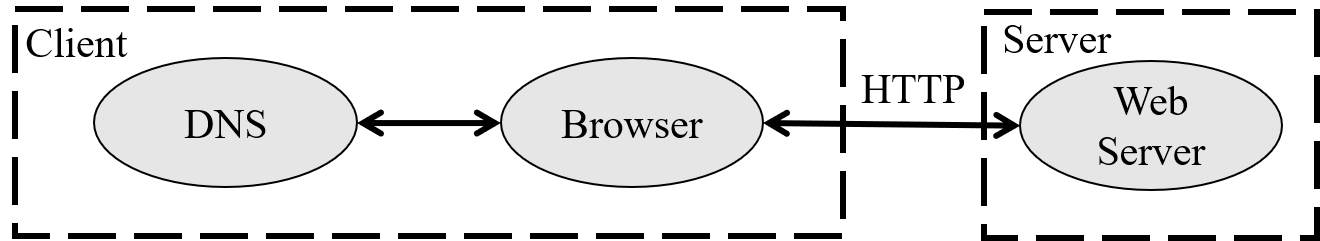
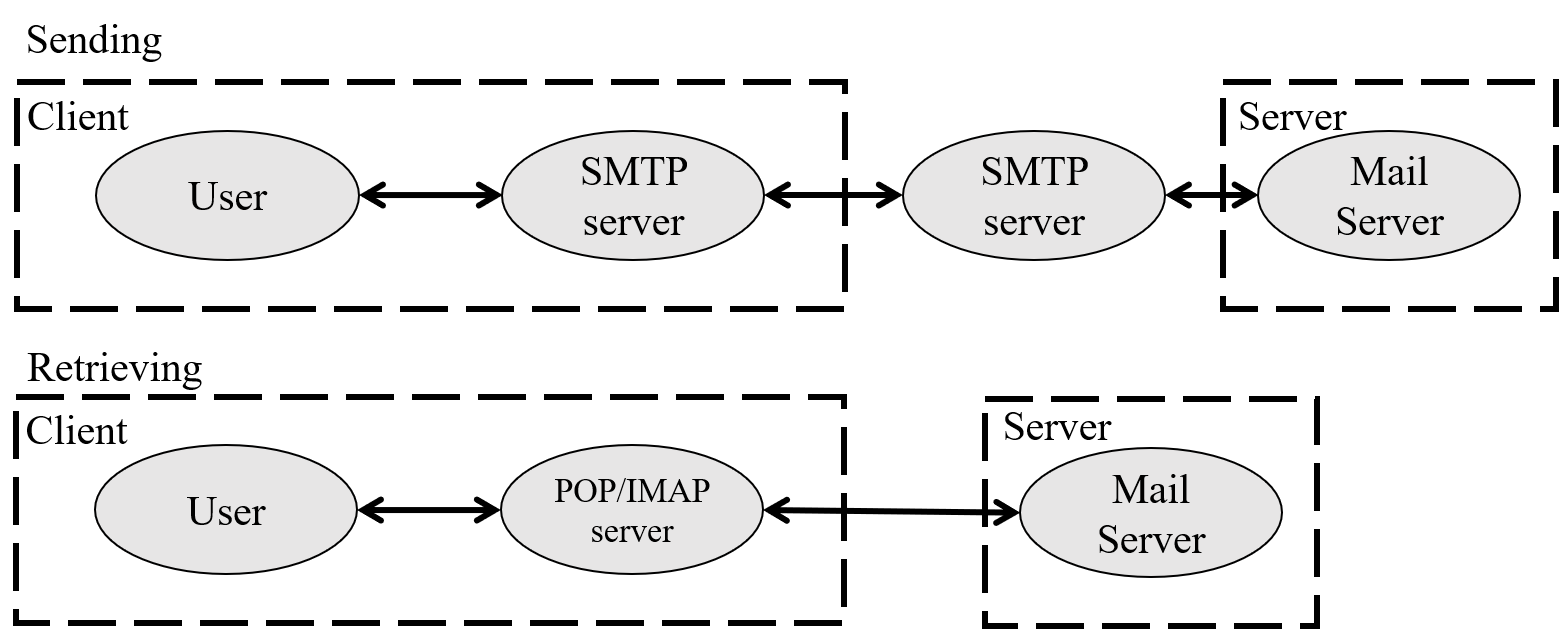
**CYEN 405 / CSC 452 / CSC 552 – Distributed and Cloud Computing, Fall 2018**

