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 «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

Отчет по лабораторной работе №3
по курсу
”Математические основы верификации ПО”

Тема Моделирование сетевого протокола

Студент Якуба Д. В.

Группа ИУ7-43М

Оценка (баллы) _____

Преподаватели Кузнецова О.В.

Москва — 2024 г.

Оглавление

ВВЕДЕНИЕ

Цель работы – описать упрощенную модель протокола TCP.

Для достижения поставленной цели потребуется:

- описать протокол и принятые допущения;
- привести описываемые UML-sequence;
- привести модель протокола;
- привести логи SPIN, демонстрирующие отправку/получение данных: пакетов, HTML-документов и прочего.

1. Выполнение

1.1 Описание протокола TCP

TCP (Transmission Control Protocol) - это протокол транспортного уровня в сетях TCP/IP. Он обеспечивает надежную, упорядоченную и точечную передачу данных между устройствами в сети. TCP гарантирует доставку данных без потерь, дублирования или искажения в правильном порядке. Для этого используются подтверждения и механизм повторной отправки данных в случае возникновения ошибок. Кроме того, данный протокол контролирует скорость передачи данных между отправителем и получателем для предотвращения переполнения буфера или перегрузки сети – это достигается за счет использование механизмов окна передачи и алгоритмов обратной связи. TCP устанавливает и разрывает виртуальные соединения между устройствами для передачи данных, что включает в себя процедуры установки и разрыва соединения с использованием трехстороннего рукопожатия. Каждое приложение на устройстве использует порт для связи в сети, а TCP использует номера портов для адресации различных служб и приложений. При передаче данные сегментируются, что позволяет оптимизировать использование сетевых ресурсов и обеспечивает более надежную передачу данных.

UML-sequence диаграмма протокола представлена на рисунке 1.1. В качестве допущения в реализуемой модели определим, что:

- модель предусматривает отправку в качестве данных только целочисленных значений, причем в самой модели будет описана отправка всего двух элементов;
- модель не предусматривает проверки целостности данных и корректности значений, для которых на схеме задействованы параметры *i, j, a, b, c, n, m*;
- в модели не будет учитываться поле пакетов *SEQ*;
- в качестве моделирования отправки одновременно двух флагов в одном пакете, например *SYN* и *ACK*, будут использоваться такие типы как *SYN_ACK*.

