CS5313: Computer Network

Fall 2020

Instructor: Dr. Deepak Tosh

Assignment 2

Erik Macik, M.S.

Aaron Espinosa, M.S

Introduction

It is important to understand how reliable data transfer works. Since reliable data transfer in a general context is considered a problem, as it does not only occur at the transport layer, it also occurs at the link and application layers. The general problem is thus of central importance to networking. [1]

Within the reliable data transfer protocol, a simple alternating bit protocol can be designed. This protocol is also known as a stop-and-wait protocol: after sending each packet the sender stops and waits for feedback from the receiver indicating that the packet has been received. [2]

The stop-and-wait protocol offers good control over the flow of errors, but it can cause significant performance problems, as the sender always waits for confirmation, even if they have the next packet ready to send. The sliding window protocol handles this efficiency problem by sending more than one packet at a time with larger sequence number. Practically it is implemented in two protocols namely: Go back N (GBN) and Selective Repeat (SR) [3]

In this report, we will explain how we implement both versions of the reliable data transfer protocol: Stop-and-Wait protocol and Go-Back-N (GBN) protocol using python programming language. Figure 1.

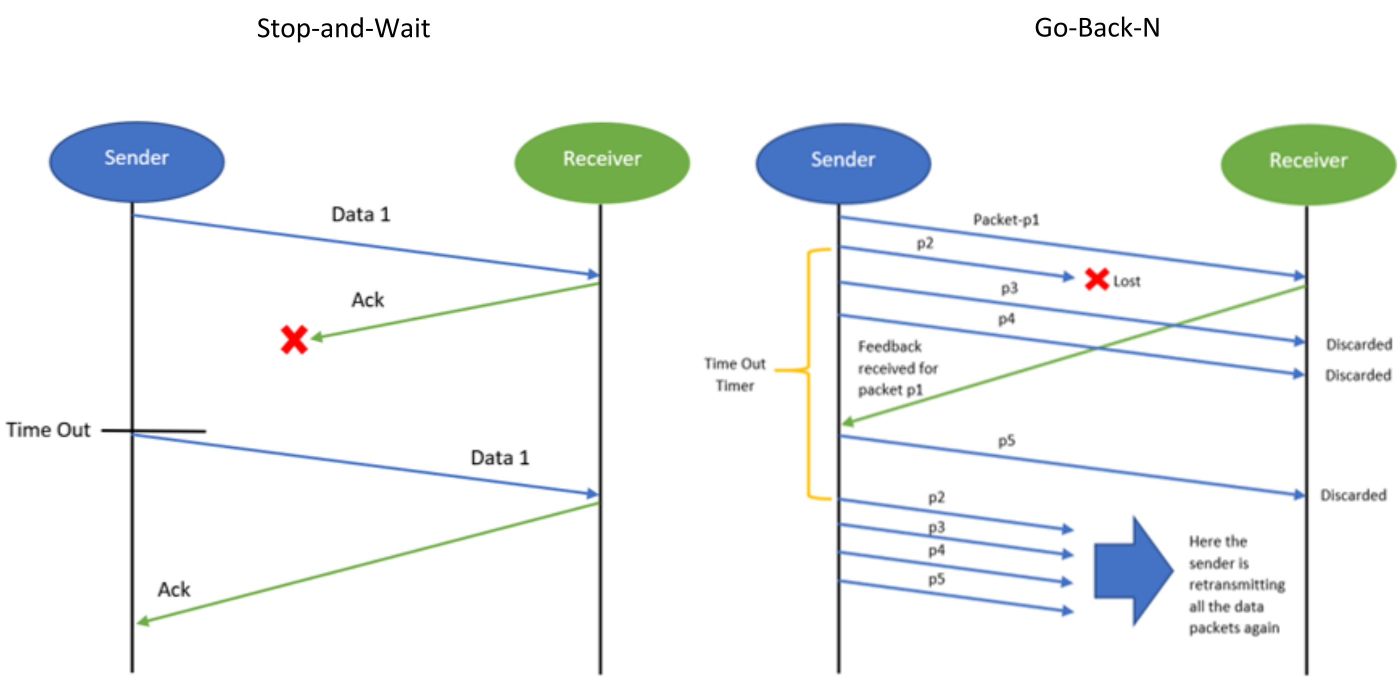


Figure 1 Stop-and-Wait and Go-Back-N protocols.

