University of California, Berkeley
College of Engineering
Computer Science Division — EECS

Fall 2006 John Kubiatowicz

Midterm II
SOLUTIONS

December 4th, 2006

CS162: Operating Systems and Systems Programming

Your Name:	
SID Number:	
Circle the letters of CS162 Login	First: a b c d e f g h I j k l m n o p q r s t u v w x y z Second: a b c d e f g h I j k l m n o p q r s t u v w x y z
Discussion Section:	

General Information:
This is a **closed book** exam. You are allowed 1 page of **hand-written** notes (both sides). You have 3 hours to complete as much of the exam as possible. Make sure to read all of the questions first, as some of the questions are substantially more time consuming.

Write all of your answers directly on this paper. *Make your answers as concise as possible.* On programming questions, we will be looking for performance as well as correctness, so think through your answers carefully. If there is something about the questions that you believe is open to interpretation, please ask us about it!

Problem	**Possible**	**Score**
1	20	
2	20	
3	27	
4	10	
5	23	
Total		

[This page left for π]

3.14159265358979323846264338327950288419716939937510582097494 4

Problem 1: True/False

In the following, it is important that you *EXPLAIN* your answer in TWO SENTENCES OR LESS (Answers longer than this may not get credit!). Also, answers without an explanation *GET NO CREDIT*.

Problem 1a[2pts]: Memory Mapped I/O devices are accessed with special instructions not used for normal computation.

True / (False)

Explain: *Memory Mapped I/O devices are accessed with normal loads and stores*

Problem 1b[2pts]: A single file server can be constructed with a Byzantine Agreement algorithm such that clients are protected against break-ins to the server.

True / (False)

Explain: *A Byzantine Agreement algorithm requires 3f+1 nodes to tolerate f malicious faults. A single file server couldn't tolerate any malicious faults.*

Problem 1c[2pts]: When initiating a secure communication over the web (such as for a bank), you need to trust the public key of a certificate authority that is compiled into your browser.

(True) / False

Explain: *When initiating, the server sends a certificate that contains its public key. To verify this certificate (and public key), you use the public key of the CA.*

Problem 1d[2pts]: The limitation in TCP for 65536 ports is due to hardware limitations.

True / (False)

Explain: *This is purely a software convention (format of TCP and UDP header)*

Problem 1e[2pts]: Randomness is essential to achieving good performance from the Ethernet communication algorithm (CSMA/CD).

(True) / False

Explain: *Randomness is required to break up collisions so that they are eventually resolved.*

Problem 1f[2pts]: "Marshalling" is the process by which Byzantine Generals are forced into making good decisions.

True / (False)

Explain: *Marshalling is the process of packaging data items (such as arguments for an RPC) into a network message.*

Problem 1g[2pts]: RAID-5 is 5 times more reliable than RAID-1.

True / (False)

Explain: *Actually, the numbers have nothing to do with the level of reliability. Both systems can handle a single disk failure.*

Problem 1h[2pts]: One advantage of a software TLB is that the same hardware platform can support both forward and inverse page tables.

(True) / False

Explain: *Since TLB faults are handled in software, the OS can choose to implement page tables in any way desired.*

Problem 1i[2pts]: The *Use* bit in the PTE is required for implementing the Second Chance List algorithm.

True / (False)

Explain: *In fact, the Second Chance List algorithm was developed for a machine that was missing the Use bit in its PTE.*

Problem 1j[2pts]: The difference between a computer Worm and a computer Virus is that a Virus requires human action to spread.

(True) / False

Explain: *This is the standard definition – Viruses are often transported as attachments in email or on a floppy boot track and require someone to actually click on the attachment or boot the disk to start. Worms can arrive all by themselves, via security flaws in the OS.*

Problem 2: Virtual Memory, Paging, and Disks

Problem 2a[2pts]: Suppose that we have a 64-bit virtual address split as follows:

9 Bits [Table ID]	13 Bits [Table ID]	13 Bits [Table ID]	13 Bits [Page ID]	16 Bits [Offset]

Show the format of a PTE complete with bits required to support the clock algorithm, sharing of dynamic libraries, and copy-on-write optimizations.

