

**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 | 高靖龙 | | 院系 | 计算机科学与技术学院 | | |
| 班级 | 3104 | | 学号 | 1161000309 | | |
| 任课教师 | 刘亚维 | | 指导教师 | 刘亚维 | | |
| 实验地点 | 213 | | 实验时间 | 2018.10.31 | | |
| 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： |
| 概述本次实验的主要内容，包含的实验项等。  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： |
| 以文字描述、实验结果截图等形式阐述实验过程，必要时可附相应的代码截图或以附件形式提交。   1. **GBN 协议数据分组格式** 编码:UTF-8   数据包格式：[“data”+\n+seqnum+checksum+data]   1. **确认分组格式** ACK格式:[“ACK”+\n+exceptseqnum+\n\n] 2. **各个域作用** 发送窗口N = 2   序号空间seqSpace = N + 1  序号参数searg = {"seqSpace": seqSpace, "N": N, "base": 0, "nextseqnum": 0, "cache": [DATAGRAM()] \* seqSpace, "timer": False,  "pend": Queue()} cache:发送窗口内未ACK分组的缓存 pend:由于发送窗口满而等待的分组的缓存   1. **协议两端程序FSM**      1. **数据分组丢失验证模拟方法** 接收方ACK模5丢失1个。这个和对发送方分组丢失的模拟是等价的。 2. **程序实现的主要类（或函数）及其主要作用。**   **class** DATAGRAM():  *#存储数据的结构*  **def** initSocket(host, port):  *# 初始化socket：产生并绑定socket到目的地址端口*  **def** gbn(host, port, thost, tport):  *# gbn主程序，facade设计模式。*  **def** procceedThread(s, q):  *# 处理命令的线程，是实质上的gbn管理器。这里将发送方和接收方结合起来了。  # 既可以当发送方，也可以当接收方*   1. **def** sendThread(s):  *# 发送方发送数据*   **def** timerThread(s):  *# 计时器*  **def** recACKandDataThread(s):  *# 以发送方角色接受ack，以接收方角色接受data发送ack的线程。*  **以下是SR协议相关内容：**   1. **SR协议数据分组格式**   编码:UTF-8  数据包格式：[“data”+\n+seqnum+checksum+data]   1. **确认分组格式** ACK格式:[“ACK”+\n+exceptseqnum+\n\n] 2. **各个域作用** *# 接受窗口* recvN = 2 *# 发送窗口* sendN = 3 *# 序号空间* seqSpace = recvN + sendN  *# 多线程参数 # cache：发送方，未ACK的分组的缓存 # timerlist：计时器列表，对应每一个发送窗口位置 # pend：发送方窗口满而等待发送的分组 # recvCache：接收方缓存 # recvOrder：接受方接受记录，成功接受的窗口位置* searg = {**"seqSpace"**: seqSpace, **"recvN"**: recvN, **"sendN"**: sendN,  **"base"**: 0, **"nextseqnum"**: 0,  **"cache"**: [DATAGRAM()] \* seqSpace,  **"timerlist"**: [-1] \* seqSpace,  **"pend"**: Queue(),  **"recvBase"**:0,  **"recvCache"**:[DATAGRAM()] \* seqSpace,  **"recvOrder"**:[**False**]\*seqSpace} 3. **协议两端程序流程** 4. **数据分组丢失验证模拟方法** 接收方ACK模5丢失1个。这个和对发送方分组丢失的模拟是等价的。 5. **程序实现的主要类（或函数）及其主要作用 def** sr(host, port, thost, tport):  *# sr主程序。facade设计模式的命令行。*   **def** procceedThread(s, q):  *# 实质上的主进程，即使发送方，也能当接收方。实现双向传输*  **def** sendThread(s):  *# 作为发送方式发送数据*  **def** recvThread(s):  *# 作为接收方时接收数据发送ACK，同时作为发送方接受ACK* |
| 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。  本次实验代码，GBN、SR均将发送方和接收方集成在一起达到双向传输的效果。  丢包使用模k丢失的方式模拟。   |  |  |  | | --- | --- | --- | | **GBN丢包模拟及双向发送** | Server start at: 127.0.0.1:8081  wait for connection...  Enter command:  **开始接受**  ACK:0 0 data： 测试0  ACK:1 1 data： 测试1  ACK:2 2 data： 测试2  ACK:0 0 data： 测试3  ACK:1 1 data： 测试4  ACK:2 2 data： 测试5  **(丢包)**  forbidACK:2 data： 测试5  ACK:2 2 data： 测试5  ACK:0 0 data： 测试6  **开始发送**  **send 测试**  Enter command:  Send: 测试0 nextseqnumTo: 1  ACKed: 0 base+To: 1  Send: 测试1 nextseqnumTo: 2  ACKed: 1 base+To: 2  Send: 测试2 nextseqnumTo: 0  ACKed: 2 base+To: 0  Send: 测试3 nextseqnumTo: 1  ACKed: 0 base+To: 1  Send: 测试4 nextseqnumTo: 2  ACKed: 1 base+To: 2  Send: 测试5 nextseqnumTo: 0  Timer: 0  Timer: 1  Timer: 2  Timer: 3  Timer: 4  Timer: 5  base: 2  nextseqnum: 0  Resend: 2 测试5  ACKed: 2 base+To: 0  Send: 测试6 nextseqnumTo: 1  ACKed: 0 base+To: 1 | Server start at: 127.0.0.1:8080  wait for connection...  