|  |  |
| --- | --- |
|  |  |
| МИНОБРНАУКИ РОССИИ |
| Федеральное государственное бюджетное образовательное учреждениевысшего профессионального образования«Московский технологический университет» МИРЭА |

Институт Информационных Технологий

Кафедра Корпоративных Информационных Систем

**ОТЧЕТ**

по Лабораторной Работе №3

на тему

«Потоковые и датаграммные сокеты»

по дисциплине

«Разработка клиент-серверных приложений»

Выполнил студент группы ИКБО-08-18 Валяев Д.А.

Принял старший преподаватель Мирзоян Д.И.

Выполнено «\_\_\_» \_\_\_\_\_\_\_\_\_ 2020 г.

Зачтено «\_\_\_» \_\_\_\_\_\_\_\_\_ 2020 г.

Москва 2020

**Задание**

Исследовать различия между потоковыми и датаграммными сокетами в поведении при передаче данных.

**Теоретическая часть**

Протокол UDP (User Datagram Protocol) является одним из основных протоколов, расположенных непосредственно над IP. Он предоставляет прикладным процессам транспортные услуги, немногим отличающиеся от услуг протокола IP. Протокол UDP обеспечивает доставку дейтограмм, но не требует подтверждения их получения. Протокол UDP не требует соединения с удаленным модулем UDP ("бессвязный" протокол). К заголовку IP-пакета UDP добавляет поля порт отправителя и порт получателя, которые обеспечивают мультиплексирование информации между различными прикладными процессами, а также поля длина UDP-дейтограммы и контрольная сумма, позволяющие поддерживать целостность данных. Таким образом, если на уровне IP для определения места доставки пакета используется адрес, на уровне UDP - номер порта.

Примерами сетевых приложений, использующих UDP, являются NFS (Network File System), TFTP (Trivial File Transfer protocol, RFC-1350), RPC (Remote Procedure Call, RFC-1057) и SNMP (Simple Network Management Protocol, RFC-1157). Малые накладные расходы, связанные с форматом UDP, а также отсутствие необходимости подтверждения получения пакета, делают этот протокол наиболее популярным при реализации приложений мультимедиа, но главное его место работы - локальные сети и мультимедиа.

Хотя протокол UDP не гарантирует доставки, по умолчанию предполагается, что вероятность потери пакета достаточно мала.

Прикладные процессы и модули UDP взаимодействуют через UDP-порты. Эти порты нумеруются, начиная с нуля. Прикладной процесс, предоставляющий некоторые услуги (сервер), ожидает сообщений, направленных в порт, специально выделенный для этих услуг. Программа-сервер ждет, когда какая-нибудь программа-клиент запросит услугу.

Например, сервер SNMP всегда ожидает сообщения, адресованного в порт 161. Если клиент snmp желает получить услугу, он посылает запрос в UDP-порт 161 на машину, где работает сервер. На каждой машине может быть только один агент SNMP, т.к. существует только один порт 161. Данный номер порта является общеизвестным, т.е. фиксированным номером, официально выделенным в сети Internet для услуг SNMP. Общеизвестные номера портов определяются стандартами Internet.

Данные, отправляемые прикладным процессом через модуль UDP, достигают места назначения как единое целое. Например, если процесс-отправитель производит 5 записей в порт, то процесс-получатель должен будет сделать 5 чтений. Размер каждого записанного сообщения будет совпадать с размером каждого прочитанного. Протокол UDP сохраняет границы сообщений, определяемые прикладным процессом. Он никогда не объединяет несколько сообщений в одно и не делит одно сообщение на части.

Протокол TCP (transmission control protocol) в отличии от UDP осуществляет доставку дейтограмм, называемых сегментами, в виде байтовых потоков с установлением соединения. Протокол TCP применяется в тех случаях, когда требуется гарантированная доставка сообщений. Он использует контрольные суммы пакетов для проверки их целостности и освобождает прикладные процессы от необходимости таймаутов и повторных передач для обеспечения надежности. Для отслеживания подтверждения доставки в TCP реализуется алгоритм "скользящего" окна. Наиболее типичными прикладными процессами, использующими TCP, являются FTP (File Transfer Protocol - протокол передачи файлов) и telnet. Кроме того, TCP используют системы SMTP, HTTP, X-window, RCP (remote copy), а также "r"-команды. Внутренняя структура модуля TCP гораздо сложнее структуры UDP. Подобно UDP прикладные процессы взаимодействуют с модулем TCP через порты. Под байтовыми потоками здесь подразумевается то, что один примитив, например, read или может вызвать посылку адресату последовательности сегментов, которые образуют некоторый блок данных (сообщение). Использование портов открывает возможность осуществлять несколько соединений между двумя сетевыми объектами (работать с разными процессами).

