Содержание

Введение

Современные вычислительные системы сложно представить без поддержки интернет-сетей. Сети являются основным средством передачи информации. От качества соединения зависит работа большого количества сервисов и учереждений.

Основными проблемами, влияющими на качество связи являются следующие:

- высокий RTT(round-trip-time);
- высокий процент потерь пакетов;
- получение пакетов в порядке, отличном от порядка отправки;
- дублирование пакетов.

В данной работе будет рассмотренна проблема дублирования пакетов.

1 Аналитический раздел

1.1 Постановка задачи

В соответствии с заданием на курсовую работу необходимо разработать и реализовать метод определения и отбрасывания дубликатов пакетов.

Для достижения цели курсовой работы необходимо решить следующие задачи:

- проанализировать существующие методы решения;
- спроектировать протокол;
- реализовать спроектированный протокол;
- протестировать реализованный протокол.

Разрабатываемый протокол должен отбрасывать дубликаты уже пришедших пакетов.

1.2 Общие сведения

Канал обычно характеризуются следующими характеристикам:

1. RTT (Round Trip Time). Представляет собой время между отправкой запроса и получением ответа. На RTT влияет как физические ограничения скорости передачи сигнала в зависимости от среды (медь, оптоволокно, радиосигнал) и расстояния, так и ограничения в скорости обработки трафика в промежуточных узлах;
2. bandwidth. Ширина канала - максимально возможное количество данных, которые могут быть переданы через канал за некоторый промежуток времени;
3. lossrate (процент потерь пакетов).

Вышеприведенные характеристики и вытекающие из них используются не только для общей оценки качества канала, но и в прикладных алгоритмах для его улучшения.

1.3 Анализ существующих решений

Уровень, который интересует нас на модели OSI - четвертый (транспортный).
Рассмотрим некоторые протоколы транспортного уровня.

1.3.1 UDP

User Datagram Protocol (UDP) - один из базовых протоколов сети. Протокол быстр, не гарантирует, что пакеты будут доставлены в том порядке, в котором они были отправлены и даже сам факт доставки. В протоколе отсутствуют пакеты подтверждения. Не требует открытия соеденения, пакеты отправляются сразу по готовности.

UDP используется когда требуется скорость доставки, а гарантией и правильностью доставки можно пренебречь, а также для широковещательной передачи. Протокол UDP определен в RFC 786.

1.3.2 TCP

Transmission Control Protocol (TCP) - второй базовый протокол транспортного уровня. Гарантирует надежную доставку пакетов в определенном порядке. Использует пакеты подтверждения и повторную отправку пакета в случае его утери или искажения. Соединение должно установиться до начала передачи данных.

TCP медленный, так как использует механизм контроля получения пакетов, что требует больших затрат времени. Используется в случаях, требующих надежную доставку сообщений. Протокол TCP определен в

RFC 793.

Протокол TCP, в отличии от UDP использует метрики, собираемые в процессе работы протокола, для определения дальнейшей работы. Из полученных метрик высчитывается размер TCP окна, показывающий количество байт, которыепринимающая сторона готова принять в текущий момент без подтверждения.

Помимо RTT и lossrate TCP вводит понятие отклонения RTT (devRTT). DevRTT и средний RTT используется для определения промежутка времени, после которого пакеты будут считаться утерянными в случае отсутствия подтверждающих пакетов. Далее приводятся формулы, которым оперирует TCP.

$$RTT_i = t_r - t_0, \tag{1}$$

где t_r - время получения подтверждающего пакета;

t_0 - время отправки пакета;

RTT_i - RTT i-ого пакета.

$$RTT_{average} = \alpha RTT_{average} + \beta RTT_i \ , \tag{2}$$

где $RTT_{average}$ - среднее RTT;

обычно $\alpha = 0{,}875$, $\beta = 0{,}125$.

$$dev_i = |RTT_i - RTT_{average}| \ , \tag{3}$$

где dev_i - devRTT i-ого пакета.

$$dev_{average} = \alpha dev_{average} + \beta dev_i \ , \tag{4}$$

где $dev_{average}$ - среднее devRTT;

обычно $\alpha = 0{,}75$, $\beta = 0{,}25$.

$$timeout = RTT_{average} + 4dev_{average} \tag{5}$$

Заголовок TCP пакетов представлен на рисунке 1.

