов между собой.

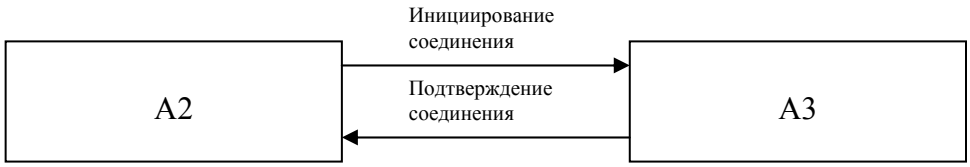

Рис. 4. Схема взаимодействия автоматов, обеспечивающих синхронизацию соединения

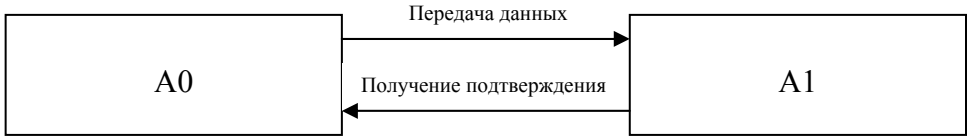

Рис. 5. Схема взаимодействия автоматов, обеспечивающих передачу данных

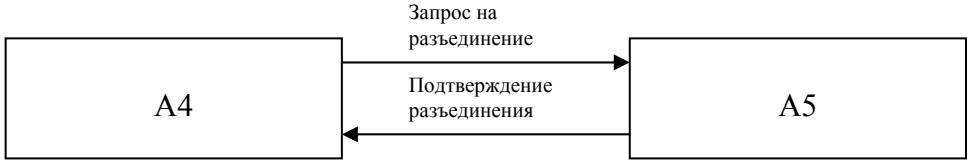

Рис. 6. Схема взаимодействия автоматов, обеспечивающих завершение соединения

5. Автомат «Передача пакета данных» (A0)

Описание автомата

Устойчивая передача пакета данных по сети требует обязательного подтверждения о приеме данных принимающей стороной. Это реализуется следующим методом.

В сеть передается пакет данных и запускается таймер. Если по истечении некоторого времени, от принимающей стороны не получено подтверждения получения, то пакет отправляется повторно. Это повторяется итеративно некоторое количество раз, пока подтверждение не будет получено. В случае если в течение этих попыток оно получено не было, отправляется сообщение о разрыве соединения и возвращается ошибка соединения. Если же подтверждение было получено, то это означает, что передача пакета была успешно завершена.

Также при передаче пакета возможно получение сообщения от принимающей стороны об окончании соединения. После отправки этого сообщения принимающая сторона перестает обрабатывать новые данные. В ответ на это передается подтверждение о согласии на окончание соединения. Это не является ошибкой, однако данные переданы не были.

В случае получения запроса на синхронизацию от принимающей стороны, генерируется ошибка и принимающей стороне передается в ответ сообщение о разрыве соединения. Эта ситуация возможна в случае, если на второй стороне произошла ошибка, и соединение было разорвано, а после этого была произведена попытка повторной его установки.

В случае если на принимающей стороне произошла ошибка, посылается сообщение об этом и соединение разрывается. Передающая сторона обрабатывает это сообщение и разрывает соединение со своей стороны.

Описание состояний автомата

0. Стартовое состояние.
1. Подготовка к передаче данных.
2. Передача пакета данных.
3. Ожидание подтверждения о получении пакета данных.
4. Проверка полученного ответа.
5. Проверка количества оставшихся попыток повторной передачи пакета.
6. Во время передачи пакета произошла ошибка. Невозможно определить, был ли пакет доставлен. Соединение было прервано.

7. Был получен запрос на завершение соединения. Вторая сторона больше не принимает пакетов.
8. Успешное завершение передачи пакета.

Обозначение входных воздействий

x0_1. Из сети получен пакет данных.

x0_2. Полученный пакет является корректным подтверждением приема данных.

x0_3. Полученный пакет является сообщением о разрыве соединения.

x0_4. Полученный пакет является запросом на завершение соединения.

x0_5. Полученный пакет является запросом на установку соединения.

x0_6. Время ожидания ответа превысило максимальное.

x0_7. Количество попыток отправки пакета не превысило максимальное.

x0_8. Произошла ошибка уровня передачи данных.

Обозначение выходных воздействий

z0_1. Обнулить счетчик совершенных попыток передачи пакета.

z0_2. Обнулить таймер.

z0_3. Передать пакет данных.

z0_4. Увеличить счетчик совершенных попыток передачи пакета.

z0_5. Отправить сообщение о разрыве соединения.

z0_6. Отправить подтверждение приема данных.

Схема связей

На рис. 7 приведена схема связей автомата *A0*, описывающая его интерфейс.

Рис. 7. Схема связей автомата *A0*

Граф переходов

На рис. 8 приведен граф переходов автомата *A0*.

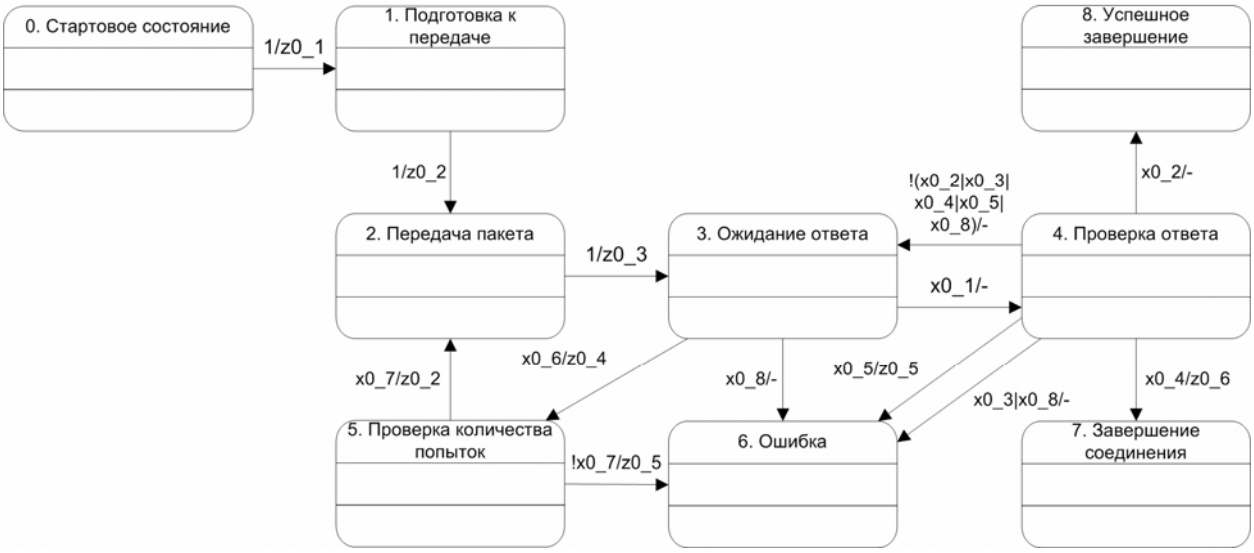

Рис. 8. Граф переходов автомата *A0*

6. Автомат «Прием пакета данных» (A1)

Описание автомата

Получение пакета при устойчивом соединении требует обязательного ответа передающей стороне о получении данных.

Сначала производится проверка на наличие новых данных от низкого уровня передачи. При отсутствии таковых работа заканчивается, возвращая сообщение об отсутствии данных. В случае если получен пакет, но он устарел, отправляется последнее из сообщений о приеме пакета и начинается обработка следующего пакета. В случае если получен новый пакет данных, отправляется подтверждение его получения и работа автомата успешно завершается.

Также при получении пакета может прийти сообщение об окончании соединения. После его отправки передающая сторона перестает обрабатывать новые данные. В ответ на это передается подтверждение о согласии на окончание соединения и возвращается сообщение об окончании соединения.

В случае получения запроса на синхронизацию от принимающей стороны, генерируется ошибка и принимающей стороне передается в ответ сообщение о разрыве соединения. Эта ситуация возможна в случае, если на второй стороне произошла ошибка и соединение было некорректно разорвано, а после этого была произведена попытка повторной его установки.

В случае если на передающей стороне произошла ошибка, она посылает сообщение об этом и разрывает соединение. Принимающая сторона обрабатывает это сообщение и разрывает соединение со своей стороны.

Описание состояний автомата

0. Стартовое состояние.
1. Ожидание получения пакета.
2. Проверка полученного пакета.
3. Проверка на наличие ошибки при передаче подтверждения о получении пакета данных.
4. Проверка на наличие ошибки при передаче подтверждения о получении сообщения о завершении соединения.
5. Пакетов на получение нет.
6. Во время получения пакета произошла ошибка. Соединение было прервано.
7. Был получен запрос на завершение соединения. Вторая сторона больше не принимает пакетов.

