编程作业2：实现可靠传输协议

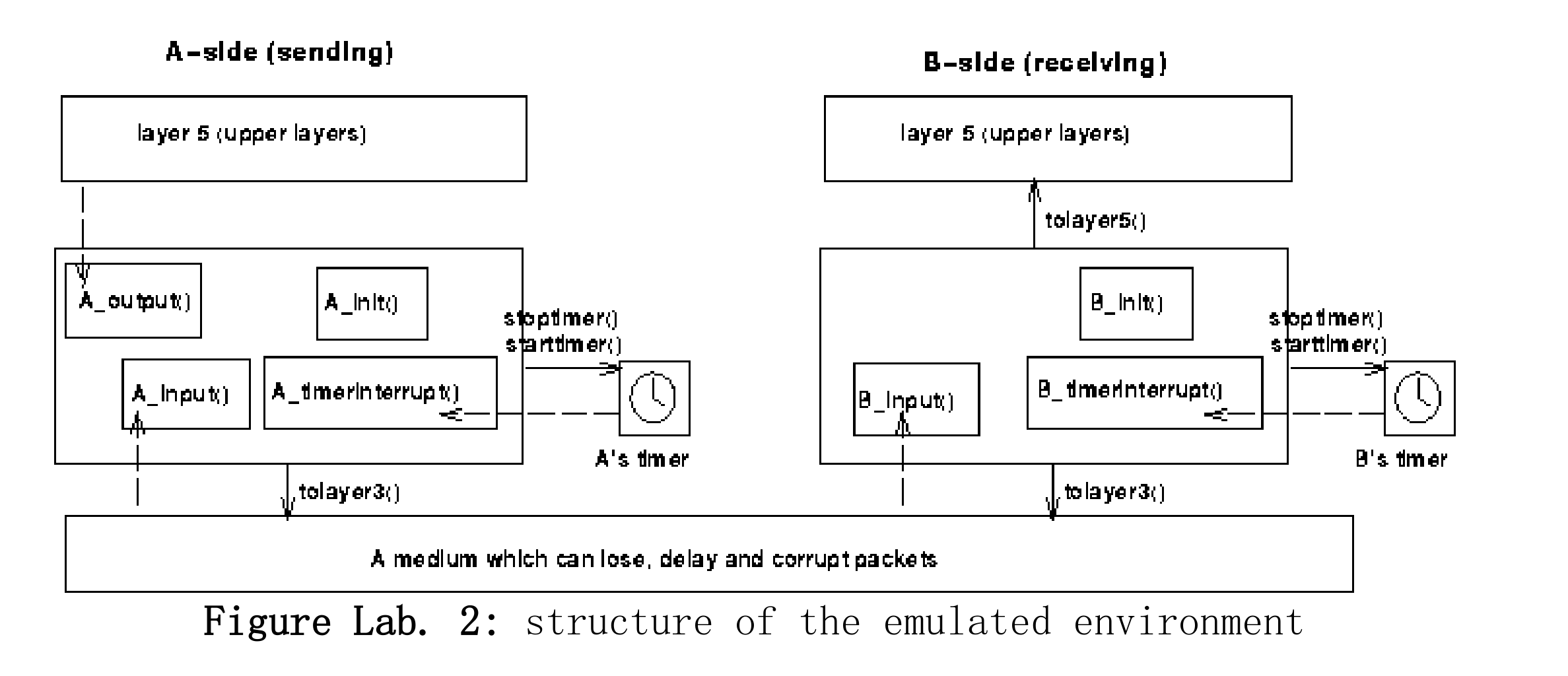
Overview

在本实验中，要求编写用于发送和接收的传输层代码以实现一个简单的可靠传输协议。本实验有两个版本：比特交换协议和回退N帧协议（GBN）。本次实验只要求做GBN。

考虑到你没有独立的计算机（有一个可以自主修改的系统），你的代码要在一个模拟的硬件/软件平台执行。编程接口为你提供了例程，代码将从上层和下层调用实体，这一过程与实际的UNIX环境下的运作很相似。实际上本次编程实验中的所用的软件接口要比其它实验中所用的无限循环的发送和接收端更真实。定时器的暂停/启动也要仿真，定时器中断会激活定时器执行程序。**编写好的代码要在安装好的mininet-VM环境下完成运行。在终端下运行mininet，使用xterm指令打开模拟终端，将代码在终端下编译并运行。**