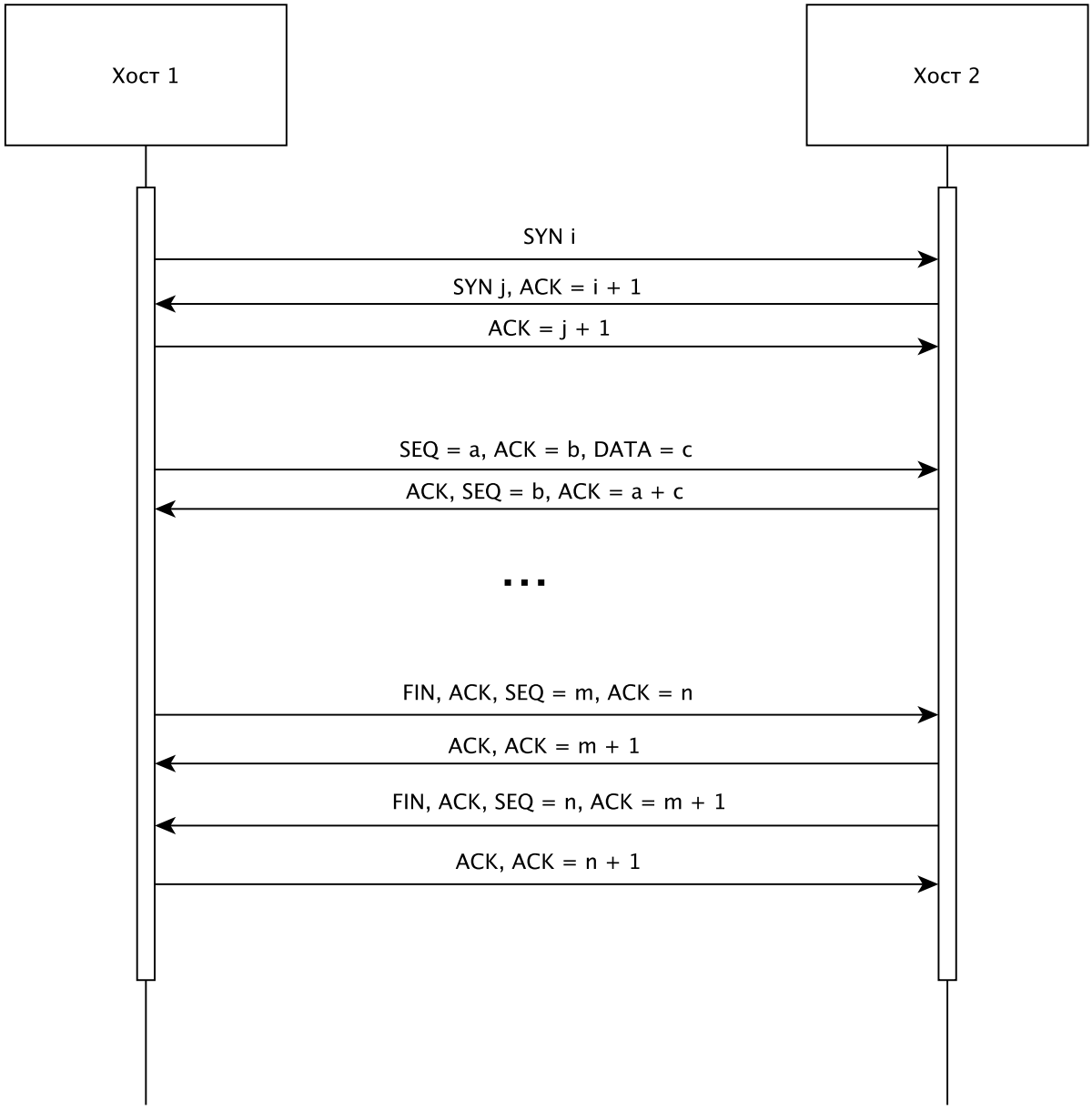

Рис. 1.1: UML-sequence диаграмма взаимодействия двух хостов с использованием протокола TCP.

1.2 Описание модели и результаты

Описание модели представлено в листинге 1.