8. Пакет был успешно получен.

Обозначение входных воздействий

x1_1. Из сети получен пакет данных.

x1_2. Полученный пакет является корректным следующим по очередности пакетом.

x1_3. Полученный пакет является запросом на завершение соединения.

x1_4. Полученный пакет является сообщением о разрыве соединения.

x1_5. Полученный пакет является запросом на установку соединения.

x1_6. Полученный пакет является дубликатом уже обработанного ранее.

x1_7. Произошла ошибка уровня передачи данных.

Обозначение выходных воздействий

z1_1. Отправить подтверждение приема данных.

z1_2. Отправить сообщение о разрыве соединения.

Схема связей

На рис. 9 приведена схема связей автомата *A1*, описывающая его интерфейс.

Рис. 9. Схема связей автомата *A1*

Граф переходов

На рис. 10 приведен граф переходов автомата *A1*.

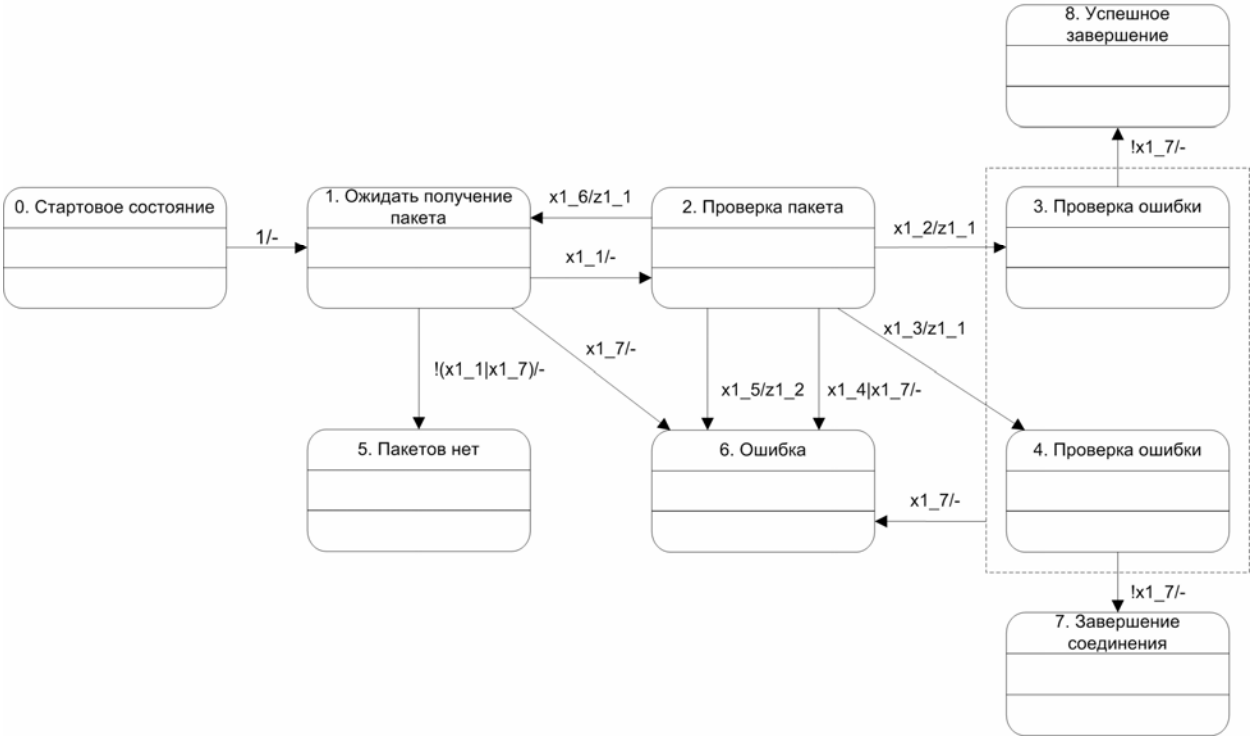

Рис. 10. Граф переходов автомата *A1*

7. Автомат «Соединение со стороны клиента» (A2)

Описание автомата

В стартовом состоянии синхронизированного соединения не существует. Для установки соединения со стороны клиента происходит отправка пакета синхронизации, ожидание ответа на него со стороны сервера и подтверждение ответа сервера.

В случае если серверная сторона ожидала другой пакет (например, если уже было начато соединение, но оно было прервано по какой-либо причине), то она отправляет пакет разрыва соединения, означающий, что клиент должен повторить свою попытку соединения. При получении любого другого ответа (запрос синхронизации, подтверждение приема данных, запрос на окончание соединения, данные) возвращается ошибка.

При успешном завершении соединение переходит в синхронизированный режим и становится возможна передача данных.

Описание состояний автомата

0. Стартовое состояние.
1. Инициировать соединение. Отправка запроса синхронизации.
2. Запрос на синхронизацию послан. Ожидание подтверждения.
3. Получен ответ.
4. Отправлено подтверждение. Проверка на ошибку передачи данных.
5. Во время работы пакета произошла ошибка. Соединение было прервано.
6. Соединение было успешно установлено.

Обозначение входных воздействий

x2_1. Из сети получен пакет данных.

x2_2. Полученный пакет является корректным подтверждением соединения со стороны сервера.

x2_3. Полученный пакет является сообщением о разрыве соединения.

x2_4. Произошла ошибка уровня передачи данных.

Обозначение выходных воздействий

z2_1. Создать сессию.

z2_2. Отправить запрос на установку соединения.

z2_3. Отправить подтверждение приема данных.

Схема связей

На рис. 11 приведена схема связей автомата *A2*, описывающая его интерфейс.

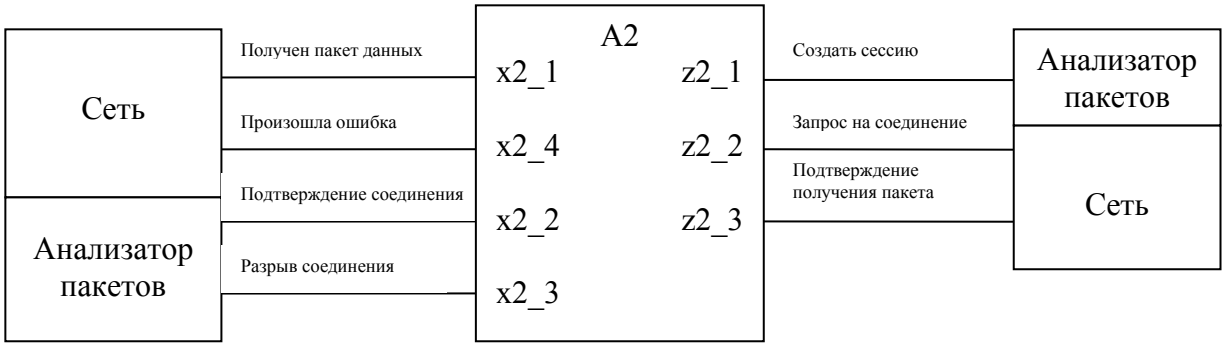

Рис. 11. Схема связей автомата *A2*

Граф переходов

На рис. 12 приведен граф переходов автомата *A2*.