Answer:

48 Bits [Physical Page Frame]	***12 Bits [Other Uses]***	***Dirty***	***Use***	***Writable***	***Valid***

Problem 2b[3pts]: Explain how to implement the clock algorithm (*not the Second Chance List algorithm)* if the hardware does not directly support *use* bits or *dirty* bits.

We assume that the Valid and Writable bits exist in hardware. Since the Use and Dirty bits don't exist, we need to emulate them in software. First, we use two spare bits in the PTE to represent our emulated versions of Use and Dirty. Second, we use another two spare bits in the PTE to track whether a page is actually valid or writable. On each pass of the clock hand, we check the (software) Use bit as usual. If Use is clear, we recycle the page. If Use is set, we clear it **and** *set the page to invalid. On the next use of the page, we get a page fault (telling us the page is in use). We use this page fault to set the software Use bit* **and** *to set the page to read-only. If the application tries to write the page, we get a page fault and can set the software Dirty bit* **and** *set the page to read-write.*

Problem 2c[4pts]: For the following problem, assume a hypothetical machine with 4 pages of physical memory and 7 pages of virtual memory. Given the access pattern:

A B C D E A A E C F F G A C G D C F

Indicate in the following table which pages are mapped to which physical pages for each of the following policies. Assume that a blank box matches the element to the left. We have given the FIFO policy as an example.

Access→		**A**	B	C	D	E	A	A	E	C	F	F	G	A	C	G	D	C	F
FIFO	1	A				E									C				
	2		B				A										D		
	3			C							F								
	4				D								G						
MIN	1	*A*																	*F*
	2		*B*			*E*					*F*		*G*						*F*
	3			*C*															*F*
	4				*D*														*F*
LRU	1	*A*				*E*								*A*					*F*
	2		*B*				*A*						*G*						
	3			*C*															
	4				*D*						*F*						*D*		

Any one of these

Problem 2d[4pts]: Suppose that we have a disk with the following parameters:

- 750GB in size
- 12000 RPM, Data transfer rate of 40 Mbytes/s (40×10^6 bytes/sec)
- Average seek time of 8ms
- ATA Controller with 2ms controller initiation time
- A block size of 4Kbytes (4096 bytes)

What is the average time to read a random block from the disk (assuming no queueing at the controller). Show your work. *Hint: there are 4 terms here.*

We need to be careful to get all of our units correct. I am going to stick with milliseconds.

T_{read} = *Controller + Seek + Rotational + Xfer Time*

$$= 2ms + 8ms + \frac{1}{2}\left(\frac{60000\frac{ms}{min}}{12000\frac{revolutions}{min}}\right) + \frac{4096bytes}{\left(40\times10^6\frac{bytes}{s}\right)\times\left(10^{-3}\frac{s}{ms}\right)} = 12.6024\ ms$$

T_{read} = *12.6024ms*

Problem 2e[2pts]: Given the same parameters from above, assume that the operating system has exploited locality by grouping related blocks together in the filesystem. As a result, the typical access pattern is not as random as in 2d. It typically retrieves 10 blocks sequentially at a time and spends only 1 ms for each seek. What is the average time to read a *single* block now? Show your work.

The key insight here is to compute the time for 10 block reads (since there is only 1 seek and 1 rotational for those 10), then divide by 10 for a single block read. Also, the seek is shorter. Since there is no queuing, each request is treated separately – thus leading to 1 controller timer/request.

T_{read10} = *10×Controller + Seek + Rotational + 10×Xfer Time*

$$= 10\times2ms + 1ms + \frac{1}{2}\left(\frac{60000\frac{ms}{min}}{12000\frac{revolutions}{min}}\right) + 10\times\frac{4096bytes}{\left(40\times10^6\frac{bytes}{s}\right)\times\left(10^{-3}\frac{s}{ms}\right)} = 24.524$$

Thus, $T_{read} = T_{read10}/10 = 2.4524ms$.

