哈爾濱工業大學
Harbin Institute of Technology

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	Cycleke		院系	计算学部		
班级			学号			
任课教师	刘亚维		指导教师	刘亚维		
实验地点	格物 207		实验时间	2020 年 11 月 7 日		
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：

理解可靠数据传输的基本原理；掌握停等协议的工作原理；掌握基于 UDP 设计并实现一个停等协议的过程与技术。

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：

1. 设计数据报文格式

在实验中，我们将报文分为两种：数据报文和确认报文。数据报文用于传输分组的数据，确认报文用于接收端确认数据报文的接收情况。

数据报文的格式如下：

MSG	空格	编号	空格	数据

数据报文以 MSG 字符串开始，后面接着数据的分组编号和数据，使用空格分隔。例如，传输的数据为 255，编号为 10，那么数据报文就是"MSG 10 255"。

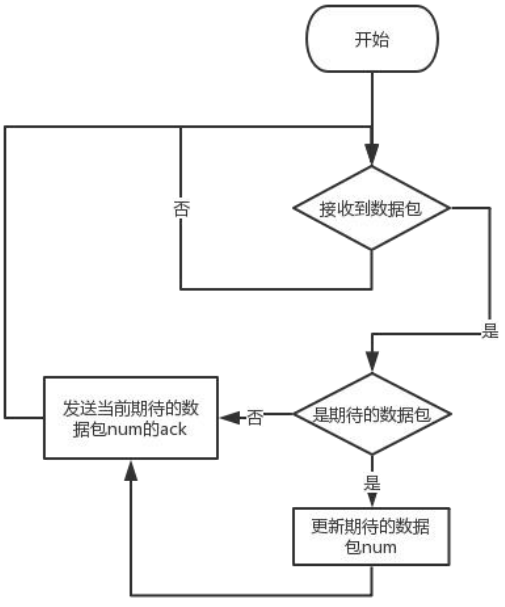
确认报文的格式如下：

ACK	空格	编号

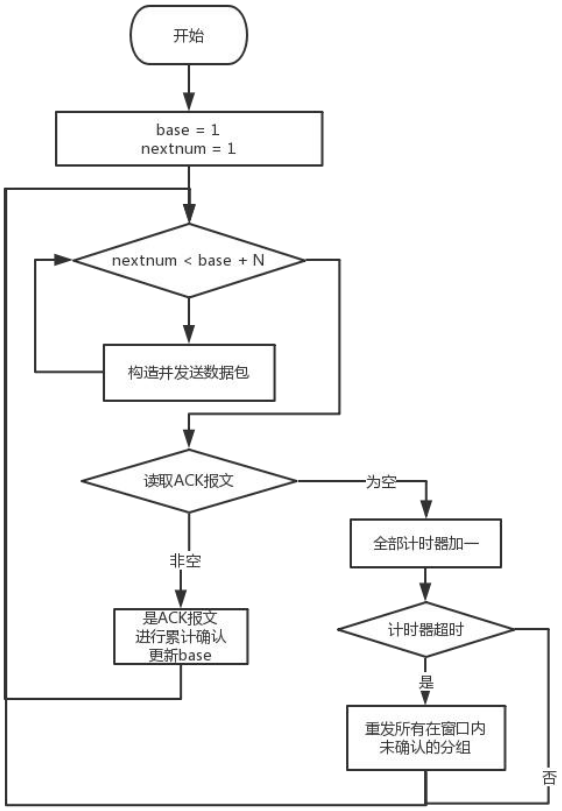
确认报文以 MSG 字符串开始，后面接着确认的编号信息，使用空格分隔。例如，在 GBN 协议中确认收到了分组 10 及以前的数据，那么的确认报文为"ACK 10"。

2. 协议两端程序流程图

GBN 协议接收端



GBN 协议发送端



3. 数据包模拟丢失

对于每种报文，我们在其对应的接收端模拟报文的丢失。对于数据报文，我们在协议的接收端模拟丢失；对于确认报文，我们在协议发送端模拟丢失。

在对应报文的接收端，我们在接收时随机产生一个 0 到 1 的随机数，若随机数低于阈值，则不对报文做处理，模拟该报文丢失。为了方便测试，接收端会在模拟丢失报文后会打印对应的日志。