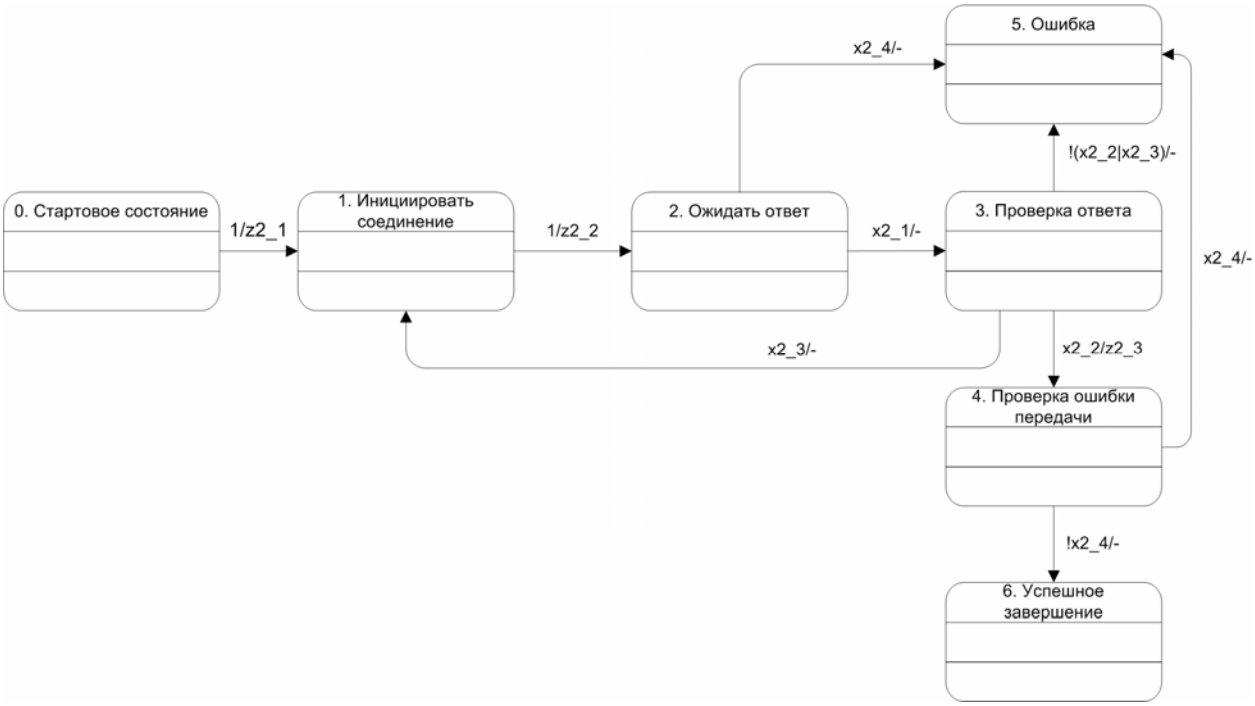

Рис. 12. Граф переходов автомата *A2*

8. Автомат «Соединение со стороны сервера» (A3)

Описание автомата

В стартовом состоянии синхронизированного соединения не существует. Для установки соединения сервер слушает и обрабатывает все полученные им пакеты.

При получении запроса на синхронизацию сервер отвечает на этот запрос подтверждением и ждет от клиентской стороны подтверждения соединения, после чего переходит в режим синхронизированного соединения, когда возможна передача данных.

В случае получения пакета неуместного на данном шаге соединения, сервер отправляет клиенту сообщение о разрыве соединения, означающее, что клиент должен повторить попытку соединения. При любой ошибке сервер переходит в состояние ожидания нового соединения.

Описание состояний автомата

0. Стартовое состояние.
1. Ожидание сообщения от клиента.
2. Получен пакет данных от клиента.
3. Ожидание подтверждения установки соединения от клиента.
4. Проверка полученного подтверждения от клиента.
5. Произошла ошибка. Соединение не было установлено.
6. Соединение было успешно установлено.

Обозначение входных воздействий

x3_1. Из сети получен пакет данных.

x3_2. Полученный пакет является корректным запросом на установку соединения.

x3_3. Полученный пакет является корректным подтверждением приема данных.

x3_4. Полученный пакет является сообщением о разрыве соединения.

x3_5. Произошла ошибка уровня передачи данных.

Обозначение выходных воздействий

z3_1. Создать сессию.

z3_2. Отправить подтверждение установки соединения.

z3_3. Отправить сообщение о разрыве соединения.

Схема связей

На рис. 13 приведена схема связей автомата *A3*, описывающая его интерфейс.

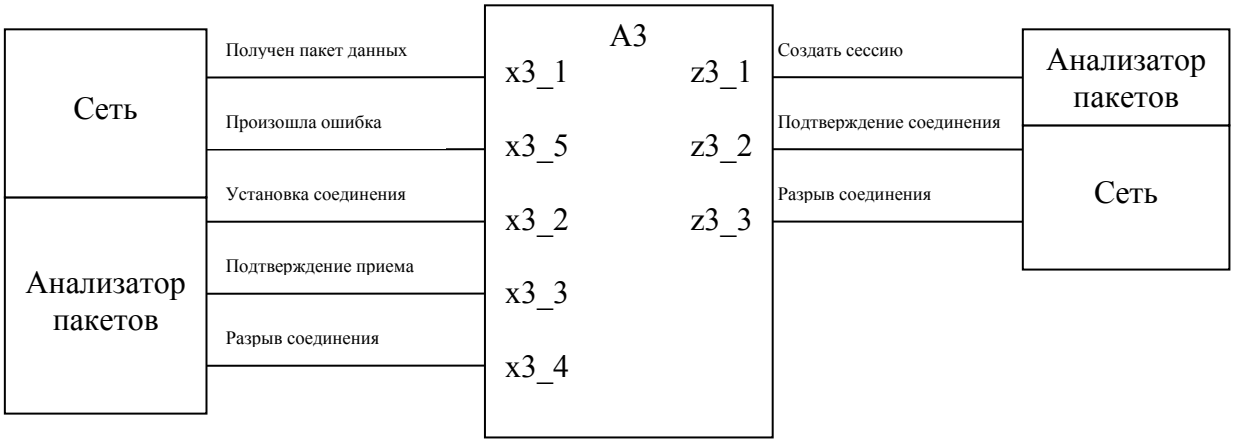

Рис. 13. Схема связей автомата *A3*

Граф переходов

На рис. 14 приведен граф переходов автомата *A3*.

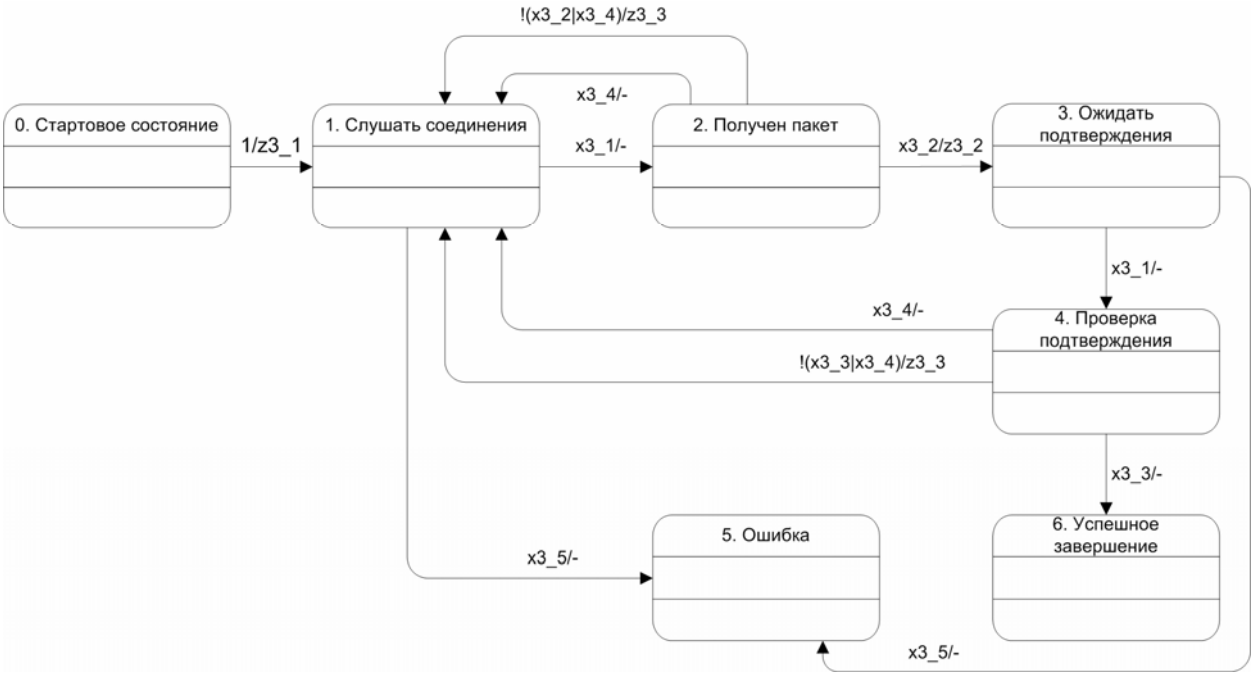

Рис. 14. Граф переходов автомата *A3*

9. Автомат «Активное разъединение» (A4)

Описание автомата

Когда одна из сторон соединения переходит в состояние, в котором она больше не передает данные и не ожидает их от второй стороны, первая сторона вызывает активное разъединение. При этом второй стороне посылается запрос на завершение соединения, который информирует ее о намерении завершить соединение, и ожидается его подтверждение. После этого пассивная сторона сообщает об окончании соединения. В ответ на это посылается сообщение о приеме окончания соединения.

Для того, чтобы можно было гарантировать, что все пакеты этого соединения больше не перемещаются по сети, требуется подождать максимальное время жизни пакета. По истечении этого времени соединение полагается завершенным, и удаляется информация о нем.

Также возможна ситуация, когда обе стороны одновременно перешли в состояние активного разъединения. При такой ситуации автомат получает пакет с запросом на завершение и отвечает на него. Таким же образом поступает вторая сторона и после таймаута, равного максимальному времени жизни пакета, соединение завершается.