Note that other arguments are possible here – state your assumptions.

Problem 2f[5pts]: Assume that there is a queue in front of the controller. Assume that the operating system behaves as in (2e), and that the distribution of service times has C = 1.5 (i.e. not quite memoryless). Also assume that requests for blocks arrive via an exponential (memoryless) process with an average arrival rate of λ blocks/second. What is the arrival rate λ such that the queue has an average length of 10 blocks? You don't need to actually compute this number as long as you given an explicit equation for λ. *Hint: use Little's law: $L_q=\lambda T_q$ and solve for utilization. Also, remember from basic algebra that $Ax^2+Bx+C=0 \Rightarrow x=\frac{-B\pm\sqrt{B^2-4AC}}{2A}$.*

This is just a straightforward use of the M/G/1 queue formula from class:

$$T_q=T_{ser}\times\frac{1}{2}(1+C)\times\frac{u}{1-u}$$

Where T_{ser} is the service time from 2g, C=1.5, and $u=\lambda\times T_{ser}$.

Solving simply (for a queue-length of 10):

$$L_q=10=\lambda\times T_q \qquad \text{(Little's Law)}$$

$$=\lambda T_{ser}\times\frac{1}{2}(1+C)\times\frac{u}{1-u}=\frac{1}{2}(1+C)\times\frac{u^2}{1-u} \qquad \text{(Substituting and rearranging)}$$

Now, we just solve for u (and remembering that C=1.5):

$$10=\frac{1}{2}(1+C)\times\frac{u^2}{1-u}\Rightarrow 40-40u=5u^2\Rightarrow u^2+8u-8=0$$

So, using quadratic formula: $u=\frac{-8\pm\sqrt{8^2-4\times(-8)}}{2}\Rightarrow u=2\sqrt{6}-4$

Since $u=\lambda\times T_{ser}\Rightarrow\lambda=\frac{(2\sqrt{6}-4)}{T_{ser}}=\frac{(2\sqrt{6}-4)}{2.4524ms}$ *=.367 requests/ms=367 requests/sec*

Problem 3: File Systems

Please keep your answers short (one or two sentences per question-mark). *We may not give credit for long answers.*

Problem 3a[3pts]: Rather than writing updated files to disk immediately when they are closed, many UNIX systems use a delayed *write-behind policy* in which dirty disk blocks are flushed to disk once every 30 seconds. List two advantages and one disadvantage of such a scheme:

Advantage 1: *The disk scheduling algorithm (i.e. SCAN) has more dirty blocks to work with at any one time and can thus do a better job of scheduling the disk arm.*

Advantage 2: *Temporary files may be written and deleted before they ever get to disk.*

Disadvantage: *File data may be lost if the computer crashes before data is written to disk.*

Problem 3b[2pts]: Describe a technique that can be used to mitigate the disadvantage of (3a) without losing the advantages of (3a). Be explicit here.

Use an NVRAM (Non volatile RAM). The idea here is that data which is written is placed in the NVRAM right away, before telling the user that it has been written. Since this process is as fast (or nearly so) as writing to RAM and doesn't involve moving the disk arm, both of the above advantages remain. Since it is nonvolatile, a disk crash will not lose data.

Problem 3c[4pts]: In the following, compare the performance of a FAT file system to a more primitive system that uses "linked allocation" (where file blocks are linked together via pointers in each block) with respect to number of disk accesses. *Be sure to state any assumptions that you are making in terms of what (if anything) is memory resident.*

a) Which is faster for *random* access? Explain. (Assume the file in question is large):

The FAT file system is much faster for random access. You read the FAT (once), trace through the links in memory to find the desired block, then read the block from memory (effectively 2 disk accesses). With the linked-allocation, you need to read through (n-1) blocks to randomly access the n^{th} block of the file.

b) Which is faster for *sequential* access? Explain. (Again assume a large file):

This depends on your assumptions. If you are unable to keep the FAT in memory, the linked allocation is faster, since the next block is contained in the previous block. If, on the other hand you assume that the FAT stays in memory, then these two techniques are of equal performance or the FAT could be faster if requests were queued enough to allow better disk scheduling. Note that you needed to state your assumptions to get full credit here.