Method

First the receiver and sender files were run on two different consoles to see how it worked. Figure 2.

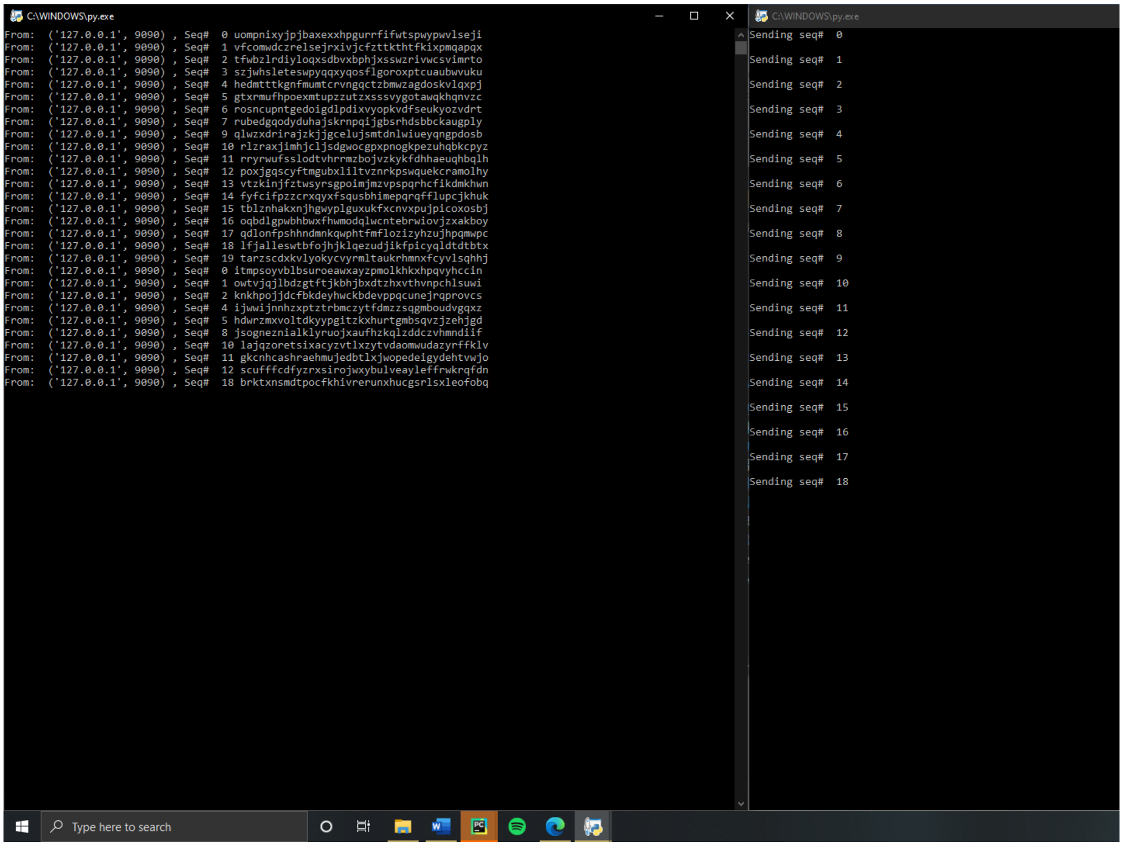


Figure 2 Receiver and Sender work.

After reviewing the code, we focus on the sender. We add the libraries that would be needed and uncomment _thread. Figure 3.

