

**计算机网络**

**课程实验报告**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 实验名称 | 可靠数据传输协议-GBN协议的设计与实现 | | | | | |
| 姓名 | 冯运 | | 院系 | 计算机学院 | | |
| 班级 | 1603108 | | 学号 | 1160300524 | | |
| 任课教师 | 聂兰顺 | | 指导教师 | 聂兰顺 | | |
| 实验地点 | 格物楼207 | | 实验时间 | 2018.11.7 | | |
| 实验课表现 | 出勤、表现得分(10) |  | 实验报告  得分(40) |  | 实验总分 |  |
| 操作结果得分(50) |  |
| 教师评语 | | | | | | |
|  | | | | | | |

****

|  |
| --- |
| 实验目的： |
| 理解滑动窗口协议的基本原理；掌握GBN的工作原理；掌握基于UDP设计并实现一个GBN协议的过程与技术。 |
| 实验内容： |
| 1. 基于UDP设计一个简单的GBN协议，实现单向可靠数据传输（服务器到客户的数据传输）。 2. 模拟引入数据包的丢失，验证所设计协议的有效性。 3. 改进所设计的GBN协议，支持双向数据传输； 4. 将所设计的GBN协议改进为SR协议。 |
| 实验过程： |
| 要实现一个GBN协议,就需要先了解GBN协议的数据传输流程。   1. GBN的原理与实现   原理:  GBN是属于传输层的协议,它负责接收应用层传来的数据,将应用层的数据报发送到目标IP和端口  滑动窗口: 假设在序号空间内，划分一个长度为N的子区间，这个区间内包含了已经被发送但未收到确认的分组的序号以及可以被立即发送的分组的序号，这个区间的长度就被称为窗口长度。（随着发送方方对ACK的接收，窗口不断的向前移动，并且窗口的大小是可变的）  GBN一个分组的发送格式是 Base(1Byte) + seq(1Byte) + data(max 1024Byte)  GBN协议的传送流程是: 从上层应用层获得到一个完整的数据报,将这个数据报进行拆分(一个GBN数据帧最大传输的数据大小限制为1024B,因为在以太网中，数据帧的MTU为1500字节，所以UDP数据报的数据部分应小于1472字节（除去IP头部20字节与UDP头的8字节）),如果发送方的滑动窗口中,如果窗口内已经被发送但未收到确认的分组数目未达到窗口长度,就将窗口剩余的分组全部用来发送新构造好的数据,剩余未能发送的数据进行缓存。发送完窗口大小的数据分组后,开始等待接收从接收方发来的确定信息(ACK),GBN协议采取了累积确认，当发送方收到一个对分组n的ACK的时候，即表明接收方对于分组n以及分组n之前的分组全部都收到了。对于已经确认的分组,就将窗口滑动到未确认的分组位置(窗口又有空闲位置,可以发送剩余分组了),对于未确认的分组,如果计时器超时,就需要重新发送,直到收到接收方的ACK为止。  对于超时的触发，GBN协议会将当前所有已发送但未被确认的分组重传，即如果当前窗口内都是已发送但未被确认的分组，一旦定时器发现窗口内的第一个分组超时，则窗口内所有分组都要被重传。每次当发送方收到一个ACK的时候，定时器都会被重置。  接收方只需要按序接收分组，对于比当前分组序号还要大的分组则直接丢弃。假设接收方正在等待接收分组n，而分组n＋1却已经到达了，于是，分组n＋1被直接丢弃，所以发送方并不会出现在连续发送分组n，分组n＋1之后，而分组n＋1的ACK却比分组n的ACK更早到达发送方的情况。  这是BGN的有穷状态机的示意图    实现:  发送方:  首先定义窗口大小,起始 base 的值, 窗口采用链表的数据结构存储  private int WindowSize = 16;  private long base = 0;  进入一个循环,循环结束条件是所有需要传送的数据都已经发送完成,并且窗口中的分组都已经全部确认。  在这个循环中,如果窗口内有空余,就开始发送分组,直到窗口被占满,计时器开始计时,之后进入接收ACK的状态,收到ACK之后,更新滑动窗口的位置,之后如果计时器超时,就将窗口内所有的分组全部重发一次。之后开始下一次循环。  接收方:  不需要有缓存,只需要记录一个seq值,每成功接收一个数据帧,seq+1,开始循环顺序接收数据帧,对于seq不是目标值得数据帧直接丢弃,如果是符合要求的数据帧,就给发送方发送一个ACK=seq的确认数据帧,直到发送方没有数据传来为止。  GBN的实现就完成了。   1. SR协议的原理与实现:   SR协议的原理:  SR协议是在GBN协议的基础上进行的改进。  对于SR协议来说，发送方需要做到：  为每一个已发送但未被确认的分组都需要设置一个定时器，当定时器超时的时候只发送它对应的分组。  当发送方收到ACK的时候，如果是窗口内的第一个分组，则窗口需要一直移动到已发送但未未确认的分组序号。  对于接收方,需要设置一个窗口大小的缓存,即使是乱序到达的数据帧也进行缓存,并发送相应序号的ACK, 并及时更新窗口的位置,窗口的更新原则同发送方。  SR协议实现:  发送方:  在GBN发送方的基础上,增加一个基于链表数据结构的计时器,对每一个未被确认的分组进行计时。在每次判断是否超时时,需要对链表中所有的计时进行判断,与GBN重传不同的是,SR只对超时的那一个分组进行重传。  发送方完整代码见附录 SR.java中的void send(byte[] content) 函数  接收方:  需要增加一个同发送方的对分组的缓存,用于缓存乱序到达的分组,同样使用链表数据结构。  List<ByteArrayOutputStream> datagramBuffer = new LinkedList<>();  首先进入一个循环, 一次循环需要进行如下工作:  接收分组,将分组的数据缓存到datagramBuffer对应的位置(因为到达的数据可能是乱序的)  然后发送数据分组对应seq的ACK,告知发送方自己已经成功接收。 之后更新滑动窗口的位置,更新的规则同发送方一样。之后进行下一次循环。  直到发送方没有新的数据传来,超过接收方设定的最大时间,就结束循环,将接收到的数据拼接成一个完整的Byte数组,传给应用层。  接收方的完整代码见附录 SR.java中的 ByteArrayOutputStream receive() 函数  SR协议的实现就完成了。  三、双向传输的实现:  发送方发送数据需要占用一个固定的端口,而接收方也需要一个固定的端口来向发送方发送 ACK,所以就可以封装一个完整的协议类,类似于TCP的有连接传输一样,发送方和接收方之间在两个固定的ip和端口之间进行数据的传输,直到双方的传输结束。