Описание состояний автомата

0. Стартовое состояние.
1. Запрос на завершение соединения отправлен. Ожидание подтверждения.
2. Получен пакет. Проверка его типа.
3. Ожидание окончания соединения со второй стороны.
4. Получен пакет данных. Проверка его типа на запрос окончания соединения со второй стороны.
5. Произошло одновременное окончание соединения. Ожидание подтверждения о приеме сообщения об окончании соединения.
6. Получен пакет данных. Проверка на корректность.
7. Соединение было разорвано. Обнуление таймера.
8. Ожидание истечения таймера. Все пакеты игнорируются.
9. Произошла ошибка при завершении соединения.
10. Успешное завершение соединения.

Обозначение входных воздействий

x4_1. Из сети получен пакет данных.

x4_2. Полученный пакет является корректным подтверждением приема данных.

x4_3. Полученный пакет является запросом на завершение соединения.

x4_4. Полученный пакет является сообщением о разрыве соединения.

x4_5. Полученный пакет является корректным запросом на установку соединения.

x4_6. Произошла ошибка уровня передачи данных.

x4_7. Время ожидания до окончания соединения истекло.

Обозначение выходных воздействий

z4_1. Отправить запрос на окончание соединения.

z4_2. Отправить подтверждение завершения соединения.

z4_3. Отправить сообщение о разрыве соединения.

z4_4. Обнулить таймер.

z4_5. Удалить сессию.

Схема связей

На рис. 15 приведена схема связей автомата *A4*, описывающая его интерфейс.

Рис. 15. Схема связей автомата *A4*

Граф переходов

На рис. 16 приведен граф переходов автомата *A4*.

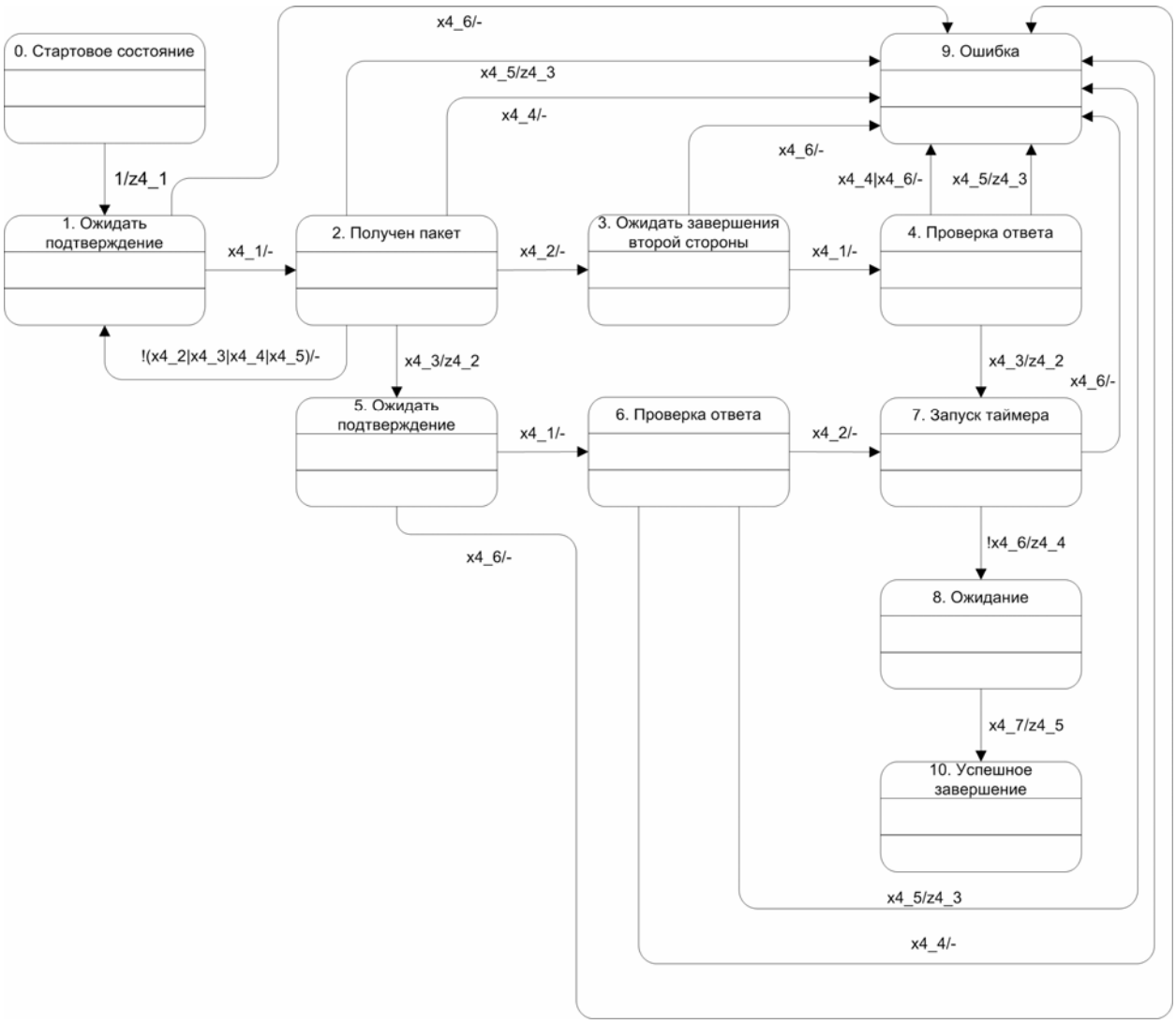

Рис. 16. Граф переходов автомата *A4*

10. Автомат «Пассивное разъединение» (A5)

Описание автомата

Когда одна из сторон соединения переходит в состояние, в котором она больше не передает данные и не ожидает их от второй стороны, первая сторона вызывается активное разъединение. При этом вторая сторона, предварительно завершив все необходимые действия, переходит в режим пассивного разъединения.

Для завершения соединения с пассивной стороны происходит отправка сообщения о завершении соединения и ожидание подтверждения его получения. При получении этого подтверждения соединение полагается завершенным и удаляется информация о нем.

Описание состояний автомата

0. Стартовое состояние.
1. Ожидание подтверждения о завершении соединения.
2. Проверка полученного ответа.
3. Во время окончания соединения произошла ошибка. Соединение было прервано.
4. Корректное завершение соединения.

Обозначение входных воздействий

x5_1. Из сети получен пакет данных.

x5_2. Полученный пакет является корректным подтверждением приема данных.

x5_3. Полученный пакет является сообщением о разрыве соединения.

x5_4. Произошла ошибка уровня передачи данных.

Обозначение выходных воздействий

z5_1. Отправить запрос на завершение соединения.

z5_2. Отправить сообщение о разрыве соединения.

z5_3. Удалить сессию.

Схема связей

На рис. 17 приведена схема связей автомата *A5*, описывающая его интерфейс.

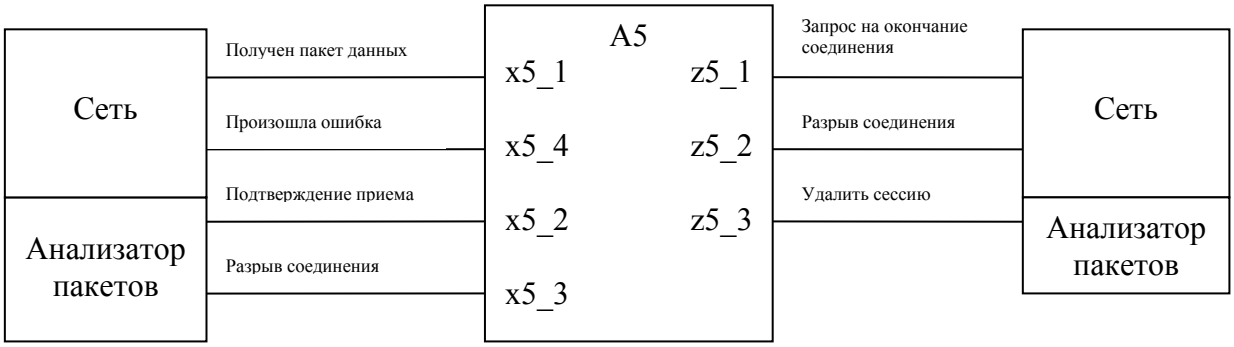

Рис. 17. Схема связей автомата *A5*

Граф переходов

На рис. 18 приведен граф переходов автомата *A5*.