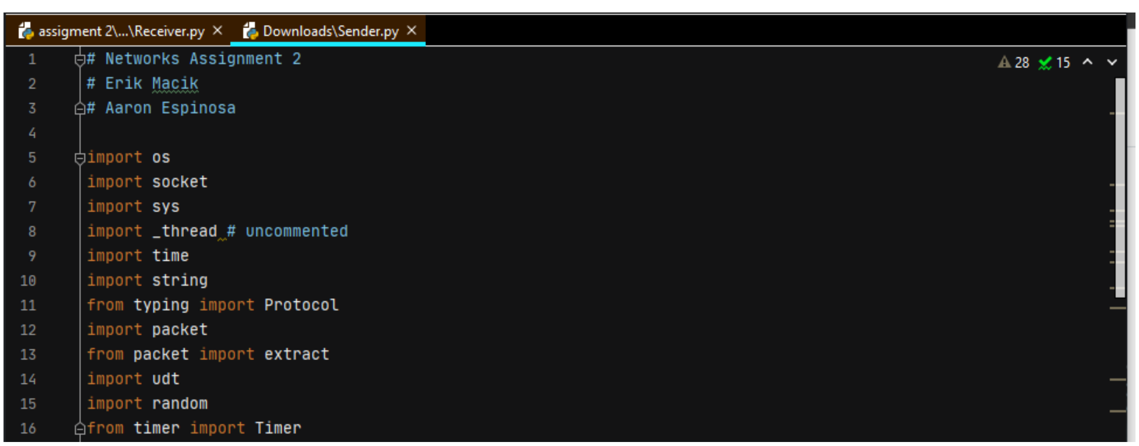


Figure 3 Import modules.

In this part the lines of code that are seen in the following image (figure 4) were uncommented and a new variable declared as *end*.

```
# Some already defined parameters
PACKET_SIZE = 512
RECEIVER_ADDR = ('localhost', 8080)
SENDER_ADDR = ('localhost', 9090)
SLEEP_INTERVAL = 0.05 # (In seconds)
TIMEOUT_INTERVAL = 0.5
WINDOW_SIZE = 4

# You can use some shared resources over the two threads
base = 0 # uncommented
end = 0
mutex = _thread.allocate_lock() # uncommented
timer = Timer(TIMEOUT_INTERVAL) # uncommented

```

Figure 4 Adding and uncomment variables.

Then, we specify the number of bytes of our file data to send assigning to our function payload the file and the size of the package. Later, all packets are getting from a file along with an ending packet. Finally, a list of packets is created of size of *packet_size* from the file. (Figure 5)

```
# Generate random payload of any length
def generate_payload(length=10):
    letters = string.ascii_lowercase
    result_str = ''.join(random.choice(letters) for i in range(length))

    return result_str

# Get specified number of bytes of data from file to send
def generate_payload_from_file(opened_file, length=PACKET_SIZE):
    return opened_file.read(length)

# Get all packets from a file along with an ending packet
def get_packets_from_file(filename, packet_size=PACKET_SIZE):
    packets = []
    file = open(filename, "r")
    file_size = os.path.getsize(filename)
    seq = 0

    # Create a list of packet of size packet_size from file
    while file_size > 0:
        data = generate_payload_from_file(file, packet_size).encode()
        file_size -= packet_size
        pkt = packet.make(seq, data)
        packets.append(pkt)
        print("Created packet seq#", seq)
        seq += 1
    end = packet.make(len(packets), "END".encode())
    packets.append(end)
    return packets

```

Figure 5 Creating and assigning the package size.

And the stop-and-wait protocol code was left the same. Figure 6.

```python
# Send packets from file using Stop_n_wait protocol
def send_snw(sock, filename, packet_size=PACKET_SIZE):
    global base

    # Get all packets from file
    packets = get_packets_from_file(filename)

    # Start thread to listen for acks from receiver
    _thread.start_new_thread(receive_snw, (sock, ))

    # Send each packet
    while base < len(packets):
        # base is only incremented by the thread listening for acks
        pkt = packets[base]
        print("Sending seq# ", base, "\n")
        udt.send(pkt, sock, RECEIVER_ADDR)
        timer.start()

        # loop while the timer is running
        while timer.running():
            # if it timed out, resend the packet
            if timer.timeout():
                timer.stop()
```

Figure 6 Stop-and-Wait protocol.

