COSC264 Networking Assignment

Ollie Chick (67683982)

Samuel Pell (\_\_number\_\_)

Percentage contribution: 50% each

9 September 2017

# 1 Deadlocks

In networking, a deadlock is where transmission of new data ceases due to the system state. A deadlock occurs in this protocol when the last acknowledgement packet from the receiver is lost. The receiver closes after it sends the last acknowledgement packet, so if this packet is lost, the sender thinks that the last data packet was lost and keeps resending the last packet. The sender is thus in a deadlock state: it tries to retransmit the last packet continuously, preventing it from transmitting any more data it may need to.

# 2 The magicno field

The magicno field ensures that the message came from this protocol and that some other program has not connected to one of the programs by mistake.

# 3 Bit errors

Bit errors were solved by adding a checksum to the header. This was calculated when a packet was created by taking the last three digits of the sum of the other header fields. When the sender and receiver received a packet, they calculated what the checksum should be and compared it to the actual checksum. If they were identical, they processed the packet, otherwise they discarded it.

# 4 The select() function

The select function, or in Python the select.select method, waits until data can be read from the socket (or a timeout, if specified, is exceeded). In channel, this method is used so that it can wait until it can read data from its sockets, and only then read from sockets that have new data. This is useful as it reduces code complexity and prevents the channel program from using CPU resources unnecessarily.

In sender, the select method is used to enact the timeout mechanism for re-sending data if an acknowledgement packet isn’t received.

# 5 Verifying correct transferral

To check that the two files were the same, diff <input\_file> <output\_file> was run. This compares the contents of the two files and outputs nothing if they are identical.

# 6 Packet loss measurement

The number of packets required to send a 512,000-byte file with different packet loss rates was tested experimentally. To do this, the programs were run 10 times for each packet loss rate. The results of this are shown in Table 1 below, and are summarised in Figure 1.

<Explanation goes here>

Table 1 - Number of packets required to send 512,000-byte file for different packet loss probabilities.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Packet Loss  Probability | 0 | 0.01 | 0.05 | 0.1 | 0.2 | 0.3 |
| Packets Sent | 134 | 126 | 142 | 162 | 185 | 227 |
| 123 | 136 | 139 | 154 | 191 | 202 |
| 128 | 125 | 138 | 151 | 178 | 272 |
| 122 | 119 | 153 | 131 | 188 | 293 |
| 119 | 130 | 137 | 159 | 167 | 269 |
| 124 | 115 | 141 | 156 | 208 | 269 |
| 124 | 128 | 139 | 147 | 219 | 292 |
| 121 | 123 | 136 | 145 | 173 | 215 |
| 121 | 126 | 142 | 146 | 183 | 274 |
| 118 | 136 | 140 | 150 | 179 | 253 |
| Average | 123.4 | 126.4 | 140.7 | 150.1 | 187.1 | 256.6 |

|  |
| --- |
|  |
| Figure 1 – The average number of packets required to transmit a file 512,000 bytes long. |

# 7 Derivation of average packets required

Let *P* be the probability that a packet is dropped or has its data length field changed, and *N* be the number of packets that is needed to transmit some arbitrary file. Assuming that each transmission is statistically independent of all others, each transmission can be defined as a Bernoulli trial. Let the event the a given packet is dropped be a success and the event that a packet is successfully transmitted be a failure.

If a series of these Bernoulli experiments are performed then the results will have a negative binomial distribution. That is, have a probability mass function of

|  |  |
| --- | --- |
|  | (1) |

where *k* is the number of successes, *r* is the number of failures, and *γ* is the probability of success. This distribution has the expected value of

|  |  |
| --- | --- |
|  | (2) |

where *µ* is the expected value (Pennsylvania State University, 2017), (Weisstein, nd).