发送方在使用send()函数进行发送时,也可以同时使用receive()函数进行接收,两个过程并不冲突,可以同时进行。如果要同时收发,就需要同时开一个发送线程和一个接收线程,两个线程独立运行,没有冲突,这样就可以实现双向数据传输了。  所以我构造了一个SR class,其中包含的成员变量有:  private InetAddress host;  private int targetPort, ownPort;  private int WindowSize = 16;  private final int sendMaxTime = 2, receiveMaxTime = 4; // max time for one datagram  private long base = 0;  private final int virtualLossRemainder = 17; // this value is used to simulate the loss of the datagram as a remainder  包含的函数有两个:  void send(byte[] content) 负责数据的发送  ByteArrayOutputStream receive() 负责数据的接收  private ByteArrayOutputStream getBytes(List<ByteArrayOutputStream> buffer, long max) 负责将接收到的数据分组拼接成一个完整的数据报  private boolean checkWindow(List<Integer> timers) 负责判断当前的窗口是否可以移动  详细的代码见附件中SR.java  在Client 主函数中先使用SR协议发送一张图片, 在Server 主函数中使用SR协议接收这张图片,并保存。然后向Client发送另一张图片, Client由发送变成接收。  这有就可以实现双向文件的发送和接收了。  详细的代码见附件中 Client.java 中 main函数和Server.java中的main函数  四、模拟丢包  在接收端,设立一个计数变量count, 然后每次收到数据帧就加一,如果count 对一个数取余=0就不发送ACK,模拟这一分组丢失的情况,然后测试发送方会不会重新发送丢失的分组。  这一部分的代码实现详见附录中 SR.java中 receive中 count这个变量。 |
| 实验结果： |
| 采用演示截图、文字说明等方式，给出本次实验的实验结果。  刚开始 Client 程序向 Server程序发送一个图片文件  Client 程序的表现:    Server 中的表现    之后Server.java 向Client发送另一张图片  Server中的表现;    Client中的表现:    由于第二个文件比较大,其中出现了模拟丢包的情况:    最后文件发送成功,两个程序都结束运行。    最后运行结束之后文件夹中的截图:  其中1.png和3.png是原图,2.png和4.png是接收到的图片。 |
| 问题讨论： |
| 对于在实现SR协议过程中数据结构的设计,尝试了两种不同的实现方法。第一种是使用数组存储窗口中的分组,另一种是使用链表存储窗口中的分组。  在滑动窗口的过程中,需要的是当窗口中的分组已经有确认的,则需要把窗口滑动到序号最小的一个未被确认的分组位置,需要把已经确认的分组数据缓存清除,则需要队列的入队和出队操作,需要频繁增删元素,如果使用链表,增删元素操作就很方便而且开销小,使用数组就不能够删除元素,所以我最终选择了使用链表实现队列的功能。 |
| 心得体会： |
| 通过自己实现GBN协议和SR协议,我充分理解了滑动窗口协议的基本原理,掌握了GBN的工作原理,掌握了如何基于UDP设计并实现一个GBN协议的过程与技术,学会了自己实现一个SR协议。同时对UDP的网络编程有了足够的实践,掌握了网络编程的基本方法和调试错误的方法。 |
| 附录: |
| SR.java 完整代码 |
| package Lab2;  import java.io.ByteArrayOutputStream;  import java.io.IOException;  import java.net.\*;  import java.util.\*;  import java.util.LinkedList;  class SR {  private InetAddress host;  private int targetPort, ownPort;  private int WindowSize = 16;  private final int sendMaxTime = 2, receiveMaxTime = 4; // max time for one datagram  private long base = 0;  private final int virtualLossRemainder = 17; // this value is used to simulate the loss of the datagram as a remainder  SR(String host, int targetPort, int ownPort) throws UnknownHostException {  this.ownPort = ownPort;  this.host = InetAddress.getByName(host);  this.targetPort = targetPort;  }  /\*\*  \* transport content data to host:targetPort  \*  \* @param content the content data to transport  \* @throws IOException  \*/  void send(byte[] content) throws IOException {  int sendIndex = 0, length;  final int MAX\_LENGTH = 1024;  DatagramSocket datagramSocket = new DatagramSocket(ownPort);  List<ByteArrayOutputStream> datagramBuffer = new LinkedList<>(); // window buffer,used to resent the data  List<Integer> timers = new LinkedList<>();  long sendSeq = base;  