TCP Header

Offsets	Octet	0								1								2								3			
Octet	Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	Source port																Destination port															
4	32	Sequence number																															
8	64	Acknowledgment number (if ACK set)																															
12	96	Data offset				Reserved 0 0 0			NS	CWR	ECE	URG	ACK	PSH	RST	SYN	FIN	Window Size															
16	128	Checksum																Urgent pointer (if URG set)															
20	160	Options (if *data offset* > 5. Padded at the end with "0" bytes if necessary.)																															
...	...	...																															

Рисунок 1 – Заголовок TCP пакета

Расшифровка полей:

Таблица 1 – Расшифровка полей заголовка TCP пакета.

Поле	Длина	Описание
Порт источника	2 байта	Номер порта источника
Порт назначения	2 байта	Номер порта назначения
Последовательный номер	4 байта	Последовательный номер генерируется источником и используется назначением, чтобы переупорядочить пакеты для создания исходного сообщения и отправить подтверждение источнику.
Номер подтверждения	4 байта	Если установлен бит ACK поля "Управление в данном поле содержится следующий ожидаемый последовательный номер.
Смещение данных	4 байта	Информация о начале пакета данных.
Резерв	6 битов	Резервируются для будущего использования

Продолжение на следующей странице...

Поле	Длина	Описание
Управление	6 битов	Биты управления содержат флаги, указывающие, верны ли поля подтверждения (ACK), указателя срочности (URG), следует ли сбрасывать соединение (RST), послан ли синхронизирующий последовательный номер (SYN) и т. д.
Размер окна	2 байта	В этом поле указывается размер приемного буфера. Используя подтверждающие сообщения, получатель может информировать отправителя о максимальном размере данных, которые тот может отправить.
Контрольная сумма	2 байта	Контрольная сумма заголовка и данных; по ней определяется, был ли искажен пакет
Указатель срочности	2 байта	В этом поле целевое устройство получает информацию о срочности данных.
Опции	переменная	Необязательные значения, которые указываются при необходимости
Дополнение	переменная	В поле дополнения добавляется столько нулей, чтобы заголовок заканчивался на 32-битной границе

В алгоритме TCP алгоритм отбрасывания пакетов, приходящих вне своей очереди или дублированных, реализован с помощью сравнения поля с последовательным номером пакета с текущим TCP-окном. В самом три-

виальном случае, если номер пакета больше или меньше крайних значений TCP порта, пакет отбрасывается. Если номер пакета находится внутри текущего окна и ещё не пришел, то его номер сохраняется и происходит принятие пакета. Если же номер уже существует, то пакет отбрасывается как дублированный.

1.4 Вывод

В аналитической части были проанализированы два основных сетевых протокола транспортного уровня (TCP и UDP), были разобраны особенности протокола TCP.

На основании проведенного анализа можно сделать вывод, что свой протокол необходимо реализовывать на основе протокола UDP и алгоритма отбрасывания пакета в TCP.

2 Конструкторский раздел

В данном разделе будут спроектирован и описан разрабатываемый протокол.

2.1 Проектирование протокола

Из аналитической части стало понятно, что для определения дублированных пакетов необходимо ввести как минимум уникальный идентификатор пакета. Таким образом добавляется заголовок, содержащий метаинформацию.

На принимающей стороне необходимо обеспечить хранение номеров принятых пакетов. Это можно сделать различными способами.

Первый способ — хранить номера пакетов отдельно, с помощью массива, списка или хэш-таблицы. У этого способа есть как плюсы так и минусы.

Плюсы:

1. простота хранения и доступа к номеру пакета;
2. простота добавления новых номеров пакета.

Главным минусом же является большое количество памяти, затрачиваемое на хранение номеров.

Второй способ предполагает групировку пакетов набором диапазонов. В этом методе номера 1, 2, 3...10 хранятся не по отдельности а в виде двух чисел — началу и конец диапазона (1, 10).

Плюсом такого подхода является намного меньший объем занимаемой памяти, а главным минусом — сложность в добавлении нового номера пакета к существующим диапазонам.