Problem 3d[4pts]: Consider a file system with 4096 byte blocks and 32-bit disk and file block pointers. Each file has 13 direct pointers, 4 singly-indirect pointers, a doubly-indirect pointer, and a triply-indirect pointer. In the following be explicit about your work:

a) What is the maximum disk size that can be supported? Explain.

Since block pointers are 32 bits, total size = $2^{32} \times 4096 = 2^{44}$ bytes maximum.

b) What is the maximum file size? Explain.

Since pointers are 32 bits (4 bytes), one indirect block can address 1024 blocks. Thus, maximum file= $4096 \times (13 + 4 \times 1024 + 1024^2 + 1024^3) = 4402358308864$ bytes

c) Give some reasonable assumptions and compute the number of **inodes** that can fit into a disk block.

An inode has 13 + 4 + 1 + 1=19 pointers = 76bytes. Assuming various other file information such as ACLs, etc, we round up to 128bytes and can fit 32 inodes/block

Problem 3e[4]: The Fast File System (FFS) of BSD 4.2 introduced several mechanisms for improving performance of the BSD 4.1 system. Name 2 of these *and explain why they improve performance*:

1. *Data from files allocated to be mostly sequential by looking for large chunks of empty blocks within a cylinder; this optimization requires blocks to be kept in reserve in order to perform this optimization. This improves performance because sequential file access translates to sequential access on disk (fast).*
2. *Placing of inodes for a file in the same cylinder group as the data⇒means that finding successive sequential chunks for files is quick.*
3. *inodes and data for files within a directory placed in same cylinder group⇒means that operations to files within a directory (such as listing, etc) are fast.*

Problem 3g[3pts]: On a single UNIX machine, if some program B reads a block of a file after it has been updated by another program A, the copy of the file block B reads will include A's updates. In NFS this behavior is not guaranteed. Assuming that there are no failures, why doesn't NFS necessarily provide such update semantics when A and B are run on different machines? What semantics does it provide instead?

In NFS (versions through 3), cached data is updated only periodically. Thus it is possible that B could read old data for a while after A has finished updating it. The semantics are those of "weak coherence". Just telling us what would happen here is sufficient.

Problem 3h[3pts]: The Andrew File System (AFS) solves the above problem (3g) using state information it maintains at the server. What state is kept? How is that state used to solve the problem?

An AFS server keeps track of which clients have read-only copies of particular files. Thus, when one client writes data (and closes the file so that the data is flushed to the server), the server contacts each of the clients that have cached copies of the file and tells them to invalidate the file.

Problem 3i[4pts]: At time XZ, the request queue for a disk contains the following requests in [track:sector] form:

[10:5], [22:9], [20:21], [21:9], [2:10], [40:45], [6:7], [38:9] …. (in this order).

Assume that the disk head is currently positioned over cylinder 20. What is the sequence of reads under the following head scheduling algorithms?

a) Shortest Seek Time First:

Only the track numbers matter here (since you can't really do inter-track optimization). So:

[20:21], [21:9], [22:9], [10:5], [6:7], [2:10], [38:9], [40:45]

b) SCAN (initially moving upwards):

Moving initially upwards (increasing tracks), we have:

[20:21], [21:9], [22:9], [38:9], [40:45], [10:5], [6:7], [2:10]

Problem 4: Security

Problem 4a[3pts]: Assume Alice and Bob have never met. Explain how Alice can use a public key infrastructure (PKI) to prove her identity to Bob. *Hint: make sure to prevent replay attacks.*

The essential insight here is that Alice needs to sign something with her private key that can be verified with her public key. To prove to Bob what her public key is, she has a certificate authority sign her public key. This is the PKI.

Once Bob is certain of Alice's public key, proving her identity is easy:she signs a message (like "Hi! I'm Alice") with her private key asserting that she is Alice. To avoid replay attacks, she asks Bob for a random value which she also includes in that message.