do {  while (timers.size() < WindowSize && sendIndex < content.length && sendSeq < 256) { // until the window is run up  timers.add(0);  datagramBuffer.add(new ByteArrayOutputStream());  length = content.length - sendIndex < MAX\_LENGTH ? content.length - sendIndex : MAX\_LENGTH;  ByteArrayOutputStream oneSend = new ByteArrayOutputStream();  byte[] temp = new byte[1];  temp[0] = new Long(base).byteValue();  oneSend.write(temp, 0, 1);  temp = new byte[1];  temp[0] = new Long(sendSeq).byteValue();  oneSend.write(temp, 0, 1);  oneSend.write(content, sendIndex, length);  DatagramPacket datagramPacket = new DatagramPacket(oneSend.toByteArray(), oneSend.size(), host, targetPort);  datagramSocket.send(datagramPacket);  datagramBuffer.get((int) (sendSeq - base)).write(content, sendIndex, length);  sendIndex += length;  System.out.println("send the datagram : base " + base + " seq " + sendSeq);  sendSeq++;  }  datagramSocket.setSoTimeout(1000);  DatagramPacket receivePacket;  try { // receive ACKs  while (!checkWindow(timers)) {  byte[] recv = new byte[1500];  receivePacket = new DatagramPacket(recv, recv.length);  datagramSocket.receive(receivePacket);  int ack = (int) ((recv[0] & 0x0FF) - base);  timers.set(ack, -1);  }  } catch (SocketTimeoutException e) { // out of time  for (int i = 0; i < timers.size(); i++) {  if (timers.get(i) != -1)  timers.set(i, timers.get(i) + 1);  }  }  for (int i = 0; i < timers.size(); i++) { // update timer  if (timers.get(i) > this.sendMaxTime) { // resend the datagram which hasn't receive ACK and over time  ByteArrayOutputStream resender = new ByteArrayOutputStream();  byte[] temp = new byte[1];  temp[0] = new Long(base).byteValue();  resender.write(temp, 0, 1);  temp = new byte[1];  temp[0] = new Long(i + base).byteValue();  resender.write(temp, 0, 1);  resender.write(datagramBuffer.get(i).toByteArray(), 0, datagramBuffer.get(i).size());  DatagramPacket datagramPacket = new DatagramPacket(resender.toByteArray(), resender.size(), host, targetPort);  datagramSocket.send(datagramPacket);  System.err.println("resend the datagram : base " + base + " seq " + (i + base));  timers.set(i, 0);  }  }  // slide the window if front datagram is acknowledged  int i = 0, s = timers.size();  while (i < s) {  if (timers.get(i) == -1) {  timers.remove(i);  datagramBuffer.remove(i);  base++;  s--;  } else {  break;  }  }  if (base >= 256) {  base = base - 256;  sendSeq = sendSeq - 256;  }  } while (sendIndex < content.length || timers.size() != 0); // until data has all transported  datagramSocket.close();  }  /\*\*  \* receive data from host:targetPort  \*  \* @return the received data  \* @throws IOException IO exception occur  \*/  ByteArrayOutputStream receive() throws IOException {  int count = 0, time = 0; // used to simulate datagram loss  long max = 0, receiveBase = -1;  ByteArrayOutputStream result = new