The negative binomial distribution measures the number of success before *r* failures occurs. Because of this definition, the average number of packets that must be sent to transmit successfully transmit the file is

|  |  |
| --- | --- |
|  | (3) |

There should be constant of one as overhead from the protocol - an empty data packet is sent to signal the end of transmission - this packet can be lost and need retransmission. However, this is ignored as the last packet must pass through unmolested for successful transmission to occur.

However, the parameter *γ* is not the same as *P* — it is in fact

|  |  |
| --- | --- |
|  | (4) |

This is because if the acknowledgement packet from receiver is lost then a retransmission also occurs.

Thus, the average number of packets needed to transmit a file that can be split into *N* packets is

|  |  |
| --- | --- |
|  | (5) |

which can be simplified to

|  |  |
| --- | --- |
|  | (6) |

Table 2 shows that the predicted average number of transmissions closely matches the experimental average. The difference will be due to the small sample size of 10 and the fact that the random number generator in Python is only pseudo-random and not truly random.

Table 2 – The predicted average number of packets required to send a 100-packet file compared to an experimental average.

|  |  |  |
| --- | --- | --- |
| Packet Drop Rate | Predicted Average | Experimental Average |
| 0 | 123.5 | 123.4 |
| 0.01 | 126.0 | 126.4 |
| 0.05 | 136.8 | 140.7 |
| 0.1 | 152.4 | 150.1 |
| 0.2 | 192.9 | 187.1 |
| 0.3 | 252.0 | 256.6 |

# References

Pennsylvania State University (2017). Key Properties of a Negative Binomial Random *Variable*. Retrieved on 30th August 2017 from https://onlinecourses.science.psu.edu/stat414/node/79

Weisstein, Eric W. (nd). *Negative Binomial Distribution.* Retrieved 30th August 2017 from http://mathworld.wolfram.com/NegativeBinomialDistribution.html

# Appendices

**Appendix 1 – packet.py**

"""

Packet

A class to represent a packet of information.

Authors: Samuel Pell and Ollie Chick

Date Modified: 30 August 2017

Contains:

\_\_init\_\_()

\_\_repr\_\_()

\_\_str\_\_()

\_\_len\_\_()

encode()

decode()

is\_valid\_ack()

is\_valid\_data()

"""

PTYPE\_DATA **=** 0

PTYPE\_ACK **=** 1

MAGIC\_NO **=** 0x497E

**class** **Packet:**

**def** \_\_init\_\_**(**self**,** magic\_no**=**0**,** packet\_type**=**0**,** seq\_no**=**0**,** data\_len**=**0**,** data**=**""**):**

self**.**magic\_no **=** magic\_no #determines if packet is dropped

self**.**packet\_type **=** packet\_type #either dataPacket or acknowledgementPacket

self**.**seq\_no **=** seq\_no #sequence number

self**.**data\_len **=** data\_len #number of bytes in the data

self**.**checksum **=** **(**magic\_no **+** packet\_type **+** seq\_no **+** data\_len**)** **%** 1000

self**.**data **=** data #data carried by packet

**def** \_\_repr\_\_**(**self**):**

**return** self**.**\_\_str\_\_**()**

**def** \_\_str\_\_**(**self**):**

pt **=** 'unknown'

**if** self**.**packet\_type **==** 0**:**

pt **=** 'data'

**elif** self**.**packet\_type **==** 1**:**

pt **=** 'ack'

s **=** 'Magic number: 0x{:X}\n'**.**format**(**self**.**magic\_no**)**

s **+=** 'Packet type: {} ({})\n'**.**format**(**self**.**packet\_type**,** pt**)**

s **+=** 'Seq no: {}\n'**.**format**(**self**.**seq\_no**)**

s **+=** 'Data len: {}\n'**.**format**(**self**.**data\_len**)**

s **+=** 'Data: "{}"'**.**format**(**self**.**data**)**

**return** s

**def** \_\_len\_\_**(**self**):**

**return** len**(**self**.**encode**())**

**def** encode**(**self**):**

""" Returns the byte representation of the packet. """

