Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ ИУ «Информатика и системы управления»

КАФЕДРА ИУ-7 «Программное обеспечение ЭВМ и информационные технологии»

ЛАБОРАТОРНАЯ РАБОТА №3

«Моделирование сетевого протокола»

Группа: ИУ7-41М

Студент: Дубовицкая Ольга Николаевна

Дисциплина: Математические основы верификации ПО

Преподаватель: Кузнецова Ольга Владимировна

Москва, 2025 г.

Задание

Выбирается любой сетевой протокол и описывается упрощённая модель этого протокола. Необязательно полностью все поля, например, IP-пакетов.

Отчёт должен содержать:

- описание протокола и принятые допущения,
- описываемые uml-sequence при работе,
- модель протокола,
- логи SPIN, демонстрирующие отправку/получение данных: пакетов, HTML-документов и т.д.,
- выводы по результатам работы.

Описание протокола и принятые допущения

Для реализации был выбран протокол сетевого уровня – Протокол управления передачей (TCP или Transmission Control Protocol). Это протокол, который обеспечивает надёжную передачу данных между двумя узлами сети (в данном случае будет рассмотрено взаимодействие клиентского и серверного приложений).

Описываемая модель протокола включает в себя следующие **этапы**:

1. ***установка соединения*** – протокол начинается с трёхстороннего рукопожатия (three-way handshake):
    - клиент отправляет сообщение SYN серверу для инициализации соединения,
    - сервер отвечает сообщением SYN_ACK, подтверждая запрос клиента,
    - клиент отправляет сообщение ACK, подтверждая получение SYN_ACK;
2. ***передача данных***:
    - после установления соединения клиент отправляет данные частями (в данном случае это 3 чанка, количество которых предварительно задано константой TOTAL_CHUNKS),

- каждое передаваемое сообщение имеет тип DATA и в поле полезной нагрузки (payload) содержит номер чанка,
- после получения каждого чанка сервер отправляет клиенту подтверждение (ACK);

3. ***завершение соединения*** – после передачи всех данных происходит трёхстороннее закрытие соединения:
   - клиент отправляет сообщение FIN, чтобы инициировать завершение соединения,
   - сервер отвечает сообщением FIN_ACK, подтверждая завершение соединения,
   - клиент отправляет сообщение ACK, подтверждая получение FIN_ACK.

При реализации протокола были сделаны следующие **допущения**:

1. ***статичное количество чанков*** – протокол предполагает фиксированное количество чанков для передачи данных по частям (оно задаётся с помощью константы TOTAL_CHUNKS), что упрощает логику обработки данных; в реальных условиях TCP может обрабатывать переменное количество чанков данных;
2. ***отсутствие потерь данных*** – протокол не учитывает возможность потери сообщений или их повреждение; в реальном TCP используются механизмы для повторной передачи потерянных пакетов и проверки целостности данных;
3. ***отсутствие управления потоком и перегрузкой*** – протокол не реализует механизмы управления потоком (*flow control*) и управления перегрузкой (*congestion control*), которые являются важными аспектами реального TCP для обеспечения эффективной передачи данных и предотвращения перегрузки сети;
4. ***упрощённая обработка ошибок*** – с помощью оператора assert в модели протокола реализуется проверка корректности получаемых

сообщений и соблюдение их чёткой последовательности отправки; в случае несоответствия выполнение передачи данных прерывается, в то время как в реальном TCP ошибки обрабатываются более сложным образом, включая тайм-ауты и повторные попытки;

5. ***синхронная обработка*** – для синхронизации операций в модели протокола используются атомарные блоки, в то время как реальный TCP может использовать асинхронные механизмы для обработки входящих и исходящих сообщений.

UML-диаграмма последовательности действий, которая приведена на рисунке 1, иллюстрирует основные шаги описанного протокола в упрощённом виде. Она включает в себя процесс трёхстороннего рукопожатия, последующую передачу данных по частям и процесс закрытия TCP-соединения.

