# Computer Systems Solutions to Assignments 0, 1, and 2/

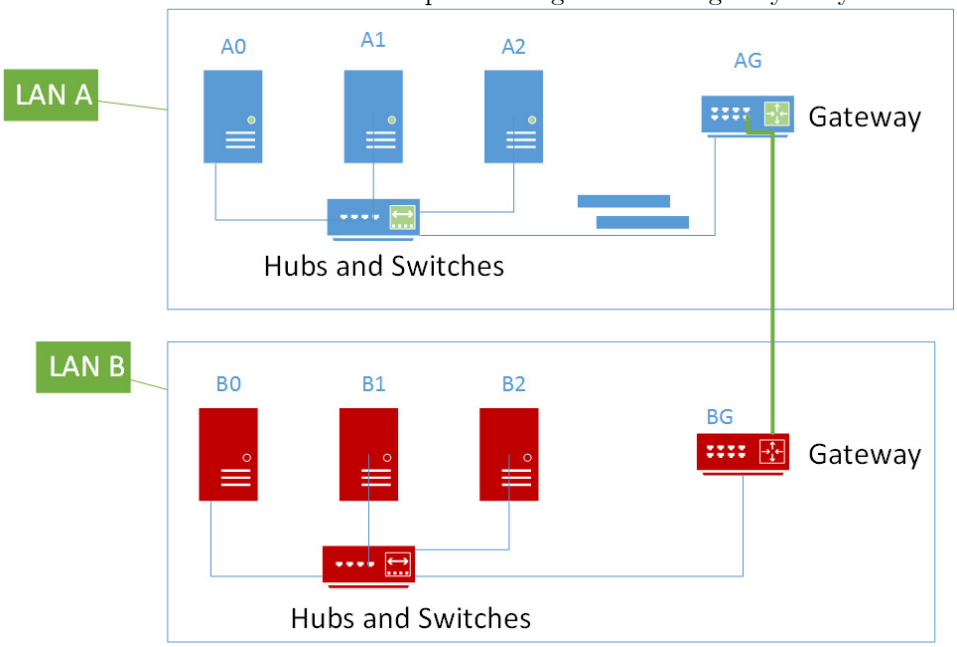
Theodore S. Norvell

June 19

**A0-Q0 [10]** Suppose two LANs are connected by a high-speed link. A host A0 on LAN A sends an single internet packet to a host B2 on LAN B. See my notes for a picture. Show a series of pictures illustrating how A0 can send a packet of total size 1000 bytes (octets), including header, to B2. Assume that on LAN A data frames are limited to a payload of 800 bytes, that the gateways communicate with data frames of at most 500 bytes of payload and that LAN B uses data frames of at most 400 bytes of payload.

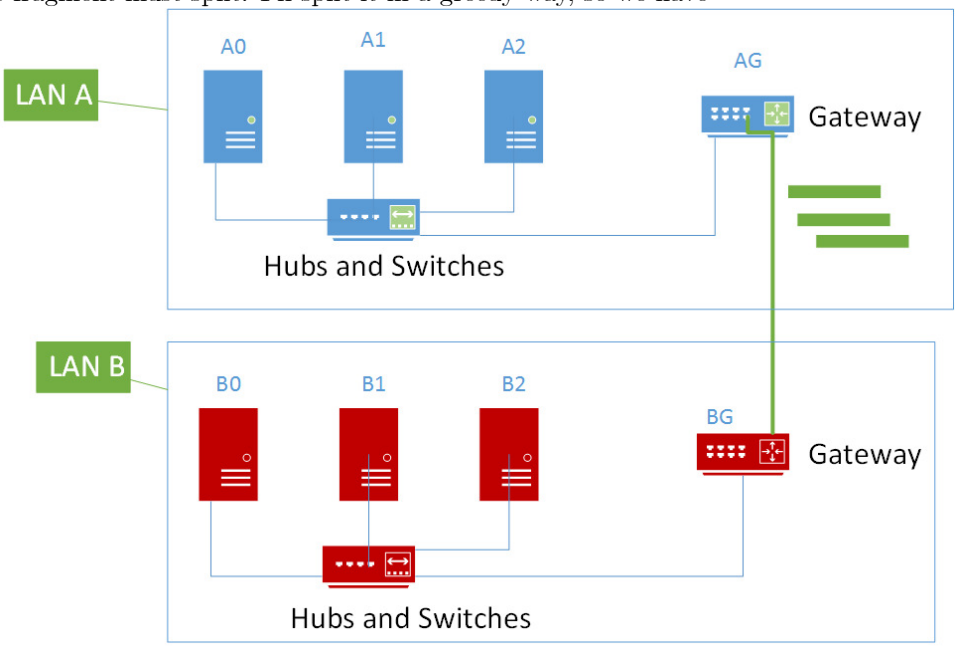
**My solution:** While there are many things that could happen, here is a sensible set of events. (I don't say sequence of events because there could be a lot of sequences made out of this set. E.g. the first fragment from A0 could be delivered to B2 before A0 gets around to sending the second fragment.) First off, how much data is there? Since an IPv4 header is 20 bytes and the total is 1000 bytes, there is 980 bytes of IP payload.

