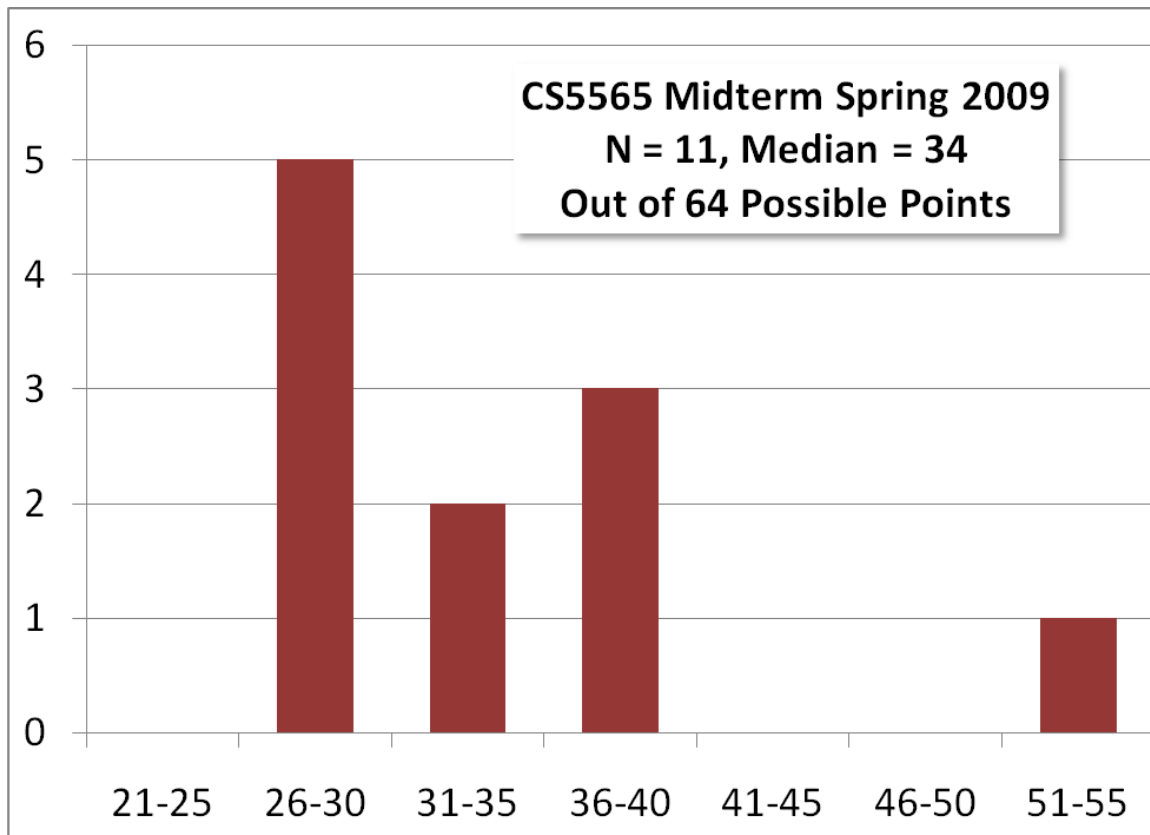


CS 5565 Midterm Solutions

11 students took the midterm. Below is a histogram of scores.

Considering the small number of questions I received before the midterm, I am surprised by these disappointingly low scores.

Problem	1	2	3	4	Total
Possible	20	16	20	8	64
Min	4	4	2	0	26
Max	16	16	16	8	52
Median	9	12	5	6	34
Average	9.8	11.5	7.2	4.7	33.2
StDev	4.1	4.1	5.2	3.6	7.9



Solutions are shown in this style.

Comments are shown in this style.

1 Reliable Data Transmission and TCP (20 pts)

a) (8 pts) Layering vs. Performance.

Name two *performance-enhancing* techniques TCP implementations must employ that directly result from the fact that these implementations reside in a *layered architecture*. For each technique, describe the relationship to layering!

i. (4 pts) Technique #1

One technique is Nagel's algorithm. If an application passes down single byte requests for transmissions to the TCP layer, a TCP implementation will send the first byte, then aggregate the sending of the additional bytes in a single segment.

ii. (4 pts) Technique #2

A second technique is the avoidance of silly window syndrome. If a receiver application drains a full receive buffer, a TCP implementation will not immediately advertise the open window space, in order to avoid forcing the sender to send many small segments to use up the newly advertised window space.

Both techniques are a) performance-enhancing (they're not required for correctness) and b) directly result from layering where a TCP implementation interacts with a layer above it.

Other examples include techniques that result from the fact the TCP is run over IP: for instance, one could argue that congestion control must be implemented by TCP because it is not part of IP's service model. However, congestion control by itself is not needed because of layering.

I gave partial credit for "flow control," if it was made clear that in which way it can be considered related to performance and layering.

b) (4 pts) Autotuning

Windows Vista added support for TCP autotuning. According to Microsoft, Vista "tunes the TCP receive window size based on the bandwidth delay product (BDP) and the rate at which the application reads data from the connection." Microsoft claims to have observed throughput improvements of up to 10x.

