

**计算机网络**

**课程实验报告**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 实验名称 | GBN协议的设计与实现 | | | | | |
| 姓名 | 梅智敏 | | 院系 | 计算学部软件工程 | | |
| 班级 | 1837101 | | 学号 | 1183710118 | | |
| 任课教师 | 李全龙 | | 指导教师 | 李全龙 | | |
| 实验地点 | 格物213 | | 实验时间 | 2020.11.7 | | |
| 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 | | | | | | |
|  | | | | | | |

**计算学部**

|  |
| --- |
| 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； * 附加掌握基于 UDP 设计并实现一个 SR 协议的过程与技术。 |
| 实验内容： |
| * 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） * 模拟引入数据包的丢失，验证所设计协议的有效性； * 改进所设计的 GBN 协议，支持双向数据传输（选作， 加分项）； * 将所设计的 GBN 协议改进为 SR 协议（选作， 加分项）。 |
| 实验过程： |
| 注意：由于停等协议只需要将GBN协议的发送窗口大小设置为1便可实现，故本实验直接展示GBN和SR的实现过程，不再赘述停等协议。   1. 首先展示各个分组的格式  * 数据分组报文      1. 其中Label的值为“data”字符串，表示这是个数据报分组 2. “$”作为各个域的分隔符 3. Sequence是分组的序列号  * ACK分组报文      1. Label的值为“ACK”字符串，表示这是个ACK报文分组 2. “$”和“Sequence”含义与数据报分组相同  * 结束表示分组报文      1. Label的值为“end”字符串，表示这是个结束标志分组 2. 结束报文分组只在数据传输结束时起作用，用来告诉对方所有数据都已经传输完毕 3. 接下来展示协议两端程序流程图  * GBN的服务器     服务器主要有2个任务：发送数据报文和接收ACK，并在内部设置一个计时器Clock，用来记录send\_base是否超时，若超时则需要重传所有空中分组。   * GBN的客户端     GBN的客户端任务相对轻松，只需要判断接收到的数据报文的序列号是否和自己当前所期望的值“exp\_seq”相同，若相同则构造ACK报文发送给服务器，否则直接丢弃。   * GBN双向传输   为实现GBN的双向传输，我将客户端的功能集成到服务器，从而构建出run\_as\_both函数；再创建2个子线程，执行函数均为run\_as\_both函数即可实现双向通信。  下面展示示意图：     * SR的服务器     SR服务器与GBN的主要区别在于：为每个分组单独设置一个Clock，当接收到ACK时便将对应分组标记为已接收；当遇到超时事件时，单独重发一个分组，而不是将全部的空中分组全部重发。   * SR的客户端     SR客户端与GBN的主要区别在于：在客户端也设置一个滑动窗口，当到达的数据报不在窗口范围内便不予理睬。反之，判断n和rcv\_base是否相同，若相同则滑动窗口并交付数据；若不同则将该数据报缓存起来，而不是像GBN那样直接丢弃，然后等待后面一起交付。   1. 数据包丢失模拟方法   本实验的模拟丢包是通过设置2个丢包阈值，再配合随机数生成函数来实现概率传输报文，从而模拟现实中丢包的情况。   * 模拟数据报文分组丢包  1. 首先在服务器的class中设置了数据报丢包阈值      1. 结束python中内置的random函数产生（0,1）区间中的随机数，若该随机数大于我们设定的丢包阈值，则构造数据报文分组进行发送；否则便不发送，这样便可以模拟出数据报文分组丢包的情况。      1. 当遇到Clock超时，需要重传空中的全部数据分组的时候，依然采用上面的方式来模拟数据包的丢失。      * 模拟ACK报文分组丢包  1. 同样，在客户端的Class中设定一个ACK丢包阈值      1. 当客户端需要发送ACK报文时，采用同前面一样的方式进行传输，以模拟ACK报文分组的丢失。      1. 