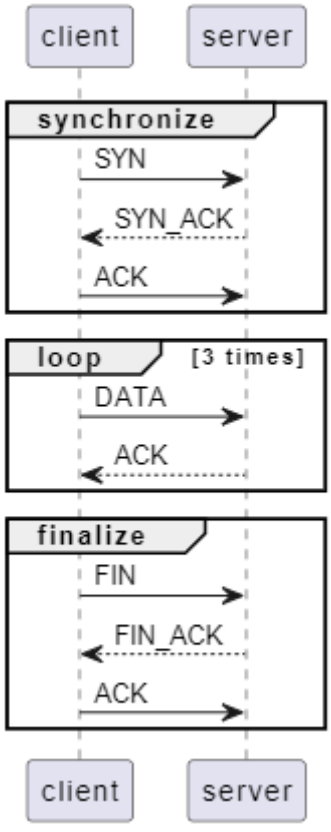

Рисунок 1. Диаграмма последовательности действий упрощённой модели протокола TCP

Модель протокола

Модель протокола включает в себя два процесса – client и server, которые взаимодействуют через каналы передачи сообщений client_to_server и server_to_client (каждый канал может содержать только одно сообщение).

Используются сообщения следующих типов:

- SYN (Synchronize) – для инициализации соединения,
- SYN_ACK (Synchronize-Acknowledgment) – ответ сервера после получения от клиента сообщения SYN,
- ACK (Acknowledgment) – для подтверждения получения других сообщений,
- DATA – для передачи фактических данных между узлами после установления соединения,
- FIN (Finish) – для завершения соединения,
- FIN_ACK (Finish-Acknowledgment) – ответ сервера после получения от клиента сообщения FIN.

Каждое из передаваемых сообщений содержит два поля:

- msg_type – тип сообщения (используется один из типов SYN, SYN_ACK, ACK, DATA, FIN или FIN_ACK),
- payload – передаваемая в сообщении полезная нагрузка.

В каждом из процессов отправка и получение сообщений выстроено таким образом, чтобы они соответствовали требованиям, которые приведены в разделе *«Описание протокола и принятые допущения»*.

В блоке init происходит запуск процессов клиента и сервера.

Promela-код реализованного протокола представлен в Листинге 1.

Листинг 1

```
#define TOTAL_CHUNKS 3

mtype = { SYN, SYN_ACK, ACK, DATA, FIN, FIN_ACK }

typedef Message {
    mtype msg_type;
    byte payload;
};

chan client_to_server = [1] of { Message };
chan server_to_client = [1] of { Message };

proctype client(){
    Message msg;
    msg.msg_type = SYN;
    msg.payload = 0;

    printf("[client] Sending SYN to server..\n");
    client_to_server ! msg;

    atomic {
        server_to_client ? msg;
        assert(msg.msg_type == SYN_ACK);
        printf("[client] Received SYN_ACK from server\n");
    }

    msg.msg_type = ACK;
    msg.payload = 0;

    printf("[client] Sending ACK to server..\n");
    client_to_server ! msg;

    int i;
    for (i : 1 .. TOTAL_CHUNKS) {
        msg.msg_type = DATA;
        msg.payload = i;
        printf("[client] Sending DATA (payload=%d) to server..\n", msg.payload);
        client_to_server ! msg;

        atomic {
            server_to_client ? msg;
            assert(msg.msg_type == ACK);
            printf("[client] Received ACK for DATA from server\n");
        }
    }

    msg.msg_type = FIN;
    msg.payload = 0;

    printf("[client] Sending FIN to server..\n");
    client_to_server ! msg;

    atomic {
        server_to_client ? msg;
        assert(msg.msg_type == FIN_ACK);
        printf("[client] Received FIN_ACK from server\n");
    }
```

```
    msg.msg_type = ACK;
    msg.payload = 0;

    printf("[client] Sending ACK to server..\n");
    client_to_server ! msg;

}

proctype server(){
    Message msg;
    atomic {
        client_to_server ? msg;
        assert(msg.msg_type == SYN);
        printf("[server] Received SYN from client\n");
    }

    msg.msg_type = SYN_ACK;
    msg.payload = 0;

    printf("[server] Sending SYN_ACK to client..\n");
    server_to_client ! msg;

    atomic {
        client_to_server ? msg;
        assert(msg.msg_type == ACK);
        printf("[server] Received ACK from client\n");
    }

    int i;
    for (i : 1 .. TOTAL_CHUNKS) {
        atomic {
            client_to_server ? msg;
            assert(msg.msg_type == DATA);
            printf("[server] Received DATA from client: %d\n", msg.payload);
        }

        msg.msg_type = ACK;
        msg.payload = 0;
        printf("[server] Sending ACK for DATA to client..\n");
        server_to_client ! msg;
    }

    atomic {
        client_to_server ? msg;
        assert(msg.msg_type == FIN);
        printf("[server] Received FIN from client\n");
    }

    msg.msg_type = FIN_ACK;
    msg.payload = 0;

    printf("[server] Sending FIN_ACK to client..\n");
    server_to_client ! msg;

    atomic {
        client_to_server ? msg;
        assert(msg.msg_type == ACK);
        printf("[server] Received ACK from client\n");
    }
}
```

```
init {
    run client();
    run server();
}
```