Problem 4b[3pts]: Explain how to utilize a PKI to establish a private session key between two parties for fast symmetric encryption.

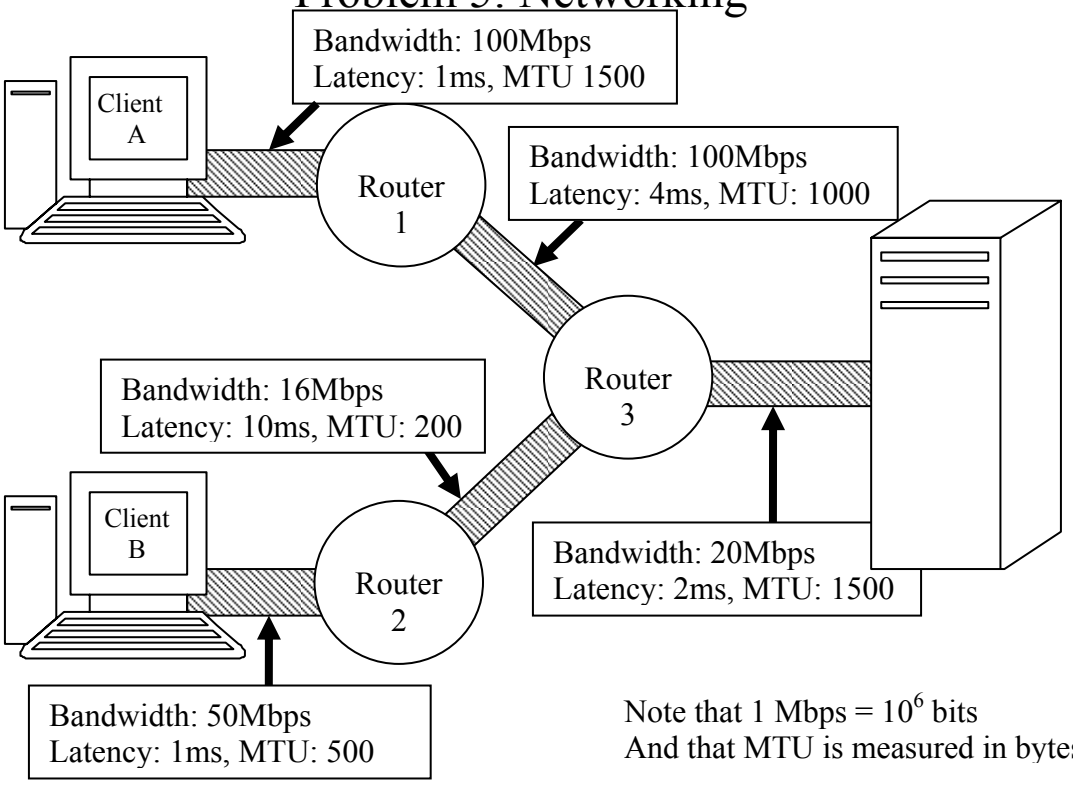
Both parties use the PKI to make sure that they have each other's appropriate public key. Then, they each choose a random number, encrypt it with the public key of the other party, and send it off. Now, both of them have the same two random numbers (which are private). They can combine them anyway they like, then use the result as a private session key.

Problem 4c[4pts]: What are two desirable properties for secure hash functions (ignoring the property where half the bits change for small changes in input). Why are these properties important for signatures?

1. *Given message M_1 with hash $h_1=h(M_1)$, it is computationally infeasible to find another message M_2 such that $h_1=h(M_2)$*
2. *Further, it is computationally infeasible to find any two messages M_A and M_B such that $h(M_A)=h(M_B)$.*

The type of signature discussed in class involves encrypting the hash of a document with the private key of the signer. The properties above prevent the possibility of a single signature from referring to multiple documents (which would defeat the purpose of a signature).