Once having the sender ready we move on to the receiver code. Here, we incorporate the missing part of the Stop-and-Wait code adding the file name and the variables that we used. Then, we added a while loop to continually receive the packets until we get the last one. Inside of the same loop, we used a condition statements if to tell the program if we already have the packet send a repeat ack, and write the data to a file if it does not end then send an ack of that packet to the sender. Figure 7.

```python
# Receive packets from the sender w/ Stop-n-wait protocol
def receive_snw(sock, filename):
    # File to write to
    file = open(filename, "w")
    ack_number = 0
    endStr = ''

    # Continually receive packets until we get the last one
    while endStr != 'END':
        pkt, senderaddr = udt.recv(sock)
        seq, data = packet.extract(pkt)

        # If we already have this packet, just send back a repeat ack
        if seq < ack_number:
            ack_packet = packet.make(seq, ''.encode())
            print("Sending Repeat ack#", seq)
            udt.send(ack_packet, sock, SENDER_ADDR)
            continue

        # Write the data to a file if it's not the end
        endStr = data.decode()
        if endStr != 'END':
            file.write(endStr)

        # Send ack of this packet to sender
        print("Received #", seq, "\n")
        ack_packet = packet.make(ack_number, ''.encode())
        print("Sending ack#", ack_number)
        udt.send(ack_packet, sock, SENDER_ADDR)

        # Keep track of sequence numbers received
        ack_number += 1
```

Figure 7 Stop-and-Wait Receiver code.

Once having the sender ready we move on to the receive code. Here, we incorporate the missing part of the GBN code adding the file name and the variables that we used. Then, we added a while loop to continually receive the packets until we get the last one. Figure 7.

```python
    # Continually receive packets until we get the last one
    while endStr != 'END':
        pkt, senderaddr = udt.recv(sock)
        seq, data = packet.extract(pkt)

        if seq == ack_num:
            ack_packet = packet.make(ack_num, ''.encode())
            print("sending ack for packet#", ack_num)
            udt.send(ack_packet, sock, SENDER_ADDR)
            ack_num += 1
            # Write the data to a file if it's not the end
            endStr = data.decode()
            if endStr != 'END':
                file.write(endStr)
```

Figure 8 Receive packets from the sender Stop-and-Wait.