4. GBN 协议和 SR 协议发送端实现

对于一个全双工通信的系统，其客户端和服务器其实是对称的，其逻辑可以拆分就是一个发送端和一个接收方。对于发送方，其核心逻辑就是将分组后的数据根据滑动窗口发送给接收方，直到所有的数据都成功发送并接受到对应的确定报文。

GBN 协议使用一个滑动窗口进行数据包的发送，当当前数据包发送完成后或者滑动窗口满时会进行 check。当所有数据发送完成后还需要进行 check，直到所有数据包都收到了对应的数据包，滑动窗口为空。

在 __check 方法中程序好接收确认报文并更新滑动窗口以及相关的变量。如果没有接收到报文，那么还需要进行超时判断，如果发生了超时，那么还需要将窗口中的数据重发。具体的实现如下：

```python
def __check(self, addr):
    """接收 ACK 报文并更新滑动窗口"""
    rlist, _, _ = select.select([self.__socket], [], [], 1)
    if rlist:
        msg_bytes, _ = self.__socket.recvfrom(BUFFER_SIZE)
        message: str = msg_bytes.decode()
        # 对确认报文处理
        if message.startswith("ACK "):
            messages = message.split()
            ack_number = int(messages[1])
            # 模拟 ACK 报文丢失
            if DEBUG_RANDOM_THROW_ON and random.random() < DEBUG_LOST_RATE:
                print("{} lose ack packet {}.".format(
                    self.__name, ack_number))
                return
            print("{} receive ACK message {}".format(self.__name, message))
            self.__base = (ack_number + 1) % SEQUENCE_LENGTH
            if self.__base == self.__next_seq_num:
                # 特殊值，表示停止计时
                self.__timer = -1
            else:
                self.__timer = time.time()
    elif self.__timer != -1 and time.time() - self.__timer > TIME_LIMIT:
        # 超时处理
        print("{} timeout on pkt {}.".format(self.__name, self.__base))
        self.__timeout(addr)

def __timeout(self, addr):
    """发送超时，将数据再次发送"""
    i = self.__base
    while i != self.__next_seq_num:
        self.__socket.sendto(self.__data_seq[i], addr)
        print("{} resend message {} to {}.\nMessage: {}".format(
            self.__name, i, addr, self.__data_seq[i].decode()))
        i = (i + 1) % SEQUENCE_LENGTH
    self.__timer = time.time()

def __send(self, addr, message: str):
    if not self.__in_window(self.__next_seq_num):
        # 滑动窗口已满，无法发送数据
        return False
    pkt = self.__make_msg_pkt(self.__next_seq_num, message)
    if self.__next_seq_num == self.__base:
        self.__timer = time.time()
    self.__socket.sendto(pkt, addr)
    print("{} send message {} to {}.\nMessage: {}".format(
        self.__name, self.__next_seq_num, addr, message))
    self.__data_seq[self.__next_seq_num] = pkt
    self.__next_seq_num = (self.__next_seq_num + 1) % SEQUENCE_LENGTH

    self.__check(addr)
    return True
```