Under which circumstances would you expect such throughput improvements to occur? Be specific!

You would expect benefits over connections whose available bandwidth-delay product exceeds the default receiver window size. This occurs in practice for underutilized wide area connections that have large amounts of bandwidth available and incur a high propagation delay.

c) (4 pts) Download Accelerators

Some so-called download accelerators purport to increase download speeds by opening multiple TCP connections to download files (for instance, by simultaneously retrieving different byte ranges using HTTP's Range: headers) Define why and under which circumstances a user would benefit from these accelerators!

Download accelerators provide a benefit if there is contention between different TCP flows in the bottleneck router between source and destination. For flows with the same round-trip time, TCP's congestion control mechanism will give roughly equal amounts of bandwidth to the competing flows, thus using more flows results in higher aggregate bandwidth.

Another argument that could be made is that if the BDP is larger than the receiver window, multiple connections could provide multiple receiver windows. See also question b).

Note that the question asked both for why users would benefit and under which circumstances such benefit would occur.

Some stated that multiple connections, for instance, multiple non-persistent HTTP connections, can reduce latency. I gave partial credit for that answer. Such latency reduction would only be significant for the download of very, very small files.

d) (4 pts) USB

The USB specification supports a number of modes for data transfer, including one mode that provides reliable data transfer (this mode is used, for instance, by mass storage devices such as USB hard drives or flash drives). However, rather than using a more general sliding window protocol with adjustable window sizes, a USB host controller uses a single "data toggle synchronization" bit to implement a simple alternating bit protocol in hardware.

Why do you think the designers of USB made this choice, despite the known weaknesses of using a single bit only?

An alternating bit protocol can be viewed as a sliding window protocol with a window size of 1. To work efficiently, the bandwidth-delay product should be less or equal than window size/RTT. USB is designed to work over small distances with little propagation delay, so the designers probably felt that this was not a problem. Using a single bit of course simplifies the designer of USB host controllers, lowering their cost.

I gave partial credit for stating that the motivation may have to do with USB's inability to reorder data (something which an alternating bit protocol cannot handle), because the same limitation is true for general sliding window protocols with finite window sizes, and the question asked for why a ABP was used "rather than using a more general sliding window protocol."

2 Congestion Control and DCCP (16 pts)

a) (8 pts) Modeling congestion control

Bansal and Balakrishnan in [MIT-LCS-TR-806] introduce the following generalization of the additive increase, multiplicative decrease congestion control model used in TCP into a binomial model that uses two parameters k and l to describe a class of congestion control schemes.

$$I: w_{t+R} \leftarrow w_t + \alpha/w_t^k; \alpha > 0$$

$$D: w_{t+\delta t} \leftarrow w_t - \beta w_t^l; 0 < \beta < 1$$

α and β are constants in this model. w_t is the window size at time t . I stands for Increase after the successful receipt of an acknowledgement, D stands for Decrease after a loss event. Consider TCP's slow start and congestion avoidance phases. What values are used for exponents k and l and for α and β ?

	k	l	α	β
TCP in Slow Start	-1	1	1	1/2
TCP in Congestion Avoidance	0	1	1	1/2

As an aside, Bansal and Balakrishnan take this model further and prove that any scheme for nonnegative k and l for which $k + l > 0$ provides fairness for competing streams.

Note that the question pointed out that alpha and beta are constants.

b) (4 pts) DCCP

What motivation led the developers of DCCP to propose to implement congestion control as a new layered protocol, rather than leaving its implementation to individual applications?

The key motivation is that applications need congestion control to protect the network (and thus themselves) in the face of congestion, but do not wish to implement it themselves, because that would add additional complexity to these applications. Moreover, congestion control algorithms have proved to be difficult to design. Secondly, implementing congestion as a separate layer allows its evolution and development independent of the applications it serves. Note that congestion control cannot be implemented as a library only.

c) (4 pts) Sequence Numbers in DCCP

We discussed in class the need for sequence numbers to implement reliable data transmission.

- i. (2 pts) DCCP does not implement reliable data transmission, yet still uses sequence numbers. Why?

DCCP implements congestion control, which in the absence of network assistance requires loss detection, which in turn requires sequence numbers.

- ii. (2 pts) Sequence numbers that wrap around make it impossible to distinguish delayed old packets from new ones. Even though DCCP does not provide any reliability guarantees to the applications, it is concerned with minimizing the wrap-around of sequence numbers. Why?

Applications using DCCP still need to be able to distinguish old data from new: as the paper points out, they will typically prefer new data to old data in many scenarios. That's the primary reason. A secondary reason is that connections are more vulnerable to attacks.

Many of you quoted the question back to me, saying that "because old delayed packets cannot be distinguished from new ones." You needed to provide a reason for why this is a problem.