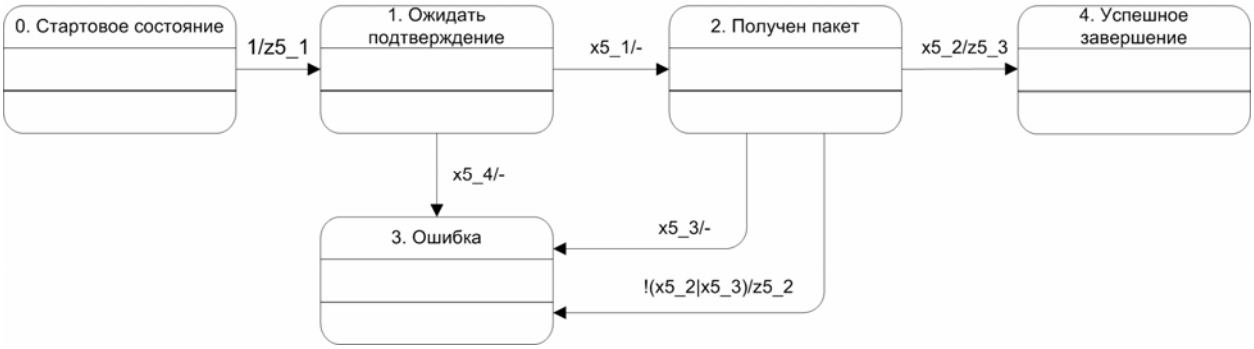

Рис. 18. Граф переходов автомата *A5*

Заключение

Полученная система передачи данных по локальным сетям является по построению надежной. В построенной системе невозможна потеря пакета данных, что позволяет передавать данные с уверенностью в том, что они будут получены.

Обратим внимание, что в спецификации также применяется автомат для описания работы протокола, но как говориться "автомат автомату рознь". Шесть разработанных автоматов совместно с функциями входных и выходных воздействий полностью описывают протокол и позволяют отказаться от словесного описания.

В приложениях приведены протокол работы тестовой программы, исходные тексты модулей, реализующих автоматы, и исходный текст демонстрационной программы для *Linux®/ Cygwin*.

Источники

1. *Агафонов К. А., Порох Д. С., Шалыто А. А.* Реализация протокола "SMTP" на основе SWITCH-технологии. http://is.ifmo.ru/projects/smtp/
2. *RFC 793* Simple Mail Transfer PROTOCOL Protocol. Information Sciences Institute, University of Southern California, September 1981. ftp://ftp.rfc-editor.org/in-notes/rfc821.txt
3. Cygwin project. http://www.cygwin.com/
4. *RFC 793* Transmission Control Protocol. Information Sciences Institute, University of Southern California, August 1982. ftp://ftp.rfc-editor.org/in-notes/rfc793.txt
5. *Танненбаум Э.* Компьютерные сети. СПб.: Питер, 2002.
6. *Куроуз Дж. Ф., Росс К. В.* Компьютерные сети. СПб.: Питер, 2004.
7. *Шалыто А. А.* Новая инициатива в программировании. Движение за открытую проектную документацию //Мир ПК. 2003. № 9. http://is.ifmo.ru/works/open_doc/
8. *Шалыто А. А., Туккель Н. И.* SWITCH-технология – автоматный подход к созданию программного обеспечения «реактивных» систем //Программирование. 2001. № 5. http://is.ifmo.ru/projects/printprj/

Приложение 1. Протокол работы программы

Сервер

```
| Automa A3 is in state 0
< z3_1 invoked by A3
| Automa A3 is in state 1
> x3_1 invoked by A3 with result true
| Automa A3 is in state 2
> x3_2 invoked by A3 with result true
< z3_2 invoked by A3
| Automa A3 is in state 3
> x3_1 invoked by A3 with result true
| Automa A3 is in state 4
> x3_3 invoked by A3 with result true
| Automa A3 is in state 6
| Automa A1 is in state 0
| Automa A1 is in state 1
> x1_1 invoked by A1 with result true
| Automa A1 is in state 2
> x1_2 invoked by A1 with result true
< z1_1 invoked by A1
| Automa A1 is in state 3
> x1_7 invoked by A1 with result false
> x1_7 invoked by A1 with result false
| Automa A1 is in state 8
| Automa A0 is in state 0
< z0_1 invoked by A0
| Automa A0 is in state 1
< z0_2 invoked by A0
| Automa A0 is in state 2
< z0_3 invoked by A0
| Automa A0 is in state 3
> x0_1 invoked by A0 with result false
> x0_6 invoked by A0 with result true
< z0_4 invoked by A0
| Automa A0 is in state 5
> x0_7 invoked by A0 with result true
< z0_2 invoked by A0
| Automa A0 is in state 2
< z0_3 invoked by A0
| Automa A0 is in state 3
> x0_1 invoked by A0 with result true
| Automa A0 is in state 4
> x0_2 invoked by A0 with result true
| Automa A0 is in state 8
| Automa A4 is in state 0
< z4_1 invoked by A4
| Automa A4 is in state 1
> x4_1 invoked by A4 with result true
| Automa A4 is in state 2
> x4_2 invoked by A4 with result false
> x4_3 invoked by A4 with result true
```

```
< z4_2 invoked by A4
| Automa A4 is in state 5
> x4_1 invoked by A4 with result true
| Automa A4 is in state 6
> x4_2 invoked by A4 with result true
| Automa A4 is in state 7
> x4_6 invoked by A4 with result false
> x4_6 invoked by A4 with result false
< z4_3 invoked by A4
| Automa A4 is in state 8
> x4_7 invoked by A4 with result true
< z4_5 invoked by A4
| Automa A4 is in state 10
```

Клиент

```
| Automa A2 is in state 0
< z2_1 invoked by A2
| Automa A2 is in state 1
< z2_2 invoked by A2
| Automa A2 is in state 2
> x2_1 invoked by A2 with result true
| Automa A2 is in state 3
> x2_2 invoked by A2 with result true
< z2_3 invoked by A2
| Automa A2 is in state 4
> x2_4 invoked by A2 with result false
> x2_4 invoked by A2 with result false
| Automa A2 is in state 6
| Automa A0 is in state 0
< z0_1 invoked by A0
| Automa A0 is in state 1
< z0_2 invoked by A0
| Automa A0 is in state 2
< z0_3 invoked by A0
| Automa A0 is in state 3
> x0_1 invoked by A0 with result true
| Automa A0 is in state 4
> x0_2 invoked by A0 with result true
| Automa A0 is in state 8
| Automa A1 is in state 0
| Automa A1 is in state 1
> x1_1 invoked by A1 with result true
| Automa A1 is in state 2
> x1_2 invoked by A1 with result true
< z1_1 invoked by A1
| Automa A1 is in state 3
> x1_7 invoked by A1 with result false
> x1_7 invoked by A1 with result false
| Automa A1 is in state 8
| Automa A4 is in state 0
< z4_1 invoked by A4
| Automa A4 is in state 1
> x4_1 invoked by A4 with result true
| Automa A4 is in state 2
> x4_2 invoked by A4 with result false
> x4_3 invoked by A4 with result false
> x4_4 invoked by A4 with result false
> x4_5 invoked by A4 with result false
> x4_2 invoked by A4 with result false
> x4_3 invoked by A4 with result false
> x4_4 invoked by A4 with result false
> x4_5 invoked by A4 with result false
| Automa A4 is in state 1
> x4_1 invoked by A4 with result true
| Automa A4 is in state 2
> x4_2 invoked by A4 with result false
```

```
> x4_3 invoked by A4 with result true
< z4_2 invoked by A4
| Automa A4 is in state 5
> x4_1 invoked by A4 with result true
| Automa A4 is in state 6
> x4_2 invoked by A4 with result true
| Automa A4 is in state 7
> x4_6 invoked by A4 with result false
> x4_6 invoked by A4 with result false
< z4_3 invoked by A4
| Automa A4 is in state 8
> x4_7 invoked by A4 with result true
< z4_5 invoked by A4
```

