COSC 264 – Introduction to Computer Networks and the Internet

Josh Bernasconi & James Toohey

68613585 27073776

Percentage contribution: 50% each

Questions:

1. A deadlock is a situation where a loop of systems are all waiting on input or output from another system. In the context of networking it could be that a server is waiting for a response packet and the receiver is waiting on a data packet because of packet losses in the system. In our case a deadlock occurs when the final data packet from the sender, the purposely empty packet, is lost. The receiver thinks there is still more data to come so is waiting for an arriving packet, but the sender has finished and is not waiting for an acknowledgement, so will not resend the final packet.
2. The magicNo field is useful for identification of related packets. For example a single sender and receiver could be attempting to send and receiver multiple files simultaneously or close together and the field could be used to differentiate between them. It is also useful as a quick check of packet validity before moving to the checksum.
3. We solved the issue of bit errors by adding a checksum field to the end of the header, which is calculated over only the preceding header fields. This means that either sender or receiver can see if the packet header has changed during transmission, protecting against bit errors.
4. The select function call is a blocking call that waits until one or more of the given file descriptors are ready for input. Blocking means that the call does not use CPU time until there is some input available, this is handled by the OS.
5. After transmission has completed, the diff command was used to check for differences between the sent and received file.

```
[jbe113@cs14142jm ~/Documents/COSC264/Cosc264_Assignment]$ clear
[jbe113@cs14142jm ~/Documents/COSC264/Cosc264_Assignment]$ diff -s in.txt out.txt
Files in.txt and out.txt are identical
```

6. The trend in figure 1 appears linear, with the number of packets sent increasing with the probability of a packet loss. This makes sense as the sender would need to resend more packets in order to account for the increase in packets lost during transmission.

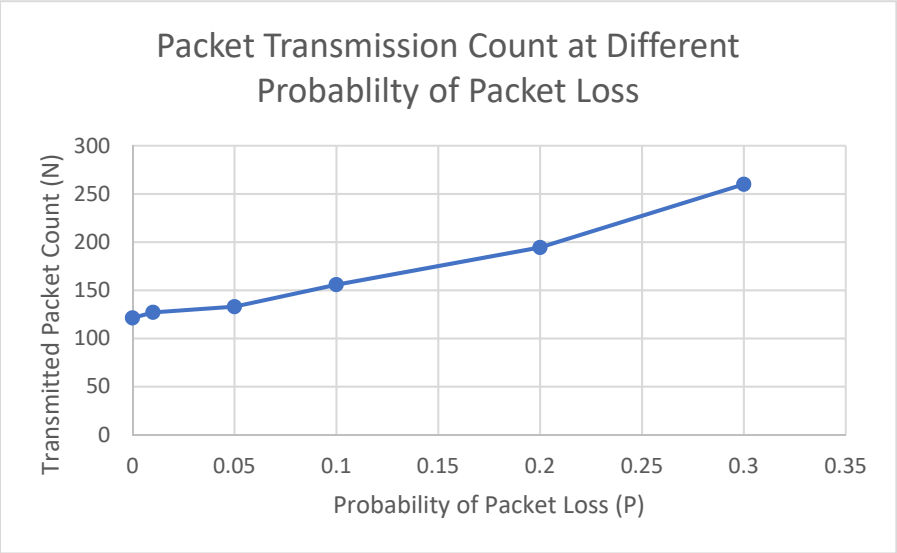

Figure 1

7. The probability that an individual packet is lost when sending in one direction is P(packet lost) = P. Since an individual packet must be sent from the sender to receiver and an acknowledgement packet sent back, this means the probability the sender sending a packet and not receiving an acknowledgement packet back is P(packet not transmitted) = 2P. It follows that the probability of successfully transmitting a packet is therefore P(packet transmitted) = 1 – 2P.
   By the use of an inverse binomial distribution, it can be found that the expression for the average amount of transmissions to successfully deliver N packets is:

$$\frac{N}{1-2p}$$

The comparison shown in figure 2 demonstrates the slight impact that the bit errors have on the number of packets that are required to send a file 51,200 bytes long.