程序主要类、方法的作用  * GBN\_Server类  1. getData方法 ：用于从txt文件中获取内容，存入send\_buf中，以便于后续传输   注意：本实验利用文件读写来模拟服务器接收到上层应用发送来的数据，以及客户端将接受到的数据传输给上层应用。  服务器所读入的文件如下：     1. sendData方法 ：服务器首先判断窗口内是否有空间，若有，则依据我们规定的格式构造数据报文分组发送给客户端，随后将next\_seq++      1. handle\_timeOut方法 ：处理计时器超时事件，会将目前空中的全部分组进行重发，并将计时器重置。      1. run\_as\_server方法 ：作为服务器的线程执行目标函数，既将前面的函数集成到一起，并不断检测是否接收到ACK，依据情况执行sendData方法和handle\_timeOut方法  * GBN\_Client类  1. writeDataToFile方法 ：将接收到的数据写入文件，模拟将数据交付给上层应用的功能      1. run\_as\_client方法 ：作为客户端的线程执行目标函数，核心功能就是检测当前收到的数据分组是否是自己当前所期望的，再分情况决定发送ACK还是直接丢弃数据分组      * SR类  1. slide\_send\_window方法：滑动服务器的发送窗口     一直滑动到未被标记为已接收的分组序号处   1. slide\_rcv\_window方法：滑动客户端的接收窗口      1. server\_run方法 ：作为SR的服务器运行函数，与GBN的主要区别在于为每个分组单独设置Clock，且接收到ACK时会将对应的分组标记为已接收，以便于后续滑动窗口      1. client\_run方法 ：作为SR的客户端运行函数，与GBN的主要不同在于设置了接收方的滑动窗口，若接收到的数据报不是rcv\_base，则先缓存起来等待一起交付，而不是像GBN一样直接丢弃      * Main函数 ：程序入口，开启2个子线程，分别执行服务器的运行函数和客户端的运行函数，从而实现彼此通信。 |
| 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。   1. 演示GBN单向传输功能  * 没丢包的情况   服务器读入的文件，共70行    服务器的传输记录：    客户端的传输记录：    客户端写入的文件内容：同样是70行，且对比后发现内容相同     * 有丢包的情况：仍然使用一样的读入文件   服务器的传输记录：    可见当窗口已满时，会等待；且数据报6超时的时候，会重发空中的所有分组，即6、7、8、9、10  客户端的写入文件：    依然成功！   1. 演示GBN双向传输功能 2. 首先在main函数中开启2个子线程，分别让主机1和主机2运行run\_as\_both函数      1. 主机1的传输记录     可见主机1既充当了服务器的角色，有充当类客户端的角色   1. 主机2的传输记录     可见主机2同样实现了服务器和客户端的功能，即实现了双向传输的功能   1. 演示SR传输功能  * 主机1的传输记录     可见当超时事件发生的时候，我们重传的只是单个分组，而不是GBN那样重传所有空中分组   * 主机2的传输记录     可见我们的客户端同样设置了滑动窗口，且当失序数据到达时先缓存，等待一起交付（伴随着窗口的连续滑动）   * 对比文件结果   通过对比服务器读入的文件内容以及客户端写入的文件内容，可以验证我们的SR协议是成功的！  结果如下 |
| 问题讨论： |
| * SR服务器与GBN的主要区别在于：   为每个分组单独设置一个Clock，当接收到ACK时便将对应分组标记为已接收；当遇到超时事件时，单独重发一个分组，而不是将全部的空中分组全部重发。   * SR客户端与GBN的主要区别在于：   在客户端也设置一个滑动窗口，当到达的数据报不在窗口范围内便不予理睬。反之，判断n和rcv\_base是否相同，若相同则滑动窗口并交付数据；若不同则将该数据报缓存起来，而不是像GBN那样直接丢弃，然后等待后面一起交付。   * 双向传输可以通过将服务器和客户端的功能集成到一起来实现 |
| 体会收获： |
| * 加深了对GBN协议的过程理解，以及双向传输的实现过程 * 理解了SR和GBN的主要区别在何处 * 掌握了使用python进行Socket编程的方法 * 熟悉了多线程编程方法，让我能够更容易地实现2端通信 * 掌握了基于UDP的通信过程，通过调用sendto(),recvfrom()等函数，我对UDP通信有了更深入的理解 |