Для реализации был выбран второй подход. На рисунках 2–4 представлена схема алгоритма проверки вхождения и добавления номера пакета

в существующий набор диапазонов.

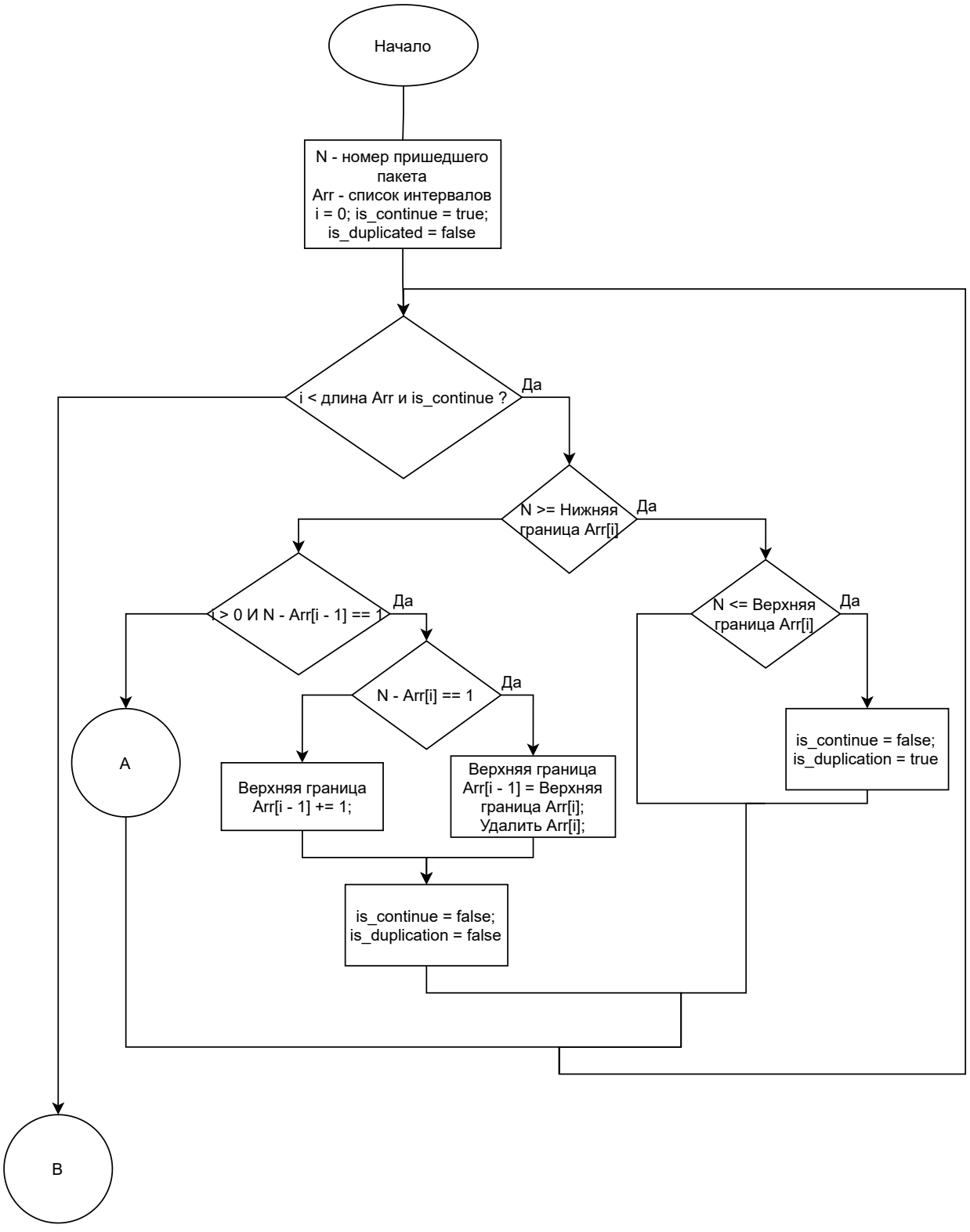

Рисунок 2 – Первая часть проверки и добавления

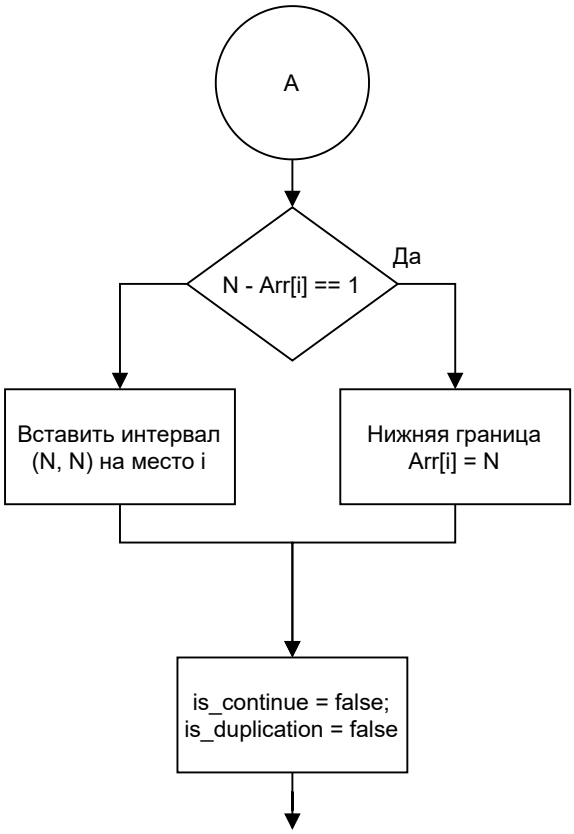

Рисунок 3 – Вторая часть проверки и добавления

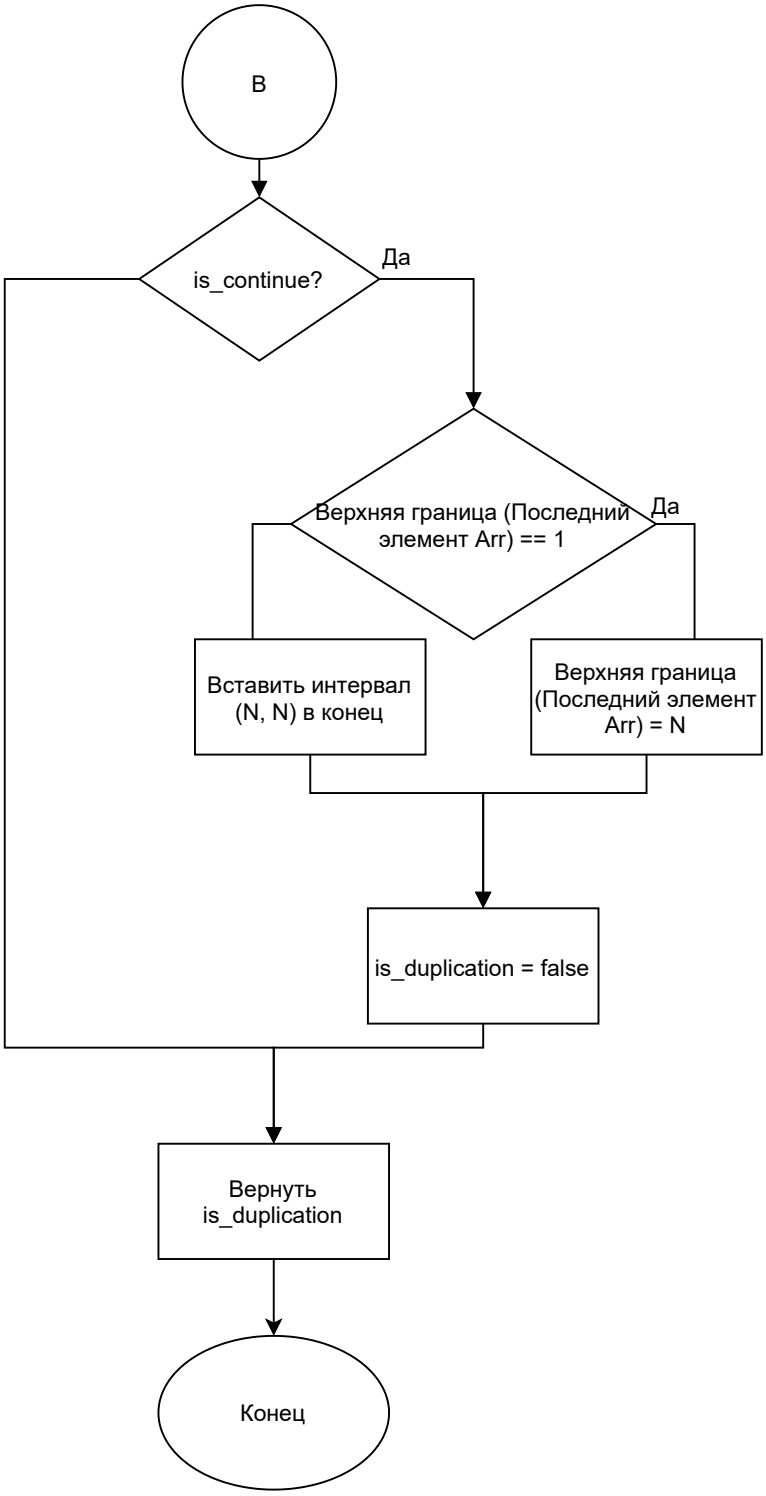

Рисунок 4 – Третья часть проверки и добавления

Для корректного отбрасывания накопленных интервалов необходимо ввести правило отбрасывания отмлеживания устаревших пакетов. По аналогии с TCP логично ввести окно, но, в отличии от TCP можно отказаться от его динамичности. Таким образом вводится статическое окно, всегда содержащее интервал в N пакетов.

Из вышеизложенного алгоритма сразу вытекает крайний случай. Если на стороне клиента произошел сбой, то нумерация отправленных пакетов начнется сначала и, если они уже были приняты в предвдущей сессии, то они начнут отбрасываться, пока номер пакета не превысит номер последнего принятого пакета.

Для решения этой проблемы стоит ввести идентификатор начала передачи пакетов, по которому будет происходить обнуление отслеживаемых интервалов. И для надежной его передачи — пакет подтверждение. На рисунке 5 представлена общая схема обработки пакета на принимающей стороне, а на рисунке 6 — со стороны отправки.