В общем виде обмен сообщениями между процессами представлен на рисунке 2. Это случай, когда все сообщения гарантированно передаются в порядке, который требует реализация упрощённой версии TCP-протокола. Листинг 2, идущий следом за рисунком, содержит последовательность действий процессов клиента и сервера, которая полностью соответствует требованиям реализуемой версии TCP-протокола.

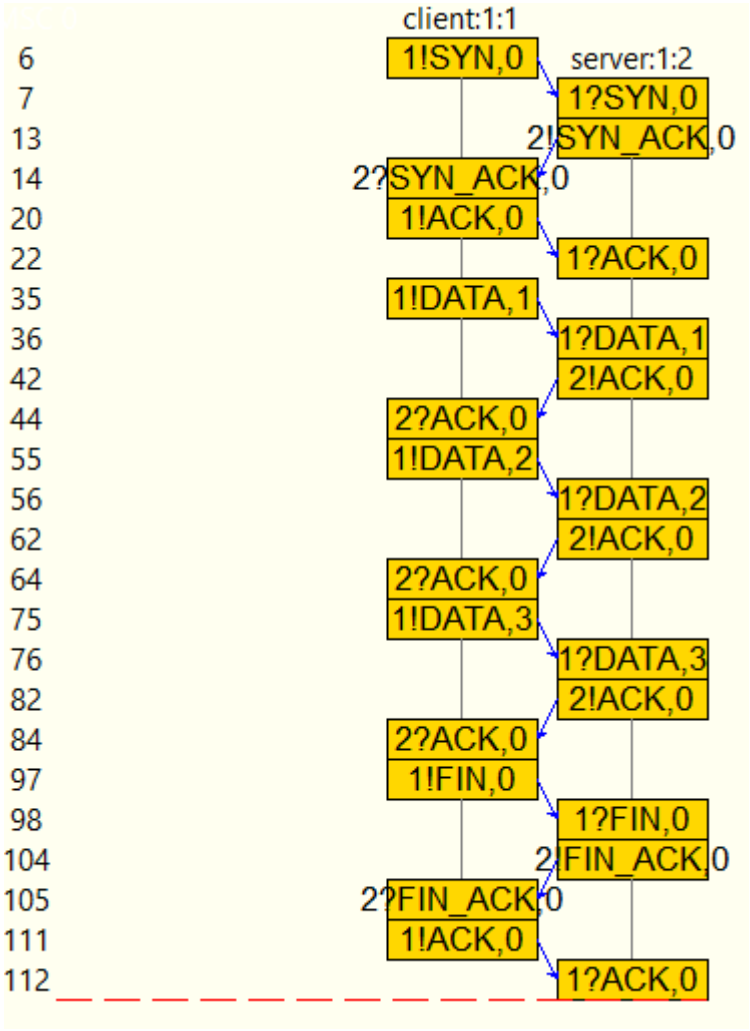

Рисунок 2. Обмен сообщениями между клиентом и сервером

Листинг 2

```
  0:   proc  - (:root:) creates proc  0 (:init:)
Starting client with pid 1
  1:   proc  0 (:init::1) creates proc  1 (client)
  1:   proc  0 (:init::1) tcp_protocol_simple.pml:123 (state 1)  [(run client())]
  2:   proc  1 (client:1) tcp_protocol_simple.pml:15 (state 1)   [msg.msg_type =
SYN]
Starting server with pid 2
  3:   proc  0 (:init::1) creates proc  2 (server)
  3:   proc  0 (:init::1) tcp_protocol_simple.pml:124 (state 2)  [(run server())]
  4:   proc  1 (client:1) tcp_protocol_simple.pml:16 (state 2)   [msg.payload = 0]
[client] Sending SYN to server..
  5:   proc  1 (client:1) tcp_protocol_simple.pml:18 (state 3)   [printf('[client]
Sending SYN to server..\\n')]
  6:   proc  1 (client:1) tcp_protocol_simple.pml:19 (state 4)
       [client_to_server!msg.msg_type,msg.payload]
  7:   proc  2 (server:1) tcp_protocol_simple.pml:71 (state 1)
       [client_to_server?msg.msg_type,msg.payload]
  8:   proc  2 (server:1) tcp_protocol_simple.pml:72 (state 2)
       [assert((msg.msg_type==SYN))]
[server] Received SYN from client
  9:   proc  2 (server:1) tcp_protocol_simple.pml:73 (state 3)   [printf('[server]
Received SYN from client\\n')]
 10:   proc  2 (server:1) tcp_protocol_simple.pml:76 (state 5)   [msg.msg_type =
SYN_ACK]
 11:   proc  2 (server:1) tcp_protocol_simple.pml:77 (state 6)   [msg.payload = 0]
[server] Sending SYN_ACK to client..
 12:   proc  2 (server:1) tcp_protocol_simple.pml:79 (state 7)   [printf('[server]
Sending SYN_ACK to client..\\n')]
 13:   proc  2 (server:1) tcp_protocol_simple.pml:80 (state 8)
       [server_to_client!msg.msg_type,msg.payload]
 14:   proc  1 (client:1) tcp_protocol_simple.pml:22 (state 5)
       [server_to_client?msg.msg_type,msg.payload]
 15:   proc  1 (client:1) tcp_protocol_simple.pml:23 (state 6)
       [assert((msg.msg_type==SYN_ACK))]
[client] Received SYN_ACK from server
 16:   proc  1 (client:1) tcp_protocol_simple.pml:24 (state 7)   [printf('[client]
Received SYN_ACK from server\\n')]
 17:   proc  1 (client:1) tcp_protocol_simple.pml:27 (state 9)   [msg.msg_type =
ACK]
 18:   proc  1 (client:1) tcp_protocol_simple.pml:28 (state 10)  [msg.payload = 0]
[client] Sending ACK to server..
 19:   proc  1 (client:1) tcp_protocol_simple.pml:30 (state 11)  [printf('[client]
Sending ACK to server..\\n')]
 20:   proc  1 (client:1) tcp_protocol_simple.pml:31 (state 12)
       [client_to_server!msg.msg_type,msg.payload]
 21:   proc  1 (client:1) tcp_protocol_simple.pml:34 (state 13)  [i = 0]
 22:   proc  2 (server:1) tcp_protocol_simple.pml:83 (state 9)