对于 SR 协议的发送端，其大体同 GBN 协议类似，不同点主要在 __check 方法。SR协议需要对于每个数据包都维护了一个 ACK 标记列表和计时器列表。因为 SR 协议的确认报文是对于一个数据包的，而不是一段数据包。代码如下：

```python
def __check(self, addr):
    """接收 ACK 报文并更新滑动窗口"""
    rlist, _, _ = select.select([self.__socket], [], [], 1)
    if rlist:
        msg_bytes, _ = self.__socket.recvfrom(BUFFER_SIZE)
        message: str = msg_bytes.decode()
        # 对确认报文处理
        if message.startswith("ACK "):
            messages = message.split()
            ack_number = int(messages[1])
            # 模拟 ACK 报文丢失
            if DEBUG_RANDOM_THROW_ON and random.random() < DEBUG_LOST_RATE:
                print("{} lose ack packet {}.".format(
                    self.__name, ack_number))
                return
            print("{} receive ACK message {}".format(self.__name, message))
            if self.__in_window(ack_number):
                self.__ack_flags[ack_number] = True
        while self.__ack_flags[self.__base]:
            self.__ack_flags[self.__base] = False
            self.__data_seq[self.__base] = b'0'
            self.__timer[self.__base] = -1
            self.__base = (self.__base + 1) % SEQUENCE_LENGTH
    else:  # 未收到 ACK 报文
        i = self.__base
        while i != self.__next_seq_num:
            if not self.__ack_flags[i] and self.__timer[i] != -1:
                if time.time() - self.__timer[i] > TIME_LIMIT:
                    # 超时处理
                    print("{} timeout on pkt {}.".format(self.__name, i))
                    self.__socket.sendto(self.__data_seq[i], addr)
                    print(
                        "{} resend message {} to {}.\nMessage: {}".format(
                            self.__name, i, addr,
                            self.__data_seq[i].decode()))
                    self.__timer[i] = time.time()
            i = (i + 1) % SEQUENCE_LENGTH
```

5. GBN 协议和 SR 协议接收端实现

接收端的核心逻辑是以一个类似守护进程的方式不断接收发送端发送的报文并发送对应的确认报文。需要注意的是，即使接收到了所有的数据包，接收端仍需要运行一段的时间以响应发送端重发的部分数据包。

GBN 的接收端只维护一个期望收到的数据包的编号，每次收到编号后判断其是否与期望编号相同，并进行相应的更新。代码如下：

```python
def __receive(self, addr):
    if DEBUG_DELAY_ON:
        # 模拟网络延迟
        time.sleep(random.random())
    rlist, _, _ = select.select([self.__socket], [], [], 1)
    if not rlist:
        return ''
    msg_bytes, _ = self.__socket.recvfrom(BUFFER_SIZE)
    message: str = msg_bytes.decode()
    if not message.startswith("MSG "):
        return ''
    messages = message.split(maxsplit=2)
    pkt_number, data = int(messages[1]), messages[2]
    if DEBUG_RANDOM_THROW_ON and random.random() < DEBUG_LOST_RATE:
        # 模拟报文丢失
        print("{} lose packet {}.".format(self.__name, pkt_number))
        return ''
    if pkt_number == self.__expected_seq_num:
        print("{} send ack message {} to {}.".format(
            self.__name, pkt_number, addr))
        self.__sendpkt = self.__make_ack_pkt(self.__expected_seq_num)
        self.__expected_seq_num = (self.__expected_seq_num +
                                   1) % SEQUENCE_LENGTH
    else:
        print(
            "{} received unexpected message {}, resend ack message {} to {}."
            .format(self.__name, pkt_number, self.__expected_seq_num,
                    addr))
        data = ''

    self.__socket.sendto(self.__sendpkt, addr)
    return data
```