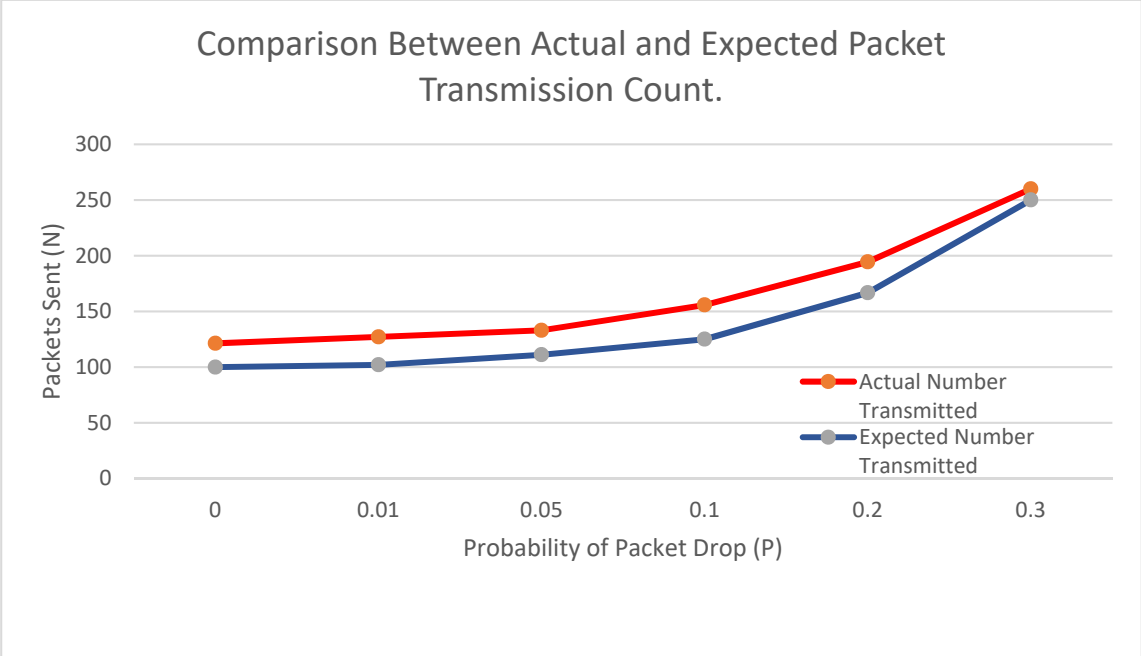

Figure 2

```python
""" Channel program for Cosc264 Assignment

    Authors: Josh Bernasconi 68613585
             James Toohey    27073776
"""

import random
import select
import socket
import sys
import time

from helpers import *


def channel(CSin_port, CSout_port, CRin_port, CRout_port, Sin_port, Rin_port, Precision):
    """ Checks ports, sets up connections, then hands over to the main loop """

    ports_ok = check_ports(CSin_port, CSout_port, CRin_port, CRout_port, Sin_port, Rin_port)

    if ports_ok:
        print("Port numbers all valid\n")
    else:
        print("There is a problem with the supplied port numbers!\n Exiting")
        sys.exit()

    # Socket init
    CSin = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    CSout = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    CRin = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    CRout = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

    # Bind
    try:  # Catching errors binding the ports
        print("Binding port CSin")
        CSin.bind(('localhost', CSin_port))
        print("CSin successfully bound\n")
        print("Binding port CSout")
        CSout.bind(('localhost', CSout_port))
        print("CSout successfully bound\n")
        CRin.bind(('localhost', CRin_port))
        print("CRin successfully bound\n")
        CRout.bind(('localhost', CRout_port))
        print("CRout successfully bound\n")
    except socket.error as msg:
        print("Bind failed. Exiting.\n Error: " + str(msg))
        sys.exit()

    # Listen and accept CSin
    CSin.listen(50)
    CSin, _ = CSin.accept()

    print("CSin accepted")

    # Connect CRout to Rin
    connected = False
    connect_attempts = 0
    while not connected:
        try:
            print("Connecting CRout to Rin")
            CRout.connect(('localhost', Rin_port))
            print("Connection successful\n")
            connected = True
        except socket.error as msg:
            connect_attempts += 1
            if msg.errno in [111, 10061] and connect_attempts < 6:
                print("Connection refused {} time(s), sleeping and retrying".format(connect_attempts))
                time.sleep(5)
                pass
            else:
                print("Connect failed. Exiting\n Error: " + str(msg))
                sys.exit()

    # Listen and accept CRin
    CRin.listen(50)
    CRin, _ = CRin.accept()

    # Connect CSout to Sin
    try:
        print("Connecting CSout to Sin")
        CSout.connect(('localhost', Sin_port))
        print("Connection successful\n")
    except socket.error as msg:
        print("Connect failed. Exiting\n Error: " + str(msg))
        sys.exit()

    # Receive, select and send
```

```
    process(CSin, CSout, CRin, CRout, CSin_port, CRin_port, Precision)

    CSin.close()
    CSout.close()
    CRin.close()
    CRout.close()
    return None


def bitError(data_in):
    """ Randomly adds/ doesn't add a bit error to the packet"""
    v = random.uniform(0, 1)
    if v < 0.1:
        print("bit error")
        packet, valid = get_packet(data_in)
        if valid:
            new_packet = Packet(packet.pac_type,
                                packet.seqno,
                                packet.data_len + int(random.uniform(1, 10)),
                                packet.data,
                                packet.checksum)
            data_in = pack_data(new_packet)

    return data_in


def process(CSin, CSout, CRin, CRout, CSin_port, CRin_port, Precision):
    """Main infinite loop which receives and processes all packets. Then sends them to the destination"""
    finished = False
    while not finished:  # while CRin doesnt receive terminating packet
        readable, _, _ = select.select([CSin, CRin], [], []) # Blocking call for input
        for sock in readable:
            host, port = sock.getsockname()

            data_in, address = sock.recvfrom(1024)

            if len(data_in) != 0:
                header = get_header_object(data_in)

                if header.magicno != 0x497E:
                    print("Sender Packet magic number != 0x497E, dropping packet.\n")
                    continue
                else:  # potentially drop packet
                    u = random.uniform(0, 1)
                    if u < Precision:  # drop packet
                        print("drop packet")
                        continue
                    else:  # potentially introduce bit error
                        data_in = bitError(data_in)

                    if port == CSin_port:
                        CRout.send(data_in)
                    elif port == CRin_port:
                        CSout.send(data_in)
                    else:  # received from invalid port number
                        print("Invalid port number")
                        continue
            else:
                print("empty data packet received, finished")
                finished = True
                break

    print("Precision {}".format(Precision))

if __name__ == '__main__':

    if len(sys.argv) != 8:
        print("Invalid command.")
        print("Usage: channel.py [CSin port] [CSout port] [CRin port] [CRout port] {Sin port] [Rin port] [Precision]")
    else:
        CSin = int(sys.argv[1])
        CSout = int(sys.argv[2])
        CRin = int(sys.argv[3])
        CRout = int(sys.argv[4])
        Sin = int(sys.argv[5])
        Rin = int(sys.argv[6])
        Precision = float(sys.argv[7])

        channel(CSin, CSout, CRin, CRout, Sin, Rin, Precision)
```

```python
""" Receiver program for Cosc264 Assignment

    Authors: Josh Bernasconi 68613585
             James Toohey    27073776
"""

import socket
import sys
import select
import os.path
import time

from helpers import *
from packet import Packet


def receiver(Rin_port, Rout_port, CRin_port, filename):
    """ Checks ports, sets up connections, then hands over to the main loop """

    ports_ok = check_ports(Rin_port, Rout_port, CRin_port)

    if ports_ok:
        print("Port numbers all valid\n")
    else:
        print("There is a problem with the supplied port numbers!\n Exiting")
        sys.exit()

    if not os.path.isfile(filename):
        file = open(filename, "wb+")
    else:
        print("File already exists, aborting")
        sys.exit()

    # Socket init
    Rin = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    Rout = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    CRin = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

    Rin.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    Rout.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    CRin.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)

    # Bind
    try:
        print("Binding port Rin")
        Rin.bind(('localhost', Rin_port))
        print("Rin successfully bound\n")
        print("Binding port Rout")
        Rout.bind(('localhost', Rout_port))
        print("Rout successfully bound\n")
    except socket.error as msg:
        print("Bind failed. Exiting.\n Error: " + str(msg))
        sys.exit()

    # Listen and accept Rin
    Rin.listen(50)
    Rin, _ = Rin.accept()

    # Connect Rout to CRin
    connected = False
    connect_attempts = 0
    while not connected:
        try:
            print("Connecting Rout to CRin")
            Rout.connect(('localhost', CRin_port))
            print("Connection successful\n")
            connected = True
        except socket.error as msg:
            connect_attempts += 1
            if msg.errno in [111, 10061] and connect_attempts < 6:
                print("Connection refused {} time(s), sleeping and retrying".format(connect_attempts))
                time.sleep(5)
                pass
            else:
                print("Connect failed. Exiting\n Error: " + str(msg))
                sys.exit()
```

```
    # Read/Write
    read_and_write(Rin, Rout, file)

    Rin.close()
    Rout.close()
    CRin.close()

    return None


def read_and_write(Rin, Rout, file):
    """ Receiver the packets, check validity, acknowledge, then write to file if valid """
    expected = 0
    finished = False
    while not finished:  # while the empty packet has not been found
        readable, _, _ = select.select([Rin], [], [])
        if len(readable) == 1:
            data_in, address = readable[0].recvfrom(1024)
            # print(len(data_in))
            # if len(data_in) == 0:
            #     print("Finished, I think...")
            #     finished = True
            #     continue
            rcvd, valid_packet = get_packet(data_in)
            if not valid_packet:
                print("Invalid packet, stop processing\n")
                continue
            elif rcvd.pac_type == 1:
                print("Packet type not dataPacket, stop processing\n")
                continue
            elif rcvd.seqno != expected:
                acknowledge(Rout, rcvd.seqno, expected)
                print("out of sequence")
                continue

            if rcvd.data_len > 0:
                print("Received valid data packet, writing...")