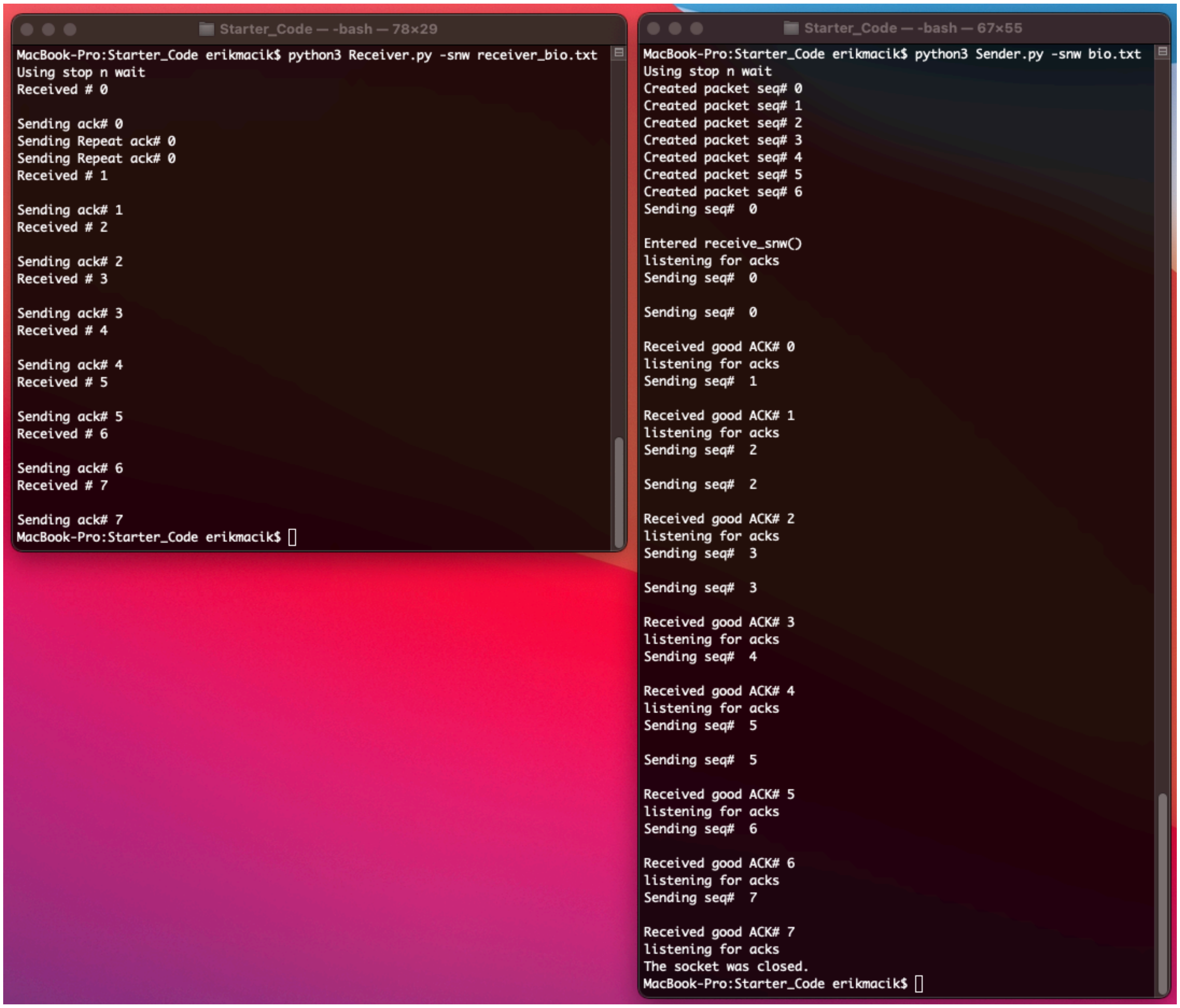
Finally, for the GBN protocol we incorporate the missing part code adding the file name and the variables that we used as well. Then, we added a while loop to continually receive the packets until we get the last one. And again, we used condition statements if to write the data to a file if it is not the end. Figure 9.

```python
# Receive packets from the sender w/ GBN protocol
def receive_gbn(sock, filename):
    file = open(filename, "w")
    ack_num = 0
    endStr = ''

    # Continually receive packets until we get the last one
    while endStr != 'END':
        pkt, senderaddr = udt.recv(sock)
        seq, data = packet.extract(pkt)

        if seq == ack_num:
            ack_packet = packet.make(ack_num, ''.encode())
            print("sending ack for packet#", ack_num)
            udt.send(ack_packet, sock, SENDER_ADDR)
            ack_num += 1
            # Write the data to a file if it's not the end
            endStr = data.decode()
            if endStr != 'END':
                file.write(endStr)
```