Problem 5: Networking



The above figure illustrates a network in which two clients (Client A and Client B) route packets through the network to the server and to each other. Each link is characterized by its Bandwidth, one-way Latency, and Maximum Transfer Unit (MTU) in *bytes*. All links are full-duplex (can handle traffic in both directions at full bandwidth).

To send data packets, a user-level process on one of the Clients issue a system call. As a result, the OS copies the data first to a kernel buffer, then uses DMA to copy the data to the network controller board. Immediately after receiving all of the data, the network controller generates one or more packets and sends them. A router fully receives a packet, after which it takes 1ms to begin forwarding the packet to the next hop. Assume that no packets will be lost unless otherwise noted.

At the destination, the receiving network controller DMAs the received data to memory. When the last bit arrives, it interrupts the CPU, which copies the data to user space. The CPU then sends back a one-byte acknowledgement to the sending OS. *For simplicity, you may treat acknowledgements as if they are zero bytes long. Assume that a sender uses a window-based protocol unless noted.*

Assume that all DMA operations can overlap with processor actions such as copying and interrupt handling. Also assume that DMA operations do not have to generate interrupts on completion.

System Parameters:

- Time to process an interrupt, T_{int}=100 μs
- Time to copy or DMA one byte, T_{copy}= 100 ns
- Retransmission timeout, T_{retran}= 400 ms

Problem 5a[4pts]: Under ideal circumstances (and ignoring interrupt and copying overheads and window size), what is the maximum bandwidth that Client A can send data to the server without causing packets to be dropped (Assuming that the headers are of zero length and that routers can handle data at wire speed)? How about Client B? Explain.

This is simply a matter of finding the slowest link (bottleneck) along the path.

So: Client A, maximum bandwidth available is 20Mbps (bottleneck from router 3→server)
Client B, maximum bandwidth available is 16Mbps (bottleneck from router 2→3)

Problem 5b[4pts]: Keeping in mind that TCP/IP involves a total header size of 40 bytes (for TCP + IP), what is the maximum *data* bandwidth that Client A could send to the server through TCP/IP? Explain. What about Client B? Explain. Again, ignore interrupt and copying overheads. *Hint: Assume that a sender runs a protocol that computes the proper maximum packet size to avoid fragmentation within the network.*

Here, the key is to compute the minimum MTU along each path. This gives the packet size that we will be using as a result. Then, the data bandwidth needs to be scaled back by subtracting 40 bytes for every packet.

$$A\rightarrow server:\ min\ MTU=1000bytes.\ \ Thus,\ bandwidth=20Mbps\times\frac{1000-40}{1000}=19.2\ Mbps$$

$$B\rightarrow server:\ min\ MTU=200bytes.\ \ Thus,\ bandwidth=16Mbps\times\frac{200-40}{200}=12.8\ \ Mbps$$

Problem 5c[6pts]: Now, assume that copying and interrupt overheads impact the maximum rate at which data can be sent or received at the endpoints (clients or servers). Routers continue to be fully pipelined and senders avoid fragmentation as in 5b. What is the maximum *data* bandwidth that Client A can transfer to the server without dropping packets? Explain. What about Client B? *Hint: make sure to account for the impact of the 40 byte TCP/IP header.*

Since DMA can be overlapped with the CPU, the key is to see what bottlenecks prevent the CPU from acting. At the sender, we have one data copy from user→kernel space and one interrupt (ACK). At the destination, we have one interrupt (packet arrival) and one data copy from kernel→user space. Thus, we have the same constraints on both sides. Notice that only the data portion of the packet is copied to and from user space.

Client A→Server overhead/packet = copy+ interrupt = (960 bytes × 100ns/byte)+100μs=196μs
Without network limits. Client A→Server =(960bytes×8bits/byte)/196μs=38.4Mbps.
However, network limits Client A→Server bandwidth to 19.2Mbps (see 5b)

Client B→Server overhead/packet = copy+ interrupt = 160 bytes × 100ns/byte+100μs=116μs
Without network limits. Client B→Server =(160bytes×8bits/byte)/116μs=11Mbps.
Since network permits 12.8Mpbs (see 5b), max Client B→Server bandwidth is 11Mbps.