conv **=** str**(**self**.**magic\_no**)**

conv **+=** str**(**self**.**packet\_type**)**

conv **+=** str**(**self**.**seq\_no**)**

conv **+=** "0" **\*** **(**3 **-** len**(**str**(**self**.**data\_len**)))** **+** str**(**self**.**data\_len**)**

conv **+=** "0" **\*** **(**3 **-** len**(**str**(**self**.**checksum**)))** **+** str**(**self**.**checksum**)**

conv **+=** str**(**self**.**data**)**

**return** bytes**(**conv**,** "utf-8"**)**

**def** decode**(**self**,** data**):**

""" Sets the fields of this packet to that of data. """

**try:**

data **=** data**.**decode**()**

self**.**magic\_no **=** int**(**data**[:**5**])**

self**.**packet\_type **=** int**(**data**[**5**])**

self**.**seq\_no **=** int**(**data**[**6**])**

self**.**data\_len **=** int**(**data**[**7**:**10**])**

self**.**checksum **=** int**(**data**[**10**:**13**])**

self**.**data **=** data**[**13**:]**

**except:**

**print(**"Error decoding data ({}). Packet is unchanged."**.**format**(**data**))**

**def** is\_valid\_ack**(**self**,** next\_no**):**

"""

Checks if the packet is a valid acknowledgement packet with the

correct sequence number, next\_no.

"""

valid\_magic **=** self**.**magic\_no **==** MAGIC\_NO

valid\_type **=** self**.**packet\_type **==** PTYPE\_ACK

valid\_length **=** self**.**data\_len **==** 0

valid\_seq\_no **=** self**.**seq\_no **==** next\_no

valid\_checksum **=** self**.**checksum **==** **(**self**.**magic\_no **+** self**.**packet\_type **+** self**.**seq\_no **+** self**.**data\_len**)** **%** 1000

**return** valid\_magic **and** valid\_type **and** valid\_length **and** valid\_seq\_no **and** valid\_checksum

**def** is\_valid\_data**(**self**):**

""" Checks if the packet is a valid data packet. """

valid\_magic **=** self**.**magic\_no **==** MAGIC\_NO

valid\_type **=** self**.**packet\_type **==** PTYPE\_DATA

valid\_checksum **=** self**.**checksum **==** **(**self**.**magic\_no **+** self**.**packet\_type **+**self**.**seq\_no **+** self**.**data\_len**)** **%** 1000

**return** valid\_magic **and** valid\_type **and** valid\_checksum

# Appendix 2 – socket\_generator.py

"""

Socket generator

Program to generate sending and listening sockets.

Authors: Samuel Pell and Ollie Chick

Date modified: 29 August 2017

"""

**import** socket

IP **=** '127.0.0.1'

**def** create\_sending\_socket**(**local\_port**,** remote\_port**):**

"""

Creates a socket on the local\_port and connects it to the

remote\_port socket, then returns that socket.

If it fails, returns None.

"""

**try:**

new\_socket **=** socket**.**socket**(**socket**.**AF\_INET**,** socket**.**SOCK\_STREAM**)**

new\_socket**.**bind**((**IP**,** local\_port**))**

new\_socket**.**connect**((**IP**,** remote\_port**))**

**except** IOError**:**

new\_socket **=** **None**

**return** new\_socket

**def** create\_listening\_socket**(**port**):**

"""

Creates a socket to listen on the port given, then returns that

socket. If it fails, returns None.

