CSCI 415 Data Communication Networks
Homework 5
Solution

Saad Mneimneh
Visiting Professor
Hunter College of CUNY

Problem 1: File Transfer
Consider a simple UDP-based protocol for requesting files (based somewhat loosely on the Trivial File Transfer Protocol, TFTP). The client sends an initial request, and the server answers (if the file can be sent) with the first data packet. Client and server continue with a stop and wait transmission mechanism.

(a) Describe a scenario by which a client might request one file but get another; you may allow the client application to exit abruptly and be restarted with the same port.

ANSWER:

1. The client sends a request for file "foo"
2. The request arrives at the server
3. The client abort locally, but restarts with the same port
4. The client sends a new request for file "goo"
5. The second request is lost
6. The server responds with the first data packet of "foo", answering the only request is has actually received
7. The client start receiving "foo" thinking its "goo"

(b) Propose a change in the protocol that will make this situation much less likely.

ANSWER: Requiring the client to use a new port number for each separate request would solve the problem. To do this, however, the client would have to trust the underlying operating system to assign a new port number each time a new socket was opened. Having the client attach a timestamp or random number to the file request, to be echoed back in each data packet from the server, would be another approach fully under the applications control (something like in TCP, but see last problem).

Problem 2: Internet checksum
In ones complement arithmetic, a negative integer $-x$ is represented as the complement of x, that is each bit of x is inverted (that's $1111111111111111 - x$, hence the name). Therefore, 5 is 0000000000000101, and -5 is 1111111111111010. Similarly, 3 is 0000000000000011 and -3 is 1111111111111100. When adding numbers in ones complement, a carryout from the most significant bit must be added to the result (unlike in twos complement that is used in most machines). Therefore, if we add 1111111111111010 and 1111111111111100 ignoring the carry we get 1111111111110110. In ones complement arithmetic, the fact that this operation caused a carry from the most significant bit causes us to increment the result, giving 1111111111110111, which is the ones complement representation of -8 (obtained by inverting the bits), as we would expect.

The Internet checksum algorithm is the following: data is checksummed as a sequence of 16 bit integers. They are added together using a 16 bit ones complement arithmetic and the checksum is obtained as the ones complement of the result. The receiver adds all 16 bit words received including the checksum and, therefore, must obtain 1111111111111111 (the ones compliment of 0000000000000000) which is also a zero in ones complement).

This problem is designed to help you appreciate the Internet checksum algorithm. We start by asking the following questions:

- Beside the ease of implementation in software, how do we justify the use of a checksum?
- If the use of a checksum is justified, why ones compliment arithmetic?

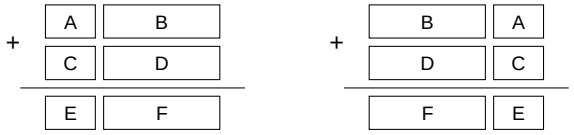
(a) Why a checksum (incremental updates)? Assume a router needs to change a word in the header from x to y before forwarding the packet, e.g. decrementing the TTL (time to live) of the packet. Argue that the new checksum can be obtained by simply adding $x + \bar{y}$ to the previous checksum, where $\bar{y}$ is the ones complement of y.

ANSWER: The checksum is $s = \overline{\sum_i x_i + x}$. If x is to be replaced by y, we need to update s to s', where $s' = \overline{\sum_i x_i + y}$. Therefore $s' = \overline{\sum_i + x + \bar{x} + y} = \overline{\sum_i + x} + x + \bar{y} = s + x + \bar{y}$.

(b) Why ones compliment (endian independent)? Little endian computers store numbers with the least significant byte first (Intel processors for example). Big endian computers put the most significant byte first (IBM mainframes for example). Show that adding 16 bit numbers in ones complement arithmetic is endian independent. More specifically, if $AB{+}CD{=}EF$, then $BA + DC = FE$ (this is not necessarily true if ones complement arithmetic is not used), where A, B, C, D, E, and F are 8 bit numbers and, for instance, AB is a 16 bit number with A being the most significant byte and B being the least significant byte.