       [client_to_server?msg.msg_type,msg.payload]
 23:   proc  2 (server:1) tcp_protocol_simple.pml:84 (state 10)
       [assert((msg.msg_type==ACK))]
[server] Received ACK from client
 24:   proc  2 (server:1) tcp_protocol_simple.pml:85 (state 11)  [printf('[server]
Received ACK from client\\n')]
 25:   proc  1 (client:1) tcp_protocol_simple.pml:34 (state 14)  [i = 1]
 27:   proc  1 (client:1) tcp_protocol_simple.pml:34 (state 15)  [((i<=3))]
 28:   proc  2 (server:1) tcp_protocol_simple.pml:89 (state 13)  [i = 0]
 29:   proc  1 (client:1) tcp_protocol_simple.pml:35 (state 16)  [msg.msg_type =
DATA]
 30:   proc  2 (server:1) tcp_protocol_simple.pml:89 (state 14)  [i = 1]
```

```
 31:   proc  1 (client:1) tcp_protocol_simple.pml:36 (state 17)  [msg.payload = i]
[client] Sending DATA (payload=1) to server..
 33:   proc  1 (client:1) tcp_protocol_simple.pml:38 (state 18)  [printf('[client]
Sending DATA (payload=%d) to server..\\n',msg.payload)]
 34:   proc  2 (server:1) tcp_protocol_simple.pml:89 (state 15)  [((i<=3))]
 35:   proc  1 (client:1) tcp_protocol_simple.pml:39 (state 19)
       [client_to_server!msg.msg_type,msg.payload]
 36:   proc  2 (server:1) tcp_protocol_simple.pml:91 (state 16)
       [client_to_server?msg.msg_type,msg.payload]
 37:   proc  2 (server:1) tcp_protocol_simple.pml:92 (state 17)
       [assert((msg.msg_type==DATA))]
[server] Received DATA from client: 1
 38:   proc  2 (server:1) tcp_protocol_simple.pml:93 (state 18)  [printf('[server]
Received DATA from client: %d\\n',msg.payload)]
 39:   proc  2 (server:1) tcp_protocol_simple.pml:96 (state 20)  [msg.msg_type =
ACK]
 40:   proc  2 (server:1) tcp_protocol_simple.pml:97 (state 21)  [msg.payload = 0]
[server] Sending ACK for DATA to client..
 41:   proc  2 (server:1) tcp_protocol_simple.pml:99 (state 22)  [printf('[server]
Sending ACK for DATA to client..\\n')]
 42:   proc  2 (server:1) tcp_protocol_simple.pml:100 (state 23)
       [server_to_client!msg.msg_type,msg.payload]
 43:   proc  2 (server:1) tcp_protocol_simple.pml:89 (state 24)  [i = (i+1)]
 44:   proc  1 (client:1) tcp_protocol_simple.pml:42 (state 20)
       [server_to_client?msg.msg_type,msg.payload]
 45:   proc  1 (client:1) tcp_protocol_simple.pml:43 (state 21)
       [assert((msg.msg_type==ACK))]
[client] Received ACK for DATA from server
 46:   proc  1 (client:1) tcp_protocol_simple.pml:44 (state 22)  [printf('[client]
Received ACK for DATA from server\\n')]
 48:   proc  1 (client:1) tcp_protocol_simple.pml:34 (state 24)  [i = (i+1)]
 49:   proc  2 (server:1) tcp_protocol_simple.pml:89 (state 15)  [((i<=3))]
 51:   proc  1 (client:1) tcp_protocol_simple.pml:34 (state 15)  [((i<=3))]
 52:   proc  1 (client:1) tcp_protocol_simple.pml:35 (state 16)  [msg.msg_type =
DATA]
 53:   proc  1 (client:1) tcp_protocol_simple.pml:36 (state 17)  [msg.payload = i]
[client] Sending DATA (payload=2) to server..
 54:   proc  1 (client:1) tcp_protocol_simple.pml:38 (state 18)  [printf('[client]
Sending DATA (payload=%d) to server..\\n',msg.payload)]
 55:   proc  1 (client:1) tcp_protocol_simple.pml:39 (state 19)