- Host A0 sends out 2 fragments to its gateway via LAN A. Since data frames are limited to payloads of 800 bytes we need at least two fragments. It could send three or four, but there is really no reason to, since host A0 is on LAN A and is presumably ignorant of the restrictions on the link and on LAN B. I'll split the fragments in a "greedy" way.



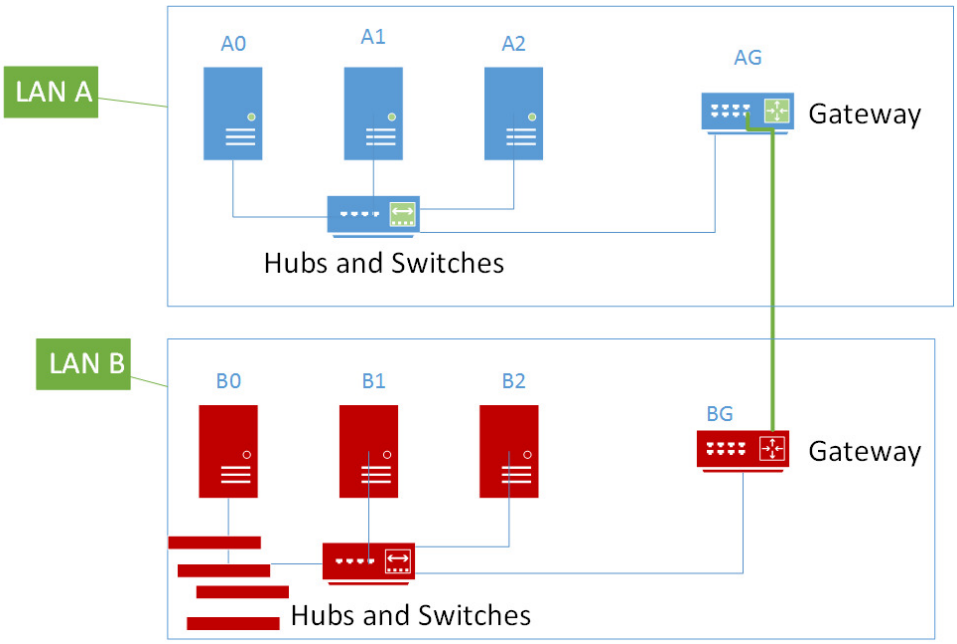
  - One fragment on LAN A has 20 bytes of header and 780 bytes of IP payload, for a total of 800 bytes of data frame payload.
  - The other has 20 bytes of header and 200 bytes of IP payload.

- For gateway to gateway communication, a data frame payload of 500 bytes is maximal. So the first fragment must split. I'll split it in a greedy way, so we have




  - A fragment of 20 + 480 bytes
  - A fragment of 20 + 300 bytes
  - A fragment of 20 + 200 bytes

- At the gateway to LAN B, the first fragment is too big for a data frame payload so it is split by the gateway. We have:




  - A fragment of 20 + 380 bytes
  - A fragment of 20 + 100 bytes
  - A fragment of 20 + 300 bytes
  - A fragment of 20 + 200 bytes

- These are delivered to host B2 and reassembled to make a complete packet.

**Discussion:** I would have given full marks for a solution that started with 3 fragments of size 400 or less, if that decision of A's was justified. There is a mechanism that hosts can use to discover the minimum maximum payload size along a routing path; it's called Path MTU Discovery; but no one mentioned it, or the need for A0 to somehow figure out that 3 fragments of less than 400 bytes each is what is needed. The internet is a big place: you can't expect each host to know about all the other LANs and their limits.

Some students thought that the packet would be reassembled at each gateway. This isn't what happens. Fragments are router (and possibly further fragmented) by routers and gateways, but are only reassembled at their final destination.

