

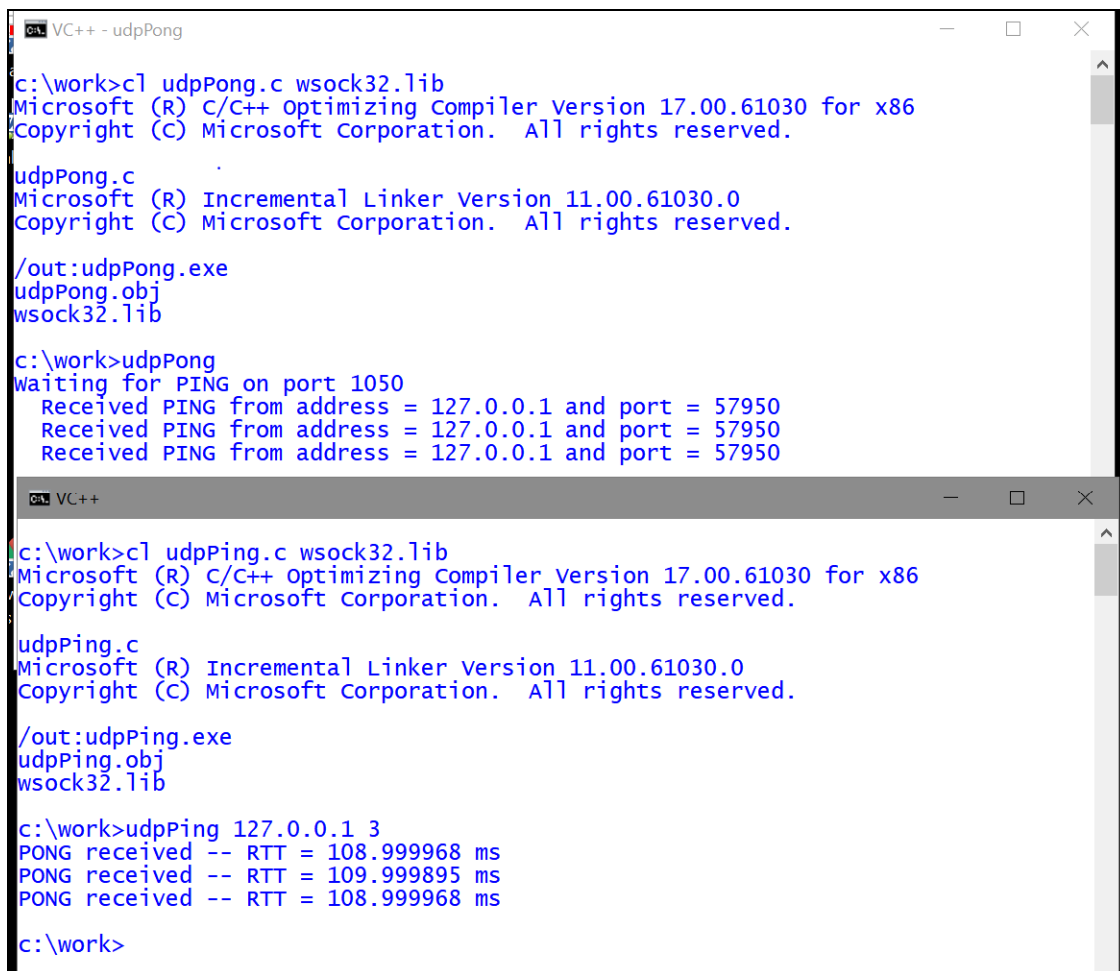
Assignment #3 for Computer Networks (CNT 4004) for Fall 2018

Due September 27, 2018 at the start of class

This assignment primarily covers material from chapters 2.7 (sockets) and 3 (through chapter 3.4 only) of the textbook and from class lecture. Each problem is worth 10 points.