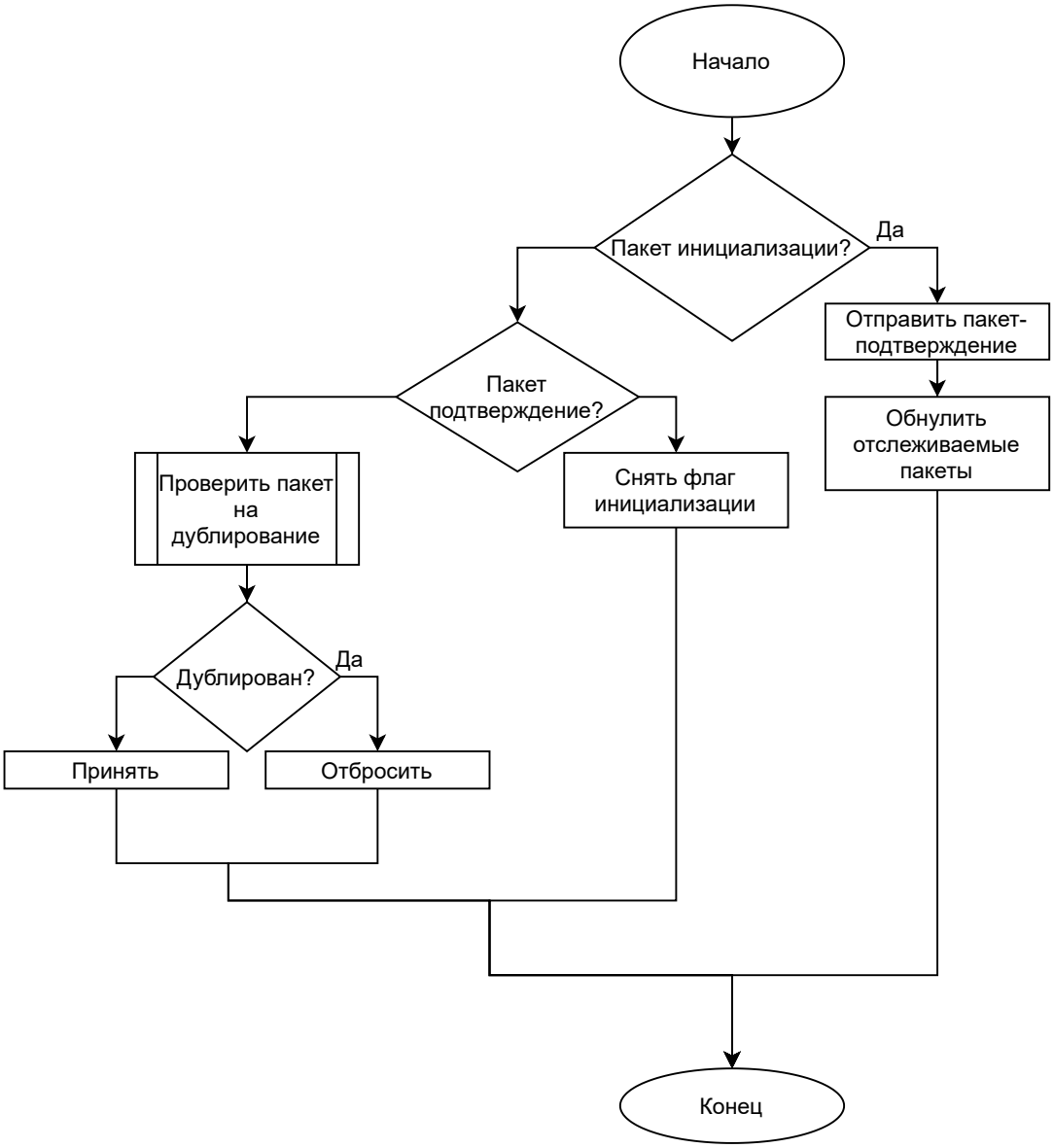

Рисунок 5 – Алгоритм получения пакеты

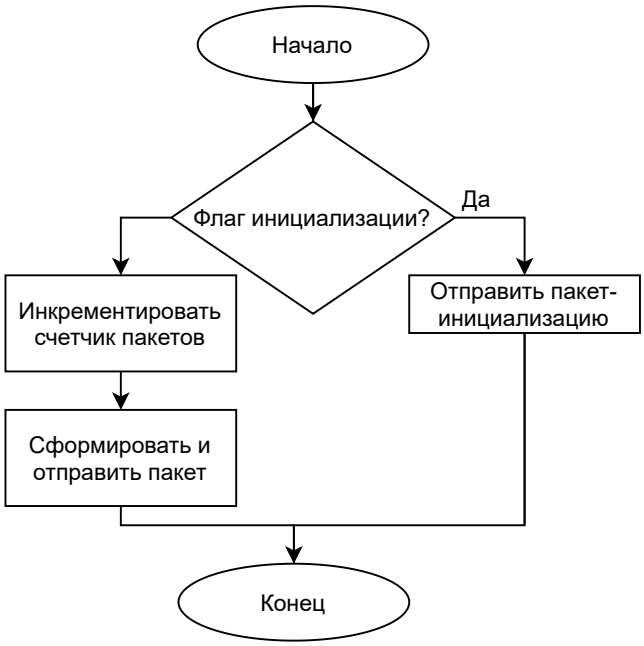

Рисунок 6 – Алгоритм отправки пакета

Для реализации передачи типа пакета необходимо добавить соответствующее поле в заголовок пакета.

2.2 Вывод

В конструкторской части был описан протокол, спроектированы основные алгоритмы.

3 Технологический раздел

3.1 Выбор языка и среды программирования

В качестве языка программирования для реализации поставленной задачи был выбран язык Rust — мультипарадигмальный компилируемый язык программирования общего назначения, часто используется для системного программирования. Средой разработки был выбран текстовый редактор Visual Studio Code.

3.2 Описание основных структур

Для реализации хранения номяров принятых пакетов была написана структура filter. На листинге 2 приведена эта структура.

Листинг 1 – filter

```
pub struct Filter {
  state: bool,
  window_size: usize,
  received_packets: Vec<(usize, usize)>,
}
```

Описание полей структуры:

- state — состояние активности фильтра;
- window_size — размер окна;
- received_packets — список хранения интервалов.