       [client_to_server!msg.msg_type,msg.payload]
 56:   proc  2 (server:1) tcp_protocol_simple.pml:91 (state 16)
       [client_to_server?msg.msg_type,msg.payload]
 57:   proc  2 (server:1) tcp_protocol_simple.pml:92 (state 17)
       [assert((msg.msg_type==DATA))]
[server] Received DATA from client: 2
 58:   proc  2 (server:1) tcp_protocol_simple.pml:93 (state 18)  [printf('[server]
Received DATA from client: %d\\n',msg.payload)]
 59:   proc  2 (server:1) tcp_protocol_simple.pml:96 (state 20)  [msg.msg_type =
ACK]
 60:   proc  2 (server:1) tcp_protocol_simple.pml:97 (state 21)  [msg.payload = 0]
[server] Sending ACK for DATA to client..
 61:   proc  2 (server:1) tcp_protocol_simple.pml:99 (state 22)  [printf('[server]
Sending ACK for DATA to client..\\n')]
 62:   proc  2 (server:1) tcp_protocol_simple.pml:100 (state 23)
       [server_to_client!msg.msg_type,msg.payload]
 63:   proc  2 (server:1) tcp_protocol_simple.pml:89 (state 24)  [i = (i+1)]
 64:   proc  1 (client:1) tcp_protocol_simple.pml:42 (state 20)
       [server_to_client?msg.msg_type,msg.payload]
 65:   proc  1 (client:1) tcp_protocol_simple.pml:43 (state 21)
       [assert((msg.msg_type==ACK))]
```

```
[client] Received ACK for DATA from server
 66:   proc  1 (client:1) tcp_protocol_simple.pml:44 (state 22)  [printf('[client]
Received ACK for DATA from server\\n')]
 67:   proc  1 (client:1) tcp_protocol_simple.pml:34 (state 24)  [i = (i+1)]
 70:   proc  1 (client:1) tcp_protocol_simple.pml:34 (state 15)  [((i<=3))]
 71:   proc  2 (server:1) tcp_protocol_simple.pml:89 (state 15)  [((i<=3))]
 72:   proc  1 (client:1) tcp_protocol_simple.pml:35 (state 16)  [msg.msg_type =
DATA]
 73:   proc  1 (client:1) tcp_protocol_simple.pml:36 (state 17)  [msg.payload = i]
[client] Sending DATA (payload=3) to server..
 74:   proc  1 (client:1) tcp_protocol_simple.pml:38 (state 18)  [printf('[client]
Sending DATA (payload=%d) to server..\\n',msg.payload)]
 75:   proc  1 (client:1) tcp_protocol_simple.pml:39 (state 19)
       [client_to_server!msg.msg_type,msg.payload]
 76:   proc  2 (server:1) tcp_protocol_simple.pml:91 (state 16)
       [client_to_server?msg.msg_type,msg.payload]
 77:   proc  2 (server:1) tcp_protocol_simple.pml:92 (state 17)
       [assert((msg.msg_type==DATA))]
[server] Received DATA from client: 3
 78:   proc  2 (server:1) tcp_protocol_simple.pml:93 (state 18)  [printf('[server]
Received DATA from client: %d\\n',msg.payload)]
 79:   proc  2 (server:1) tcp_protocol_simple.pml:96 (state 20)  [msg.msg_type =
ACK]
 80:   proc  2 (server:1) tcp_protocol_simple.pml:97 (state 21)  [msg.payload = 0]
[server] Sending ACK for DATA to client..
 81:   proc  2 (server:1) tcp_protocol_simple.pml:99 (state 22)  [printf('[server]
Sending ACK for DATA to client..\\n')]
 82:   proc  2 (server:1) tcp_protocol_simple.pml:100 (state 23)
       [server_to_client!msg.msg_type,msg.payload]
 83:   proc  2 (server:1) tcp_protocol_simple.pml:89 (state 24)  [i = (i+1)]
 84:   proc  1 (client:1) tcp_protocol_simple.pml:42 (state 20)
       [server_to_client?msg.msg_type,msg.payload]