                acknowledge(Rout, rcvd.seqno, expected)
                file.write(rcvd.data)
                expected = 1 - expected
            else:
                print("Finished")
                finished = True


def acknowledge(Rout, seqno, expected):
    """Creates appropriate acknowledgement packets and sends them through Rout."""
    if seqno != expected:
        packet = Packet(1, seqno, 0, "")  # Still needs a data parameter. Page 6
        acknowledgement_packet = pack_data(packet)
        Rout.send(acknowledgement_packet)

    elif seqno == expected:
        packet = Packet(1, seqno, 0, "")
        acknowledgement_packet = pack_data(packet)
        Rout.send(acknowledgement_packet)


if __name__ == '__main__':
    if len(sys.argv) != 5:
        print("Invalid command.")
        print("Usage: receiver.py [Rin port] [Rout port] [CRin port] dest_filename")
    else:
        Rin = int(sys.argv[1])
        Rout = int(sys.argv[2])
        CRin = int(sys.argv[3])
        filename = sys.argv[4]

        receiver(Rin, Rout, CRin, filename)
```

```python
""" Sender program for Cosc264 Assignment

    Authors: Josh Bernasconi 68613585
             James Toohey    27073776
"""

import socket
import select
import sys
from helpers import *
from packet import Packet
import time


def sender(Sin_port, Sout_port, CSin_port, filename):
    """ Checks ports, sets up connections, then hands over to the main loop """

    ports_ok = check_ports(Sin_port, Sout_port, CSin_port)

    if ports_ok:
        print("Port numbers all valid\n")
    else:
        print("There is a problem with the supplied port numbers!\n Exiting")
        sys.exit()

    file = open(filename, "rb")  # Check it exists. If not, exit.

    # Socket init
    Sin = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    Sout = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    CSin = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

    Sin.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    Sout.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    CSin.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)

    # Bind
    try:
        print("Binding port Rin")
        Sin.bind(('localhost', Sin_port))
        print("Rin successfully bound\n")
        print("Binding port Rout")
        Sout.bind(('localhost', Sout_port))
        print("Rout successfully bound\n")
    except socket.error as msg:
        print("Bind failed. Exiting.\n Error: " + str(msg))
        sys.exit()

    # Try to connect Sout 5 times before giving up (waiting 5 seconds between attempts)
    connected = False
    connect_attempts = 0
    while not connected:
        try:
            print("Connecting Sout to CSin")
            Sout.connect(('localhost', CSin_port))
            print("Connection successful\n")
            connected = True
        except socket.error as msg:
            connect_attempts += 1
            if msg.errno in [111, 10061] and connect_attempts < 6:
                print("Connection refused {} time(s), sleeping and retrying".format(connect_attempts))
                time.sleep(5)
                pass
            else:
                print("Connect failed. Exiting\n Error: " + str(msg))
                sys.exit()

    # Listen and Accept Sin
    Sin.listen(50)
    Sin, _ = Sin.accept()

    # Read file
    next, size, count = read_file(Sin, Sout, file)

    last_packet = Packet(0, next, 0, "")
    data_packet = pack_data(last_packet)
    Sout.send(data_packet)
    print("Sent {} bytes".format(size))
    print("Number needed for perfect transmission: {}".format(size//512))
```

```
    print("Actually took: {}".format(count))

    Sin.close()
    Sout.close()
    CSin.close()

def read_file(Sin, Sout, file):
    """
    An outer loop which reads the file and sends packets of its content. Also receives
    acknowledgement packets to ensure successful delivery of packets.
    """
    byte_file = file.read()
    n = len(byte_file)
    size = n
    sent = 0
    count = 0
    next = 0
    exit_flag = False

    # Send
    while not exit_flag:
        if n == 0:
            packet = Packet(0, next, 0, '')
            data_packet = pack_data(packet)
            exit_flag = True
        else:
            if n - sent > 512:
                data = byte_file[sent:sent + 512]
                packet = Packet(0, next, 512, data)
                data_packet = pack_data(packet)
                sent += 512
            else:
                data = byte_file[sent:]
                packet = Packet(0, next, len(data), data)
                data_packet = pack_data(packet)