**Homework #2: Assigned Tues, Sept 25. Due in PDF format on Moodle Thurs, Oct 4 at 11:55pm.**

1. [20 pts.] Describe and illustrate the client-server architecture of the following two major Internet applications:
   1. **Pt. A:**The Web
      1. When a user is connecting to the Web through a browser, they are first connecting to a Domain Name Server, or DNS, then to a web server through HTTP. In this case the browser and DNS are the client and the web server is the server in the client-server architecture.
      2. 
   2. **Pt.** **B:**E-mail
      1. When a user sends an E-mail, the message is sent to an SMTP server where it is routed to the next SMTP server until it reaches the receivers Mail server. The receiver then retrieves the message by connecting to the Mail server and downloading the message through a POP or IMAP server.
      2. 
2. [15 pts.] Using the Python socket example given in Note 2.1 (pp. 59-60), design two new Python programs that act as server and client for a wireless sensor network.  The server should collect sensor readings from each of the clients and write them all to a file in the order they are received.  Explain how your code took the following considerations into account:

# Server.py

import socket

s = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

s.bind(('localhost', 80))

s.listen(1)

# Returns new socket and addr. client

conn, addr = s.accept()

print 'Connected by', addr

f = open('workfile.txt', 'w')

while True:

# Receive data from client

data = conn.recv(1024)

if (data == "quit" or not data):

break

print(data)

f.write(data + '\n')

# Close the connection

conn.close()

f.close()

print("Connection Closed.")

# Client.py

import socket

s = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

# Connect to server (block until accepted)

s.connect(('localhost', 80))

print("Type 'quit' to exit.")

while True:

user = raw\_input("Input data:")

# Send some data

s.send(user)

if (user == "quit"):

break

# Close the connection

s.close()

print("Connection Closed.")

* 1. The blocking or non-blocking nature of the send and recv calls
     1. The blocking nature of the recv call helped to keep the server from infinitely looping and not being where it needed to be when a request was received.
  2. How you prevent any deadlocks
     1. TO prevent deadlock, I made the client-server connection 1 way. The client sends what data it wants to be saved to the server and the server saves the data and waits for the next request.
  3. How you accommodate low client duty cycles
     1. The client duty cycle is determined by the user, as the client sends data given by the user to the server. Given this setup, the server waits however long it takes to receive a request.

1. [15 pts.] Using the RESTful architecture described in Sec. 2.1 and the Amazon S3 SOAP interface given in Note 2.2 (pp. 65-66), write the code that would be needed to perform the following actions using both interface:
   1. Create a new bucket named “DCCclass”

REST

PUT http://DCCclass.s3.amazonsws.com/

SOAP

import bucket

bucket.create("DCCclass")

* 1. Create a new object named “Homework2grades”

REST

PUT http://DCCclass.s3.amazonsws.com/Homework2grades

SOAP

AmazonS3Client client = new AmazonS3Client();

PutObjectRequest request = new PutObjectRequest

{

BucketName = "DCCclass",

Key = "Homework2grades",

ContentBody = "90"

},

{

BucketName = "DCCclass",

Key = "Homework2grades",

ContentBody = "77"

};

PutObjectResponse response = client.PutObject(request);

* 1. Get a list of objects contained within the bucket “DCCclass”

REST

GET http://DCCclass.s3.amazonsws.com/Homework2grades

SOAP

for object in DCCclass.Key.all():

print(object)

1. [10 pts.] Using the Linda tuple spaces example given in Note 2.3 (pp. 68-70),
   1. Extend the code fragments such that a next message will be selected instead of a random one

{

int i = 1;

blog = linda.universe\_rd(("MicroBlog", linda.TupleSpace))[1]

blog.\_out((i = 1, "bob","distsys","I am studying chap 2"))

blog.\_out((i = 2, "bob","distsys","The linda example's pretty simple"))

blog.\_out((i = 3, "bob","gtcn","Cool book!"))

blog = linda.universe\_rd(("MicroBlog", linda.TupleSpace))[1]

blog.\_out((i = 4, "alice","gtcn","This graph theory stuff is not easy"))

blog.\_out((i = 5, "alice","distsys","I like systems more than graphs"))