**A0-Q1 [5]** Suppose that the gateways of two LAN networks, A and B, are connected by a series of links that can only handle IPv6 packets. Can a computer on LAN A send IPv4 packets to a computer on LAN B? What must the gateway do to make this possible.

**My solution**: The point I was hoping to reenforce via this question is that IPv4 fragments can be moved over almost any sort of network technology. This is the key idea behind the internet. The internet is layered above a heterogeneous collection of networks. So with that in mind, we can use IPv6 to move IPv4 fragments, just the same as we would use Ethernet to move IPv4 fragments. So my solution is that the gateways send IPv6 packets between themselves (i.e. addressed to the other gateway) with IPv4 fragments as the payloads.

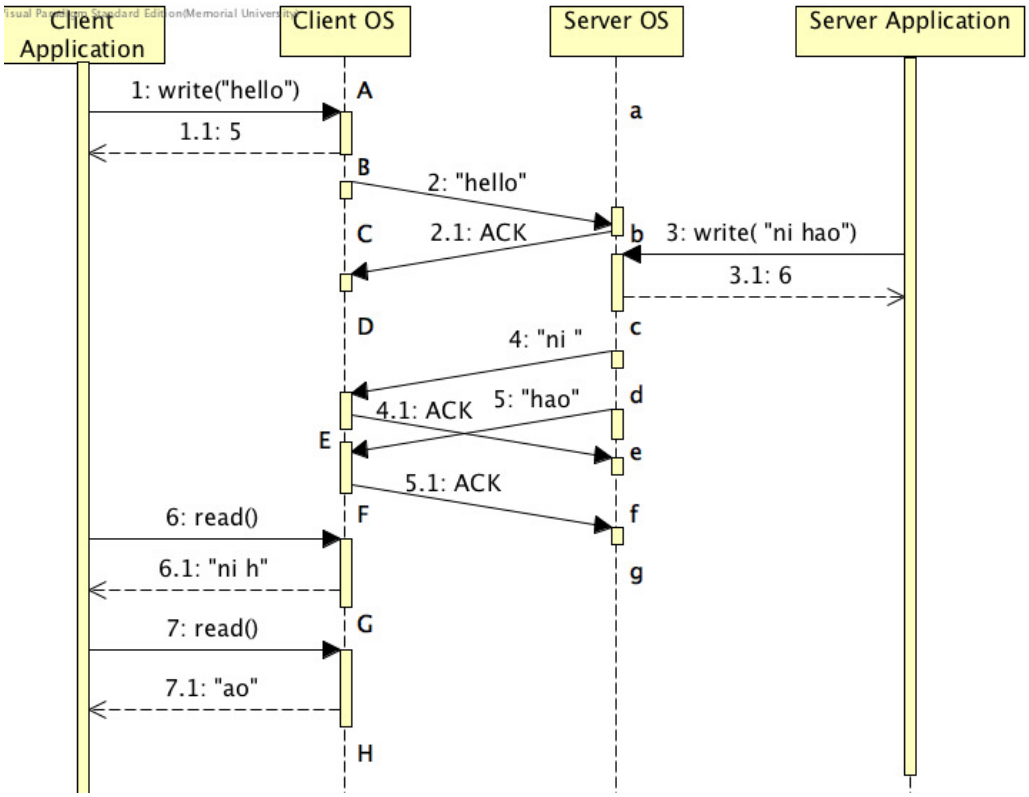



Figure 0: Times when the OSs are inactive with respect to this TCP connection are labelled by letters.

**Discussion:** My solution is an example of tunnelling. Tunnelling is useful for a lot of things. For example a VPN might tunnel IPv4 traffic over an IPSec (secure IP) link, running over an IPv4 link.

The other approach is translation from one kind of packet to another. This in not as simple as using an IPv6 address in place of an IPv4 address, because IPv6 is a completely different protocol. For example IPv6 does not support fragmentation. I was more inclined to give high marks for translation based answers if they showed a little research had been done.

**A1-Q0 [10]** Make a copy of the diagram on page 8 of slide set networks 2. For each point in time show the contents of the read buffer and the write buffer for both the client and the server.

**My solution:** Figure 0 is the diagram with time periods that the OS is inactive with respect to this TCP connection labelled by letters

The write buffer on the client.