Problem 5d[5pts]: Now, accounting for copying and interrupt overheads, what is the total time to *reliably* send a single maximal-sized packet from Client A to a user-level process on the server without fragmentation? Account for all latencies (including the reception of an ACK, which can be assumed to be zero length but which will generate an interrupt). Explain your work.
We need to compute the path from A to server with the data as well as the time back for the ACK. Note that the data involves raw latency, in addition to copying, DMA, and transfer time along the ling. The ACK has only latency elements. Note that the Copies between user space and kernel space involve the data payload (packet size – 40 bytes). We will assume that the DMAs involve the complete packet (although this is perhaps open to interpretation). Note that we convert everything to ms. We could easily have converted to seconds instead….

$$\text{Client A}\rightarrow\text{Server: } Copy + DMA + [A\rightarrow R1]_{Latency} + [A\rightarrow R1]_{Xfertime} + R1_{Latency} + [R1\rightarrow R2]_{Latency} + [R1\rightarrow R2]_{Xfertime} + R2_{Latency} + [R2\rightarrow Ser]_{Latency} + [R2\rightarrow Ser]_{Xfertime} + DMA + Int + Copy$$

$$\text{Server}\rightarrow\text{Client A: } [Ser\rightarrow R2]_{Latency} + R2_{Latency} + [R2\rightarrow R1]_{Xfertime} + R1_{Latency} + [R1\rightarrow A]_{Latency} + Int$$

$$Total = 2\times(Copy+DMA+Int) + 2\times([A\rightarrow R1]_{Latency}+R1_{Latency}+[R1\rightarrow R2]_{Latency}+R2_{Latency}+[R2\rightarrow Ser]_{Latency}) + [A\rightarrow R1]_{Xfertime}+[R1\rightarrow R2]_{Xfertime}+[R2\rightarrow Ser]_{Xfertime} =$$

$$= 2\times(960bytes\times100ns/byte\times10^{-6}ms/ns + 1000bytes\times100ns/byte\times10^{-6}ms/ns + 100\mu s\times10^{-3}ms/ns) + 2\times(1ms + 1ms + 4ms + 1ms + 2ms) + 1000bytes/(100\times10^{6}bytes/sec\times10^{-3}s/ms) + 1000bytes/(100\times10^{6}bytes/sec\times10^{-3}s/ms) + 1000bytes/(20\times10^{6}bytes/sec\times10^{-3}s/ms)$$

$$= 2\times(.296ms) + 2\times(9ms) + .01ms + .01ms + .05ms = 18.662ms$$

Problem 5e[4pts]: Assume the conditions from 5d. Client A sends a continuous stream of packets to the server (and no other clients are talking to the server). How big should the send window be so that the TCP/IP algorithm will achieve maximum bandwidth without dropping packets? Explain. *Hint: don't forget to account for the 40 bytes of header.*

*We merely need to compute the (bandwidth×latency) product, since this is the amount of data we need outstanding to completely fill up the pipeline without putting too much into the network. Since the send window is for real data (not headers), we use the bandwidth numbers from 5c with the latency from 5d. We need to make sure to cancel terms (watch out for seconds vs ms!). Note: Technically speaking, the user copy doesn't really impact the window. So, you should really ignore the term of 2×(.096ms) in the answer to 5d, yield*ing a *correct window of:*

$$\text{Send window} = [19.2\times10^{6}\ bits/s \times \frac{1}{8} bytes/bit] \times [18.47ms\times(10^{-3}s/ms)] = 4432\ 8.8\ bytes$$

Since a fraction of a byte doesn't make sense, we round down to 44328.

Note that we also gave you full credit *if you simply multiplied answer from 5d with 19.2Mbps (which would be correct if window handled by user—not really true):*

$$\text{Send window} = [19.2\times10^{6}\ bits/s \times \frac{1}{8} bytes/bit] \times [18.662ms\times(10^{-3}s/ms)] = 44788.8\ bytes$$

We can round down 44788 bytes.

[This page intentionally left blank]