Приложение 2. Исходные тексты

auto.h

```
#ifndef __AUTO_H_
#define __AUTO_H_

#define bool  int
#define true  (1)
#define false (0)

#define LOG

/* Macroses for log output */
#ifdef LOG
#include "stdio.h"
#define PRINT_LEVEL { for(int l = 0; l < log_level; l++) \
                    { printf("\t"); }}
#define LOG_AUTO_ENTER(a) { log_stack[log_stack_pos++] = a; \
                          log_level++; PRINT_LEVEL; \
           printf("| Automa A%i is in state %i\n", a, y##a);}
#define LOG_AUTO_EXIT(a) { log_stack_pos--; log_level--;}
#define LOG_AUTO_CHANGE(a) { PRINT_LEVEL; \
       printf("| Automa A%i switched to state %i\n", a, y##a);}
#define LOG_X(a, c) { int res = (c); PRINT_LEVEL; \
     printf("> %s invoked by A%i with result %s\n", #a, \
     log_stack[log_stack_pos - 1], (res == 0) ? "false" : \
     "true"); return res;}
#define LOG_Z(a) { PRINT_LEVEL; \
      printf("< %s invoked by A%i\n", \
      #a, log_stack[log_stack_pos - 1]);}
#define LOG_MAX_STACK 16
#else
#define PRINT_LEVEL
#define LOG_AUTO_ENTER(a)
#define LOG_AUTO_EXIT(a)
#define LOG_AUTO_CHANGE(a)
#define LOG_X(a, c) return (c)
#define LOG_Z(a)
#endif

/* Automata */
void A0(void);
void A1(void);
void A2(void);
void A3(void);
void A4(void);
void A5(void);


/* Automata state */
extern int y0;
extern int y1;
```

```
extern int y2;
extern int y3;
extern int y4;
extern int y5;

/* Automata 0 events */
bool x0_1(void);
bool x0_2(void);
bool x0_3(void);
bool x0_4(void);
bool x0_5(void);
bool x0_6(void);
bool x0_7(void);
bool x0_8(void);

/* Automata 1 events */
bool x1_1(void);
bool x1_2(void);
bool x1_3(void);
bool x1_4(void);
bool x1_5(void);
bool x1_6(void);
bool x1_7(void);

/* Automata 2 events */
bool x2_1(void);
bool x2_2(void);
bool x2_3(void);
bool x2_4(void);

/* Automata 3 events */
bool x3_1(void);
bool x3_2(void);
bool x3_3(void);
bool x3_4(void);
bool x3_5(void);

/* Automata 4 events */
bool x4_1(void);
bool x4_2(void);
bool x4_3(void);
bool x4_4(void);
bool x4_5(void);
bool x4_6(void);
bool x4_7(void);

/* Automata 5 events */
bool x5_1(void);
bool x5_2(void);
bool x5_3(void);
bool x5_4(void);

/* Automata 0 actions */
```

```
void z0_1(void);
void z0_2(void);
void z0_3(void);
void z0_4(void);
void z0_5(void);
void z0_6(void);

/* Automata 1 actions */
void z1_1(void);
void z1_2(void);

/* Automata 2 actions */
void z2_1(void);
void z2_2(void);
void z2_3(void);

/* Automata 3 actions */
void z3_1(void);
void z3_2(void);
void z3_3(void);

/* Automata 4 actions */
void z4_1(void);
void z4_2(void);
void z4_3(void);
void z4_4(void);
void z4_5(void);

/* Automata 5 actions */
void z5_1(void);
void z5_2(void);
void z5_3(void);

#ifdef LOG
extern int log_level;
extern int log_stack[LOG_MAX_STACK];
extern int log_stack_pos;
#endif

#endif
```

supp.h

```
#ifndef __SUPP_H__
#define __SUPP_H__

/* Data packet size */
#define DATA_SIZE 1024

/* Flags */
#define FLAG_SYN 1
#define FLAG_FIN 2
#define FLAG_ACK 4
#define FLAG_RST 8

/* Data packet */
typedef struct
{
    unsigned char from[4];
    unsigned char to[4];
    unsigned int from_port;
    unsigned int to_port;
    unsigned long id1, id2;
    unsigned int length;
    unsigned long flags;
    unsigned char data[DATA_SIZE + 1];
} PACKET;

/* Packet size */
#define PACKET_SIZE sizeof(PACKET)

/* Network timeout */
#define NET_TIMEOUT 1000

/* Network retries */
#define NET_RETRIES 10
```

```
/* Sleep time at connection end */
#define FINAL_SLEEP 2

/* Retries counter */
extern int retries;

/* Timer counter */
extern int timer;

/* Send and recive packets */
extern PACKET snd, rcv;

/* Data to be sent */
extern char data[DATA_SIZE];

/* Length of data to send */
extern int data_len;

/* Error number */
extern bool error;

/* Is connection timed out */
extern bool timedout;

/* Is packet outtimed */
extern bool old_pack;

/* Port number */
extern unsigned int port;

/* Server address */
extern char *to_adr;

/* Initialize low level connection */
bool init(void);
```

```
/* Close low level connection */
bool end(void);

/* Low level listen for connection */
bool listen(void);

/* Low level send packet */
bool send(void);

/* Low level receive packet */
bool recive(void);

/* Low level check for new packet */
bool check_data(void);

/* Low level wait for new packet */
bool wait_data(void);

/* Low level send RST packet */
void send_rst(void);

/* Low level send SYN packet */
void send_syn(void);

/* Low level send SYN ACK packet */
void send_syn_ack(void);

/* Low level send FIN packet */
void send_fin(void);

/* Low level send ACK packet */
void send_ack(void);

/* Low level send data packet */
void send_data(void);
```

```
#endif
```

a0.cpp

```
#include "auto.h"

void A0(void)
{
    LOG_AUTO_ENTER(0);

    switch(y0)
    {
        case 0:
        {
            z0_1();
            y0 = 1;
            break;
        }

        case 1:
        {
            z0_2();
            y0 = 2;
            break;
        }

        case 2:
        {
            z0_3();
            y0 = 3;
            break;
        }

        case 3:
        {
            if (x0_1())
            {
                y0 = 4;
                break;
            }

            if (x0_6())
            {
                z0_4();
                y0 = 5;
                break;
            }

            if (x0_8())
            {
                y0 = 6;
                break;
            }

            break;
```

```
}

case 4:
{
    if (x0_2())
    {
        y0 = 8;
        break;
    }

    if (x0_3() || x0_8())
    {
        y0 = 6;
        break;
    }

    if (x0_4())
    {
        z0_6();
        y0 = 7;
        break;
    }

    if (x0_5())
    {
        z0_5();
        y0 = 6;
        break;
    }

    if (!(x0_2() || x0_3() || x0_4() ||
          x0_5() || x0_8()))
    {
        y0 = 3;
        break;
    }

    break;
}

case 5:
{
    if(x0_7())
    {
        z0_2();
        y0 = 2;
        break;
    }

    if(!x0_7())
    {
        z0_5();
        y0 = 6;
```

```
                break;
            }

            break;
        }

        case 6:
        {
            break;
        }

        case 7:
        {
            break;
        }

        case 8:
        {
            break;
        }
    }

    LOG_AUTO_EXIT(0);
}
```

a1.cpp

```
#include "auto.h"

void A1(void)
{
    LOG_AUTO_ENTER(1);

    switch(y1)
    {
        case 0:
        {
            y1 = 1;
            break;
        }

        case 1:
        {
            if (x1_1())
            {
                y1 = 2;
                break;
            }

            if (x1_7())
            {
                y1 = 6;
                break;
            }

            if (1 /*|| !(x1_1() || x1_7())*/)
            {
                y1 = 5;
                break;
            }

            break;
        }

        case 2:
        {
            if (x1_2())
            {
                z1_1();
                y1 = 3;
                break;
            }

            if (x1_3())
            {
                z1_1();
                y1 = 4;
                break;
```

```
    }

    if (x1_4() || x1_7())
    {
        y1 = 6;
        break;
    }

    if (x1_5())
    {
        z1_2();
        y1 = 6;
        break;
    }

    break;
}

case 3:
{
    if (x1_7())
    {
        y1 = 6;
        break;
    }

    if (!x1_7())
    {
        y1 = 8;
        break;
    }

    break;
}

case 4:
{
    if (x1_7())
    {
        y1 = 6;
        break;
    }

    if (!x1_7())
    {
        y1 = 7;
        break;
    }

    break;
}

case 5:
```

```
        {
            break;
        }

        case 6:
        {
            break;
        }

        case 7:
        {
            break;
        }

        case 8:
        {
            break;
        }
    }

    LOG_AUTO_EXIT(1);
}
```

a2.cpp

```
#include "auto.h"

void A2(void)
{
    LOG_AUTO_ENTER(2);

    switch(y2)
    {
        case 0:
        {
            z2_1();
            y2 = 1;
            break;
        }

        case 1:
        {
            z2_2();
            y2 = 2;
            break;
        }

        case 2:
        {
            if (x2_1())
            {
                y2 = 3;
                break;
            }

            if (x2_4())
            {
                y2 = 5;
                break;
            }

            break;
        }

        case 3:
        {
            if (x2_2())
            {
                z2_3();
                y2 = 4;
                break;
            }

            if (x2_3())
            {
                y2 = 1;
```

```
                break;
            }

            if (!(x2_2() || x2_3()))
            {
                y2 = 5;
                break;
            }

            break;
        }

        case 4:
        {
            if (x2_4())
            {
                y2 = 5;
                break;
            }

            if (!x2_4())
            {
                y2 = 6;
                break;
            }

            break;
        }

        case 5:
        {
            break;
        }

        case 6:
        {
            break;
        }
    }

    LOG_AUTO_EXIT(2);
}
```