- A: "" by assumption
- B: "hello"
- C: This is an interesting one. The message has been sent but the OS can not delete the data until receipt is acknowledged. (Recall that IP is unreliable, so the packet could be lost in

which case the client OS must resend it.) We could say that the buffer is "" but that the data is retained elsewhere. Or we could say that the buffer is "hello", but that the OS must remember that the first 5 bytes (assuming ASCII coding) of the buffer have already been sent and are awaiting acknowledgement. I'll pick the first approach and say that the buffer is "".

- D, E, F, G, H: The write buffer is empty and the OS does not need to retain the first 5 bytes sent.

The write buffer on the server:

- a: "" by assumption
- b: ""
- c: "ni hao".
- d: "hao". The first 3 bytes must be retained.
- e: "". The first 6 bytes must be retained.
- f: "". Only bytes 3,4 & 5 must be retained.
- g: "". No bytes need be retained.

The read buffer on the client:

- A: "" by assumption
- B, C, D: ""
- E: "ni "
- F: "ni hao"
- G: "oa"
- H: ""

The read buffer on the server:

- a: "" by assumption.
- b, c, d, e, f, g: "hello".

**Discussion:** The most interesting part of this question was at times C, d, e, and f. Unfortunately almost no student had any comment on the situation at these point. Many simply ignored some or all of these times, leading to incomplete answers.

Some students got confused between the OS read buffer (which is what the question was intended to address) and the application's private storage. We have no way of knowing what the application does with the data. It may collect it all in it's own buffer or it may not.

**A1-Q1 [5]** On the same diagram, suppose that the packets containing "ni " and "hao" arrived out of sequence. Explain how the read buffer on the client side would look during the time between the arrival of the packet containing "hao" and the packet containing "ni ".

**My solution:** If the packet containing later bytes arrives first, we could consider that the read buffer remains empty, but that the bytes are retained for later insertion, or we could say the bytes are put in the buffer, but that there is now a gap in the buffer. (This gap will cause any application read call to block until the right bytes arrive.) In the latter case we could notate the buffer like this: "□□□hao" where the boxes represent a gap in the data. In any case once the first packet sent arrives, the read buffer will be "ni hao".

**Discussion:** The important principle here is that TCP is a streaming protocol. This means two things. First that the way that the data is broken into packets is entirely transparent to the application: It just sees a sequence of bytes. The second thing —and this is the important point that many students missed— is that the sequence of bytes that receiving application reads must be exactly the same as the sequence of bytes that the writing process writes. So it is important that the receiving process somehow gets the lumps of data ("segments" is the technical term) into the right order before delivering them to the application.

**A2-Q0 [10]** Suppose you want to find the sum of all the values in an array of length $n$. Show how this can be computed with $n$ processors in time roughly proportional to $\log n$. Give the pseudocode for the algorithm. If you wish you may assume $n$ is a power of 2.

**My first solution:** We can use parallel recursive calls to solve this very neatly. Suppose the data is in an array $a$ of length $n$. We write a function "sum" to compute the sum of the segment from $a[p]$ up to and including $a[p+l-1]$. The whole sum can the be computed by calling $sum(a, 0, n)$.

```
function sum(a : array n of real, p : int, l : int)
    if l = 0 then return 0.0
    elseif l = 1 then return a[p]
    else
        var m := ⌊l/2⌋
        var x, y : real
        concurrently
            x := sum(a, p, m)
        ||
            y := sum(a, p + m, l − m)
        end concurrently
        return x + y
    end if
end function
```

This requires up to $n$ processors and time proportional to $\log_2 n$.

**My second solution:** Another approach is to use iterative parallelism. Again assume the data is in an array $a$ of length $n$. We use $\lceil \log_2 n \rceil$ rounds. The loop invariant for the while loop is that, each element of array $s$ whose index $k2^i$ is a multiple of $2^i$ will contain $\sum_{j=k2^i}^{\min((k+1)2^i-1,n-1)} a[j]$.

```
concurrently for j ∈ {0, .., n − 1}
    s[j] := a[j]
end for
var i : int := 0
while 2^i < n
    concurrently for k ∈ {0, .., ⌈n/2^(i+1)⌉ − 1}
```

```
        if (2k+1)2^i < n then s[2k2^i] := s[2k2^i] + s[(2k+1)2^i]
        end if
    end for
    i := i + 1
end while
```

This needs only $n/2$ processors and takes time roughly proportional to $\log_2 n$.

**Discussion:** Oddly, no students used the recursive approach above. My advice is to learn to use recursive techniques. Several students proposed something similar to my second solution. Some variations included using a 2D array.