 85:   proc  1 (client:1) tcp_protocol_simple.pml:43 (state 21)
       [assert((msg.msg_type==ACK))]
[client] Received ACK for DATA from server
 86:   proc  1 (client:1) tcp_protocol_simple.pml:44 (state 22)  [printf('[client]
Received ACK for DATA from server\\n')]
 87:   proc  1 (client:1) tcp_protocol_simple.pml:34 (state 24)  [i = (i+1)]
 89:   proc  2 (server:1) tcp_protocol_simple.pml:101 (state 25) [else]
 92:   proc  1 (client:1) tcp_protocol_simple.pml:46 (state 25)  [else]
 94:   proc  1 (client:1) tcp_protocol_simple.pml:48 (state 30)  [msg.msg_type =
FIN]
 95:   proc  1 (client:1) tcp_protocol_simple.pml:49 (state 31)  [msg.payload = 0]
[client] Sending FIN to server..
 96:   proc  1 (client:1) tcp_protocol_simple.pml:51 (state 32)  [printf('[client]
Sending FIN to server..\\n')]
 97:   proc  1 (client:1) tcp_protocol_simple.pml:52 (state 33)
       [client_to_server!msg.msg_type,msg.payload]
 98:   proc  2 (server:1) tcp_protocol_simple.pml:104 (state 30)
       [client_to_server?msg.msg_type,msg.payload]
 99:   proc  2 (server:1) tcp_protocol_simple.pml:105 (state 31)
       [assert((msg.msg_type==FIN))]
[server] Received FIN from client
100:   proc  2 (server:1) tcp_protocol_simple.pml:106 (state 32) [printf('[server]
Received FIN from client\\n')]
101:   proc  2 (server:1) tcp_protocol_simple.pml:109 (state 34) [msg.msg_type =
FIN_ACK]
102:   proc  2 (server:1) tcp_protocol_simple.pml:110 (state 35) [msg.payload = 0]
[server] Sending FIN_ACK to client..
103:   proc  2 (server:1) tcp_protocol_simple.pml:112 (state 36) [printf('[server]
Sending FIN_ACK to client..\\n')]
```

```
104:   proc  2 (server:1) tcp_protocol_simple.pml:113 (state 37)
       [server_to_client!msg.msg_type,msg.payload]
105:   proc  1 (client:1) tcp_protocol_simple.pml:55 (state 34)
       [server_to_client?msg.msg_type,msg.payload]
106:   proc  1 (client:1) tcp_protocol_simple.pml:56 (state 35)
       [assert((msg.msg_type==FIN_ACK))]
[client] Received FIN_ACK from server
107:   proc  1 (client:1) tcp_protocol_simple.pml:57 (state 36)  [printf('[client]
Received FIN_ACK from server\\n')]
108:   proc  1 (client:1) tcp_protocol_simple.pml:60 (state 38)  [msg.msg_type =
ACK]
109:   proc  1 (client:1) tcp_protocol_simple.pml:61 (state 39)  [msg.payload = 0]
[client] Sending ACK to server..
110:   proc  1 (client:1) tcp_protocol_simple.pml:63 (state 40)  [printf('[client]
Sending ACK to server..\\n')]
111:   proc  1 (client:1) tcp_protocol_simple.pml:64 (state 41)
       [client_to_server!msg.msg_type,msg.payload]
112:   proc  2 (server:1) tcp_protocol_simple.pml:116 (state 38)
       [client_to_server?msg.msg_type,msg.payload]
113:   proc  2 (server:1) tcp_protocol_simple.pml:117 (state 39)
       [assert((msg.msg_type==ACK))]
[server] Received ACK from client
114:   proc  2 (server:1) tcp_protocol_simple.pml:118 (state 40) [printf('[server]
Received ACK from client\\n')]
114:   proc  2 (server:1) terminates
114:   proc  1 (client:1) terminates
114:   proc  0 (:init::1) terminates
3 processes created
```