SR 的接收端也需要维护一个滑动窗口。每次在接收时会发送对应的确认报文并更新

ACK 标记列表。如果当前窗口的头部已经发送了确认报文，那么需要更新滑动数组。代码如下：

```python
def __receive(self, addr):
    if DEBUG_DELAY_ON:
        # 模拟网络延迟
        time.sleep(random.random())
    rlist, _, _ = select.select([self.__socket], [], [], 1)
    if not rlist:
        return ''
    msg_bytes, _ = self.__socket.recvfrom(BUFFER_SIZE)
    message: str = msg_bytes.decode()
    if not message.startswith("MSG "):
        return ''
    messages = message.split(maxsplit=2)
    pkt_number, data = int(messages[1]), messages[2]
    if DEBUG_RANDOM_THROW_ON and random.random() < DEBUG_LOST_RATE:
        # 模拟报文丢失
        print("{} lose packet {}.".format(self.__name, pkt_number))
        return ''
    ack_pkt = self.__make_ack_pkt(pkt_number)
    self.__socket.sendto(ack_pkt, addr)
    if (not self.__in_window(pkt_number)) or self.__ack_flags[pkt_number]:
        print("{} have received packet {}, resend ACK message {} to {}.".
              format(self.__name, pkt_number, ack_pkt.decode(), addr))
    else:
        print("{} send ack message {} to {}.".format(
            self.__name, pkt_number, addr))
        self.__ack_flags[pkt_number] = True
        self.__data_seq[pkt_number] = data

    data = ''
    while self.__ack_flags[self.__base]:
        data += self.__data_seq[self.__base]

        self.__data_seq[self.__base] = ''
        self.__ack_flags[self.__base] = False
        self.__base = (self.__base + 1) % SEQUENCE_LENGTH

    return data
```

实验结果：

1. 对于 GBN 协议全双工通信的测试

   具体运行结果见 gbn.log 文件，下面是日志的一部分：

```
ζ py gbn_host.py
Alice send message 0 to ('', 19269).
Message: Alice
Bob send message 0 to ('', 19267).
Message: Bob0
Alice send ack message 0 to ('', 19270).
Bob receive ACK message ACK 0
Bob send message 1 to ('', 19267).
Message: Bob1
Bob send ack message 0 to ('', 19268).
Alice receive ACK message ACK 0
Alice send message 1 to ('', 19269).
Message: EOF
Alice lose packet 1.
Bob send ack message 1 to ('', 19268).

##############
Finally Bob recevied message:
Alice
##############

Alice lose ack packet 1.
Bob send message 2 to ('', 19267).
Message: Bob2
Alice received unexpected message 2, resend ack message 1 to ('', 19270).
Bob receive ACK message ACK 0
Bob send message 3 to ('', 19267).
Message: Bob3
Alice received unexpected message 3, resend ack message 1 to ('', 19270).
Bob lose ack packet 0.
Bob send message 4 to ('', 19267).
Message: Bob4
Alice received unexpected message 4, resend ack message 1 to ('', 19270).
Bob lose ack packet 0.
Bob send message 5 to ('', 19267).
Message: Bob5
Alice received unexpected message 5, resend ack message 1 to ('', 19270).
Bob receive ACK message ACK 0
Bob send message 6 to ('', 19267).
Message: Bob6
```

2. 对于 SR 协议全双工通信的测试

   具体运行结果见 sr.log 文件，下面是日志的一部分：

```
Alice send message 0 to ('', 19269).
Message: Alice
Bob send message 0 to ('', 19267).
Message: Bob0
Bob send ack message 0 to ('', 19268).
Alice receive ACK message ACK 0
Alice send message 1 to ('', 19269).
Message: EOF
Alice send ack message 0 to ('', 19270).
Bob receive ACK message ACK 0
Bob send message 1 to ('', 19267).
Message: Bob1
Bob send ack message 1 to ('', 19268).

#############
Finally Bob recevied message:
Alice
#############

Alice receive ACK message ACK 1
Alice send ack message 1 to ('', 19270).
Bob receive ACK message ACK 1
Bob send message 2 to ('', 19267).
Message: Bob2
Alice send ack message 2 to ('', 19270).
```

问题讨论：

SR 协议和 GBN 协议的区别

我之前十分简单的认为 GBN 协议就是接收窗口为一的特殊的 SR 协议，但其实这是十分错误的。GBN 协议的 ACK 信息表示的是一段报文的确认而 SR 协议的 ACK 信息是对于单个报文的确认。GBN 协议无需接收方准备一定空间的缓存来储存分组，而 SR 协议可以减少重发次数。

心得体会：

通过实现GBN协议与SR协议的全双工通信，我对于可靠数据通信的认识有了提高，并且实际编程中，对于一些细节的处理是在理论的学习中无法得到的，比如序列号用完了如何解决，如何实现全双工通信等。

这次实验使得对于协议的理解更加深刻，体会到了计算机网络的魅力。