你需要写的代码：

为发送实体A和接收实体B写程序。只要求由A到B的单向数据传输。当然，B也需要向A发送接收确认数据包，以便了解数据的接收情况。所编写的程序将与给出的仿真网络环境代码互相调用。环境的总体结构如图2所示：



在上层与可靠传输协议之间传输的数据单元是报文（message）,其定义格式为：

struct msg {

char data[20];

};

发送端从第5层接收20字节的数据；接收端正确接收20字节的数据并传递给接收端第5层。

在程序与网络层之间传输的数据单元是包，其声明格式为：

struct pkt {

int seqnum;

int acknum;

int checksum;

char payload[20];

};

程序要用第5层传递下来的报文填充payload字段。协议将利用其它字段来确保数据包的可靠传输。以下是代码编写的细节，这些代码是系统的一部分，并会被操作系统里的其它程序调用。

**A\_output(message),**message是个msg类型的结构，包含了将要发送到B的数据。在A有要发送的报文时，这个函数将被上层调用。编写要求是确保报文中的数据有序正确地传输到接收端上层。

**A\_input(packet)**,packet是pkt类型。当一个packet从B发送到A时，这个程序会被调用。从B发过来的数据包可能是被损坏的。

**A\_timerinterrupt()**当A的定时器超时，该程序将会被调用。该程序可以用于数据包的重传，下述的starttimer()和stoptimer()将会解释如何启动和停止定时器。

**A\_init()**这个程序最先被调用，其作用是进行必要的初始化。

**B\_input(packet)**,packet时一个pkt类型的结构。当数据包从A到达B，这个程序就会被调用。

**B\_init()**作用与A\_init()相同。

**软件接口：**

上面所述程序是要求自己编写的，下述程序是已经编写好的，可供调用：

**starttimer(calling\_entity,increment)**，其中calling\_entity为0（用于启动A侧计时器）或1（用于启动B侧计时器），而increment为浮点值，表示计时器中断之前经过的时间。 A的计时器只能由A侧例程启动（或停止），B侧计时器也应类似。 作为参考，当网络中没有其它数据包时，发送到网络的数据包平均需要5个时间单位才能到达另一端。

**stoptimer(calling\_entity)** ，其中calling\_entity为0（用于停止A侧计时器）或1（用于停止B侧计时器）。

**tolayer3(calling\_entity,packet)**，其中calling\_entity为0（对于A侧发送）或1（对于B侧发送），而packet是pkt类型的结构。 调用此例程将数据包发送到介质中，该数据包将发送给其他终端。

**tolayer5（calling\_entity，message）**，其中calling\_entity是0（用于将A端传递到第5层）或1（用于将B端传递到第5层），并且message是msg类型的结构。 使用单向数据传输时，仅在calling\_entity等于1（传递到B端）的情况下进行调用。 调用此例程将数据传递到第5层。

**模拟网络环境：**

调用tolayer3()会将数据包发送到介质中。当包从传输介质中到达协议层，A\_input()和B\_input()将会被调用。该介质能够破坏和丢失分组。 它不会重新排序数据包。 当运行整体程序时，终端会要求输入确定的值来设置网络环境。

**Number of messages to simulate.**当传递完该指定数量的报文，无论报文正确传输与否，程序都将会停止运行。注意，若将该值设置为1，程序会在数据到达接收端之前终止。因此该值应当大于1.

**Loss.**指定丢包率，0.1表示平均十分之一的数据包会丢失。

**Corruption**。 设置包错误率。值0.2表示平均五分之一的数据包被破坏。 请注意，有效内容，序列，确认或校验和字段的内容可能会损坏。 因此校验和应包括数据，序列和确认字段。

**Tracing**。 将追踪值设置为1或2将打印出有关仿真内部正在发生的事情的有用信息（例如，数据包和计时器正在发生的事情）。 追踪值0将关闭此功能。 大于2的跟踪值会显示无意义信息，该信息是用于调试仿真的。

**Average time between messages from sender's layer5.**可以将该值设为任意非0正数，设置值越小，发送的数据包就越快到达。

The Alternating- - Bit- - Protocol Version of this lab.