                print("Last data packet sent")
                exit_flag = True

        count, next, exit_flag = check(Sin, Sout, count, data_packet, next, file, exit_flag)

    return next, size, count


def check(Sin, Sout, count, data_packet, next, file, exit_flag):
    """An inner loop which checks that the packet has been successfully sent."""
    successfully_sent = False
    while not successfully_sent:
        count += 1
        Sout.send(data_packet)
        readable, _, _ = select.select([Sin], [], [], 1)  # Timeout after 1 second. If timeout, retransmit.

        if len(readable) == 1:
            data_in, address = readable[0].recvfrom(1024)
            rcvd, valid_packet = get_packet(data_in)
            if valid_packet and rcvd.pac_type == 1 and rcvd.data_len == 0 and rcvd.seqno == next:
                next = 1 - next
                print("Received acknowledgement packet.")
                successfully_sent = True
                if exit_flag:  # Go to beginning of outer loop
                    file.close()
                    break
        else:  # retransmit
            print("timed out")
            print("Resending packet")
    return count, next, exit_flag


if __name__ == '__main__':

    if len(sys.argv) != 5:
        print("Invalid command.")
        print("Usage: sender.py [Sin port] [Sout port] [CSin port] input_filename")
    else:
        Sin = int(sys.argv[1])
        Sout = int(sys.argv[2])
        CSin = int(sys.argv[3])
        filename = sys.argv[4]

        sender(Sin, Sout, CSin, filename)
```

```python
""" extra functions and classes used by channel, receiver and sender

    Authors: Josh Bernasconi 68613585
             James Toohey    27073776
"""

from struct import *
from packet import Packet
from packet import Header


def check_ports(*ports):
    """ Returns True if there are no duplicate ports and if all
        ports are within the acceptable range, False otherwise.
        """

    all_clear = True
    set_ports = set(ports)

    if len(ports) != len(set_ports):  # Check for duplicate ports
        all_clear = False

    for port in ports:  # check each port is within acceptable port range
        if port < 1024 or port > 64000 or not (isinstance(port, int)):
            all_clear = False

    return all_clear


def pack_data(packet):
    if type(packet.data) != bytes:
        packed = pack('!2I3i' + str(packet.data_len) + 's',
                      packet.magicno, packet.checksum, packet.pac_type, packet.seqno, packet.data_len,
                      bytes(packet.data, 'utf8'))
    else:
        packed = pack('!2I3i' + str(packet.data_len) + 's',
                      packet.magicno, packet.checksum, packet.pac_type, packet.seqno, packet.data_len, packet.data)

    return packed


def get_header(packet):
    header = unpack('!2I3i', packet[:20])

    return header


def get_header_object(packet):
    header_object = Header(get_header(packet))

    return header_object


def get_data(packet, data_len):
    data = unpack(str(data_len) + 's', packet[20:20 + data_len])
    return data[0]


def get_packet(in_data):
    valid_packet = False
    packet = None

    if in_data != b'':
        header = get_header(in_data)
        checksum = header[1]
        pac_type = header[2]
        seq_no = header[3]
        data_len = header[4]

        if data_len >= len(in_data) - 20:
            data = get_data(in_data, data_len)

            packet = Packet(pac_type, seq_no, data_len, data, checksum)

            valid_packet = packet.checksum == packet.calculate_checksum()

    return packet, valid_packet
```

```
""" Packet Class for sender, receiver and channel.

    Authors: Josh Bernasconi 68613585
             James Toohey    27073776
    """


class Packet(object):
    def __init__(self, pac_type, seqno, data_len, data, checksum=0):
        self.magicno = 0x497E
        self.pac_type = pac_type  # integer, 0 = dataPacket, 1 = acknowledgementPacket
        self.seqno = seqno  # integer
        self.data_len = data_len  # integer
        self.checksum = checksum  # hex number
        self.data = data  # string

        if self.checksum == 0:
            self.checksum = self.calculate_checksum()

    def calculate_checksum(self):
        checksum = self.magicno + self.pac_type + self.seqno + self.data_len

        return checksum


class Header(object):
    def __init__(self, header):
        self.magicno = header[0]
        self.checksum = header[1]
        self.pac_type = header[2]
        self.seqno = header[3]
        self.datalen = header[4]
```