a3.cpp

```
#include "auto.h"

void A3(void)
{
    LOG_AUTO_ENTER(3);

    switch(y3)
    {
        case 0:
        {
            z3_1();
            y3 = 1;
            break;
        }

        case 1:
        {
            if (x3_1())
            {
                y3 = 2;
                break;
            }

            if (x3_5())
            {
                y3 = 5;
                break;
            }

            break;
        }

        case 2:
        {
            if (x3_2())
            {
                z3_2();
                y3 = 3;
                break;
            }

            if (x3_4())
            {
                y3 = 1;
                break;
            }

            if (!(x3_2() || x3_4()))
            {
                z3_3();
                y3 = 1;
```

```
            break;
        }

        break;
    }

    case 3:
    {
        if (x3_1())
        {
            y3 = 4;
            break;
        }

        if (x3_5())
        {
            y3 = 5;
            break;
        }

        break;
    }

    case 4:
    {
        if (x3_3())
        {
            y3 = 6;
            break;
        }

        if (x3_4())
        {
            y3 = 1;
            break;
        }

        if (!(x3_3() || x3_4()))
        {
            z3_3();
            y3 = 1;
            break;
        }

        break;
    }

    case 5:
    {
        break;
    }

    case 6:
```

```
            {
                break;
            }
        }

        LOG_AUTO_EXIT(3);
    }
```

a4.cpp

```
#include "auto.h"

void A4(void)
{
    LOG_AUTO_ENTER(4);

    switch(y4)
    {
        case 0:
        {
            z4_1();
            y4 = 1;
            break;
        }

        case 1:
        {
            if (x4_1())
            {
                y4 = 2;
                break;
            }

            if (x4_6())
            {
                y4 = 9;
                break;
            }

            break;
        }

        case 2:
        {
            if (x4_2())
            {
                y4 = 3;
                break;
            }

            if (x4_3())
            {
                z4_2();
                y4 = 5;
                break;
            }

            if (x4_4())
            {
                y4 = 9;
                break;
```

```
    }

    if (x4_5())
    {
        z4_3();
        y4 = 9;
        break;
    }

    if (!(x4_2() || x4_3() || x4_4() || x4_5()))
    {
       y4 = 1;
       break;
    }

    break;
}

case 3:
{
    if (x4_1())
    {
        y4 = 4;
        break;
    }

    if (x4_6())
    {
        y4 = 9;
        break;
    }

    break;
}

case 4:
{
    if (x4_3())
    {
        z4_2();
        y4 = 7;
        break;
    }

    if (x4_4() || x4_6())
    {
        y4 = 9;
        break;
    }

    if (x4_5())
    {
        z4_3();
```

```
            y4 = 9;
            break;
        }

        break;
    }

    case 5:
    {
        if (x4_1())
        {
            y4 = 6;
            break;
        }

        if (x4_6())
        {
            y4 = 9;
            break;
        }

        break;
    }

    case 6:
    {
        if (x4_2())
        {
            y4 = 7;
            break;
        }

        if (x4_4())
        {
            y4 = 9;
            break;
        }

        if (x4_5())
        {
            z4_3();
            y4 = 9;
            break;
        }

        break;
    }

    case 7:
    {
        if (x4_6())
        {
            y4 = 9;
```

```
                break;
            }

            if (!x4_6())
            {
                z4_4();
                y4 = 8;
                break;
            }

            break;
        }

        case 8:
        {
            if (x4_7())
            {
                z4_5();
                y4 = 10;
                break;
            }

            break;
        }

        case 9:
        {
            break;
        }

        case 10:
        {
            break;
        }
    }

    LOG_AUTO_EXIT(4);
}
```

a5.cpp

```
#include "auto.h"

void A5(void)
{
    LOG_AUTO_ENTER(5);

    switch(y5)
    {
        case 0:
        {
            z5_1();
            y5 = 1;
            break;
        }

        case 1:
        {
            if (x5_1())
            {
                y5 = 2;
                break;
            }

            if (x5_4())
            {
                y5 = 3;
                break;
            }

            break;
        }

        case 2:
        {
            if (x5_2())
            {
                z5_3();
                y5 = 4;
                break;
            }

            if (x5_3())
            {
                y5 = 3;
                break;
            }

            if (!(x5_2() || x5_3()))
            {
                z5_2();
                y5 = 3;
```

```
                break;
            }

            break;
        }

        case 3:
        {
            break;
        }

        case 4:
        {
            break;
        }
    }

    LOG_AUTO_EXIT(5);
}
```

auto.cpp

```
#include "auto.h"

int y0;
int y1;
int y2;
int y3;
int y4;
int y5;

#ifdef LOG
int log_level;
int log_stack[LOG_MAX_STACK];
int log_stack_pos;
#endif
```

x.cpp

```
#include "auto.h"
#include "supp.h"

#include <unistd.h>

bool x0_1(void)
{
    LOG_X(x0_1, wait_data());
}

bool x0_2(void)
{
    LOG_X(x0_2, rcv.flags == FLAG_ACK);
}

bool x0_3(void)
{
    LOG_X(x0_3, rcv.flags == FLAG_RST);
}

bool x0_4(void)
{
    LOG_X(x0_4, rcv.flags == FLAG_FIN);
}

bool x0_5(void)
{
    LOG_X(x0_5, rcv.flags == FLAG_SYN);
}

bool x0_6(void)
{
    LOG_X(x0_6, timedout);
}
bool x0_7(void)
{
    LOG_X(x0_7, retries <= NET_RETRIES);
}

bool x0_8(void)
{
    LOG_X(x0_8, error);
}

bool x1_1(void)
{
    LOG_X(x1_1, check_data());
}

bool x1_2(void)
{
```

```
    LOG_X(x1_2, rcv.flags == 0);
}

bool x1_3(void)
{
    LOG_X(x1_3, rcv.flags == FLAG_FIN);
}

bool x1_4(void)
{
    LOG_X(x1_4, rcv.flags == FLAG_RST);
}

bool x1_5(void)
{
    LOG_X(x1_5, rcv.flags == FLAG_SYN);
}

bool x1_6(void)
{
    LOG_X(x1_6, old_pack);
}

bool x1_7(void)
{
    LOG_X(x1_7, error);
}

bool x2_1(void)
{
    LOG_X(x2_1, recive());
}
bool x2_2(void)
{
    LOG_X(x2_2, rcv.flags == (FLAG_ACK | FLAG_SYN));
}

bool x2_3(void)
{
    LOG_X(x2_3, rcv.flags == FLAG_RST);
}

bool x2_4(void)
{
    LOG_X(x2_4, error);
}

bool x3_1(void)
{
    LOG_X(x3_1, recive());
}

bool x3_2(void)
```

```
{
    LOG_X(x3_2, rcv.flags == FLAG_SYN);
}

bool x3_3(void)
{
    LOG_X(x3_3, rcv.flags == FLAG_ACK);
}

bool x3_4(void)
{
    LOG_X(x3_4, rcv.flags == FLAG_RST);
}

bool x3_5(void)
{
    LOG_X(x3_5, error);
}

bool x4_1(void)
{
    LOG_X(x4_1, recive());
}

bool x4_2(void)
{
    LOG_X(x4_2, rcv.flags == FLAG_ACK);
}

bool x4_3(void)
{
    LOG_X(x4_3, rcv.flags == FLAG_FIN);
}

bool x4_4(void)
{
    LOG_X(x4_4, rcv.flags == FLAG_RST);
}

bool x4_5(void)
{
    LOG_X(x4_5, rcv.flags == FLAG_SYN);
}

bool x4_6(void)
{
    LOG_X(x4_6, error);
}

bool x4_7(void)
{
    sleep(FINAL_SLEEP);
    LOG_X(x4_7, true);
```

```
}

bool x5_1(void)
{
    LOG_X(x5_1, recive());
}

bool x5_2(void)
{
    LOG_X(x5_2, rcv.flags == FLAG_ACK);
}

bool x5_3(void)
{
    LOG_X(x5_3, rcv.flags == FLAG_RST);
}

bool x5_4(void)
{
    LOG_X(x5_4, error);
}
```