ANSWER: One can actually prove a stronger statement. Let $x + y = z$ in ones complement arithmetic. If x' and y' and z' are circular shifts (by the same

amount) of x, y, and z respectively, then $x' + y' = z'$ in ones complement arithmetic.



case 1	No carry from F to E and no carry out of E	No carry from E to F and no carry out of F
case 2	There is a carry from F to E and no carry out of E	There is a carry out of F that wraps around and gets added to E (which produces no more carry)
case 3	symmetric to case 2	
case 4	There is a carry from F to E and a carry out of E that wraps around and gets added to F (not necessarily in that order)	There is a carry from E to F and a carry out of F that wraps around and gets added to E (not necessarily in that order)

Problem 3: Wait before closing the connection
Read about the netstat Unix/Window utility (e.g. the man page for netstat on Unix). Use the netstat utility to see the state of your local TCP connections. Find out how long closing connections spend in the waiting state. For instance, you can use your server/client program from the second homework and observe how long it takes the client to close the connection. What happens if another client contacts the server while the previous client is in the wait state (observe the port numbers).

ANSWER: 2 minutes. Each client gets a new port number.

Problem 4: Designing a protocol header
You are hired to design a reliable byte-stream protocol that uses a sliding window (like TCP). This protocol will run over a 1 Gbps network. The RTT of the network is 140 ms, and the maximum segment life is 60 seconds. How many bits would you include in the Advertised Window and Sequence Number fields of your protocol header?

ANSWER: The window size (in bytes) must be RTTxBandwidth = $\frac{10^9}{8} \times 0.14 = 17500000$ bytes. We need therefore, 25 bits for the advertized window (allows a maximum window size of 33554431 bytes).

In 60 seconds, $\frac{19^9}{8} \times 60 = 7500000000$ bytes can be transmitted. They must all have unique sequence numbers. Therefore, we need 33 bits for the sequence numbers.

Problem 5: Sequence numbers and fooling the server
If server A accepts a TCP connection from client B, then during the three-way handshake A sent its initial sequence number to B and received an acknowledgment from it. Therefore, another client C can pretend to be B in the following way:

- C sends a SYN packet to open TCP connection to A pretending to be B
- A sends its SYN+ACK packet with its initial sequence number as part of the three-way handshake to B, and waits for the acknowledgement
- B ignores, i.e. does not respond to A's SYN+ACK packet
- C sends an acknowledgement to A pretending to be B

One would argue, however, that C cannot properly acknowledge A because it does not receive A's initial sequence number.

The algorithm for choosing the initial sequence number gives unrelated hosts a fair chance of guessing it. Specifically, A selects the initial sequence number based on a clock value at the time of connection. RFC 793 specifies that this clock value be incremented every 4 μs; common Berkeley implementations once simplified this to incrementing by 250000 once per second.

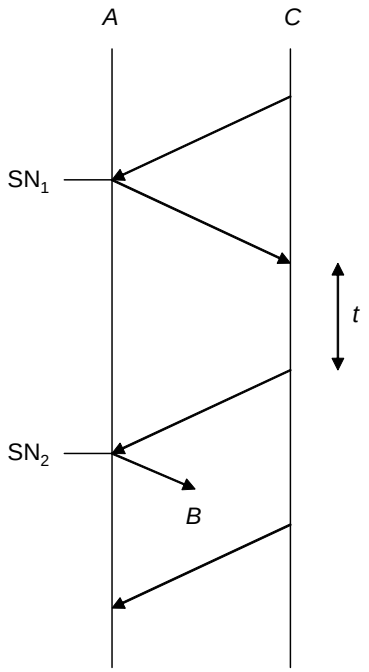
(a) Given this simplified increment-once-per-second implementation, explain how C could masquerade as B in at least the opening of a TCP connection.

ANSWER:

1. C connects to A, and gets As current clock-based sequence number SN_1
2. C sends a SYN packet to A, pretending to be B
3. A sends SYN+ACK, with SN_2 to B, which we are assuming is ignored
4. C makes a guess at SN_2, eg SN_1 plus some suitable increment (RTT+t seconds passed, see diagram below), and sends the appropriate ACK to A (pretending to be B), along with some data that has some possibly malign effect on A
5. C does nothing further, and the connection either remains half-open indefinitely or else is reset, but the damage is done

(b) Assuming real RTT can be estimated to within 40 ms, about how many tries would you expect it to take to implement the strategy of part (a) with the unsimplified "increment every 4 μs" TCP implementation?

ANSWER: In one 40ms period there are 40ms/4μsec = 10000 possible sequence numbers; we would expect to need about 10000 tries.


A
C
SN1
t
SN2
B