Enter command:  **开始发送**  **send 测试**  Enter command:  Send: 测试0 nextseqnumTo: 1  ACKed: 0 base+To: 1  Send: 测试1 nextseqnumTo: 2  ACKed: 1 base+To: 2  Send: 测试2 nextseqnumTo: 0  ACKed: 2 base+To: 0  Send: 测试3 nextseqnumTo: 1  ACKed: 0 base+To: 1  Send: 测试4 nextseqnumTo: 2  ACKed: 1 base+To: 2  Send: 测试5 nextseqnumTo: 0  Timer: 0  Timer: 1  Timer: 2  Timer: 3  Timer: 4  Timer: 5  base: 2  nextseqnum: 0  **(重发)**  Resend: 2 测试5  ACKed: 2 base+To: 0  Send: 测试6 nextseqnumTo: 1  ACKed: 0 base+To: 1  **开始接收**  ACK:0 0 data： 测试0  ACK:1 1 data： 测试1  ACK:2 2 data： 测试2  ACK:0 0 data： 测试3  ACK:1 1 data： 测试4  ACK:2 2 data： 测试5  forbidACK:2 data： 测试5  ACK:2 2 data： 测试5  ACK:0 0 data： 测试6 | | **SR丢包模拟及双向发送** | Server start at: 127.0.0.1:8081  wait for connection...  Enter command:  **开始发送**  **send 测试**  Send: 测试0 nextseqnumTo: 1  ACKed:0 base+To: 1  Send: 测试1 nextseqnumTo: 2  ACKed:1 base+To: 2  Send: 测试2 nextseqnumTo: 3  ACKed:2 base+To: 3  Send: 测试3 nextseqnumTo: 4  ACKed:3 base+To: 4  Send: 测试4 nextseqnumTo: 0  Send: 测试5 nextseqnumTo: 1  ACKed:0 base+To: 4  **重发**  Resend: 测试4 nextseqnumTo: 1  Send: 测试6 nextseqnumTo: 2  ACKed:4 base+To: 1  ACKed:1 base+To: 2  Send: 测试7 nextseqnumTo: 3  ACKed:2 base+To: 3  Send: 测试8 nextseqnumTo: 4  ACKed:3 base+To: 4  **开始接受**  ACK:0 data： 测试0 recvbaseTo 1  ACK:1 data： 测试1 recvbaseTo 2  ACK:2 data： 测试2 recvbaseTo 3  ACK:3 data： 测试3 recvbaseTo 4  ACK:4 data： 测试4 recvbaseTo 0  **丢包**  forbidACK:4 data： 测试4  ACK:0 data： 测试5 recvbaseTo 1  ACK:4 data： 测试4 recvbaseTo 1  ACK:1 data： 测试6 recvbaseTo 2  ACK:2 data： 测试7 recvbaseTo 3  ACK:3 data： 测试8 recvbaseTo 3 | Server start at: 127.0.0.1:8080  wait for connection...  Enter command:  **开始接受**  ACK:0 data： 测试0 recvbaseTo 1  ACK:1 data： 测试1 recvbaseTo 2  ACK:2 data： 测试2 recvbaseTo 3  ACK:3 data： 测试3 recvbaseTo 4  ACK:4 data： 测试4 recvbaseTo 0  **丢包**  forbidACK:4 data： 测试4  ACK:0 data： 测试5 recvbaseTo 1  ACK:4 data： 测试4 recvbaseTo 1  ACK:1 data： 测试6 recvbaseTo 2  ACK:2 data： 测试7 recvbaseTo 3  ACK:3 data： 测试8 recvbaseTo 3  **开始发送**  **send 测试**  Enter command:  Send: 测试0 nextseqnumTo: 1  ACKed:0 base+To: 1  Send: 测试1 nextseqnumTo: 2  ACKed:1 base+To: 2  Send: 测试2 nextseqnumTo: 3  ACKed:2 base+To: 3  Send: 测试3 nextseqnumTo: 4  ACKed:3 base+To: 4  Send: 测试4 nextseqnumTo: 0  Send: 测试5 nextseqnumTo: 1  ACKed:0 base+To: 4  **重发**  Resend: 测试4 nextseqnumTo: 1  Send: 测试6 nextseqnumTo: 2  ACKed:4 base+To: 1  ACKed:1 base+To: 2  Send: 测试7 nextseqnumTo: 3  ACKed:2 base+To: 3  Send: 测试8 nextseqnumTo: 4  ACKed:3 base+To: 4 | |
| 问题讨论： |
| 对实验过程中的思考问题进行讨论或回答。   1. 使用FSM来表现设计不仅仅清晰，更容易将不同的设计更好的融合在一起，本次实验就将发送方和接收方集成在一起，作为一个FSM运行。 2. FSM特别适合多线程编程，定义不同的状态，完成状态间转换，特别的，FSM对如何避免竞争，合理安排共享资源访问有着天然的指导性。 |
| 心得体会： |
| 结合实验过程和结果给出实验的体会和收获。   1. Python3的socket编程相比python2麻烦不少，特别是在编码转换上，发送接收方都需要显示的编码解码。 2. Socket的阻塞、非阻塞模式切换能帮助我们更好的完成设计。 3. 注释非常重要， 4. Seq序列空间为程序编写带来复杂性。 |