Если нарушить ожидаемый порядок передаваемых сообщений, (например, тем, что данные начнут передаваться сразу после получения клиентом сообщения SYN_ACK от сервера, без отправки клиентом ACK-подтверждения серверу), то процесс передачи данных и TCP-соединение в целом окажутся прерваны. Листинг 3 содержит соответствующую последовательность действий, которая выводится до момента, когда срабатывает нарушение assert-утверждения (утверждения, проверяющего, что получаемое на данном этапе сообщение может иметь только тип ACK).

Листинг 3

```
using statement merging
Starting client with pid 1
  1:   proc  0 (:init::1) tcp_protocol_error.pml:117 (state 1)   [(run client())]
  2:   proc  1 (client:1) tcp_protocol_error.pml:15 (state 1)    [msg.msg_type =
SYN]
  2:   proc  1 (client:1) tcp_protocol_error.pml:16 (state 2)    [msg.payload = 0]
[client] Sending SYN to server..
  2:   proc  1 (client:1) tcp_protocol_error.pml:18 (state 3)    [printf('[client]
Sending SYN to server..\\n')]
```

```
Starting server with pid 2
  3:   proc  0 (:init::1) tcp_protocol_error.pml:118 (state 2)   [(run server())]
  4:   proc  1 (client:1) tcp_protocol_error.pml:19 (state 4)
       [client_to_server!msg.msg_type,msg.payload]
  5:   proc  2 (server:1) tcp_protocol_error.pml:65 (state 1)
       [client_to_server?msg.msg_type,msg.payload]
  5:   proc  2 (server:1) tcp_protocol_error.pml:66 (state 2)
       [assert((msg.msg_type==SYN))]
[server] Received SYN from client
  5:   proc  2 (server:1) tcp_protocol_error.pml:67 (state 3)   [printf('[server]
Received SYN from client\\n')]
  5:   proc  2 (server:1) tcp_protocol_error.pml:70 (state 5)   [msg.msg_type =
SYN_ACK]
  5:   proc  2 (server:1) tcp_protocol_error.pml:71 (state 6)   [msg.payload = 0]
[server] Sending SYN_ACK to client..
  5:   proc  2 (server:1) tcp_protocol_error.pml:73 (state 7)   [printf('[server]
Sending SYN_ACK to client..\\n')]
  6:   proc  2 (server:1) tcp_protocol_error.pml:74 (state 8)
       [server_to_client!msg.msg_type,msg.payload]
  7:   proc  1 (client:1) tcp_protocol_error.pml:22 (state 5)
       [server_to_client?msg.msg_type,msg.payload]
  7:   proc  1 (client:1) tcp_protocol_error.pml:23 (state 6)
       [assert((msg.msg_type==SYN_ACK))]
[client] Received SYN_ACK from server
  7:   proc  1 (client:1) tcp_protocol_error.pml:24 (state 7)   [printf('[client]
Received SYN_ACK from server\\n')]
  7:   proc  1 (client:1) tcp_protocol_error.pml:28 (state 9)   [i = 0]
  7:   proc  1 (client:1) tcp_protocol_error.pml:28 (state 10)  [i = 1]
  8:   proc  1 (client:1) tcp_protocol_error.pml:28 (state 11)  [((i<=3))]
  8:   proc  1 (client:1) tcp_protocol_error.pml:29 (state 12)  [msg.msg_type =
DATA]
  8:   proc  1 (client:1) tcp_protocol_error.pml:30 (state 13)  [msg.payload = i]
[client] Sending DATA (payload=1) to server..
  8:   proc  1 (client:1) tcp_protocol_error.pml:32 (state 14)  [printf('[client]
Sending DATA (payload=%d) to server..\\n',msg.payload)]
  9:   proc  1 (client:1) tcp_protocol_error.pml:33 (state 15)
       [client_to_server!msg.msg_type,msg.payload]
 10:   proc  2 (server:1) tcp_protocol_error.pml:77 (state 9)
       [client_to_server?msg.msg_type,msg.payload]
spin: tcp_protocol_error.pml:78, Error: assertion violated
spin: text of failed assertion: assert((msg.msg_type==ACK))
#processes: 3
 10:   proc  2 (server:1) tcp_protocol_error.pml:78 (state 10)
 10:   proc  1 (client:1) tcp_protocol_error.pml:35 (state 19)
 10:   proc  0 (:init::1) tcp_protocol_error.pml:119 (state 3)
3 processes created
Exit-Status 0
```