3 Implementation Issues (20 pts)

a) (8 pts) Server Models

We had discussed the following options for implementing servers in class:

- A – for each connection, a new process is spawned
- B – for each connection, a new thread is spawned
- C – a constant number of processes that handle connections, with each process dedicated to one connection
- D – a constant number of threads that handle connections, with each thread dedicated to a connection
- E – a single thread or a small number of threads that multiplex multiple connections using select() or a similar mechanism

Consider the following applications and state which model you would prefer, and name one reason why. Justify your reason briefly.

- i. (2 pts) A multi-tier intranet application using the server-centric ZK AJAX framework for Rich Internet Applications (RIA) you reverse-engineered in problem set 2.

A threaded model (D or B) would be preferred since state must be maintained and shared between requests, excluding a per-process model. The complexity of operations (e.g., its multi-tiered nature) would make E difficult because of the need to stack-rip. If many requests trigger long-running activities, consider B; if robustness under load is a concern, use D. In practice, uses D.

- ii. (2 pts) A local DNS name server.

A local DNS server receives requests from local clients and performs recursive DNS lookups, caching the results. Needs a cache, so single-process is preferred. Processing per request is short and uncomplicated, and many requests must be handled. Clear preference for E. In practice, BIND9 uses E with a thread pool to exploit multiple CPUs where present.

- iii. (2 pts) The CS department's web server.

Robustness is important since the web server handles many types of dynamic content written by many people, plus static content. Strong preference for C (which is also Apache's default model). Single-process models such as B, D, and E are ruled out because of the need to support different frameworks (PHP, WSGI, etc.). A and B are ruled out since process and even thread start overhead is too large for typical HTTP request.

- iv. (2 pts) The campus mirror hosting large DVD images of Linux (mirror.cc.vt.edu).

Would prefer something close to A or B – connections are long-lived, so startup overhead is unimportant. Each request involves I/O, best handled in its own execution context. Traditionally, ftpd used A, though modern FTP servers use B as well. C and D would also work and are preferred if limiting the number of users is desired, but it may be difficult to find the right number of threads/process to pre-spawn, and difficult to find an appropriate limit that does not lead to underutilization. E is also possible, and could lead to higher peak performance, but would require substantial additional implementation effort.

These questions had multiple possible answers. I awarded 0, 1, or 2 points based on how much sense your justification made in the context of the given application scenario.

b) (12 pts) AZ-SDP

Considering the following snippet, which may have come from a web server serving large static files:

```
void copy_file_to_socket(int conn_to_client, const char *filename)
{
    static char buf[128];
    int fd = open(filename, O_RDONLY);
    int bread;
    while ((bread = read(fd, buf, sizeof buf)) > 0)
        send(conn_to_client, buf, bread);
    close (fd);
}
```

where error checking has been omitted for brevity.

- i. (4 pts) What performance would you expect assuming that `conn_to_client` is a socket referring to a high-speed interconnect and that AZ-SDP is being used?

Extremely dismal performance, likely. The client immediately touches the buffer it passes to the socket call in order to copy in the next chunk of data from disk. This will cause a page fault in the AZ-SDP model. Refer to Figure 6(b), which shows that AZ-SDP performs much worse than the copying implementation for small message sizes.

- ii. (8 pts) Optimize this code for AZ-SDP!

At a minimum, you need to increase the buffer size drastically. A smarter approach would be to overlay disk transfers and socket sends, like so:

```
void copy_file_to_socket(int conn_to_client, const char *filename)
{
#define CHUNK 65536
    static char buf[128*CHUNK];
    int fd = open(filename, O_RDONLY);
    int bread;
    char *p = buf;
    while ((bread = read(fd, p, CHUNK)) > 0) {
        send(conn_to_client, p, bread);
        p += bread;
        if (p + CHUNK >= buf + sizeof buf)
            p = buf;
    }
    close (fd);
}
```

This will read 64 Kbyte from disk, then start the socket transfer, then read the next 64K into a non-overlapping buffer, and so on.

4 Know Your Current Events (8 pts)

Conficker C Worm

As of today, April 1, 2009, the Conficker C worm has emerged as a potential threat to stability on the Internet. This worm, which by some estimates has infected over 11.9M machines, is programmed to contact a sample of 50,000 randomly generated domain names every day in order to receive updates for its code. An alliance of researchers and industry, the Conficker Cabal, has formed to combat the worm. This group recently decoded the algorithm the worm uses to compute these domain names based on the current date and time.

One possible approach to fight the worm would be to reprogram the root DNS servers to redirect these domain names away from the servers that would provide these potentially malicious updates. Since there are only 13 distinct root servers (presented by a total of 169 IP-anycast-based replicas), this could be done relatively cheaply.

Would that approach work? Justify your answer!

It would not work. Root servers are rarely contacted because local name servers cache the addresses of the top-level domain servers, then contact those servers directly.

Instead, the Conficker Cabal must cooperate with the domain name registries that run the top-level domain servers of all possible domains the worm could use (involving over 110 domains currently). This makes it a much more challenging task, but not an infeasible one.

Some of you stated that the domain names cannot be known – but the question provided the information that the algorithm to compute them was decoded.