Questions (5 points)

1. (1 point)
   (a) Define what is meant by a scalable system.
   (b) Discuss why message delays in the Internet grow only very slowly with the number of networks in the Internet. How are the networks in the Internet configured to achieve this type of scalability?
   (c) State three techniques commonly used to achieve scalability.

Answer (a):
A system is described as scalable if it will remain effective when there is a significant increase in the number of resources and in the number of users in it.

Answer (b):
The Internet is constructed from many subnets. A subnet is a unit of routing (delivering data from one part of the Internet to another). Some substantial changes to the addressing and routing mechanisms are planned in order to handle the next phase of the Internet’s growth. These include addressing schemes (e.g. IP addressing, and etc.) and routing algorithms (RIP, OSPF, default routing scheme, and etc.), and etc. Read Chapter 3 for details.

The Internet is configured as a tree hierarchy, with “small subnets” combined together to form a “bigger” network, from the bottom level to the top level. By configuring hosts like this, only a few gateway or routers provide access to each subnet. The packet transmission process is as follows. If a packet is addressed to a host on the same network as the sender, it is transmitted in a single hop, using the host identifier part of the address to obtain the address of the destination host on the same network. If the destination is on a different network, the packet should be sent to a local router. Then the network address of the gateway or router will be obtained and the packet is transmitted there. After that the packet is transmitted locally.

Answer (c):
Hierarchical partitioning of resources; replication; caching; multiple servers…

2. (1 point)
   (a) Distinguish between buffering and caching.
   (b) Consider two communication services for use in asynchronous distributed systems. In service A, messages may be lost, duplicated or delayed and checksums apply only to headers. In service B, messages may be lost, delayed or delivered too fast for the recipient to handle them, but those that are delivered arrive with the correct contents. Describe the classes of failure exhibited by each service. Classify their failures according to their effect on the properties of validity and integrity. Can service B be described as a reliable communication service?
Answer (a):
Buffering: a technique for storing data transmitted from a sending process to a receiving process in local memory or secondary (disk) storage until the receiving process is ready to consume it. For example, when reading data from a file or transmitting messages through a network, it is beneficial to handle it in large blocks. The blocks are held in buffer storage in the receiving process’ memory space. The buffer is released when the data has been consumed by the process.

Caching: a technique for optimizing access to remote data objects by holding a copy of them in local memory or secondary (disk) storage. Accesses to parts of the remote object are translated into accesses to the corresponding parts of the local copy. Unlike buffering, the local copy may be retained as long as there is local memory available to hold it. A cache management algorithm and a release strategy are needed to manage the use of the memory allocated to the cache. (If we interpret the word ‘remote’ in the sense of ‘further from the processor’, then this definition is valid not only for client caches in distributed systems but also for disk block caches in operating systems and processor caches in cpu chips.)

Answer (b):
Service A can have:
Arbitrary failures:
– as checksums do not apply to message bodies, message bodies can corrupted.
– duplicated messages,
Omission failures (lost messages).
Because the distributed system in which it is used is asynchronous, it cannot suffer from timing failures.
Validity - is denied by lost messages
Integrity - is denied by corrupted messages and duplicated messages.

Service B can have:
 omission failures (lost messages, dropped messages).
Because the distributed system in which it is used is asynchronous, it cannot suffer from timing failures.
It passes the integrity test, but not the validity test; therefore it cannot be called reliable.

3. (1 point)
(a) If a system does not provide a blocking receive, can you simulate it by a non-blocking receive? If so, show a pseudo code. If not, explain why. Discuss your answer for the case of a single thread and for the case where you can use multiple threads.

(b) If a system does not provide a non-blocking send, can you simulate it by a blocking send? If so, show the pseudo code. If not, explain why. Discuss your answer for the case of a single thread and for the case where you can use multiple threads.
Answer (a):
For both single-threaded and multi-threaded cases, the situation is the same. The pseudo code is as follows:

```java
blocking_receive(...) {
    while (nonblocking_recieve(...) == -1) 
        ;
}
```

Answer (b):
For single-threaded case, there is no way to simulate a non-blocking send using a blocking send since there is just only one thread.

For multi-threaded case, it is possible to do it as follows:

```java
int buffer[...];
in thread_id;
nonblocking_send(...) {
    Down the semaphores on buffer and thread_id;
    Set thread_id to be the calling thread’s thread id;
    Spawn a thread which executes the blocking send (After it returns, it sends a signal to the calling thread and ups the semaphores on buffer and thread_id);
    The calling thread continues its execution by returning from this nonblocking_send;
}
```

Note: The solution to this problem (multi-thread non-blocking send simulation) is not unique.

4. (1 point) Use the program in Figure 4.3 to make a client program that repeatedly reads a line of input from the user, sends it to the server in a UDP datagram message, then receives a message from the server. The client sets a timeout on its socket so that it can inform the user when the server does not reply. Test this client program with the server in Figure 4.4.

Answer:
The program is as Figure 4.3 with the following amendments:

```java
DatagramSocket aSocket = new DatagramSocket();
aSocket.setSoTimeout(3000); // in milliseconds
while ( // not eof ) {
    try{
        // get user’s input and put in request
        aSocket.send(request);
        ....
        aSocket.receive(reply);
    }catch (InterruptedIOException e){System.out.println("server not responding");}
```
5. (1 point) To get you started in Java programming, this question asks you to understand a simple Java program, run it, and explain how it produces the output.

Compile and run the Time-of-Day Server (*ToDServer.java*) with the class *Connection* (*Connection.java*) downloadable from the course web page. Type

```
telnet 127.0.0.1 5555
```
and observe the output.

(a) Explain the sequence of events in the server and client that led to the observed output.

(b) Next, use *telnet* from a remote machine. In this case, you need to replace the first argument (127.0.0.1) by something else. Give two possible replacements.

**Answer (a):**
On the server side, the server first starts up. It then creates a server socket with the port number. After that the server goes into an infinite loop, listening and waiting for clients requests. On the client side, the client uses the command telnet to make a connection request. When the server accepts the connection request from the client, it creates a new socket. Then it spawns a new thread by creating a connection object. The server passes the new socket (as a parameter to the constructor of the connection object) to the new thread. The new thread is running, sending the date and time back to the client, and then it closes the new socket. After that, it terminates. This finishes a service from the server to the client. The client receives the date and time and then prints them out. The client then closes the socket it created for this communication. Finally telnet exits.

**Answer (b):**
We can specify either the IP address or the hostname of the server host.