Using the same number of processors we can compute the suffix sum so that at the end $s[k] = \sum_{j=k}^{n-1} a[j]$, for all $k$. The loop invariant for the while loop is that, each element of array $s[k]$ contain $\sum_{j=k}^{\min(k+2^i-1,n-1)} a[j]$.

```
concurrently for j ∈ {0,..,n-1}
    s[j] := a[j]
end for
var i : int := 0
while 2^i < n
    concurrently for k ∈ {0,..,n}
        if (2k+1)2^i < n then s[2k2^i] := s[2k2^i] + s[(2k+1)2^i]
        end if
    end for
    i := i + 1
end while
```

However there is a bug here as there is a race between threads reading locations in $s$ and other threads writing the same location in $s$. To fix this we can copy all the values to a second array like this

```
concurrently for j ∈ {0,..,n-1}
    s[j] := a[j]
end for
var i : int := 0
while 2^i < n
    var t : array n of real
    concurrently for k ∈ {0,..,n}
        t[k] := s[k]
    end for
    concurrently for k ∈ {0,..,n}
        if (2k+1)2^i < n then s[2k2^i] := t[2k2^i] + t[(2k+1)2^i]
        end if
    end for
    i := i + 1
end while
```

There was a disturbing amount of copying on this question. If several people hand in copies of the same assignment, it is doubtful that more than one or two are learning much from it.

**A2-Q1 [10]** Suppose we represent a mutable set of numbers with the following subroutines.

```
/* Add i to the set */
void addNumber( int i )
/* Return 1 if the set contains i. Otherwise return 0. */
int contains( int i )
/* Return 1 if the set is empty. Otherwise return 0. */
int isEmpty()
/* Return any member of a nonempty set */
int anyMember()
/* Remove i from the set.*/
void remove(int i)
```

It is intended that this set will be shared by a number of threads. It is intended that the following code will obtain a member of the set and also remove that member from the set so that no other thread has the same member.

```
while( isEmpty() ) { wait a little while }
int n = anyMember() ;
remove( n ) ;
```

Is there a problem with this design? What is it? How could you fix it?

**My solution:** There are a few problems with this approach. For example a thread may exit the while loop when the set is not empty, only to find that when it obtains "any member" the set has been made empty by another process. Also two threads may can anyMember at about the same time and thus take the same value, which was not the intention.

The way that I would solve this is to change the interface of the set object. I would change it to

```
/* Add i to the set */
void addNumber( int i )
/* Block until the set is not empty, then atomically remove one item and return it */
int takeAnyMember()
```

For the purpose of the disk copying program from the notes, this will do what we need.

Then the code to take a member simply becomes

```
int n = takeAnyMember() ;
```

Discussion: Many students identified the problem. In some cases though the wording was so vague that it was not clear what the problem was. Interestingly no one proposed changing the interface. Some proposed making the set's lock available to its clients. This will work, but it is not elegant.

On this question also, there was a disturbing level of copying.

**Some further questions:**

We didn't get to have any homework using mutexes or conditions. Here are two problems to solve. These will not be marked, but if you send me your solutions, I'll send back my comments.

**A3-Q0:** Implement the mutable set with the following interface. The second and third func-

tions should be suitable for multiple threads to use.

```
/* Initialize the set */
void initSet() ;
/* Add i to the set */
void addNumber( int i )
/* Block until the set is not empty, then atomically remove one item and return it */
int takeAnyMember()
```

Use the mutex and condition variable types from pthreads. You should need one mutex and one condition variable.

**A3-Q1:** Consider a narrow bridge that goes north/south. Up to 5 cars are allowed on the bridge, but they all must be going the same direction. Implement the following interface

```
void init() ;
northBoundEnters() ;
northBoundExits() ;
southBoundEnters() ;
southBoundExits() ;
```

The two 'enters' routines should block until it is safe for the car to enter the bridge. You may assume that each thread alternately calls an enter routine and the corresponding exit routine.

Use the mutex and condition variable types from pthreads. You should need one mutex and one condition variable.

As a first step ensure that the controller has the following safety property:

$$0 \leq n \leq 5 \text{ and } 0 \leq s \leq 5 \text{ and } (n = 0 \text{ or } s = 0)$$

where $n$ is the number of threads that have called a northBoundEnter routine but not subsequently the northBoundExit routine, and $s$ is the number that have called the southBoundEnter routine but not subsequently the southBoundExit routine.

As a second step ensure that an infinite supply of north bound traffic does not lockout the south bound traffic and vice versa. This is a liveness property.