Problem #1

Create a UDP ping program as follows. Write a UDP ping server (call it udpPong) that will listen on port 1050. When it receives a UDP message with “PING” in the data field, it should send a UDP message with “PONG” in the data field to the sender. Call the sender program udpPing. In the appendix are two program headers – one for udpPong and one for udpPing. Use these (note that they describe the command line parameters for udpPing). Note that you do not have to consider the case of the PONG message not received (that is, it is OK if your udpPing implementation hangs on the case of sending a PING and no PONG received). Here below is a screenshot showing operation. In this case, a 100 millisecond delay has been added to udpPong as described below to show that timing works.



```
VC++ - udpPong
c:\work>cl udpPong.c wsock32.lib
Microsoft (R) C/C++ Optimizing Compiler Version 17.00.61030 for x86
Copyright (C) Microsoft Corporation. All rights reserved.

udpPong.c
Microsoft (R) Incremental Linker Version 11.00.61030.0
Copyright (C) Microsoft Corporation. All rights reserved.

/out:udpPong.exe
udpPong.obj
wsock32.lib

c:\work>udpPong
waiting for PING on port 1050
Received PING from address = 127.0.0.1 and port = 57950
Received PING from address = 127.0.0.1 and port = 57950
Received PING from address = 127.0.0.1 and port = 57950

VC++
c:\work>cl udpPing.c wsock32.lib
Microsoft (R) C/C++ Optimizing Compiler Version 17.00.61030 for x86
Copyright (C) Microsoft Corporation. All rights reserved.

udpPing.c
Microsoft (R) Incremental Linker Version 11.00.61030.0
Copyright (C) Microsoft Corporation. All rights reserved.

/out:udpPing.exe
udpPing.obj
wsock32.lib

c:\work>udpPing 127.0.0.1 3
PONG received -- RTT = 108.999968 ms
PONG received -- RTT = 109.999895 ms
PONG received -- RTT = 108.999968 ms

c:\work>
```

Submit your source code and a screenshot (that should be very similar to the above) showing build and operation. Be ready to demo if the TA asks for a demo. To test your timing on udpPing, you can insert a delay in udpPong between PING receive and PONG send. For Windows, Sleep() can be used. Don't worry too much on the accuracy of timing (within +/- 10 ms is OK).

Problem #2

For the above problem, the case of PONG not returned was not considered. In reality, a ping program that cannot time-out on failure to receive a response is not very useful. Explain how you could fix your udpPing to have a 1 second time-out on receive. There are, no doubt, multiple possible solutions – full credit only for a solution that does not “burn” excessive CPU (so, no spin loops!) and is not overly complicated. A good solution must work on both Windows and Unix. **Hint:** You will have to dig a little into what other sockets functions are available. The solution was not given in class, it is also not in the book.

Problem #3

Use an FSM to describe the protocol rules for udpPing where a time-out occurs if a PONG message is not received. The FSM should also show what is output by udpPing to the console.

Problem #4

Consider a UDP receiver that receives a packet and determines that the checksum computed on the received packet matches the checksum carried in the checksum field. Is it now certain that no bit errors have occurred during the transmission of the packet? Explain.

Problem #5

Consider the RDT protocol developed in the book using RDT 2.2 receiver and RDT 3.0 sender.

- a) If the time-out is less than the RTT between a sender and receiver sketch a timing diagram of the packet and ACK flows between the sender and receiver.
- b) If the time-out is much greater than the RTT between a sender and receiver sketch the time diagram for the case of a packet loss.
- c) Comment on the trade-off between time-out less than RTT and time-out much greater than RTT. Which condition could cause the most harm to other senders and receivers (that is, to other users of the network)? Explain why.

Problem #6

Consider a UDP receiver that receives a packet and determines that the checksum computed on the received packet matches the checksum carried in the checksum field. Is it now certain that no bit errors have occurred during the transmission of the packet? Explain.

Problem #7

~~What if a channel has a maximum delay that is known? Can the rdt 2.1 receiver be modified such that it can now communicate correctly (with the rdt 3.0 sender)? By “communicate correctly” we mean~~

reliably transfer data. If it can, describe the change needed and explain why it works. **Hint:** Think about a time-out event.

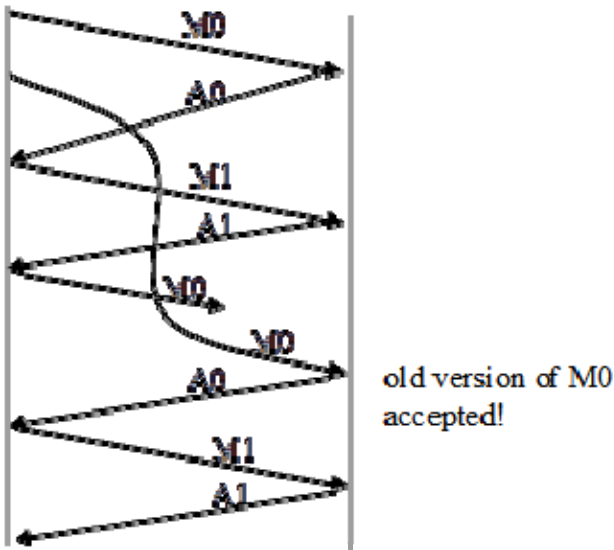
Consider a channel that can lose packets but has a maximum delay that is known. Modify protocol rdt 2.1 to include sender timeout and retransmit. Informally argue why your protocol can communicate correctly over this channel.