编写例程A\_output（），A\_input（），A\_timerinterrupt（），A\_init（），B\_input（）和B\_init（），这些例程将共同实现停止等待（即交换比特协议）， 我们在本文中将其称为rdt3.0）从A端到B端的单向数据传输。 协议应同时使用ACK和NACK报文。

发送端第五层发送报文间隔的平均时间应选择一个非常大的值，以使发送端在尝试发送给接收端的报文中仍然有未确认的未确认报文时，不会对其进行调用。建议选择值1000。还应该检查发送方，以确保在调用A\_output（）时，当前没有正在传输的报文。 如果存在，可以简单地忽略（丢掉）传递给A\_output（）例程的数据。

写好的例程放在prog2.c文件中。此文件的初始版本中包含了编写好的仿真例程。

上交代码清单，设计文档和示例输出。 对于示例输出，只要发送端或接收端发送数据（报文/数据包到达或计时器中断）以及作为响应而采取任何措施，例程就可能打印出一条消息。数据设置为要求接收端正确确认10条消息，丢失概率为0.1，损坏概率为0.3，跟踪级别为2。

The Go- - Back- - N version of this lab

编写A\_output(),A\_input(),A\_timerinterrupt(),A\_init(),B\_input(), B\_init()，用于实现从A到B的GBN单向数据传输，窗口大小为8.协议要求使用ACK和NACK报文。

强烈建议先实现更简单的交换比特实验，再扩展代码实现更难一些的GBN实验。

**A\_output(message)**, message是msg类型的结构,包含了要发送到B的数据。

当介质中有未解决的、未确认消息时，有时会调用A\_output（）例程-这意味着必须在发送端中缓冲多个消息。 另外，由于Go-Back-N的性质，也需要在发送端中进行缓冲：有时会调用发送端，但由于新消息不在窗口之内，因此它无法发送新消息。可以在发送端使用一些有限的最大缓冲区数（比如50个报文），并使发送端可以简单地中止（放弃并退出） 同时使用的所有50个缓冲区。使用下文给定的值，不会发生这种情况。

**A\_timerinterrupt（）**当A的计时器到期（从而生成计时器中断）时，将调用此例程。由于只有一个计时器，并且介质中可能包含许多未完成的未确认数据包，因此必须考虑如何使用该单个计时器。

上交代码清单，设计文档和示例输出。对于示例输出，只要发送端或接收端发送数据（报文/数据包到达或计时器中断）以及作为响应而采取任何措施，例程就可能打印出一条消息。发送端至少成功传输20条报文（即，发送端收到这些消息的ACK），丢包率为0.2，损坏概率为0.2，追踪级别为2，平均到达间隔10。要在代码部分中用彩色字体注明丢包和损坏是如何恢复的。

**扩展：**你可以实现数据的双向传输。 在这种情况下，实体A和B既充当发送者又充当接收者。 你也可以在数据包上附带确认（或者可以选择不这样做）。要使仿真器将报文从第5层传递到B\_output（）例程，你需要将BIDIRECTIONAL的声明值从0更改为1。

Helpful Hints and the like

校验和。你可以使用任何方法进行所需的校验和。但是序列号和ack字段也可能会损坏，建议使用类似TCP的校验和，该校验和由（整数）序列和ack字段值的总和组成，并添加到数据包有效负载字段的每个字符的总和中（即，将每个字符视为是8位整数，然后将它们加在一起）。

例程之间的任何共享”state”都必须采用全局变量的形式。例程需要从一个调用保存到下一个调用的任何信息也必须是全局（或静态）变量。 例如，例程将需要保留数据包的副本以进行可能的重传。但是如果发送端使用了全局变量之一，则接收端不应访问该变量，因为在现实生活中，仅通过通信通道连接的通信实体不能共享全局变量。

你可以在代码中访问一个浮点变量time，以帮助你检验信息。

**START SIMPLE.** 将丢失和损坏的可能性设置为零，然后测试一下例程。然后在无丢包，无损坏的情况下设计并运行程序。 然后处理以下情况：这些概率之一为非零，然后最终都是非零。

**Debugging.** 建议在调试过程时将跟踪级别设置为2，并在代码中放入大量的printf。

**Random Numbers**.仿真器使用随机数生成器来生成丢包率和错误率，就经验来看随机数生成器的取值差异较大，你可能需要调整提供的随机数生成代码。