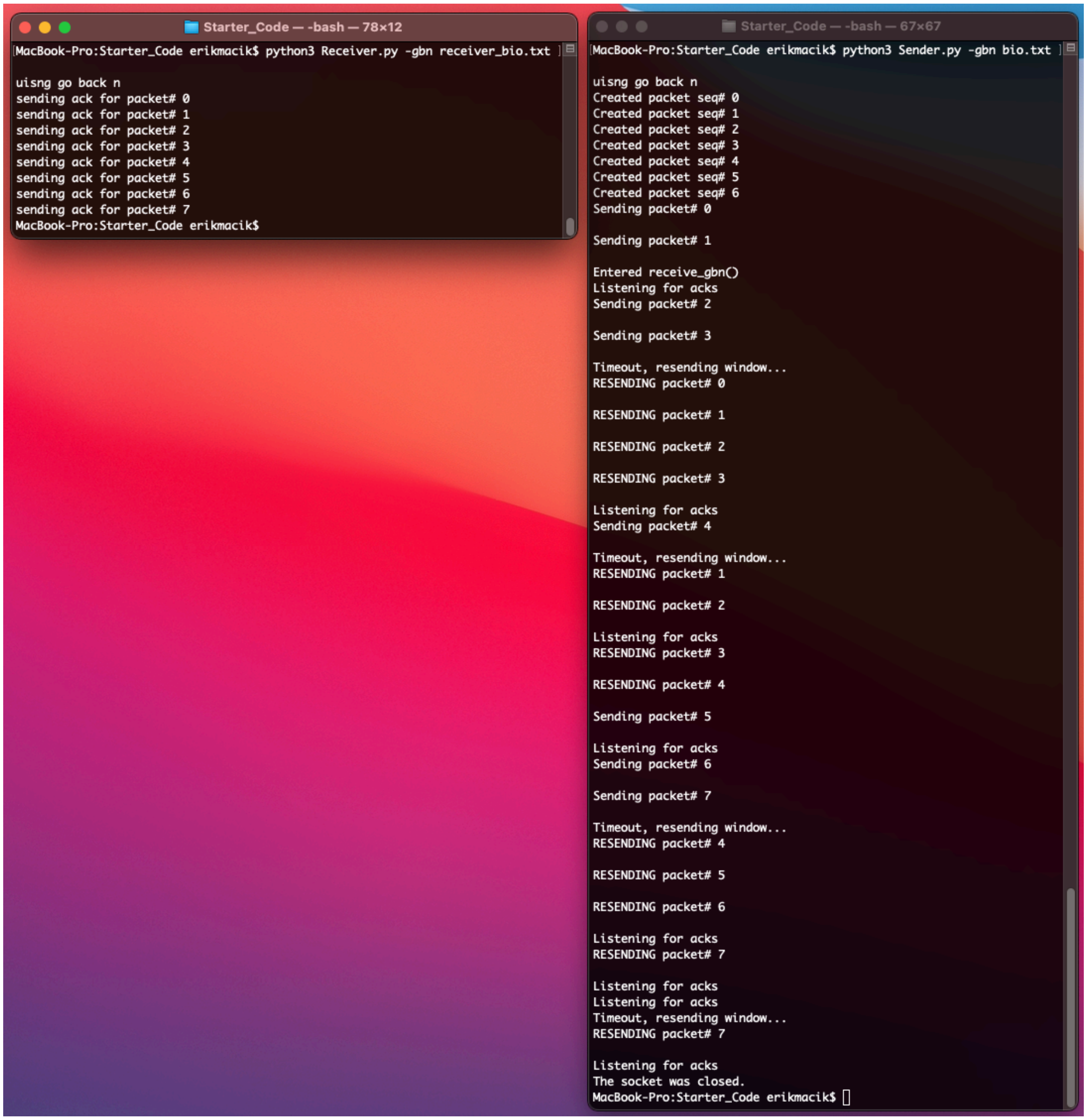
Figure 9 Receive packets from the sender GBN.

Conclusions

Stop-and-Wait Protocol



Go-Back-N Protocol



Students contribution

Task list	Erik Macik	Aaron Espinosa
Code	✓	✓
Report	✓	✓

References

[1] James F. Kurose, Keith W. Ross, «3.4 Principles of Reliable Data Transfer,» de *Computer Networking A Top-Down Approach*, New Jersey, Pearson, 2017.

[2] G. Shute, «Reliable Data Transfer,» University Of Minnesota Duluth, [En línea]. Available: https://www.d.umn.edu/~gshute/net/reliable-data-transfer.xhtml#:~:text=For%20connection%2Doriented%20service%20provided,designed%20using%20some%20basic%20tools.. [Último acceso: 5 Octubre 2020].

[3] James F. Kurose, Keith W. Ross, «Computer Networking A Top-Down Approach,» de *3.4.3 Go-Back-N (GBN)*, New Jersey, Pearson, 2017.