Problem #8

Consider the rdt 2.2 receiver and rdt 3.0 sender (from your textbook, also from class lecture). Would the protocol communicate correctly (that is, reliably transfer data) if the actions in the self-loop for “Wait for 0 from below” and “Wait for 1 from below” from rdt 2.2 were to be removed? Explain your answer carefully (if it would not work, given an example where failure occurs).

Problem #9

Consider the below timing diagram. Explain what has happened. Propose at least one reasonable fix for rdt 2.2 receiver and/or rdt 3.0 sender that can prevent the wrong packet from be accepted.



Problem #10

You are given a 24,000 mile link with a 1 Mb/s data rate between a sender and receiver. All data is sent in 1500 byte packets. For the Stop-and-Wait (SAW) protocol determine the link utilization assuming that 1) the sender always has data to send, 2) the overhead from processing and ACK transmission time is negligible, 3) the overhead from packet headers is negligible, and the link is error-free (no packets are lost).

- Determine the link utilization for the above described scenario.
- Change the above original scenario to be a 24 mile link, determine the link utilization.
- Change the above original scenario (the 24,000 mile link case) to use a Sliding Window protocol with windows size 10 packets, determine the link Utilization.

d) For the above original scenario, what is the window size needed for a Sliding Window protocol to achieve 100% link utilization.

Appendix

Program headers for problem #1.

```
//===== file = udpPing.c =====
//= A UDP ping client that sends a "PING" and listens for a "PONG" =
//=====
//= Notes: =
//= 1) This program conditionally compiles for Winsock and BSD sockets. =
//= Set the initial #define to WIN or BSD as appropriate. =
//= 2) This program needs udpPong to be running on the other host =
//= Program udpPong must be started first. =
//= 3) This program takes command line input for udpPong host IP address =
//= and number of PING to send =
//= 4) If a PONG response is not received, this program hangs =
//=====
//= Example execution: (udpPong and udpPing running on host 127.0.0.1) =
//= udpPing 127.0.0.1 5 =
//= PONG received from IP = 127.0.0.1 -- RTT = 110 ms =
//= PONG received from IP = 127.0.0.1 -- RTT = 110 ms =
//= PONG received from IP = 127.0.0.1 -- RTT = 110 ms =
//= PONG received from IP = 127.0.0.1 -- RTT = 110 ms =
//= PONG received from IP = 127.0.0.1 -- RTT = 110 ms =
//=====
//= Build: =
//= Windows (WIN): Borland: bcc32 udpPing.c =
//= MinGW: gcc udpPing.c -lws2_32 -o udpPing =
//= Visual C: cl ucpPing.c wsock32.lib =
//= Unix/Mac (BSD): gcc ucpPing.c -lnsl -o ucpPing =
//=====
//= Execute: udpPing ip_addr num_ping =
//=====
//= Author: << YOUR NAME >> =
//= University of South Florida =
//= Email: << YOUR EMAIL ADDRESS >> =
//=====
//= History: << YOUR INITIALS >> (09/__/18) - Genesis (from udpClient.c) =
//=====

//===== file = udpPong.c =====
//= A UDP ping server that listens on for "PING" and returns "PONG" =
//=====
//= Notes: =
//= 1) This program conditionally compiles for Winsock and BSD sockets. =
//= Set the initial #define to WIN or BSD as appropriate. =
//= 2) This program listens on port PORT_NUM for a "PING" message and then =
//= returns a "PONG" message =
//= 3) Ignore build warning for unreachable code =
//=====
//= Example execution: (udpPing and udpPong running on host 127.0.0.1) =
//= Waiting for PING on port 1024... =
//= Received PING from address = 127.0.0.1 and port = 55476 =
//=====
//= Build: =
//= Windows (WIN): Borland: bcc32 udpPong.c =
//= MinGW: gcc udpPong.c -lws2_32 -o udpPong =
//= Visual C: cl ucpPong.c wsock32.lib =
//= Unix/Mac (BSD): gcc ucpPong.c -lnsl -o ucpPong =
```

```
//=====
// Execute: udpPong
//=====
// Author: << YOUR NAME >>
//         University of South Florida
//         Email: << YOUR EMAIL ADDRESS >>
//=====
// History: << YOUR INITIALS >> (09/__/18) - Genesis (from udpServer.c)
//=====
```