"""

**try:**

new\_socket **=** socket**.**socket**(**socket**.**AF\_INET**,** socket**.**SOCK\_STREAM**)**

new\_socket**.**bind**((**IP**,** port**))**

new\_socket**.**listen**(**1**)**

**except** IOError**:**

new\_socket **=** **None**

**return** new\_socket

# Appendix 3 – channel.py

"""

channel

A program for the COSC264-17S2 Assignment

Authors: Samuel Pell and Ollie Chick

Date Modified: 30 August 2017

"""

**import** socket**,** select**,** sys**,** packet**,** random

**from** packet **import** Packet**,** MAGIC\_NO**,** PTYPE\_DATA**,** PTYPE\_ACK

**from** socket\_generator **import** create\_sending\_socket**,** create\_listening\_socket

BIT\_ERR\_RATE **=** 0.1

**def** process\_packet**(**data**,** drop\_rate**):**

"""

Process an input packet (as bytes) and randomly drop it or change

its header. Returns the input packet as bytes.

Returns the null byte if the input data is the null byte.

"""

**if** data **==** b''**:**

**return** data

p **=** Packet**()**

p**.**decode**(**data**)**

**if** p**.**magic\_no **!=** MAGIC\_NO**:**

# magic numbers is wrong: drop it

**return** **None**

**elif** random**.**uniform**(**0**,** 1**)** **<** drop\_rate**:**

# drop packet by random chance

**return** **None**

**elif** random**.**uniform**(**0**,**1**)** **<** BIT\_ERR\_RATE**:**

# create a bit error by random chance (increase data len field randomly)

p**.**data\_len **+=** random**.**randint**(**1**,** 10**)**

**return** p**.**encode**()** # return the packet's byte conversion

**def** main\_loop**(**sender\_in**,** sender\_out**,** recv\_in**,** recv\_out**,** drop\_rate**):**

"""

Wait to recieve packets on sender\_in and recv\_in. When it does,

process the packet and send it on to either recv\_out or sender\_out

respectively. Takes the four socket objects as arguments.

When one of the sockets indicates it has closed it will stop

watching it and when both sockets have closed it will return None

"""

sockets\_to\_watch **=** **[**sender\_in**,** recv\_in**]**

**while** **True:**

readable**,** \_**,** \_ **=** select**.**select**(**sockets\_to\_watch**,** **[],** **[])**

**for** s **in** readable**:**

data **=** s**.**recv**(**1024**)**

**if** data **==** b''**:**

# a socket sent out the null byte (indicating it has closed)

**if** s **==** recv\_in**:**

# receiver has closed; stop watching it

sockets\_to\_watch**.**remove**(**recv\_in**)**

**else:**

# sender has closed; stop watching it

sockets\_to\_watch**.**remove**(**sender\_in**)**

**if** len**(**sockets\_to\_watch**)** **==** 0**:**

# sender and receiver have closed; exit loop

**print(**'\nChannel shut down.'**)**

**return**

**elif** len**(**sockets\_to\_watch**)** **==** 2**:**

# sender and receiver are both open

data\_to\_forward **=** process\_packet**(**data**,** drop\_rate**)**

**if** data\_to\_forward **!=** **None:**

# the packet hasn't been dropped

**if** s **==** sender\_in**:**

# came from sender, send to receiver

**print(**"sender -> channel -> receiver"**,** end **=** '\r'**)**

recv\_out**.**send**(**data\_to\_forward**)**

**else:**

# came from receiver, send to sender

**print(**"sender <- channel <- receiver"**,** end **=** '\r'**)**

sender\_out**.**send**(**data\_to\_forward**)**

**def** main**(**args**):**

"""

Pull the relevant numbers out of the command line arguments, check

They are valid input, then create the appropriate sockets before

entering into the main loop

"""

# Check arguments are valid

**try:**

# Port numbers for this program

sender\_in\_port **=** int**(**args**[**1**])**

sender\_out\_port **=** int**(**args**[**2**])**

recv\_in\_port **=** int**(**args**[**3**])**

recv\_out\_port **=** int**(**args**[**4**])**

# Port numbers of the sender and reciver

sender **=** int**(**args**[**5**])**

recv **=** int**(**args**[**6**])**

# Probability of dropping a packet

drop\_rate **=** float**(**args**[**7**])**

**except:**

# User inputted wrong number of arguments, or non-ints/floats, etc.

**print(**"Usage: {} <sender\_in\_port> <sender\_out\_port> <recv\_in\_port> <recv\_out\_port> <sender> <recv> <drop\_rate>"**.**format**(**args**[**0**]))**