Заголовок пакета был реализован с помощью следующих структур:

Листинг 2 – Заголовок пакета

```
pub enum PacketKind {
  Init,
  Data,
  Ack,
```

```
}

pub struct Header {
  index: u32,
  payload_len: u32,
  kind: PacketKind,
}
```

3.3 Реализация

На основе алгоритма из конструкторской части была написана функция проверки дублирования пакета. На листинге 3 представлена эта функция.

Листинг 3 – Проверка на дублирование

```
fn check_duplicate(&mut self, i: usize) -> bool {
 let len = self.received_packets.len();
 for c in 0..end {
   if i >= self.received_packets[c].0 {
     if i <= self.received_packets[c].1 {
       return true;
     }
     continue;
   } else if (c != 0) && ((i - self.received_packets[c - 1].1) ==
   1) {
     if (self.received_packets[c].0 - i) == 1 {
       self.received_packets[c - 1].1 = self.received_packets[c
  ].1;
       self.received_packets.remove(c);
     } else {
       self.received_packets[c - 1].1 = i;
     }
     return false;
   } else if (self.received_packets[c].0 - i) == 1 {
     self.received_packets[c].0 = i;
```

```
19          return false;
20        }
21        self.received_packets.insert(c, (i, i));
22        return false;
23      }
24      if i - self.received_packets[len - 1].1 == 1 {
25        self.received_packets[len - 1].1 = i;
26      } else {
27        self.received_packets.insert(len, (i, i));
28      }
29      false
30  }
```

На основе алгоритма из конструкторской части была написана функция обработки получения пакета. На листинге 4 представлена часть этой функции.

Листинг 4 – Получение пакета

```
1  if packet.is_init() {
2    dup_filter.init();
3  } else if !dup_filter.is_duplicate(packet.index() as usize) {
4    // обработка
5  }
```

3.4 Вывод

В технологической части были реализаны спроектированные функций.

4 Технологическая часть

Для тестирования реализованного протокола были выбраны следующие средства:

- iperf3 — утилита для генерации трафика;

- netem — утилита для изменения параметров канала.

На рисунке 7 показана настройка утилиты netem. В данном случае произведена настройка канала на 20% дублирования пакетов.

```
61f4b1c912:/home/py/rust/chagg# tc qdisc replace dev eth0 root netem duplicate 20
root@aa61f4b1c912:/home/py/rust/chagg# tc qdisc show
qdisc noqueue 0: dev lo root refcnt 2
qdisc netem 8001: dev eth0 root refcnt 2 limit 1000 duplicate 20%
```

Рисунок 7 – Настройка netem

На рисунке 8 показан результат работы утилиты iperf3 при выключенной дедупликации. Как видно из отчета работы, примерно 20% пакетов приходят вне очереди, что в данном случае показывает приход дублированных пакетов.

```
Server listening on 5201
-----------------------------------------------------------
Accepted connection from 10.0.0.1, port 48150
[  5] local 10.0.0.2 port 5201 connected to 10.0.0.1 port 47738
[ ID] Interval           Transfer     Bitrate         Jitter    Lost/Total Datagrams
[  5]   0.00-1.00   sec   153 KBytes  1.25 Mbits/sec  0.796 ms  0/95 (0%)
[  5]   1.00-2.00   sec   150 KBytes  1.23 Mbits/sec  1.012 ms  0/94 (0%)
[  5]   2.00-3.00   sec   145 KBytes  1.19 Mbits/sec  1.130 ms  0/95 (0%)
[  5]   3.00-4.00   sec   148 KBytes  1.21 Mbits/sec  1.103 ms  0/94 (0%)
[  5]   4.00-5.00   sec   155 KBytes  1.27 Mbits/sec  1.063 ms  0/94 (0%)
[  5]   5.00-6.00   sec   156 KBytes  1.28 Mbits/sec  0.764 ms  0/95 (0%)
[  5]   6.00-7.00   sec   156 KBytes  1.28 Mbits/sec  0.930 ms  0/94 (0%)
[  5]   7.00-8.00   sec   159 KBytes  1.30 Mbits/sec  0.838 ms  0/95 (0%)
[  5]   8.00-9.00   sec   155 KBytes  1.27 Mbits/sec  1.092 ms  0/94 (0%)
[  5]   9.00-10.00  sec   160 KBytes  1.31 Mbits/sec  0.784 ms  0/94 (0%)
[  5]  10.00-10.01  sec  2.71 KBytes  4.21 Mbits/sec  0.846 ms  0/1 (0%)
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bitrate         Jitter    Lost/Total Datagrams
[SUM]  0.0-10.0 sec  190 datagrams received out-of-order
[  5]   0.00-10.01  sec  1.50 MBytes  1.26 Mbits/sec  0.846 ms  0/945 (0%)  receiver
```