ByteArrayOutputStream(); // store the received content  DatagramSocket datagramSocket = new DatagramSocket(ownPort); // UDP socket to receive datagram and send ACKs  List<ByteArrayOutputStream> datagramBuffer = new LinkedList<>(); // window buffer,used to store the datagram out of order  DatagramPacket receivePacket; // one temp datagram packet  datagramSocket.setSoTimeout(1000);  for (int i = 0; i < WindowSize; i++) {  datagramBuffer.add(new ByteArrayOutputStream());  }  while (true) {  // receive one datagram and send ACK  try {  byte[] recv = new byte[1500];  receivePacket = new DatagramPacket(recv, recv.length, host, targetPort);  datagramSocket.receive(receivePacket);  // simulate datagram loss when count%virtualLossRemainder ==0  if (count % virtualLossRemainder != 0) {  long base = recv[0] & 0x0FF;  long seq = recv[1] & 0x0FF;  if (receiveBase == -1)  receiveBase = base;  // slide the window  if (base != receiveBase) {  ByteArrayOutputStream temp = getBytes(datagramBuffer, (base - receiveBase) > 0 ? (base - receiveBase) : max + 1);  for (int i = 0; i < base - receiveBase; i++) {  datagramBuffer.remove(0);  datagramBuffer.add(new ByteArrayOutputStream());  }  result.write(temp.toByteArray(), 0, temp.size());  receiveBase = base;  max = max - (base - receiveBase);  }  if (seq - base > max) {  max = seq - base;  }  ByteArrayOutputStream recvBytes = new ByteArrayOutputStream();  recvBytes.write(recv, 2, receivePacket.getLength() - 2);  datagramBuffer.set((int) (seq - base), recvBytes);  // send ACK  recv = new byte[1];  recv[0] = new Long(seq).byteValue();  receivePacket = new DatagramPacket(recv, recv.length, host, targetPort);  datagramSocket.send(receivePacket);  System.out.println("receive datagram : base " + base + " seq " + seq);  }  count++;  time = 0;  } catch (SocketTimeoutException e) {  time++;  }  if (time > receiveMaxTime) { // check if the connect out of time  ByteArrayOutputStream temp = getBytes(datagramBuffer, max + 1);  result.write(temp.toByteArray(), 0, temp.size());  break;  }  }  datagramSocket.close();  return result;  }  /\*\*  \* splice the ByteArrays(datagram) to one ByteArray(one datagram)  \*  \* @param buffer the datagram in current window  \* @param max the max datagram  \* @return spliced datagram(ByteArray)  \*/  private ByteArrayOutputStream getBytes(List<ByteArrayOutputStream> buffer, long max) {  ByteArrayOutputStream result = new ByteArrayOutputStream();  for (int i = 0; i < max; i++) {  if (buffer.get(i) != null)  result.write(buffer.get(i).toByteArray(), 0, buffer.get(i).size());  }  return result;  }  /\*\*  \* check if it's ok to slide window  \*  \* @param timers the timer to mark the window datagram  \* @return boolean true-> it's ok to slide window ;false-> it can slide window  \*/  private boolean checkWindow(List<Integer> timers) {  for (Integer timer : timers) {  if (timer != -1)  return false;  }  return true;  }  } |