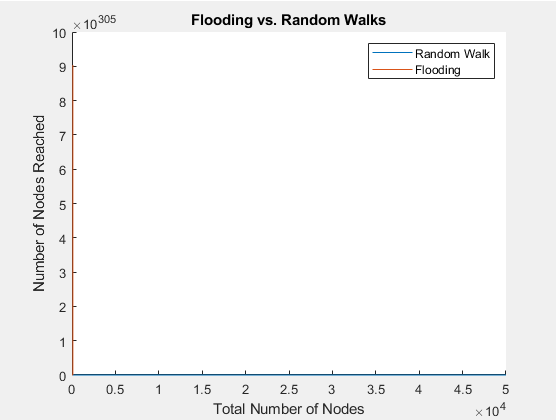
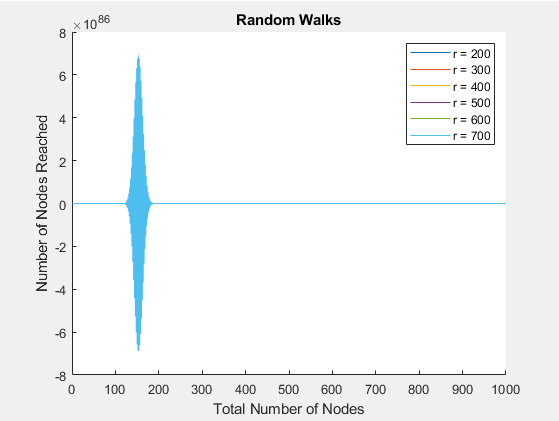
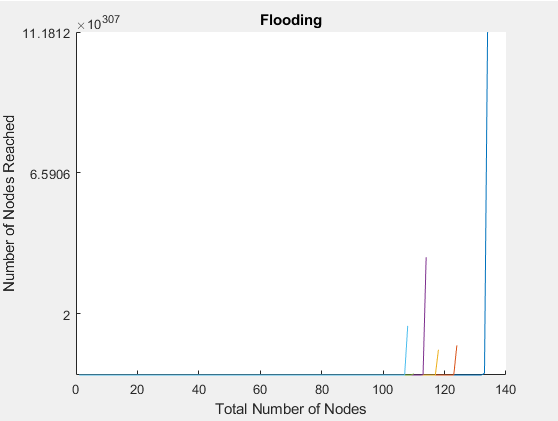
while result.topic == "distsys":

i = i + 1

t = blog\_rd((i, str, "distsys", str))

}

* 1. How could you extend the code fragments to ensure that any user cannot post messages under a different user’s name?
     1. By adding a layer of security and requiring a form of authentication before posting as a certain name should prevent poster name thievery.

1. [20 pts.] Considering the Flooding vs. Random Walks discussion in Note. 2.6 (pp. 85-86).
   1. Create a plot with two lines (one for flooding & one for random walk) of the number of nodes reached/probed before finding the requested data item versus the total number of nodes in the P2P network, *N*.  Determine a reasonable range of *N* values, and use realistic constant values for numbers of replicated nodes/neighbors *r* and *d*.
      1. 
   2. Create a new plot of the number of nodes reached/probed versus *N* using random walk but choosing a reasonable range of 3-10 *r* values to be represented as different lines.
      1. 
   3. Create a new plot of the number of nodes reached/probed versus *N* using flooding but choosing a reasonable range of 3-10 *d* values to be represented as different lines.
      1. 
   4. Referencing the plots that you made in parts (a)-(c), state under which circumstances flooding is preferable to random walk and vice versa.
      1. Random Walk, being more communication efficient, surpasses Flooding when time is not an obstacle, but communication is. Meaning the user is willing to wait for the data item to be found and communication resources may be occupied with other tasks. Conversely, Flooding surpasses Random Walk when the user needs the data item as fast as possible and connection resources are available.
2. [10 pts.] In your own words, describe the purpose of the Resource Pooling architecture described in Erl Ch. 11 and and list common types of resources that are grouped into resource. Further, list the three common forms of resource pool groups, explain resource pool hierarchies, and list the cloud computing mechanisms associated with this architecture.
   1. Resource pooling architecture is used to group homogenous resources so that they can be maintained and synchronized by a system. The resources that are commonly grouped together into a single resource are: physical or virtual servers, storage, network devices, CPUs, and memory. These pools have three main forms, parent, sibling, and nested pools Resource pool hierarchies are used to group different resource pools together for different purposes. Sibling pools are used to create separate pools for different consumers. Nested pools are used to divide a pool into smaller sections, often for different subsections of the cloud consumer. The cloud computing mechanisms commonly associated with this architecture are: audit monitors, cloud usage monitors, hypervisors, logical network perimeters, pay-per-use monitors, remote administration systems, resource management systems, and resource replication.
3. [10 pts.] In your own words, describe the purpose of the Redundant Storage architecture and explain the related use of primary and secondary storage devices as part of a failover system. List the cloud computing mechanisms associated with this architecture.
   1. The purpose of redundant storage architecture is to allow access to cloud storage even when the main database is unavailable. The primary storage device is simply the device that data is stored on under normal operations, data is copied from this device onto the secondary storage device. When the primary storage device goes offline the storage service gateway forwards the requests for the data to the secondary storage device, which will return or update the data as needed. When the primary storage device comes back online the changes made on the secondary storage device will be copied back to the primary device. This system prevents the failure of one storage device from interrupting normal operations. The primary cloud computing mechanism associated with this architecture is the storage replication system.