**return**

# Check that ports are in the valid range

**for** port **in** **[**sender\_in\_port**,** sender\_out\_port**,** recv\_in\_port**,** \

recv\_out\_port**,** sender**,** recv**]:**

**if** port **<** 1024 **or** port **>** 64000**:**

**print(**"All port numbers should be integers in the range [1024, 64000]."**)**

**return**

# Check that the drop rate is between 0 (inclusive) and 1 (exclusive)

**if** **(**drop\_rate **>=** 1**)** **or** **(**drop\_rate **<** 0**):**

**print(**"drop\_rate should be in the range [0, 1)."**)**

**return**

# Create in sockets

sender\_in **=** create\_listening\_socket**(**sender\_in\_port**)**

recv\_in **=** create\_listening\_socket**(**recv\_in\_port**)**

**if** **None** **in** **[**sender\_in**,** recv\_in**]:**

sys**.**exit**(**"One of the in sockets failed to be created."**)**

input**(**"Please start sender and receiver then press enter."**)**

# Create out sockets and connect them

sender\_out **=** create\_sending\_socket**(**sender\_out\_port**,** sender**)**

recv\_out **=** create\_sending\_socket**(**recv\_out\_port**,** recv**)**

**if** **None** **in** **[**sender\_out**,** recv\_out**]:**

sys**.**exit**(**"One of the out sockets failed to be created."**)**

# Accept incomming connections to sender\_in and recv\_in

**try:**

sender\_in**,** addr **=** sender\_in**.**accept**()**

recv\_in**,** addr **=** recv\_in**.**accept**()**

**except** IOError**:**

sys**.**exit**(**"Error connecting sender\_in or recv\_in"**)**

# Enter the main loop

main\_loop**(**sender\_in**,** sender\_out**,** recv\_in**,** recv\_out**,** drop\_rate**)**

# Shut down then close all the sockets (then the program)

sender\_in**.**shutdown**(**socket**.**SHUT\_RDWR**)**

sender\_in**.**close**()**

sender\_out**.**shutdown**(**socket**.**SHUT\_RDWR**)**

sender\_out**.**close**()**

recv\_in**.**shutdown**(**socket**.**SHUT\_RDWR**)**

recv\_in**.**close**()**

recv\_out**.**shutdown**(**socket**.**SHUT\_RDWR**)**

recv\_out**.**close**()**

**if** \_\_name\_\_ **==** "\_\_main\_\_"**:**

# Get arguments from the command line.

# These should be:

# \* four port numbers to use for the sockets c\_s\_in, c\_s\_out, c\_r\_in, and c\_r\_out

# \* the port number where the socket s\_in should be found

# \* the port number where the socket r\_in should be found

# \* a packet loss rate P such that 0 <= P < 1

args **=** sys**.**argv

main**(**args**)**

# Appendix 4 – receiver.py

"""

A program to receive packets from a channel.

For a COSC264 assignment.

Author: Ollie Chick

Date modified: 29 August 2017

"""

**import** sys**,** socket**,** os**,** select

**from** packet **import** Packet**,** MAGIC\_NO**,** PTYPE\_DATA**,** PTYPE\_ACK

**from** socket\_generator **import** create\_sending\_socket**,** create\_listening\_socket

**def** main**(**args**):**

# Check arguments are valid

**try:**

in\_port **=** int**(**args**[**1**])**

out\_port **=** int**(**args**[**2**])**

channel\_in\_port **=** int**(**args**[**3**])**

filename **=** args**[**4**]**

**except:**

**print(**"Usage: {} <in\_port> <out\_port> <channel\_in\_port> <filename>"**.**format**(**args**[**0**]))**

# Check that ports are in the valid range

**for** port **in** **[**in\_port**,** out\_port**,** channel\_in\_port**]:**

**if** port **<** 1024 **or** port **>** 64000**:**

**print(**"All port numbers should be integers in the range [1024, 64000]."**)**

**return**

# Create sockets (and connect socket\_out)

socket\_in **=** create\_listening\_socket**(**in\_port**)**

socket\_out **=** create\_sending\_socket**(**out\_port**,** channel\_in\_port**)**