Аналогичным образом это прерывание отображено на рисунке 3, который иллюстрирует обмен сообщениями между процессами.

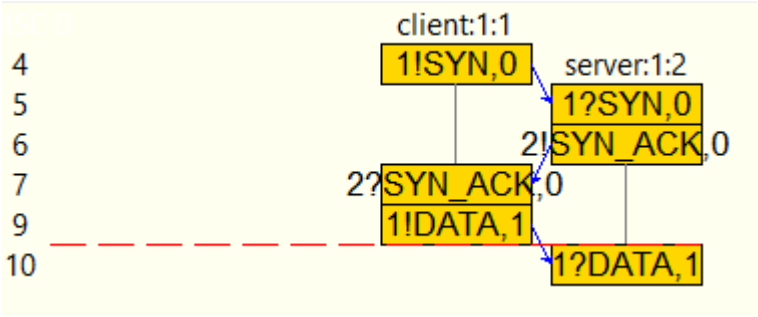

Рисунок 3. Прерывание обмена сообщениями между клиентом и сервером в момент нарушения порядка передаваемых сообщений

Рассмотренный пример ошибки подтверждает возможность отправки сообщений только в порядке, который задан протоколом.

Выводы

В результате выполнения лабораторной работы была реализована модель сетевого протокола для передачи данных между клиентским и серверным приложением. Был описан TCP-протокол и принятые допущения для его упрощения. Приведена UML-диаграмма последовательности действий для упрощённой модели TCP-протокола.

Была рассмотрена работа этой модели в двух ситуациях – в случае последовательности обмена сообщениями, заданной протоколом, и в случае её нарушения. Продемонстрировано прерывание работы протокола при нарушении ожидаемой последовательности передачи сообщений.