Рисунок 8 – Iperf3 без дедупликации

На рисунке 9 показан результат работы утилиты iperf3 при включенной дедупликации. Как видно из отчета работы, дублирование отсутствует, что значит, что реализованный алгоритм работает.

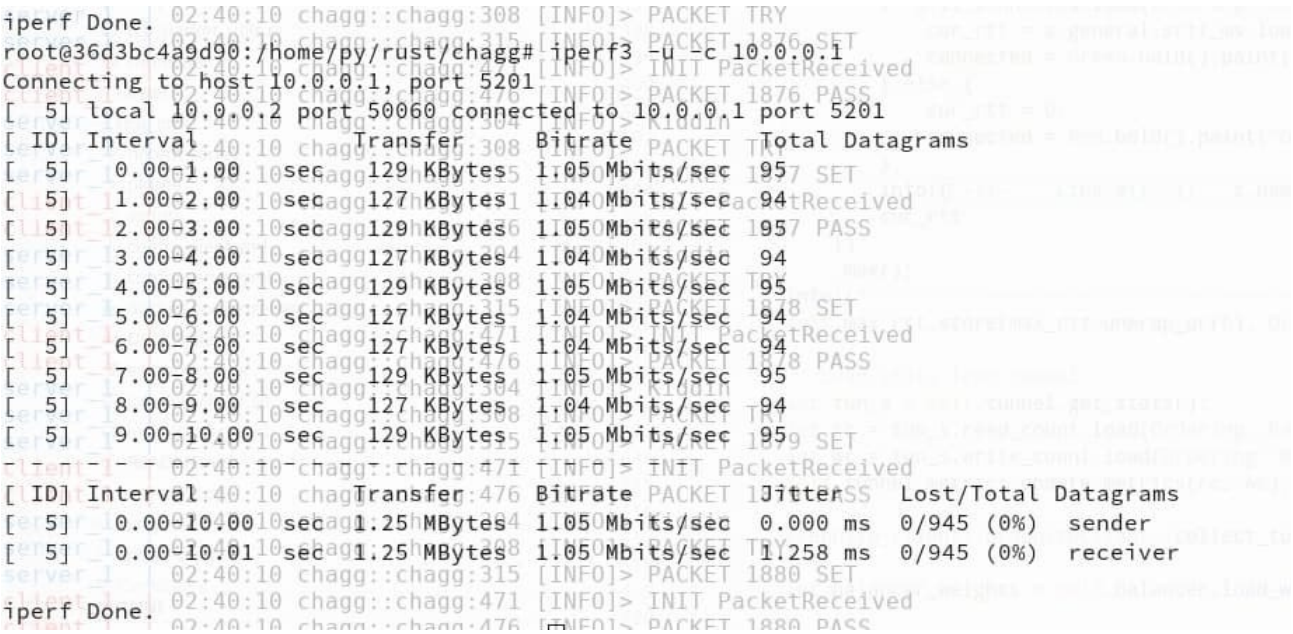

Рисунок 9 – Iperf3 со дедупликацией

Заключение

Список литературы

1. ANDREW S. TANENBAUM HERBERT BOS / Modern Operating Systems FOURTH EDITION [Электронный ресурс]479-480 (дата

2. TEST [Электронный ресурс] URL: https://www.cisco.com/c/en/us/products/colla nx-os-software/ios-netflow/prod_white_paper0900aecd80406232.html (дата обращения: 09.06.2021).

Рисунок 10 – Example images

$$timeout = RTT_{average} + 4dev_{average} \tag{6}$$

Листинг 5 – lst example