| Client.java 完整代码 |
| package Lab2;  import java.io.ByteArrayOutputStream;  import java.io.\*;  import java.io.IOException;  public class Client {  public static void main(String[] args) throws IOException, InterruptedException {  File file1 = new File("./src/Lab2/1.png");  File file2 = new File("./src/Lab2/4.png");  if (!file2.exists()) {  file2.createNewFile();  }  ByteArrayOutputStream byteArrayOutputStream = new ByteArrayOutputStream();  CloneStream(byteArrayOutputStream, new FileInputStream(file1));  SR client = new SR("localhost", 7070, 8080);  System.out.println("Start to send file 1.png to " + "localhost " + 7070);  client.send(byteArrayOutputStream.toByteArray());  System.out.println("\nStart to receive file 3.png from " + "localhost " + 7070);  while (true) {  byteArrayOutputStream = client.receive();  if (byteArrayOutputStream.size() != 0) {  FileOutputStream fileOutputStream = new FileOutputStream(file2);  fileOutputStream.write(byteArrayOutputStream.toByteArray(), 0, byteArrayOutputStream.size());  fileOutputStream.close();  System.out.println("Get the file ");  System.out.println("Saved as 4.png");  break;  }  Thread.sleep(50);  }  }  /\*\*  \* clone the input stream to a ByteArrayOutputStream object  \*  \* @param CloneResult the clone result of input stream  \* @param InputStream the input stream to be cloned  \* @throws IOException when read input stream, some exception occur  \*/  static void CloneStream(ByteArrayOutputStream CloneResult, InputStream InputStream) throws IOException {  byte[] buffer = new byte[1024];  int length;  while ((length = InputStream.read(buffer)) != -1) {  CloneResult.write(buffer, 0, length);  }  CloneResult.flush();  }  } |
| Server.java 完整代码 |
| package Lab2;  import java.io.\*;  public class Server {  public static void main(String[] args) throws IOException, InterruptedException {  File file1 = new File("./src/Lab2/2.png");  File file2 = new File("./src/Lab2/3.png");  if (!file1.exists()) {  file1.createNewFile();  }  SR server = new SR("localhost", 8080, 7070);  System.out.println("Start to receive file 1.png from " + "localhost " + 8080);  while (true) {  ByteArrayOutputStream byteArrayOutputStream = server.receive();  if (byteArrayOutputStream.size() != 0) {  FileOutputStream fileOutputStream = new FileOutputStream(file1);  fileOutputStream.write(byteArrayOutputStream.toByteArray(), 0, byteArrayOutputStream.size());  fileOutputStream.close();  System.out.println("Get the file ");  System.out.println("Saved as 2.png");  fileOutputStream.close();  break;  }  Thread.sleep(50);  }  ByteArrayOutputStream byteArrayOutputStream = new ByteArrayOutputStream();  Client.CloneStream(byteArrayOutputStream, new FileInputStream(file2));  System.out.println("\nStart to send file 3.png to " + "localhost" + 8080);  server.send(byteArrayOutputStream.toByteArray());  }  } |