**if** **None** **in** **[**socket\_in**,** socket\_out**]:**

sys**.**exit**(**"One of the sockets failed to be created."**)**

# Check if file exists

**if** os**.**path**.**isfile**(**filename**):**

sys**.**exit**(**"Error: {} already exists."**.**format**(**filename**))**

# Initialisation

expected **=** 0

file **=** open**(**filename**,** 'w'**)**

input**(**"Please acknowledge on the channel that you have started the receiver, then press enter."**)**

# Accept connection from channel

socket\_in**,** addr **=** socket\_in**.**accept**()**

**print(**"Receiving data..."**)**

# Main loop

i **=** 0

**while** **True:**

readable**,** \_**,** \_ **=** select**.**select**([**socket\_in**],** **[],** **[])**

# got a response

**print(**"Got packet {}"**.**format**(**i**),** end **=** '\r'**)**

i **+=** 1

s **=** readable**[**0**]**

data **=** s**.**recv**(**1024**)**

rcvd **=** Packet**()**

rcvd**.**decode**(**data**)**

**if** rcvd**.**is\_valid\_data**():**

# got a valid data packet

# Prepare an acknowledgement packet and send it

magic\_no **=** MAGIC\_NO

packet\_type **=** PTYPE\_ACK

seq\_no **=** rcvd**.**seq\_no

data\_len **=** 0

data **=** ""

pack **=** Packet**(**magic\_no**,** packet\_type**,** seq\_no**,** data\_len**,** data**)**

socket\_out**.**send**(**pack**.**encode**())**

**if** rcvd**.**seq\_no **==** expected**:**

expected **=** 1 **-** expected

**if** rcvd**.**data\_len **>** 0**:**

# has some data

file**.**write**(**rcvd**.**data**)**

**else:**

# no data - indicates end of file

file**.**close**()**

socket\_in**.**shutdown**(**socket**.**SHUT\_RDWR**)**

socket\_in**.**close**()**

socket\_out**.**shutdown**(**socket**.**SHUT\_RDWR**)**

socket\_out**.**close**()**

**print(**"\nData received."**)**

**return**

**if** \_\_name\_\_ **==** "\_\_main\_\_"**:**

# Get arguments from the command line.

# These should be:

# \* two port numbers to use for the two receiver sockets r\_in and r\_out

# \* the port number where the socket c\_r\_in should be found

# \* a file name, indicating where the received data should be stored

args **=** sys**.**argv

main**(**args**)**

# Appendix 5 – sender.py

"""

A program to send packets to a channel.

For a COSC264 assignment.

Author: Ollie Chick and Samuel Pell

Date modified: 29 August 2017

"""

**import** sys**,** socket**,** os**,** select

**from** packet **import** Packet**,** MAGIC\_NO**,** PTYPE\_DATA**,** PTYPE\_ACK

**from** socket\_generator **import** create\_sending\_socket**,** create\_listening\_socket

TIMEOUT **=** 1 #seconds

FILE\_ENCODING **=** 'utf8'

**def** inner\_loop**(**socket\_out**,** socket\_in**,** bytes\_to\_send**,** next\_no**):**

"""

Function to continuously send a packet until a valid acknowledgement

packet is received. Returns the number of packets sent from sender

to achieve successful transmission.