Листинг 1: Описание модели взаимодействия процессов.

```
1  #define MAX_BUF 2
2  #define DATA_MESSAGES_COUNT 2
3
4  mtype {DATA, ACK, SYN, SYN_ACK, FIN_ACK};
```

```
5
6     chan client_to_server = [MAX_BUF] of {int, mtype};
7     chan server_to_client = [MAX_BUF] of {int, mtype};
8
9     proctype Client() {
10        int data = 1337;
11        int seq_num = 0;
12        mtype msg_type;
13
14        printf("Клиент отправляет SYN\n");
15        client_to_server ! data, SYN;
16
17        server_to_client ? _, msg_type;
18        if
19        :: (msg_type == SYN_ACK) ->
20            printf("Клиент получает SYN-ACK от сервера и направляет в
    ответ ACK\n");
21            client_to_server ! 0, ACK;
22        fi
23
24        do
25            :: seq_num < DATA_MESSAGES_COUNT ->
26                printf("Клиент отправляет данные %d\n", data);
27                seq_num++;
28                msg_type = DATA;
29                client_to_server ! data, DATA;
30                data++;
31                server_to_client ? _, msg_type;
32                if
33                :: (msg_type == ACK) ->
34                    printf("Клиент получает ACK от сервера\n");
35                fi
36
37            :: else ->
38                break;
39        od
40
41        server_to_client ? _, msg_type;
42        if
43        :: (msg_type == ACK) ->
44            printf("Клиент получает ACK от сервера\n");
45        fi
46
47        printf("Клиент направляет FIN_ACK серверу\n")
48        client_to_server ! 0, FIN_ACK;
49
50        server_to_client ? _, msg_type;
51        if
```

```
52          :: (msg_type == ACK) ->
53              printf("Клиент получил ACK от сервера\n")
54          fi
55
56          server_to_client ? _, msg_type;
57          if
58          :: (msg_type == FIN_ACK) ->
59              printf("Клиент получил FIN_ACK от сервера\n")
60          fi
61
62          printf("Клиент направляет последний ACK серверу и на этом
    заканчивает работу\n")
63          client_to_server ! 0, ACK;
64      }
65
66      proctype Server() {
67          int received_data;
68          mtype msg_type;
69
70          client_to_server ? _, msg_type;
71          if
72          :: (msg_type == SYN) ->
73              printf("Сервер получает SYN от клиента и направляет в ответ
    SYN_ACK\n");
74              server_to_client ! 0, SYN_ACK;
75          fi
76
77          client_to_server ? _, msg_type;
78          if
79          :: (msg_type == ACK) ->
80              printf("Сервер получает ACK от клиента\n");
81              server_to_client ! 0, ACK;
82          fi
83
84          printf("Сервер ожидает данные\n");
85          client_to_server ? received_data, msg_type;
86
87          do
88              :: (msg_type == DATA) ->
89                  printf("Сервер получает данные %d и отправляет ACK в ответ
    клиенту\n", received_data);
90                  server_to_client ! 0, ACK;
91                  client_to_server ? received_data, msg_type;
92
93              :: (msg_type == FIN_ACK) ->
94                  printf("Сервер получает FIN_ACK от клиента и направляет
    клиенту ACK и FIN_ACK\n")
95                  server_to_client ! 0, ACK;
```

```
96                  server_to_client ! 0, FIN_ACK;
97                  break;
98          od
99
100         client_to_server ? 0, msg_type;
101
102         if
103         :: (msg_type == ACK) ->
104             printf("Сервер получил ACK от клиента и завершает работу\n")
105         fi
106     }
107
108     init {
109         atomic {
110             run Client();
111             run Server();
112         }
113     }
114
```

На рисунке 1.2 представлены логи SPIN, демонстрирующие работу модели.

```
❯ make pml
spin lab.pml
          Клиент отправляет SYN
              Сервер получает SYN от клиента и направляет в ответ SYN_ACK
          Клиент получает SYN-ACK от сервера и направляет в ответ ACK
              Сервер получает ACK от клиента
              Сервер ожидает данные
          Клиент отправляет данные 1337
          Клиент получает ACK от сервера
              Сервер получает данные 1337 и отправляет ACK в ответ клиенту
          Клиент отправляет данные 1338
              Сервер получает данные 1338 и отправляет ACK в ответ клиенту
          Клиент получает ACK от сервера
          Клиент получает ACK от сервера
          Клиент направляет FIN_ACK серверу
              Сервер получает FIN_ACK от клиента и направляет клиенту ACK и FIN_ACK
          Клиент получил ACK от сервера
          Клиент получил FIN_ACK от сервера
          Клиент направляет последний ACK серверу и на этом заканчивает работу
              Сервер получил ACK от клиента и завершает работу
3 processes created
```

Рис. 1.2: Логи SPIN, демонстрирующие работу модели.

ЗАКЛЮЧЕНИЕ

В результате выполнения работы была описана упрощенная модель протокола TCP на ЯП Promela.

Были решены следующие задачи:

- описан протокол TCP и принятые допущения;
- приведено описание EML-sequence;
- приведена модель протокола;
- приведены логи SPIN, демонстрирующие отправку/получение данных.