Предположим, что из точки А передаются данные в точку Б и эти точки непосредственно связаны каналом с пропускной способностью В, которая больше максимальной скорости передачи узла А (FA). Пусть узел Б способен принимать данные со скоростью FБ. Если узел А будет передавать данные со скоростью FБ, потерь не будет. Но если А не знает значения FБ, он будет пытаться осуществлять передачу со скоростью FA. Это в конце концов приведет к переполнению буфера в узле Б и к потере пакетов. Отсутствие подтверждения получения пакета уведомит узел А о том, что скорость передачи слишком велика.

Нужно учитывать, что потерей пакета будет считаться, как вариант потери информационного сегмента, так и потеря отклика на него. Эти два варианта не различимы.

Проблема может быть решена методом итеративного алгоритма выбора скорости передачи с использованием детектирования отсутствия подтверждений или путем присылки узлом Б уровня заполнения его буфера. Второй вариант представляется на первый взгляд более привлекательным. Беда в том, что переполнение буфера и потери пакетов могут происходить не только в узле Б, но в любом другом промежуточном узле. Причем переполнение может происходить за счет какого-то конкурирующего трафика, источником которого не является узел А. Для задания скорости передачи отправителя был создан механизм “медленного” старта, который определял значения окна перегрузки (cwnd) и порога медленного старта (ssthresh).

**Алгоритм решения задачи**

1. Разместить на форме поле ввода, кнопку и 2 поля для вывода результатов.
2. Реализовать следующие четыре сокета: потоковый на прием, потоковый на передачу, датаграммный на прием, датаграммный на передачу.
3. Реализовать прием и передачу данных.
4. Подобрать размер пакета и задержку на таймере чтобы размеры принятых фрагментов в случае потоковой и датаграммной передачей не совпадали.

**Тестирование**

Проверка работоспособности приложения. Как видно принятые фрагменты в случае потоковой и датаграммной передачей не совпадают



Рис. 1 – Отправка пакетов на 127.0.0.1

**Заключение**

В результате лабораторной работы была реализована передача и прием сетевых пакетов с помощью потокового и датаграммного сокетов.

**Исходный код**

****

using System;

using System.Collections.Generic;

using System.Net;

using System.Net.Sockets;

using System.Windows.Forms;

namespace labNet3

{

 public partial class Form1 : Form

 {

 int packet\_size = 256;

 static byte[] buffer;

 private Socket tcp, udp;

 private Socket trans\_tcp, trans\_udp;

 List<Socket> receivers = new List<Socket>();

 public Form1()

 {

 InitializeComponent();

 }

 private void Form1\_Load(object sender, EventArgs e)

 {

 buffer = new byte[packet\_size];

 timer2.Enabled = true;

 tcp = new Socket(SocketType.Stream, ProtocolType.IP);

 udp = new Socket(SocketType.Dgram, ProtocolType.IP);

 trans\_tcp = new Socket(SocketType.Stream, ProtocolType.IP);

 trans\_udp = new Socket(SocketType.Dgram, ProtocolType.IP);

 tcp.Bind(new IPEndPoint(IPAddress.Any, 2002));

 udp.Bind(new IPEndPoint(IPAddress.Any, 2003));

 tcp.Listen(5);

 }

 private void button1\_Click(object sender, EventArgs e)

 {

 IPAddress destination = IPAddress.Parse(textBox1.Text.Trim());

 trans\_tcp.Connect(new IPEndPoint(destination, 2002));

 trans\_udp.Connect(new IPEndPoint(destination, 2003));

 timer1.Enabled = true;

 }

 private void timer1\_Tick(object sender, EventArgs e)

 {

 trans\_tcp.Send(buffer, packet\_size, SocketFlags.None);

 trans\_udp.Send(buffer, packet\_size, SocketFlags.None);

 packet\_size \*= 2;

 buffer = new byte[packet\_size];

 if (packet\_size > 32768)

 {

 packet\_size = 256;

 Stop();

 }

 }

 private void timer2\_Tick(object sender, EventArgs e)

 {

 if (tcp.Poll(0, SelectMode.SelectRead))

 {

 receivers.Add(tcp.Accept());

 }

 if (udp.Poll(0, SelectMode.SelectRead))

 {

 int bytes = udp.Receive(buffer);

 textBox3.AppendText(String.Format("Received {0} bytes\r\n", bytes));

 }

 for (int i = 0; i < receivers.Count; i++)

 {

 var receiver = receivers[i];

 if (receiver.Poll(0, SelectMode.SelectRead))

 {

 int bytes = receiver.Receive(buffer);

 if (bytes == 0)

 {

 receivers.Remove(receiver);

 i--;

 }

 else

 {

 textBox2.AppendText(String.Format("Received {0} bytes\r\n", bytes));

 }

 }

 }

 }

 private void Stop()

 {

 timer1.Enabled = false;

 trans\_tcp.Close();

 trans\_udp.Close();

 trans\_tcp = new Socket(SocketType.Stream, ProtocolType.IP);

 trans\_udp = new Socket(SocketType.Dgram, ProtocolType.IP);

 }

 }

}