"""

packets\_sent **=** 0

**while** **True:**

# Send packet

socket\_out**.**send**(**bytes\_to\_send**)**

packets\_sent **+=** 1

# Await a response

readable**,** \_**,** \_ **=** select**.**select**([**socket\_in**],** **[],** **[],** TIMEOUT**)**

**if** readable**:**

# got a response

s **=** readable**[**0**]**

data **=** s**.**recv**(**1024**)**

rcvd **=** Packet**()**

rcvd**.**decode**(**data**)**

**if** rcvd**.**is\_valid\_ack**(**next\_no**):**

# got a valid acknowledgement packet

next\_no **=** 1 **-** next\_no

**return** packets\_sent**,** next\_no

**def** main**(**args**):**

# Check arguments are valid

**try:**

in\_port **=** int**(**args**[**1**])**

out\_port **=** int**(**args**[**2**])**

channel\_in\_port **=** int**(**args**[**3**])**

filename **=** args**[**4**]**

**except:**

**print(**"Usage: {} <in\_port> <out\_port> <channel\_in\_port> <filename>"**.**format**(**args**[**0**]))**

**return**

# Check that ports are in the valid range

**for** port **in** **[**in\_port**,** out\_port**,** channel\_in\_port**]:**

**if** port **<** 1024 **or** port **>** 64000**:**

**print(**"All port numbers should be integers in the range [1024, 64000]."**)**

**return**

# Create sockets (and connect socket\_out)

socket\_in **=** create\_listening\_socket**(**in\_port**)**

socket\_out **=** create\_sending\_socket**(**out\_port**,** channel\_in\_port**)**

**if** **None** **in** **[**socket\_in**,** socket\_out**]:**

sys**.**exit**(**"One of the sockets failed to be created."**)**

# Check if file exists

**if** **not** os**.**path**.**isfile**(**filename**):**

# file does not exist

sys**.**exit**(**"Error: {} does not exist."**.**format**(**filename**))**

# Initialisation

next\_no **=** 0

packets\_sent **=** 0

exit\_flag **=** **False**

file **=** open**(**filename**,** "rb"**)**

input**(**"Please acknowledge on the channel that you have started the sender, then press enter."**)**

# Accept connection from channel

socket\_in**,** addr **=** socket\_in**.**accept**()**

**print(**"Sending data..."**)**

# Outer loop

i **=** 0

**while** **not** exit\_flag**:**

# Read 512 bytes from file

data **=** file**.**read**(**512**)**

data **=** data**.**decode**(**FILE\_ENCODING**)**

# Prepare packet

packet\_type **=** PTYPE\_DATA

seq\_no **=** next\_no

data\_len **=** len**(**data**)**

**if** data\_len **==** 0**:**

exit\_flag **=** **True**

pack **=** Packet**(**MAGIC\_NO**,** packet\_type**,** seq\_no**,** data\_len**,** data**)**

# Inner loop

bytes\_to\_send **=** pack**.**encode**()**

**print(**"Sending datum {}"**.**format**(**i**),** end **=** "\r"**)**

packets\_used**,** next\_no **=** inner\_loop**(**socket\_out**,** socket\_in**,** bytes\_to\_send**,**

next\_no**)**

packets\_sent **+=** packets\_used

i**+=**1

# Clean up and close

file**.**close**()**

socket\_in**.**shutdown**(**socket**.**SHUT\_RDWR**)**

socket\_in**.**close**()**

socket\_out**.**shutdown**(**socket**.**SHUT\_RDWR**)**

socket\_out**.**close**()**

**print(**"\nData sent.\nPackets sent: {}"**.**format**(**packets\_sent**))**

**if** \_\_name\_\_ **==** "\_\_main\_\_"**:**

# Get arguments from the command line.

# These should be:

# \* two port numbers to use for the two sender sockets s\_in and s\_out

# \* the port number where the socket c\_s\_in should be found

# \* a file name, indicating the file whose contents should be sent

args **=** sys**.**argv

main**(**args**)**