z.cpp

```
#include "auto.h"
#include "supp.h"

void z0_1(void)
{
    retries = 0;
    LOG_Z(z0_1);
}

void z0_2(void)
{
    timer = 0;
    LOG_Z(z0_2);
}

void z0_3(void)
{
    send_data();
    LOG_Z(z0_3);
}
void z0_4(void)
{
    retries++;
    LOG_Z(z0_4);
}

void z0_5(void)
{
    send_rst();
    LOG_Z(z0_5);
}

void z0_6(void)
{
    send_ack();
    LOG_Z(z0_6);
}

void z1_1(void)
{
    send_ack();
    LOG_Z(z1_1);
}

void z1_2(void)
{
    send_rst();
    LOG_Z(z1_2);
}

void z2_1(void)
```

```
{
    init();
    LOG_Z(z2_1);
}

void z2_2(void)
{
    send_syn();
    LOG_Z(z2_2);
}

void z2_3(void)
{
    send_ack();
    LOG_Z(z2_3);
}

void z3_1(void)
{
    init();
    listen();
    LOG_Z(z3_1);
}

void z3_2(void)
{
    send_syn_ack();
    LOG_Z(z3_2);
}

void z3_3(void)
{
    send_rst();
    LOG_Z(z3_3);
}

void z4_1(void)
{
    send_fin();
    LOG_Z(z4_1);
}

void z4_2(void)
{
    send_ack();
    LOG_Z(z4_2);
}

void z4_3(void)
{
    send_rst();
    LOG_Z(z4_3);
}
```

```
void z4_4(void)
{
    timer = 0;
    LOG_Z(z4_3);
}

void z4_5(void)
{
    end();
    LOG_Z(z4_5);
}

void z5_1(void)
{
    send_fin();
    LOG_Z(z5_1);
}

void z5_2(void)
{
    send_rst();
    LOG_Z(z5_2);
}
void z5_3(void)
{
    end();
    LOG_Z(z5_3);
}
```

supp.cpp

```
#include "auto.h"
#include "supp.h"

#include <fcntl.h>
#include <stdio.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <netinet/in.h>
#include <sys/poll.h>

int retries;

int timer;

PACKET snd, rcv;

char data[DATA_SIZE];

int data_len = 0;

bool error = true;

bool timedout;

bool old_pack;

unsigned int port;

int sock = -1;

struct sockaddr_in self;
struct sockaddr_in other;

char *to_adr;

bool init(void)
{
    snd.to_port = port;
    snd.from_port = port;
    error = false;
    if ((sock = socket(PF_INET, SOCK_DGRAM, IPPROTO_IP)) < 0)
        error = true;
}

bool end(void)
{
    close(sock);
}
```

```
bool listen(void)
{
    memset(&self, 0, sizeof(self));
    self.sin_family = AF_INET;
    self.sin_addr.s_addr = htonl(INADDR_ANY);
    self.sin_port = htons(port);

    if (bind(sock, (struct sockaddr *) &self, sizeof(self)) < 0)
        error = true;

    return !error;
}

bool send(void)
{
    memcpy(snd.data, data, DATA_SIZE);
    snd.length = data_len;
    snd.id1++;
    snd.id2 = rcv.id1;
    if (sendto(sock, (void *)(&snd), PACKET_SIZE, 0,
               (struct sockaddr *) &other,
               sizeof(other)) != PACKET_SIZE)
    {
        error = true;
    }

    return error;
}

bool check_data(void)
{
    pollfd p;
    memset(&p, 0, sizeof(p));
    p.fd = sock;
    p.events = POLLIN;

    if (poll(&p, 1, 0) > 0)
        return recive();

    return false;
}

bool wait_data(void)
{
    pollfd p;
    memset(&p, 0, sizeof(p));
    p.fd = sock;
    p.events = POLLIN;
    timedout = false;

    if (poll(&p, 1, NET_TIMEOUT) > 0)
        return recive();
```

```
    timedout = true;
    return false;
}

bool recive(void)
{
    int clientlen = sizeof(other);
    if (recvfrom(sock, (void *)(&rcv), sizeof(rcv), 0,
                 (struct sockaddr *) &other,
                 &clientlen) < 0)
    {
        error = true;
        return !error;
    }

    memcpy(data, rcv.data, DATA_SIZE);
    data_len = rcv.length;

//    if (rcv.id1 != snd.id2)
//      error = true;

    return !error;
}

void send_rst(void)
{
    snd.flags = FLAG_RST;
    send();
}

void send_syn(void)
{
    other.sin_family = AF_INET;
    other.sin_addr.s_addr = inet_addr(to_adr);
    other.sin_port = htons(port);

    snd.flags = FLAG_SYN;
    snd.id1 = 100;
    send();
}

void send_syn_ack(void)
{
    snd.flags = (FLAG_SYN | FLAG_ACK);
    snd.id1 = 100;
    to_adr = inet_ntoa(other.sin_addr);
    send();
}

void send_fin(void)
{
    snd.flags = FLAG_FIN;
```

```
    send();
}

void send_ack(void)
{
    snd.flags = FLAG_ACK;
    send();
}

void send_data(void)
{
    snd.flags = 0;
    send();
}
```

server.cpp

```
#include "stdlib.h"
#include "string.h"
#include "stdio.h"

#include "auto.h"
#include "supp.h"

int main(int argc, char *argv[])
{
    if (argc != 2)
    {
      printf("USAGE: %s <port>\n", argv[0]);
      return 1;
    }

    // Listen
    printf("[Status] Listening for new connections "
           "on port \"%s\".\n", argv[1]);
    port = atoi(argv[1]);
    y3 = 0;
    while (y3 != 5 && y3 != 6)
        A3();
    A3();
    if (y3 == 5)
    {
        printf("[Error]  Connection error.\n");
        return 1;
    }
    printf("[Done]   Client \"%s\" connected.\n", to_adr);

    // Recive
    printf("[Status] Reciving data.\n");
    y1 = 0;
    while (y1 != 6 && y1 != 7 && y1 != 8)
        A1();
    A1();
    if (y1 == 5)
        printf("[Status] No data to recieve.\n");
    else if (y1 == 6)
    {
        printf("[Error]  Recive error.\n");
        return 1;
    }
    else if (y1 == 7)
        goto passive_diconnect;
    else
        printf("[Done]   Recived: \"%s\".\n", data);

    // Send
    printf("[Status] Sending \"%s\".\n", data);
    y0 = 0;
```

```
    while (y0 != 6 && y0 != 7 && y0 != 8)
        A0();
    A0();
    if (y0 == 6)
    {
        printf("[Error]  Send error.\n");
        return 1;
    }
    if (y0 == 7)
        goto passive_diconnect;
    printf("[Done]   Sent \"%s\".\n", data);

    // Disconnect
    acive_diconnect:
    printf("[Status] Active disconnect forced.\n");
    y4 = 0;
    while (y4 != 9 && y4 != 10)
        A4();
    A4();
    goto end_diconnect;

    passive_diconnect:
    printf("[Status] Got disconnect signal.\n");
    y5 = 0;
    while (y5 != 3 && y5 != 4)
        A5();
    A5();
    goto end_diconnect;

    end_diconnect:
    printf("[Done]   Disconencted.\n");

    return 0;
}
```

client.cpp

```
#include "stdlib.h"
#include "string.h"
#include "stdio.h"
#include "unistd.h"

#include "auto.h"
#include "supp.h"

int main(int argc, char *argv[])
{
    if (argc != 4)
    {
      printf("USAGE: %s <server_ip> <word> <port>\n", argv[0]);
      return 1;
    }

    // Connect
    printf("[Status] Conneting to \"%s\" on port "
           "\"%s\".\n", argv[1], argv[3]);
    to_adr = argv[1];
    port = atoi(argv[3]);
    y2 = 0;
    while (y2 != 5 && y2 != 6)
        A2();
    A2();
    if (y2 == 5)
    {
        printf("[Error]  Connection error.\n");
        return 1;
    }
    printf("[Done]   Conneted to \"%s\" on port %s.\n",
           argv[1], argv[3]);

    // Send
    printf("[Status] Sending \"%s\".\n", argv[2]);
    data_len = sizeof(argv[2]);
    strcpy(data, argv[2]);
    y0 = 0;
    while (y0 != 6 && y0 != 7 && y0 != 8)
        A0();
    A0();
    if (y0 == 6)
    {
        printf("[Error]  Send error.\n");
        return 1;
    }
    if (y0 == 7)
        goto passive_diconnect;

    printf("[Done]   Sent \"%s\".", data);
```

```
    // Recive
    sleep(1);
    printf("[Status] Reciving data.\n");
    y1 = 0;
    while (y1 != 5 && y1 != 6 && y1 != 7 && y1 != 8)
        A1();
    A1();
    if (y1 == 5)
        printf("[Status] No data to recieve.\n");
    else if (y1 == 6)
    {
        printf("[Error]  Recive error.\n");
        return 1;
    }
    else if (y1 == 7)
        goto passive_diconnect;
    else
        printf("[Done]   Recived: \"%s\".\n", data);

    // Disconnect
    acive_diconnect:
    printf("[Status] Active disconnect forced.\n");
    y4 = 0;
    while (y4 != 9 && y4 != 10)
        A4();
    goto end_diconnect;

    passive_diconnect:
    printf("[Status] Got disconnect signal.\n");
    y5 = 0;
    while (y5 != 3 && y5 != 4)
        A5();
    goto end_diconnect;

    end_diconnect:
    printf("[Done]   Disconencted.\n");

    return 0;
}
```